In [1]:
```python
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```python
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
import statsmodels
import statsmodels.api as sm
import sklearn
from sklearn.model_selection import train_test_split
```

In [3]:
```python
cars = pd.read_csv(r"C:\Users\dell\Downloads\CarPrice_Assignment.csv")
```

In [4]:
```python
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
```

In [5]:
```python
cars.head(10)
```

Out[5]:

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	...	enginesize	fuelsy
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	front	88.6	...	130	
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	front	88.6	...	130	
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	front	94.5	...	152	
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	front	99.8	...	109	
4	5	2	audi 100ls	gas	std	four	sedan	4wd	front	99.4	...	136	
5	6	2	audi fox	gas	std	two	sedan	fwd	front	99.8	...	136	
6	7	1	audi 100ls	gas	std	four	sedan	fwd	front	105.8	...	136	
7	8	1	audi 5000	gas	std	four	wagon	fwd	front	105.8	...	136	
8	9	1	audi 4000	gas	turbo	four	sedan	fwd	front	105.8	...	131	
9	10	0	audi 5000s (diesel)	gas	turbo	two	hatchback	4wd	front	99.5	...	131	

10 rows × 26 columns

Understanding the Data Dictionary

The data dictionary contains the meaning of various attributes; some non-obvious ones are:

In [6]:
```
# Symboling: -2 (least risky) to +3 most risky
# 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
```

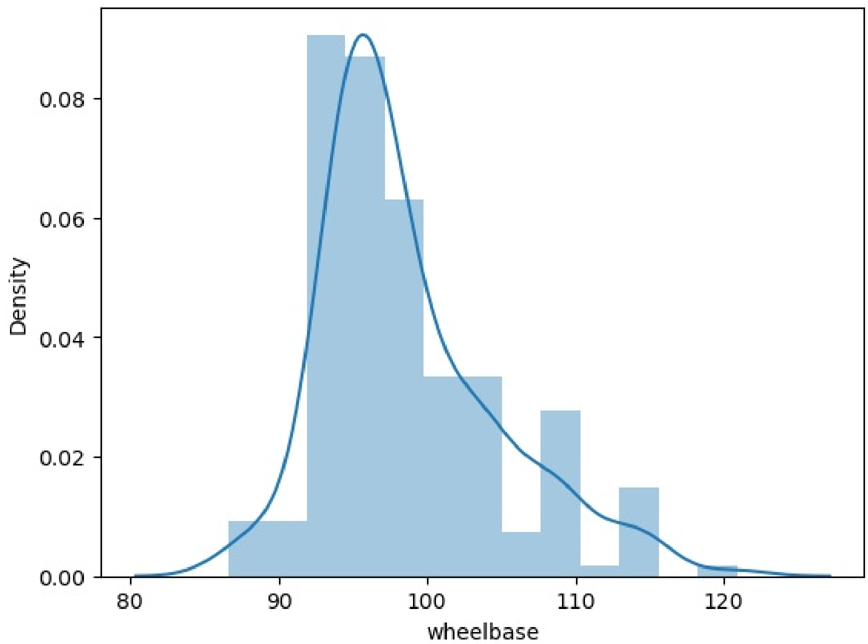
In [7]:
```
# Aspiration: An (internal combustion) engine property showing whether the oxygen intake is standard (through a
# pressure) or through turbocharging (pressurised oxygen intake)

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

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

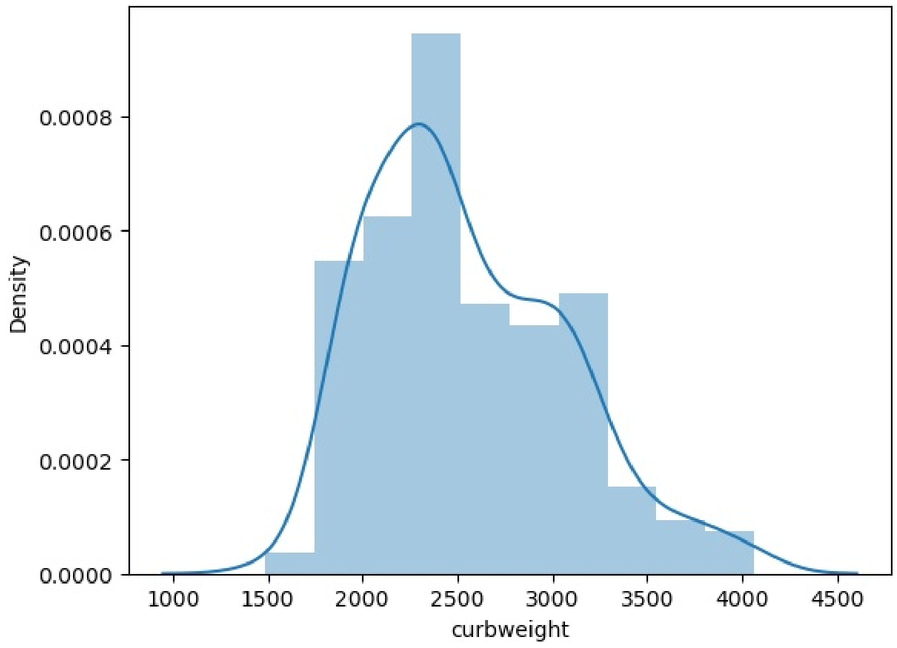
In [8]:
```
# Aspiration: An (internal combustion) engine property showing whether the oxygen intake is standard (through a
# pressure) or through turbocharging (pressurised oxygen intake)

