Multiple Regression Project

1 Introduction

1.1 Motivation

In the second assignment, I look at a Chinese automobile company 'Teclov_chinese' which aspires to enter the US market by setting up its manufacturing unit there and producing its line of cars. In this regard, an automobile consulting company has been given a contract to understand the factors on which the pricing of cars depends. They are mainly trying to assess the significant variable in predicting the price of the car and how well they describe the price based on various market surveys.

I have tried to model the price of the cars with the available information. The essential aim is to see how exactly prices vary with independent variables. I have approached this in a structured way, starting with some essential data cleaning and exploratory analysis. Following this, I have delved into the variable selection and a few descriptive analyses and then subsequently model building and interpretation.

As a result, I can identify the key factors that influence the price and how the company can manipulate various things to meet the desired price levels. It also helps in understanding the pricing system in a new market.

1.2 Contents

1.3 Data Dictionary

1 Car_ID: Unique id of each observation
2 Symboling: Its assigned insurance risk rating, A value of +3 indicates that the auto is risky, -3 that it is probably pretty safe
3 carCompany: Name of car company
4 fueltype: Car fuel type i.e gas or diesel
5 aspiration: Aspiration used in a car
6 doornumber: Number of doors in a car
7 carbody: Body of car
8 drivewheel: Type of drive wheel
9 enginelocation: Location of car engine
10 wheelbase: Weelbase of car
11 carlength: Length of car
12 carwidth: Width of car
13 carheight: Height of car
14 curbweight: The weight of a car without occupants or baggage

15 enginetype: Type of engine
16 cylindernumber: Cylinder placed in the car
17 enginesize: Size of engine
18 fuelsystem: Fuel system of car
19 boreratio: Boreratio of car
20 stroke: Stroke or volume inside the engine
21 compressionratio: Compression ratio of car
22 horsepower: Horsepower
23 peakrpm: Car peak rpm
24 citympg: Mileage in city
25 highwaympg: Mileage on highway
26 price: Price of car

2 Data Cleaning

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import scipy.stats as stats
import statsmodels.api as sm
import statsmodels.formula.api as smf
```

In [2]:
```
data = pd.read_csv('CarPrice_Assignment.csv')
data.head()
```

Out[2]:

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	...	enginesize	fuelsystem	boreratio	s
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	
4	5	2	audi 100ls	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	

5 rows × 26 columns

In [3]:
```
data.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 
```

In [4]:
```python
# Look for any missing observations
data.isnull().any()
```

Out[4]:
```
car_ID              False
symboling           False
CarName             False
fueltype            False
aspiration          False
doornumber          False
carbody             False
drivewheel          False
enginelocation      False
wheelbase           False
carlength           False
carwidth            False
carheight           False
curbweight          False
enginetype          False
cylindernumber      False
enginesize          False
fuelsystem          False
boreratio           False
```

In [5]:
```python
# Check basic values
data.nunique()
```

```
enginelocation        2
wheelbase            53
carlength            75
carwidth             44
carheight            49
curbweight          171
enginetype            7
cylindernumber        7
enginesize           44
fuelsystem            8
boreratio            38
stroke               37
compressionratio     32
horsepower           59
peakrpm              23
citympg              29
highwaympg           30
price               189
dtype: int64
```

Something Important:

Every var has more than one values.

I think car_ID has no relationship with the model, so I drop it in the next step.

There are 147 distinct values for CarName, I will turn CarName into Region(Country) later.

In [6]:
```python
# Drop car_ID
data = data.drop('car_ID', axis = 1)
```

In [7]:
```python
data.head()
```

Out[7]:

	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	carlength	...	enginesize	fuelsystem	boreratio
0	3	alfa-romero giulia	gas	std	two	convertible	rwd	front	88.6	168.8	...	130	mpfi	3.47
1	3	alfa-romero stelvio	gas	std	two	convertible	rwd	front	88.6	168.8	...	130	mpfi	3.47
2	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	front	94.5	171.2	...	152	mpfi	2.68
3	2	audi 100 ls	gas	std	four	sedan	fwd	front	99.8	176.6	...	109	mpfi	3.19
4	2	audi 100ls	gas	std	four	sedan	4wd	front	99.4	176.6	...	136	mpfi	3.19

5 rows × 25 columns

In [8]:
```python
# CarName --> CarBrand and Correct typo error --> Country
data['Brand'] = data['CarName'].str.split(expand = True)[0]
data['Brand'] = data['Brand'].replace({'maxda': 'mazda', 'Nissan' : 'nissan', 'porcshce' : 'porsche',
                                      'toyouta' : 'toyota', 'vokswagen' : 'volkswagen', 'vw' : 'volkswagen'})
data['country'] = data['Brand'].apply(lambda x: 'Japan' if x in ['honda', 'isuzu', 'mazda', 'mitsubishi', 'nissan','su
```

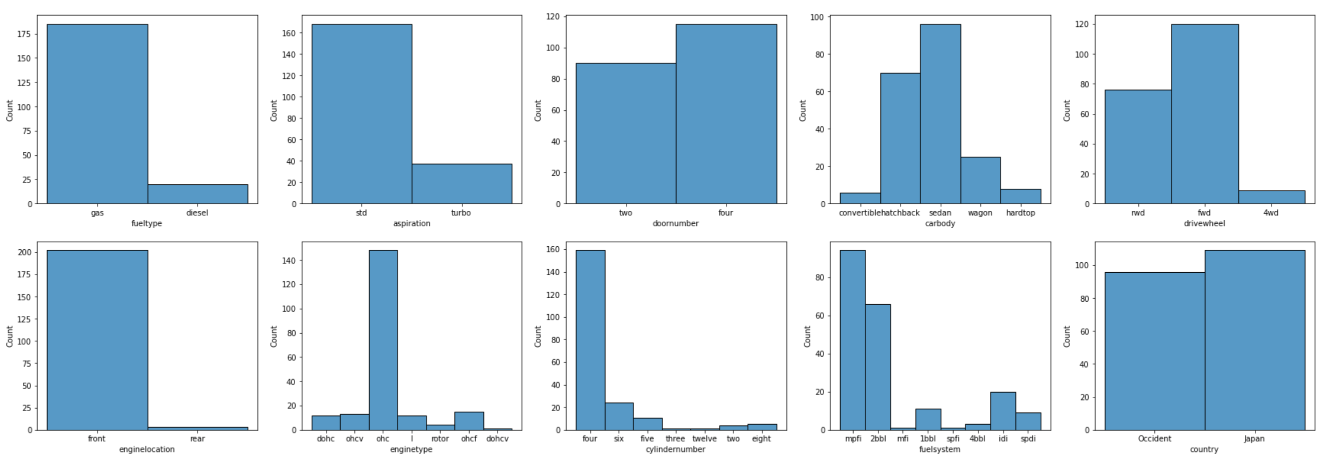
In [9]:
```python
# Numeric Variables
all_colums = list (data)
numeric_colums = ['symboling', 'wheelbase', 'carlength', 'carwidth', 'carheight', 'curbweight', 'enginesize',
                  'boreratio','stroke', 'compressionratio', 'horsepower', 'peakrpm', 'citympg', 'highwaympg']