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

Out[8]:
```
std      168
turbo     37
Name: aspiration, dtype: int64
```

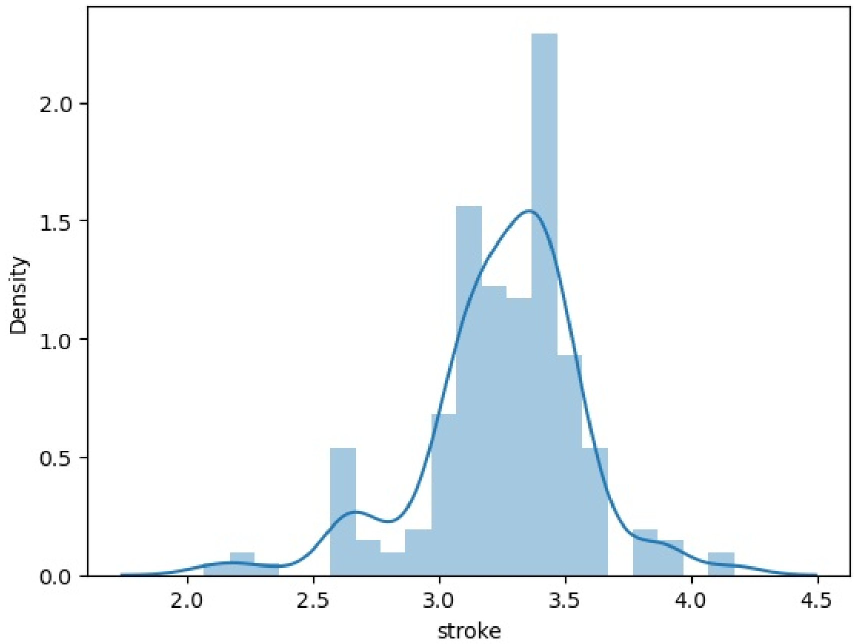
In [9]:
```
# Wheelbase: distance between centre of front and rarewheels
sns.distplot(cars['wheelbase'])
plt.show()
```



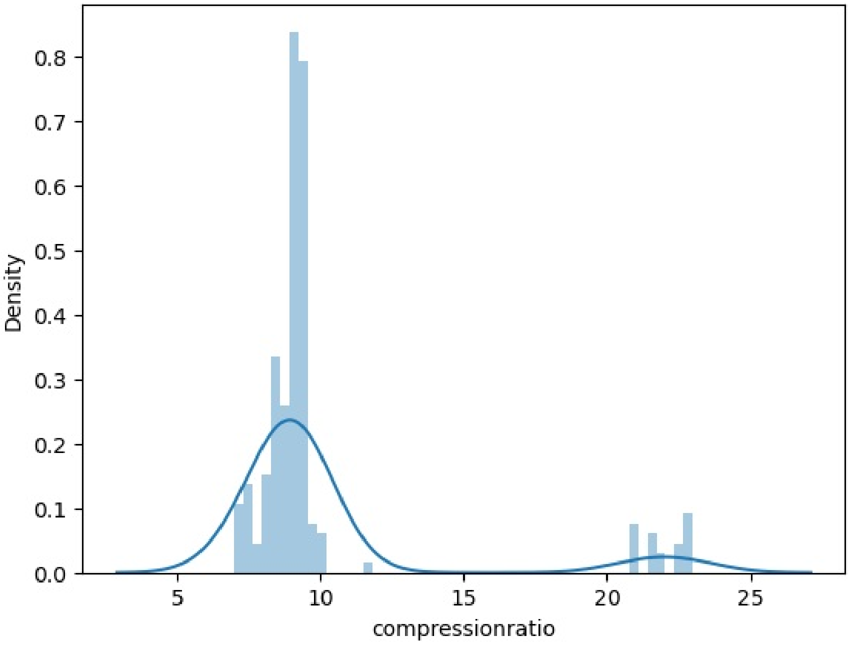
In [10]:
```
# Curbweight: weight of car without occupants or baggage
sns.distplot(cars['curbweight'])
plt.show()
```



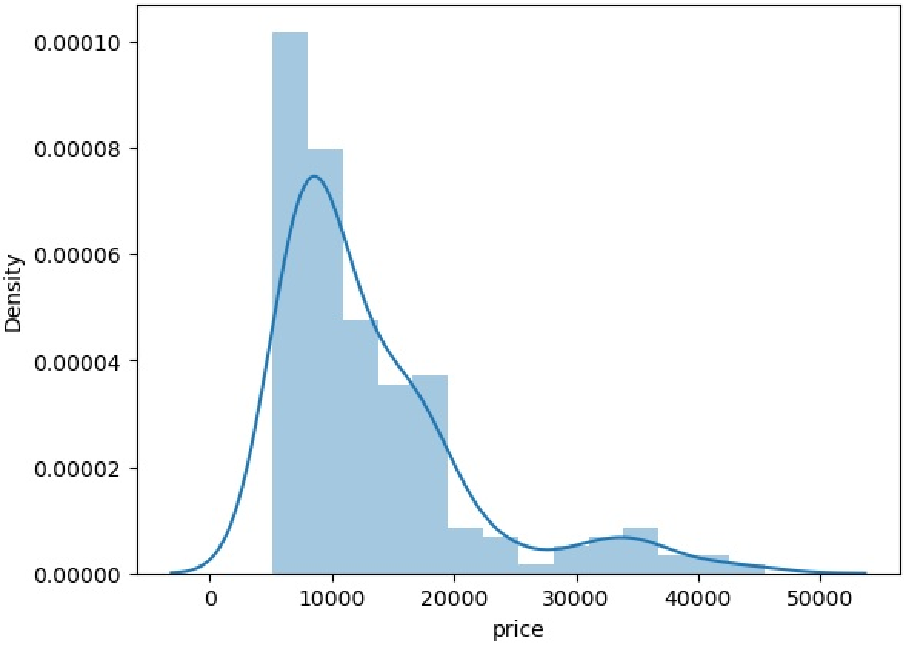
In [11]:
```
# Stroke: volume of the engine (the distance traveled by the piston in each cycle)
sns.distplot(cars['stroke'])
plt.show()
```