# Categorical Variables
categorical_colums = ['fueltype', 'aspiration', 'doornumber', 'carbody', 'drivewheel',
                      'enginelocation', 'enginetype', 'cylindernumber', 'fuelsystem']
```

Since there are 22 variables in my dataset, I want to conduct preliminary selection before variables selection.

I mainly focus on categorical variables, because the robustness of the model will be significantly affected by class-imbalance. Given that I haven't learned how to handle such situation yet, I choose to remove thoese categorical variables with extremely unbalanced sample sizes first.

In [10]:
```python
fig = plt.figure(figsize = (30,10))
ax1 = fig.add_subplot(2,5,1)
sns.histplot(data.fueltype)
ax2 = fig.add_subplot(2,5,2)
sns.histplot(data.aspiration)
ax3 = fig.add_subplot(2,5,3)
sns.histplot(data.doornumber)
ax4 = fig.add_subplot(2,5,4)
sns.histplot(data.carbody)
ax5 = fig.add_subplot(2,5,5)
sns.histplot(data.drivewheel)
ax6 = fig.add_subplot(2,5,6)
sns.histplot(data.enginelocation)
ax7 = fig.add_subplot(2,5,7)
sns.histplot(data.enginetype)
ax8 = fig.add_subplot(2,5,8)
sns.histplot(data.cylindernumber)
ax9 = fig.add_subplot(2,5,9)
sns.histplot(data.fuelsystem)
ax10 = fig.add_subplot(2,5,10)
sns.histplot(data.country)
```

Out[10]:
```
<AxesSubplot:xlabel='country', ylabel='Count'>
```



From the histograms matrix, I decid to keep 'doornumber', 'carbody', 'drivewheel', 'country'.

For 'doornumber' and 'country', I find that there is not much difference in sample size between the two groups.

For 'carbody', I found that there is little difference in the number of samples in sedan and hatchback, while the numbers of samples in other classifications are relatively small. Combined with the information found on the Internet, I learned that the mainstream market is sedans and hatchbacks, so I combine samples of other types.

For 'drivewheel', I leared that FWD(front-wheel drive) vehicles are generally extremely efficient in terms of cost, mass, space and fuel consumption and therefore FWD is the drive system adopted by the vast majority of vehicles in the market. RWD(rear-wheel drive) vehicles can typically handle more horsepower and higher vehicle weight and are often found in sports cars and racing cars. 4WD(four-wheel drive) can often be found in large SUVs and trucks. I intuitively believe that both RWD and 4WD vehicles will be more expensive than other RWD vehicles with similar conditions, so I combined the RWD and 4WD sample data.

In [11]:
```python
data['doornumber'] = data['doornumber'].apply(lambda x: 0 if x == 'two' else 1)
data['carbody'] = data['carbody'].apply(lambda x: 0 if x == 'sedan' else
                                       (1 if x =='hatchback' else 2))
data['drivewheel'] = data['drivewheel'].apply(lambda x: 0 if x == 'fwd' else 1)
data['country'] = data['country'].apply(lambda x: 0 if x == 'occident' else 1)
```

In [12]:
```python
data = data.drop(['fueltype', 'aspiration','enginelocation',
                  'enginetype', 'cylindernumber', 'fuelsystem','CarName', 'Brand'], axis = 1)
```

Now I have numeric variables: 'symboling', 'wheelbase', 'carlength',' carwidth', 'carheight', 'curbweight', enginesize', 'boreratio', 'stroke', 'compressionratio', 'horsepower', 'peakrpm', 'citympg', 'highwaympg'.

Categorical Variables: 'doornumber', 'carbody', 'drivewheel', 'country'.

In [13]:
```python
data.head()
```

Out[13]:

	symboling	doornumber	carbody	drivewheel	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	boreratio	stroke	compressionratio	horsep
0	3	0	2	1	88.6	168.8	64.1	48.8	2548	130	3.47	2.68	9.0	
1	3	0	2	1	88.6	168.8	64.1	48.8	2548	130	3.47	2.68	9.0	
2	1	0	1	1	94.5	171.2	65.5	52.4	2823	152	2.68	3.47	9.0	
3	2	1	0	0	99.8	176.6	66.2	54.3	2337	109	3.19	3.40	10.0	
4	2	1	0	1	99.4	176.6	66.4	54.3	2824	136	3.19	3.40	8.0	

3 Variable Selection

3.1 Boruta Algorithm

In [14]:
```python
pip install Boruta
```