In [12]:
```
# Compression ration: ration of volume of compression chamber at largest capacity to least capacity
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. Let's see whether that's true in this case.

We'll first 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	compressionratio	horsepower	pe
0	1	3	88.6	168.8	64.1	48.8	2548	130	3.47	2.68	9.0	111	
1	2	3	88.6	168.8	64.1	48.8	2548	130	3.47	2.68	9.0	111	
2	3	1	94.5	171.2	65.5	52.4	2823	152	2.68	3.47	9.0	154	
3	4	2	99.8	176.6	66.2	54.3	2337	109	3.19	3.40	10.0	102	
4	5	2	99.4	176.6	66.4	54.3	2824	136	3.19	3.40	8.0	115	

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
cars_numeric = cars_numeric.drop(['car_ID'], axis=1)
cars_numeric.head()
```

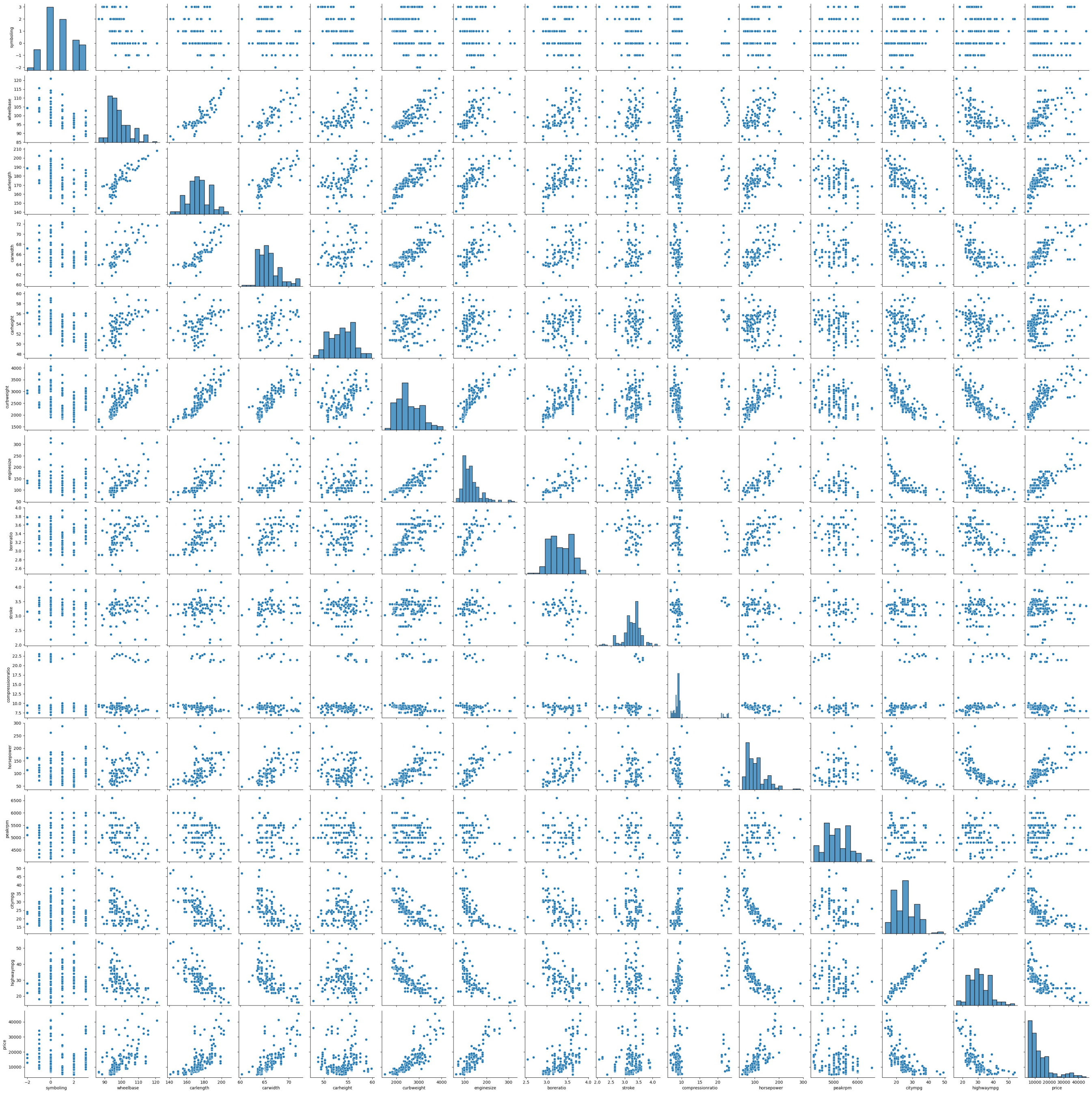
Out[15]:

	symboling	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	boreratio	stroke	compressionratio	horsepower	peakrpm
0	3	88.6	168.8	64.1	48.8	2548	130	3.47	2.68	9.0	111	5000
1	3	88.6	168.8	64.1	48.8	2548	130	3.47	2.68	9.0	111	5000
2	1	94.5	171.2	65.5	52.4	2823	152	2.68	3.47	9.0	154	5000
3	2	99.8	176.6	66.2	54.3	2337	109	3.19	3.40	10.0	102	5500
4	2	99.4	176.6	66.4	54.3	2824	136	3.19	3.40	8.0	115	5500

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.