```
Requirement already satisfied: Boruta in /opt/anaconda3/lib/python3.9/site-packages (0.3)
Requirement already satisfied: scikit-learn>=0.17.1 in /opt/anaconda3/lib/python3.9/site-packages (from Boruta) (0.2
4.2)
Requirement already satisfied: numpy>=1.10.4 in /opt/anaconda3/lib/python3.9/site-packages (from Boruta) (1.20.3)
Requirement already satisfied: scipy>=0.17.0 in /opt/anaconda3/lib/python3.9/site-packages (from Boruta) (1.7.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/anaconda3/lib/python3.9/site-packages (from scikit-learn
>=0.17.1->Boruta) (2.2.0)
Requirement already satisfied: joblib>=0.11 in /opt/anaconda3/lib/python3.9/site-packages (from scikit-learn>=0.17.1
->Boruta) (1.1.0)
Note: you may need to restart the kernel to use updated packages.
```

In [15]:
```python
pip install BorutaShap
```

```
Requirement already satisfied: BorutaShap in /opt/anaconda3/lib/python3.9/site-packages (1.0.16)
Requirement already satisfied: scikit-learn in /opt/anaconda3/lib/python3.9/site-packages (from BorutaShap) (0.24.2)
Requirement already satisfied: seaborn in /opt/anaconda3/lib/python3.9/site-packages (from BorutaShap) (0.11.2)
Requirement already satisfied: tqdm in /opt/anaconda3/lib/python3.9/site-packages (from BorutaShap) (4.62.3)
Requirement already satisfied: numpy in /opt/anaconda3/lib/python3.9/site-packages (from BorutaShap) (1.20.3)
Requirement already satisfied: scipy in /opt/anaconda3/lib/python3.9/site-packages (from BorutaShap) (1.7.1)
Requirement already satisfied: matplotlib in /opt/anaconda3/lib/python3.9/site-packages (from BorutaShap) (3.4.3)
Requirement already satisfied: statsmodels in /opt/anaconda3/lib/python3.9/site-packages (from BorutaShap) (0.12.2)
Requirement already satisfied: pandas in /opt/anaconda3/lib/python3.9/site-packages (from BorutaShap) (1.3.4)
Requirement already satisfied: shap>=0.34.0 in /opt/anaconda3/lib/python3.9/site-packages (from BorutaShap) (0.41.0)
Requirement already satisfied: numba in /opt/anaconda3/lib/python3.9/site-packages (from shap>=0.34.0->BorutaShap)
(0.54.1)
Requirement already satisfied: cloudpickle in /opt/anaconda3/lib/python3.9/site-packages (from shap>=0.34.0->BorutaS
hap) (2.0.0)
Requirement already satisfied: slicer==0.0.7 in /opt/anaconda3/lib/python3.9/site-packages (from shap>=0.34.0->Borut
aShap) (0.0.7)
Requirement already satisfied: packaging>20.9 in /opt/anaconda3/lib/python3.9/site-packages (from shap>=0.34.0->Boru
taShap) (21.0)
Requirement already satisfied: pyparsing>=2.0.2 in /opt/anaconda3/lib/python3.9/site-packages (from packaging>20.9->
```

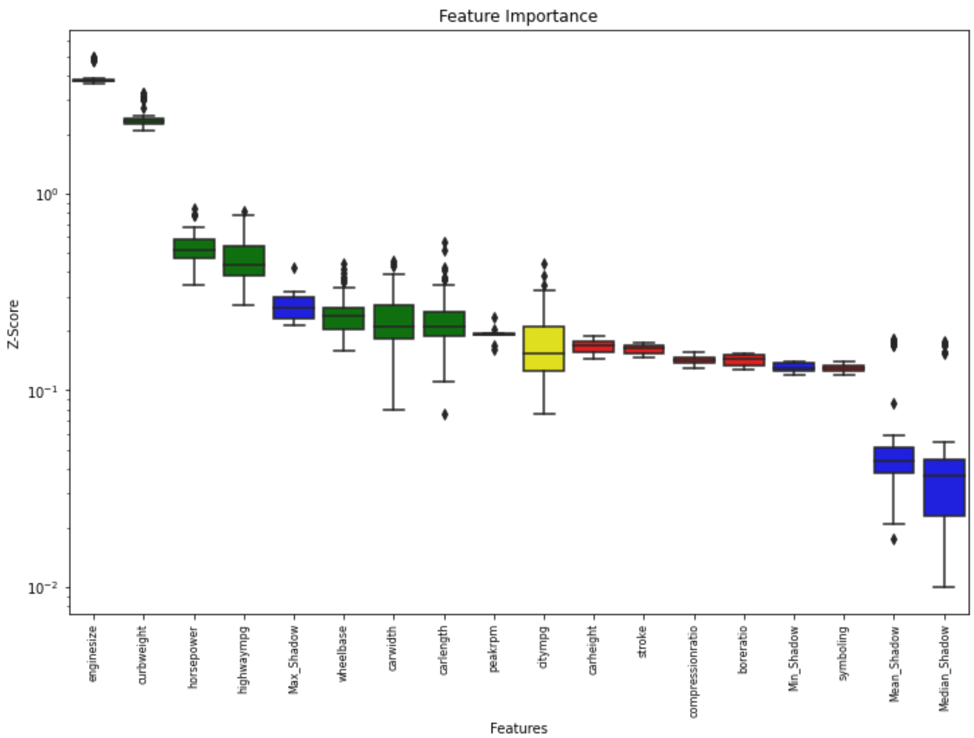
In [16]:
```python
pip install scikit-learn
```