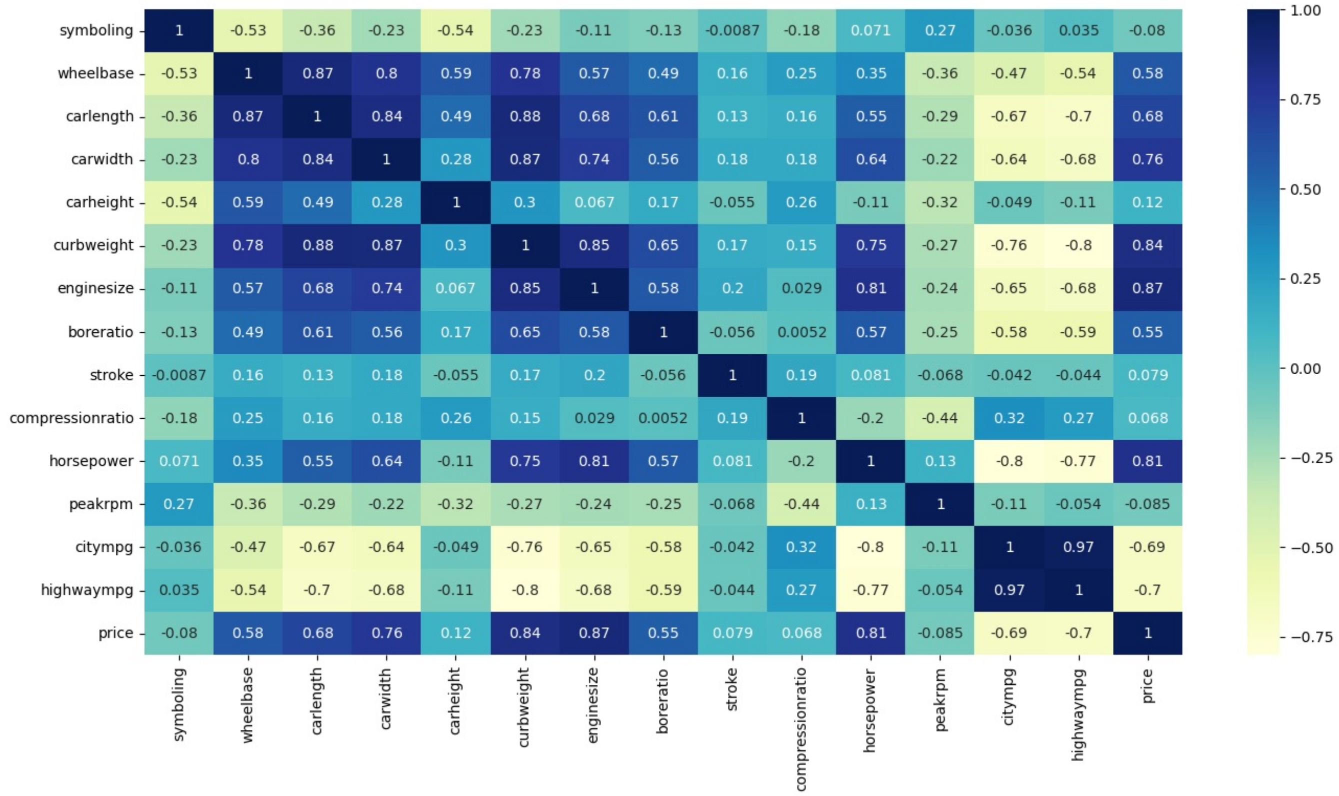
In [17]:
```
# Correlation matrix
cor = cars_numeric.corr()
cor
```

Out[17]:

	symboling	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	boreratio	stroke	compressionratio	hor
symboling	1.000000	-0.531954	-0.357612	-0.232919	-0.541038	-0.227691	-0.105790	-0.130051	-0.008735	-0.178515	(
wheelbase	-0.531954	1.000000	0.874587	0.795144	0.589435	0.776386	0.569329	0.488750	0.160959	0.249786	(
carlength	-0.357612	0.874587	1.000000	0.841118	0.491029	0.877728	0.683360	0.606454	0.129533	0.158414	(
carwidth	-0.232919	0.795144	0.841118	1.000000	0.279210	0.867032	0.735433	0.559150	0.182942	0.181129	(
carheight	-0.541038	0.589435	0.491029	0.279210	1.000000	0.295572	0.067149	0.171071	-0.055307	0.261214	-(
curbweight	-0.227691	0.776386	0.877728	0.867032	0.295572	1.000000	0.850594	0.648480	0.168790	0.151362	(
enginesize	-0.105790	0.569329	0.683360	0.735433	0.067149	0.850594	1.000000	0.583774	0.203129	0.028971	(
boreratio	-0.130051	0.488750	0.606454	0.559150	0.171071	0.648480	0.583774	1.000000	-0.055909	0.005197	(
stroke	-0.008735	0.160959	0.129533	0.182942	-0.055307	0.168790	0.203129	-0.055909	1.000000	0.186110	(
compressionratio	-0.178515	0.249786	0.158414	0.181129	0.261214	0.151362	0.028971	0.005197	0.186110	1.000000	-(
horsepower	0.070873	0.353294	0.552623	0.640732	-0.108802	0.750739	0.809769	0.573677	0.080940	-0.204326	'
peakrpm	0.273606	-0.360469	-0.287242	-0.220012	-0.320411	-0.266243	-0.244660	-0.254976	-0.067964	-0.435741	(
citympg	-0.035823	-0.470414	-0.670909	-0.642704	-0.048640	-0.757414	-0.653658	-0.584532	-0.042145	0.324701	-(
highwaympg	0.034606	-0.544082	-0.704662	-0.677218	-0.107358	-0.797465	-0.677470	-0.587012	-0.043931	0.265201	-(
price	-0.079978	0.577816	0.682920	0.759325	0.119336	0.835305	0.874145	0.553173	0.079443	0.067984	(

Let's plot 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 heatmap shows some useful insights:

Correlation of price with independent variables:

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

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:

Many independent variables are highly correlated (look at the top-left part of matrix): wheelbase, carlength, curbweight, enginesize etc.

are all measures of 'size/weight', and are positively correlated Thus, while building the model, we'll have to pay attention to multicollinearity.

2. Data Cleaning

Let's now conduct some data cleaning steps.

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

In [19]:
```
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
```

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

In [20]:
```
# Method 2: Use regular expressions (Note: This hasn't been included in the module and thus, it is not expected
# this method in your assignment. This is just for you to learn a new method)

import re

# regex: any alphanumeric sequence before a space, may contain a hyphen
p = re.compile(r'\w+-?\w+')
carnames = cars['CarName'].apply(lambda x: re.findall(p, x)[0])
print(carnames)
```

```
0      alfa-romero
1      alfa-romero
2      alfa-romero
3             audi
4             audi
          ...     
200          volvo
201          volvo
202          volvo
203          volvo
204          volvo
Name: CarName, Length: 205, dtype: object
```

In [21]:
```
# CarName: first few entries
cars['CarName'][:30]
```

Out[21]:
```
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
```

Notice that the car name is what occurs before a space, e.g. alfa-romero, audi, chevrolet, dodge, bmx etc.

Thus, we need to simply extract the string before a space. Let's see how we can do that.

In [22]:
```python
# Extracting carname

# Method: str.split() by space
carnames = cars['CarName'].apply(lambda x: x.split(" ")[0])
carnames[:30]
```

Out[22]:
```
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
Name: CarName, dtype: object
```

Let's create a new column to store the company name and check whether it looks okay.

In [23]:
```python
# Create a new column named car_company
cars['car_company'] = cars['CarName'].apply(lambda x: re.findall(p, x)[0])
```

In [24]:
```python
# Look at all values since this column will be used as a categorical variable
cars['car_company'].astype('category').value_counts()
```

Out[24]:
```
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
```

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, let's solve it.

In [25]:
```
# Replacing the misspelled car_company names

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

# 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 [26]:
```
cars['car_company'].astype('category').value_counts()
```

Out[26]:
```
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 looks okay now. Let's now drop the car name variable.

In [27]:
```
# Drop carname variable
cars = cars.drop('CarName', axis=1)
```

In [28]:
```
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   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
 25  car_company       205 non-null    object 
dtypes: float64(8), int64(8), object(10)
memory usage: 41.8+ KB
```

In [29]:
```python
# Let's check for any outliers
cars.describe()
```

Out[29]:

	car_ID	symboling	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	boreratio	stroke	compressic
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.0
mean	103.000000	0.834146	98.756585	174.049268	65.907805	53.724878	2555.565854	126.907317	3.329756	3.255415	10.1
std	59.322565	1.245307	6.021776	12.337289	2.145204	2.443522	520.680204	41.642693	0.270844	0.313597	3.9
min	1.000000	-2.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	2.540000	2.070000	7.0
25%	52.000000	0.000000	94.500000	166.300000	64.100000	52.000000	2145.000000	97.000000	3.150000	3.110000	8.6
50%	103.000000	1.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000	3.310000	3.290000	9.0
75%	154.000000	2.000000	102.400000	183.100000	66.900000	55.500000	2935.000000	141.000000	3.580000	3.410000	9.4
max	205.000000	3.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	326.000000	3.940000	4.170000	23.0

3. Data Preparation

Data Preparation Let's now prepare the data and build the model. First, let's take a look at the dataset again.

In [30]:
```python
cars.head()
```

Out[30]:

	car_ID	symboling	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	carlength	...	fuelsystem	borerati
0	1	3	gas	std	two	convertible	rwd	front	88.6	168.8	...	mpfi	3.4
1	2	3	gas	std	two	convertible	rwd	front	88.6	168.8	...	mpfi	3.4
2	3	1	gas	std	two	hatchback	rwd	front	94.5	171.2	...	mpfi	2.6
3	4	2	gas	std	four	sedan	fwd	front	99.8	176.6	...	mpfi	3.1
4	5	2	gas	std	four	sedan	4wd	front	99.4	176.6	...	mpfi	3.1

5 rows × 26 columns

Notice that two of the variables - doornumber and cylindernumber are numeric types with the numbers written as words. Let's map these to actual numbers to avoid too many dummy variable creations ahead.

Note that you can also treat them as categorical variables (these two and also, symboling) and create dummy variables for them. It's upto you.

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

Out[31]:
```
four      159
six        24
five       11
eight       5
two         4
three       1
twelve      1
Name: cylindernumber, dtype: int64
```

In [32]:
```
# Checking the different levels of 'doornumber'
cars['doornumber'].astype('category').value_counts()
```

Out[32]:
```
four    115
two      90
Name: doornumber, dtype: int64
```

In [33]:
```
# A function to map the categorical levels to actual numbers. You can see the categorical levels above and use
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)
```

Let's now create dummy variables for the categorical variables

In [34]:
```
# Subset all categorical variables
cars_categorical = cars.select_dtypes(include=['object'])
cars_categorical.head()
```

Out[34]:

	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 [35]:
```
# Convert into dummies
cars_dummies = pd.get_dummies(cars_categorical, drop_first=True)
cars_dummies.head()
```

Out[35]:

	fueltype_gas	aspiration_turbo	carbody_hardtop	carbody_hatchback	carbody_sedan	carbody_wagon	drivewheel_fwd	drivewheel_rwd	eng
0	1	0	0	0	0	0	0	1	
1	1	0	0	0	0	0	0	1	
2	1	0	0	1	0	0	0	1	
3	1	0	0	0	1	0	1	0	
4	1	0	0	0	1	0	0	0	

5 rows × 43 columns

In [36]:
```
# Drop categorical variable columns
cars = cars.drop(list(cars_categorical.columns), axis=1)
```

In [37]:
```
# Concatenate dummy variables with X
cars = pd.concat([cars, cars_dummies], axis=1)
```

In [38]:
```
cars.head()
```

Out[38]:

	car_ID	symboling	doornumber	wheelbase	carlength	carwidth	carheight	curbweight	cylindernumber	enginesize	...	car_company_niss
0	1	3	2	88.6	168.8	64.1	48.8	2548	4	130	...	
1	2	3	2	88.6	168.8	64.1	48.8	2548	4	130	...	
2	3	1	2	94.5	171.2	65.5	52.4	2823	6	152	...	
3	4	2	4	99.8	176.6	66.2	54.3	2337	4	109	...	
4	5	2	4	99.4	176.6	66.4	54.3	2824	5	136	...	

5 rows × 61 columns

Notice that the car_ID column is still there. We had dropped it from the 'cars_numeric' dataframe but not from the original. Let's drop it now.

In [39]:
```
# Drop the 'car_ID' column
cars.drop('car_ID', axis = 1, inplace = True)
```

3. Model Building and Evaluation

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

In [40]:
```
# Split the datafram into train and test sets
df_train, df_test = train_test_split(cars, train_size=0.7, test_size=0.3, random_state=100)
```

Scaling Now that we have done the test-train split, we need to scale the variables for better interpretability. But we only need the 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 it to learn anything from the test data.