```
Requirement already satisfied: scikit-learn in /opt/anaconda3/lib/python3.9/site-packages (0.24.2)
Requirement already satisfied: numpy>=1.13.3 in /opt/anaconda3/lib/python3.9/site-packages (from scikit-learn) (1.2
0.3)
Requirement already satisfied: scipy>=0.19.1 in /opt/anaconda3/lib/python3.9/site-packages (from scikit-learn) (1.7.
1)
Requirement already satisfied: joblib>=0.11 in /opt/anaconda3/lib/python3.9/site-packages (from scikit-learn) (1.1.
0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/anaconda3/lib/python3.9/site-packages (from scikit-lear
n) (2.2.0)
Note: you may need to restart the kernel to use updated packages.
```

In [17]:
```python
from BorutaShap import BorutaShap
from sklearn.ensemble import RandomForestRegressor
```

In [18]:
```
boruta_data = data[['price','symboling', 'wheelbase', 'carlength', 'carwidth', 'carheight',
                    'curbweight', 'enginesize', 'boreratio', 'stroke', 'compressionratio',
                    'horsepower', 'peakrpm', 'citympg', 'highwaympg']].copy()
x = boruta_data.iloc[:,1:]
y = boruta_data['price']
Feature_Selector = BorutaShap(importance_measure = 'shap', classification = False)
Feature_Selector.fit(X = x, y = y, n_trials = 50, random_state = 0)
Feature_Selector.plot(which_features='all')
```

```
  0%|          | 0/50 [00:00<?, ?it/s]

7 attributes confirmed important: ['enginesize', 'curbweight', 'carlength', 'wheelbase', 'highwaympg', 'carwidth',
'horsepower']
6 attributes confirmed unimportant: ['carheight', 'peakrpm', 'boreratio', 'compressionratio', 'stroke', 'symboling']
1 tentative attributes remains: ['citympg']
```



In [19]:
```
Feature_Selector.Subset()
```

Out[19]:

	enginesize	curbweight	carlength	wheelbase	highwaympg	carwidth	horsepower
0	130	2548	168.8	88.6	27	64.1	111
1	130	2548	168.8	88.6	27	64.1	111
2	152	2823	171.2	94.5	26	65.5	154
3	109	2337	176.6	99.8	30	66.2	102
4	136	2824	176.6	99.4	22	66.4	115
...	...	...	...	...	...	...	...
200	141	2952	188.8	109.1	28	68.9	114
201	141	3049	188.8	109.1	25	68.8	160
202	173	3012	188.8	109.1	23	68.9	134
203	145	3217	188.8	109.1	27	68.9	106
204	141	3062	188.8	109.1	25	68.9	114

3.2 Mallows Cp

Based on the results from Boruta Algorithm, I select 7 variables which preform well in the the Boruta Algorithm test.

7 attributes confirmed important in Boruta Algorithm test: 'carwidth', 'wheelbase', 'carlength', 'enginesize', 'curbweight', 'horsepower', 'highwaympg'.

In [20]:
```
pip install RegscorePy
```

```
Requirement already satisfied: RegscorePy in /opt/anaconda3/lib/python3.9/site-packages (1.1)
Requirement already satisfied: pandas in /opt/anaconda3/lib/python3.9/site-packages (from RegscorePy) (1.3.4)
Requirement already satisfied: numpy in /opt/anaconda3/lib/python3.9/site-packages (from RegscorePy) (1.20.3)
Requirement already satisfied: python-dateutil>=2.7.3 in /opt/anaconda3/lib/python3.9/site-packages (from pandas->Re
gscorePy) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in /opt/anaconda3/lib/python3.9/site-packages (from pandas->RegscorePy)
(2021.3)
Requirement already satisfied: six>=1.5 in /opt/anaconda3/lib/python3.9/site-packages (from python-dateutil>=2.7.3->
pandas->RegscorePy) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

In [21]:
```
from RegscorePy import mallow
import itertools
```

In [22]:
```
subdat = data[['price','carwidth', 'wheelbase', 'carlength', 'enginesize', 'curbweight', 'horsepower', 'highwaympg',
               'doornumber', 'carbody', 'drivewheel','country']].copy()

model_Mallows = smf.ols(formula = 'price ~ carwidth + wheelbase + carlength + enginesize + curbweight + horsepower + h
results_Mallows = model_Mallows.fit()
y = data['price']
y_pred = results_Mallows.fittedvalues

storage_cp = pd.DataFrame(columns = ['Variables', 'CP'])
k = 12
```