In [41]:
```
cars_numeric.columns
```

Out[41]:
```
Index(['symboling', 'wheelbase', 'carlength', 'carwidth', 'carheight',
       'curbweight', 'enginesize', 'boreratio', 'stroke', 'compressionratio',
       'horsepower', 'peakrpm', 'citympg', 'highwaympg', 'price'],
      dtype='object')
```

Let's scale all these columns using StandardScaler. You can use any other scaling method as well; it is totally up to you. Also, note that you had converted two other variables, viz., 'doornumber' and 'cylindernumber' to numeric types. So you would need to include them in the varlist as well

In [42]:
```
# Import the StandardScaler()
from sklearn.preprocessing import StandardScaler

# Create a scaling object
scaler = StandardScaler()

# Create a list of the variables that you need to scale
varlist = ['symboling', 'wheelbase', 'carlength', 'carwidth', 'carheight','curbweight', 'enginesize', 'borerati
           'compressionratio','horsepower', 'peakrpm', 'citympg', 'highwaympg', 'doornumber', 'cylindernumber',

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

In [43]:
```
# Let's take a look at the train dataframe now
df_train.head()
```

Out[43]:

	symboling	doornumber	wheelbase	carlength	carwidth	carheight	curbweight	cylindernumber	enginesize	boreratio	...	car_company
122	0.170159	0.887412	-0.811836	-0.487238	-0.924500	-1.134628	-0.642128	-0.351431	-0.660242	-1.297329	...	
125	1.848278	-1.126872	-0.677177	-0.359789	1.114978	-1.382026	0.439415	-0.351431	0.637806	2.432256	...	
166	0.170159	-1.126872	-0.677177	-0.375720	-0.833856	-0.392434	-0.441296	-0.351431	-0.660242	-0.259197	...	
1	1.848278	-1.126872	-1.670284	-0.367754	-0.788535	-1.959288	0.015642	-0.351431	0.123485	0.625138	...	
199	-1.507960	0.887412	0.972390	1.225364	0.616439	1.627983	1.137720	-0.351431	0.123485	1.201877	...	

5 rows × 60 columns

As expected, the variables have been appropriately scaled.

In [44]:
```
# Split the train dataset into X and y

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

Building the first model with all the features

Let's now build our first model with all the features.

In [45]:
```
# Instantiate
lm = LinearRegression()

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

Out[45]:
```
▾ LinearRegression
LinearRegression()
```

In [46]:
```
# Print the coefficients and intercept
print(lm.coef_)
print(lm.intercept_)
```

```
[-8.85082799e-03  4.33698434e-02  1.92456997e-01 -4.35193269e-02
  2.00950555e-01 -1.55840908e-01  1.79748275e-01 -3.15467916e-01
  1.09391381e+00 -3.52820754e-01 -1.21008441e-01 -4.64239744e-01
 -1.42549382e-01  1.90699409e-01  2.68476746e-02  1.03705263e-01
  2.46360464e+13  4.21878405e-01 -5.90545220e-01 -6.77378177e-01
 -6.50876562e-01 -5.68188787e-01 -9.03470033e-02  9.57244687e-02
 -2.39226920e+12  1.00816629e+00  1.05373204e+00  3.35533716e-01
  2.39226920e+12  8.58589794e-02  1.17597081e+00  1.88484766e-01
 -2.61429243e-01  2.46360464e+13 -2.72341698e+11  2.35686642e-02
  4.97921897e-03 -6.67394404e+11 -7.41899632e-02  1.05786815e+00
  8.71467594e-02 -5.03533489e-01 -6.59479884e-01 -3.28161291e-01
 -2.32568741e-01 -2.25754073e-01 -9.55652200e-02 -1.11979617e+10
 -7.16568650e-01 -1.83390866e-01 -1.27280381e+00 -6.56872863e-01
  9.26632298e-01 -3.03071658e-01  7.66157798e-01 -2.39226920e+12
 -1.45001846e-01 -9.92957814e-02  7.04709223e-02]
-24636046353111.77
```

Model Building Using RFE

Now, you have close to 60 features. It is obviously not possible to manually eliminate these features. So let's now build a model using recursive feature elimination 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 shortlisted features (this is also because SKLearn doesn't have Adjusted R-squared that statsmodels has).

In [47]:
```
# Import RFE
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression  # Make sure to import LinearRegression

# RFE with 15 features
lm = LinearRegression()
rfe1 = RFE(lm, n_features_to_select=15)  # Specify the number of features to select

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

# Print the boolean results
print(rfe1.support_)
print(rfe1.ranking_)
```

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

Model Building and Evaluation

Let's now check the summary of this model using statsmodels.

In [48]:
```
# Import statsmodels
import statsmodels.api as sm

# Subset 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
X_train_rfe1 = sm.add_constant(X_train_rfe1)
X_train_rfe1.head()
```

Out[48]:

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

In [52]:
```
# Import statsmodels
import statsmodels.api as sm

# Subset the features selected by rfe1
col1 = X_train.columns[rfe1.support_]
```

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

# Add a constant to the model
X_train_rfe1 = sm.add_constant(X_train_rfe1)
X_train_rfe1.head()
```

Out[52]:

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

In [49]:

```python
# 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:                Wed, 10 Jan 2024   Prob (F-statistic):           4.59e-64
Time:                        08:26:43   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.                     7.20e+16
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 5.58e-32. 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. Let's also quickly take a look at the VIF values.

In [50]:

```python
# Check for the VIF values of the feature variables.
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [51]:

```python
# Create a dataframe that will contain the names of all the feature variables and their respective VIFs
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[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[51]:

	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

Notice that there are a few variables which have an infinite VIF. These variables aren't of use. But manually elimination is time consuming and makes the code unnecessarily long. Let's try and build a model with 10 features this time using RFE and see what we get.

In [54]:

```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression

# Create a linear regression model
lm = LinearRegression()

# Initialize RFE with the linear regression model and the desired number of features
rfe2 = RFE(estimator=lm, n_features_to_select=10)

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

Out[54]:

```
▸         RFE
▸ estimator: LinearRegression
      ▸ LinearRegression
```

In [55]:

```
# 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:                Wed, 10 Jan 2024   Prob (F-statistic):           3.98e-64
Time:                        08:39:18   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.                     6.59e+16
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 5.8e-32. 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 practically. It has gone from 0.912 to 0.901. So 10 variables seems to be a good number to start with.

In [56]:
```
# Check for the VIF values of the feature variables.
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [57]:
```
# Create a dataframe that will contain the names of all the feature variables and their respective VIFs
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[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[57]:

	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

In [58]:
```
X_train_rfe2.drop('car_company_subaru', axis = 1, inplace = True)
```

In [59]:
```
# 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:                Wed, 10 Jan 2024   Prob (F-statistic):           3.98e-64
Time:                        08:40:22   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 [60]:

```
# Create a dataframe that will contain the names of all the feature variables and their respective VIFs

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[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[60]:

	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 workable value. But from the p-value perspective, enginetype_ohfc has become insignificant. So let's drop that.

In [61]:

```
X_train_rfe2.drop('enginetype_ohcf', axis = 1, inplace = True)
```

In [62]:

```
# 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:                Wed, 10 Jan 2024   Prob (F-statistic):           8.22e-65
Time:                        08:41:06   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 [63]:

```
# Create a dataframe that will contain the names of all the feature variables and their respective VIFs

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[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[63]:

	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. Let's drop them and see if the Adjusted R-squared score is getting affected.

In [64]:

```
X_train_rfe2.drop('car_company_peugeot', axis = 1, inplace = True)
```

In [65]:

```
# 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:                Wed, 10 Jan 2024   Prob (F-statistic):           2.49e-63
Time:                        08:42:27   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 [66]:
```python
# Create a dataframe that will contain the names of all the feature variables and their respective VIFs
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[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[66]:

	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

The enginetype_l variables now has a p-value of 0.3. Let's drop it and see if it affects the model much.

In [67]:
```python
# 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:                Wed, 10 Jan 2024   Prob (F-statistic):           2.92e-64
Time:                        08:43:00   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 [68]:

```
# Create a dataframe that will contain the names of all the feature variables and their respective VIFs
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[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[68]:

	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 and p-values 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. So let's proceed with this model.

Residual Analysis

Before we make predictions on the test set, let's first analyse the residuals.

In [69]:

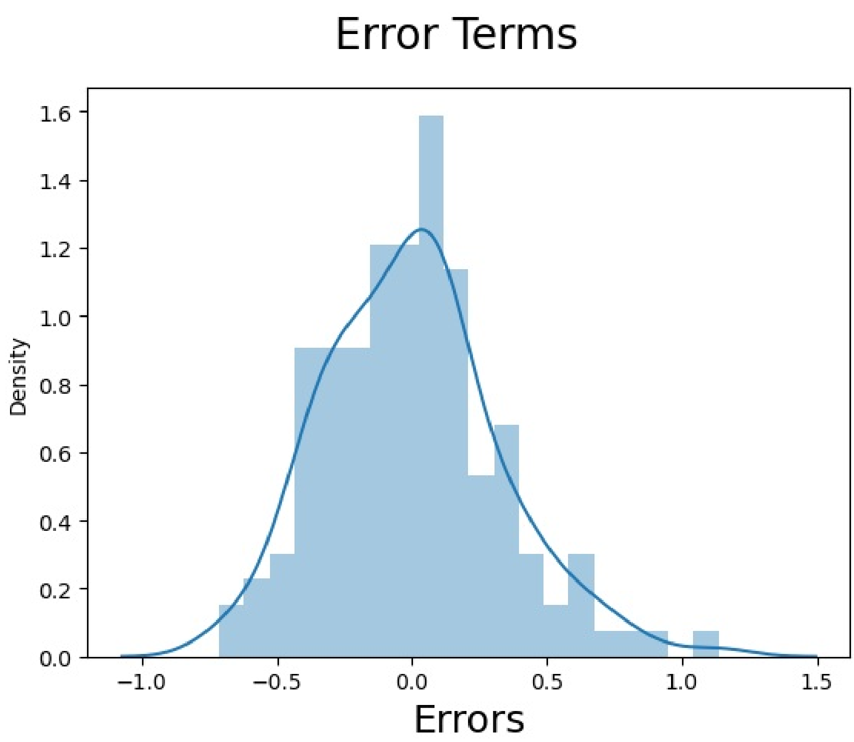
```
y_train_price = lm2.predict(X_train_rfe2)
```

In [70]:

```
# 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[70]:

```
Text(0.5, 0, 'Errors')
```



The error terms are fairly normally distributed and we can surely live with this. Let's now make predictions on the test-set.

Making Predictions

We would first need to scale the test set as well. So let's start with that.

In [71]:
```
df_test[varlist] = scaler.transform(df_test[varlist])
```

In [72]:
```
# Split the 'df_test' set into X and y
y_test = df_test.pop('price')
X_test = df_test
```

In [73]:
```
# Let's check the list 'col2' which had the 10 variables RFE had selected
col2
```