In [23]:
```
for L in range(1, len(subdat.columns[1:]) + 1):
    for subset in itertools.combinations(subdat.columns[1:],L):
        formula1 = 'price ~ ' + '+'.join(subset)
        results = smf.ols(formula = formula1, data = data).fit()
        y_sub = results.fittedvalues
        p = len(subset)+1
        cp = mallow.mallow(y, y_pred, y_sub, k, p)
        storage_cp = storage_cp.append({'Variables': subset, 'CP': cp}, ignore_index = True)
```

In [24]:
```
storage_cp = storage_cp.sort_values(by ='CP', axis = 0)
print(storage_cp.head())

print(storage_cp.iloc[1,0]) # first combination with 5 var
print(storage_cp.iloc[8,0]) # first combination with 6 var
```

```
                                            Variables        CP
304     (carwidth, enginesize, horsepower, drivewheel) -0.143509
723  (carwidth, enginesize, horsepower, carbody, dr...  0.838380
718  (carwidth, enginesize, horsepower, highwaympg,...  1.156281
721  (carwidth, enginesize, horsepower, doornumber,...  1.680224
654  (carwidth, carlength, enginesize, horsepower, ...  1.728533
('carwidth', 'enginesize', 'horsepower', 'carbody', 'drivewheel')
('carwidth', 'enginesize', 'horsepower', 'highwaympg', 'carbody', 'drivewheel')
```

Because the CP values of the previous groups do not have much difference, and I find that 'highwaympg' has a high correlation coefficient with price in the follow-up, I choose six variables here for subsequent analysis.

In [25]:
```
# Variables Information
```

· price : Price of car
· carwidth : Width of car
· enginesize : Size of engine
· horsepower : Horsepower · highwaympg : Mileage on highway
· carbody : Body of car

```
= 0 if it's sedean
= 1 if it's hatchback
= 2 others
```

· drivewheel : Types of drivewheel

```
= 0 if it's FWD
= 1 others
```

4 Descriptive Analysis

In [26]:
```python
selectedvar_columns = ['price','carwidth', 'enginesize', 'horsepower', 'highwaympg', 'carbody', 'drivewheel']
selecteddata = data[selectedvar_columns]
round(selecteddata.describe(),4)
```

Out[26]:

	price	carwidth	enginesize	horsepower	highwaympg	carbody	drivewheel
count	205.0000	205.0000	205.0000	205.0000	205.0000	205.0000	205.0000
mean	13276.7106	65.9078	126.9073	104.1171	30.7512	0.7220	0.4146
std	7988.8523	2.1452	41.6427	39.5442	6.8864	0.7642	0.4939
min	5118.0000	60.3000	61.0000	48.0000	16.0000	0.0000	0.0000
25%	7788.0000	64.1000	97.0000	70.0000	25.0000	0.0000	0.0000
50%	10295.0000	65.5000	120.0000	95.0000	30.0000	1.0000	0.0000
75%	16503.0000	66.9000	141.0000	116.0000	34.0000	1.0000	1.0000
max	45400.0000	72.3000	326.0000	288.0000	54.0000	2.0000	1.0000

I find that the standard deviation of price is very large, suggesting that it may need to be transformed for the model building.

4.1 Distributions: Histogram with Density Plot

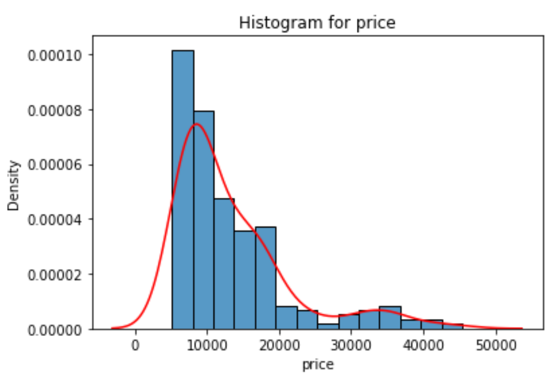
(1) Numeric Variables

In [27]:
```python
numeric_columns = ['price', 'carwidth', 'enginesize', 'horsepower', 'highwaympg']
```

price

In [28]:
```python
plt.title('Histogram for price')
sns.histplot(data.price, stat = 'density')
sns.kdeplot(data.price, color = 'red')
```

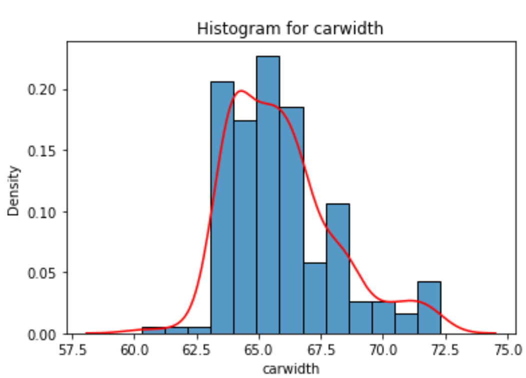
Out[28]: <AxesSubplot:title={'center':'Histogram for price'}, xlabel='price', ylabel='Density'>



carwidth

In [29]:
```python
plt.title('Histogram for carwidth')
sns.histplot(data.carwidth, stat = 'density')
sns.kdeplot(data.carwidth, color = 'red')
```

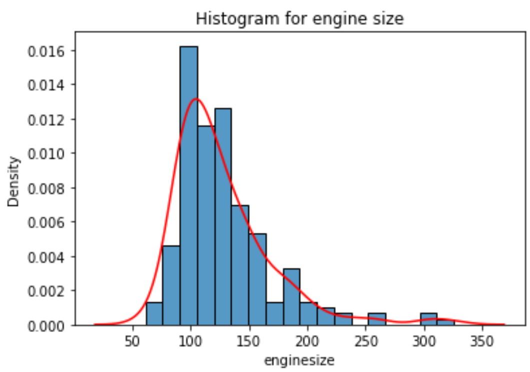
Out[29]: <AxesSubplot:title={'center':'Histogram for carwidth'}, xlabel='carwidth', ylabel='Density'>



enginesize

In [30]:
```
plt.title('Histogram for engine size')
sns.histplot(data.enginesize, stat = 'density')
sns.kdeplot(data.enginesize, color = 'red')
```

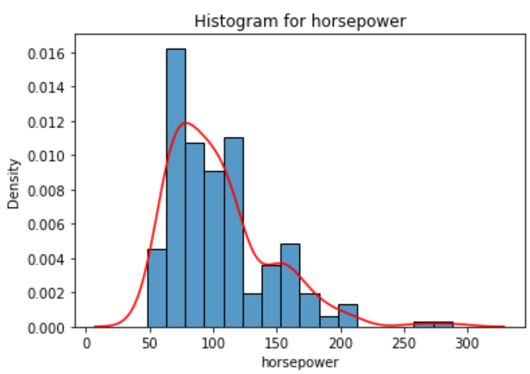
Out[30]: <AxesSubplot:title={'center':'Histogram for engine size'}, xlabel='enginesize', ylabel='Density'>



horsepower

In [31]:
```
plt.title('Histogram for horsepower')
sns.histplot(data.horsepower, stat = 'density')
sns.kdeplot(data.horsepower, color = 'red')
```

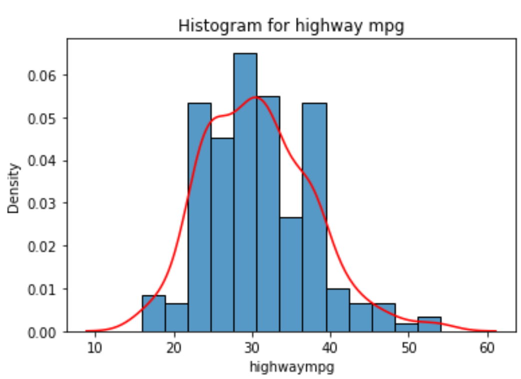
Out[31]: <AxesSubplot:title={'center':'Histogram for horsepower'}, xlabel='horsepower', ylabel='Density'>



highwaympg

In [32]:
```
plt.title('Histogram for highway mpg')
sns.histplot(data.highwaympg, stat = 'density')
sns.kdeplot(data.highwaympg, color = 'red')
```

Out[32]: <AxesSubplot:title={'center':'Histogram for highway mpg'}, xlabel='highwaympg', ylabel='Density'>



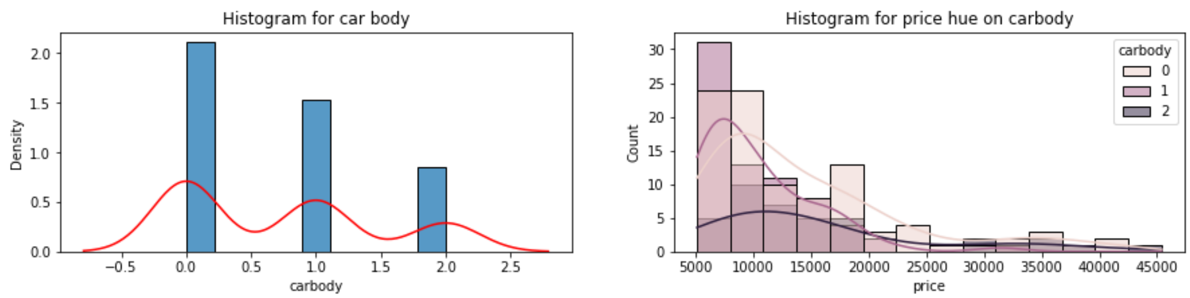
(2) Categorical Variables

carbody

0:sedan; 1:hatchback; 2:others

In [33]:
```
fig = plt.figure(figsize = (15,3))
ax1 = fig.add_subplot(1,2,1)
plt.title('Histogram for car body')
sns.histplot(data.carbody, stat = 'density')
sns.kdeplot(data.carbody, color = 'red')
ax2 = fig.add_subplot(1,2,2)
plt.title('Histogram for price hue on carbody')
sns.histplot(data = data, x = 'price', kde = True, hue = 'carbody' )
```

Out[33]: <AxesSubplot:title={'center':'Histogram for price hue on carbody'}, xlabel='price', ylabel='Count'>