Out[73]:
```
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 [74]:
```
X_test_rfe2 = X_test[col2]
```

In [75]:
```
# Let's now drop the variables we had manually eliminated as well
X_test_rfe2 = X_test_rfe2.drop(['enginetype_ohcf', 'car_company_peugeot', 'enginetype_l', 'car_company_subaru']
```

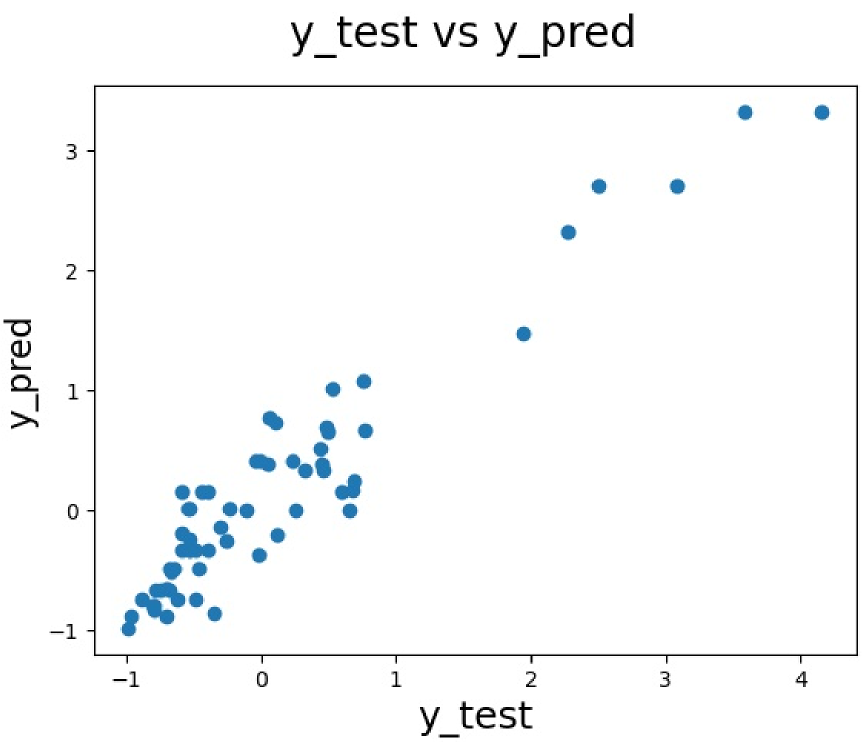
In [76]:
```
# Add a constant to the test set created
X_test_rfe2 = sm.add_constant(X_test_rfe2)
X_test_rfe2.info()
```

```
<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 [77]:
```
# Making predictions
y_pred = lm2.predict(X_test_rfe2)
```

In [78]:
```
# 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 = 18)                          # X-label
plt.ylabel('y_pred', fontsize = 16)
```

Out[78]:
```
Text(0, 0.5, 'y_pred')
```



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

In [79]:
```
# r2_score for 6 variables
from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
```

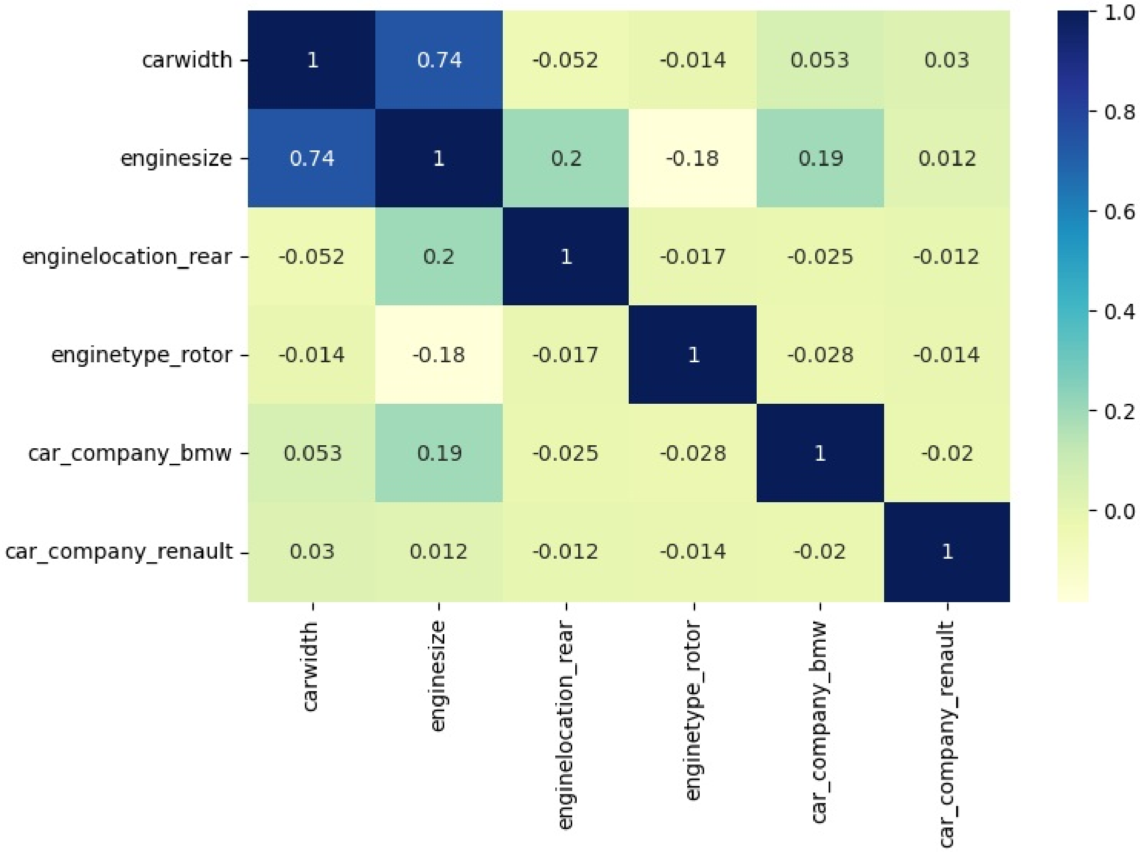
Out[79]:
```
0.8997211435182686
```

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%.

Checking the correlations between the final predictor variables

In [80]:
```
col2 = col2.drop(['enginetype_ohcf', 'car_company_peugeot', 'enginetype_l', 'car_company_subaru'])
```

In [81]:
```
# Figure size
plt.figure(figsize=(8,5))

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



Though this is the most simple model we've built till now, few final predictors still seem to have high correlations. One can go ahead and remove some of these features, though that will affect the adjusted-r2 score significantly (you should try doing that).

Thus, for now, the final model consists of the 6 variables mentioned above.

```
In [ ]:
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js