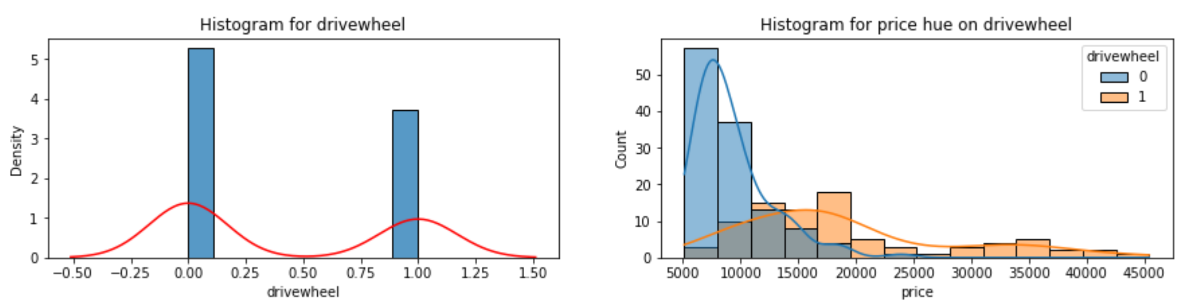


drivewheel

0:fwd; 1:others

In [34]:
```
fig = plt.figure(figsize = (15,3))
ax1 = fig.add_subplot(1,2,1)
plt.title('Histogram for drivewheel')
sns.histplot(data.drivewheel, stat = 'density')
sns.kdeplot(data.drivewheel, color = 'red')
ax2 = fig.add_subplot(1,2,2)
plt.title('Histogram for price hue on drivewheel')
sns.histplot(data = data, x = 'price', kde = True, hue = 'drivewheel' )
```

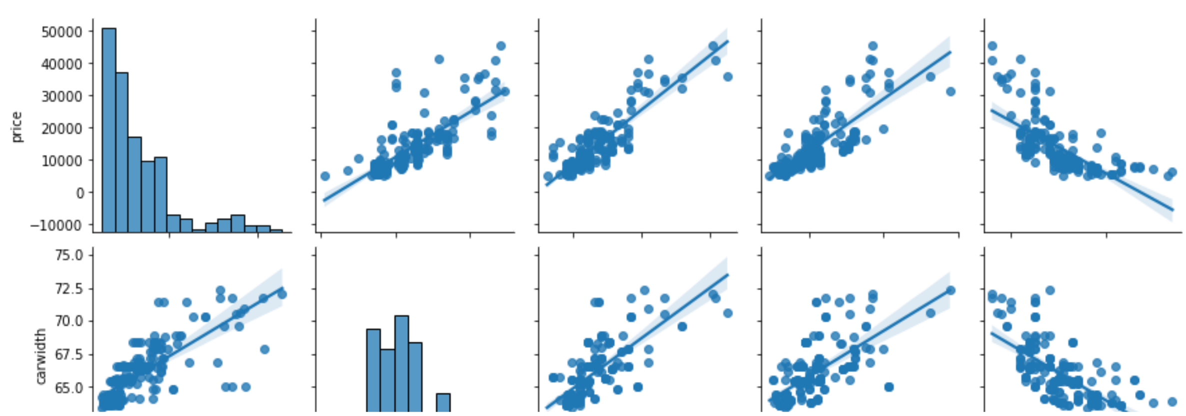
Out[34]: <AxesSubplot:title={'center':'Histogram for price hue on drivewheel'}, xlabel='price', ylabel='Count'>



4.2 Scatterplot Matrix

In [35]:
```
sns.pairplot(data[numeric_columns], kind = 'reg')
```

Out[35]: <seaborn.axisgrid.PairGrid at 0x7fc82cb7a8b0>

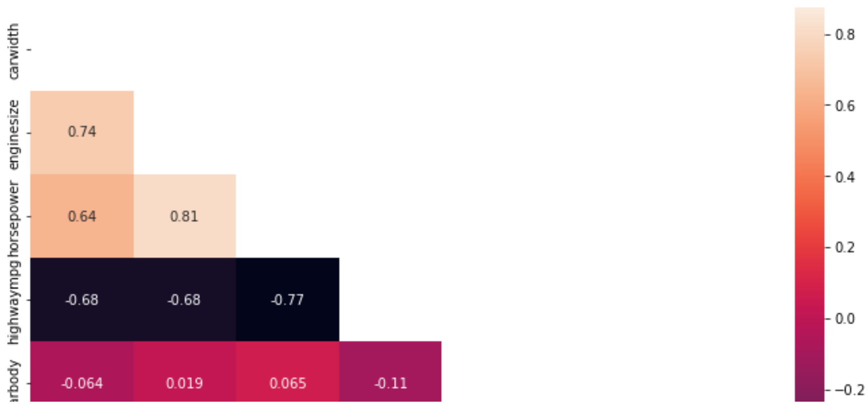


Through the scatter plot matrix, I find that there is a certain linear relationship between independent variables and the dependent variable, but whether to carry out transformation still needs to be tested and compared.

4.3 Correlation Matrix

In [36]:
```
plt.figure(figsize = (12,8))
corr = data[[ 'carwidth', 'enginesize', 'horsepower', 'highwaympg', 'carbody', 'drivewheel', 'price']].corr()
mask = np.zeros(corr.shape, dtype = bool)
mask[np.triu_indices(len(mask))] = True
sns.heatmap(corr, annot = True, mask = mask)
```

Out[36]: <AxesSubplot:>



high correlation:

enginesize VS horsepower (0.81)
highwaympg VS horsepower (-0.77)

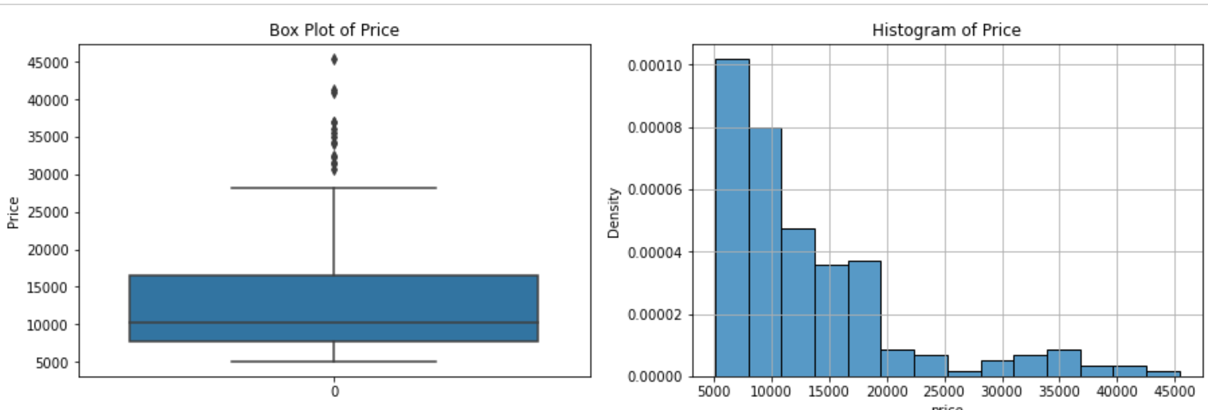
Taking multicollinearity into account, I will focus on these three variables, especially since enginesize and highwaympg are highly correlated with price, and I will discuss whether horsepower should be removed later.

4.4 Outliers/Unusual Features

(1) Numeric Variables

price

In [37]:
```
fig = plt.figure(figsize = (15,10))
ax1 = fig.add_subplot(2,2,1)
plt.title("Box Plot of Price")
sns.boxplot(data = data.price)
plt.ylabel("Price")

ax2 = fig.add_subplot(2,2,2)
plt.title("Histogram of Price")
sns.histplot(data.price, stat = "density")
plt.grid()
```



In [38]:
```
data.price.describe()
```

Out[38]:
```
count      205.000000
mean     13276.710571
std       7988.852332
min       5118.000000
25%       7788.000000
50%      10295.000000
75%      16503.000000
max      45400.000000
Name: price, dtype: float64
```

In [39]:
```
def up(x):
    return x.mean() + 3*x.std()
def down(x):
    return x.mean() - 3*x.std()
```

In [40]:
```
data[data.price > up(data.price)]
```

Out[40]:

	symboling	doornumber	carbody	drivewheel	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	boreratio	stroke	compressionratio	horse
16	0	0	0	1	103.5	193.8	67.9	53.7	3380	209	3.62	3.39	8.0	
73	0	1	0	1	120.9	208.1	71.7	56.7	3900	308	3.80	3.35	8.0	
74	1	0	2	1	112.0	199.2	72.0	55.4	3715	304	3.80	3.35	8.0	

In [41]:
```
data[data.price < down(data.price)]
```

Out[41]:

symboling	doornumber	carbody	drivewheel	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	boreratio	stroke	compressionratio	horsep

Although I find that there are not many outliers under the three sigma principle, I notice that according to the boxplot and histogram, the data distribution is not normal, so I transform the data.

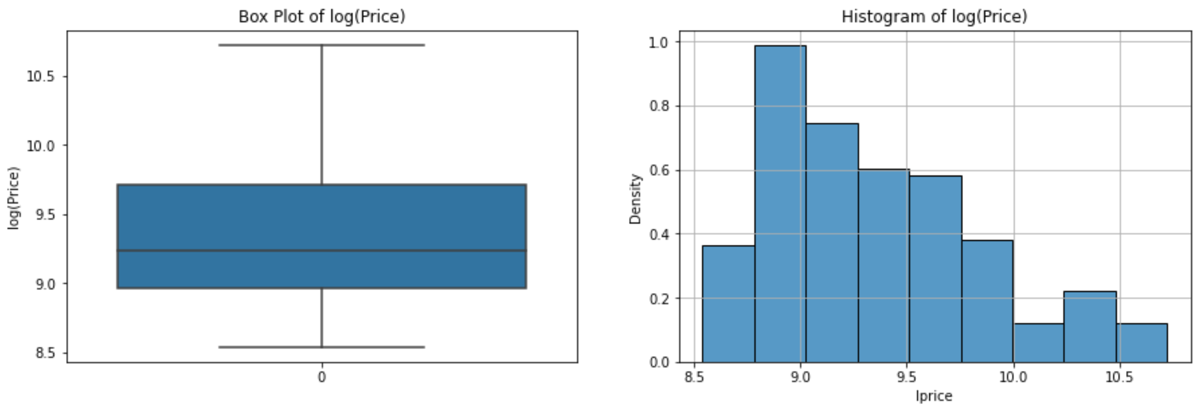
In [42]:
```
# price: -0.6280809555716815
bc_price, lambda_price = stats.boxcox(data['price'])
print('price:' ,lambda_price)
fig = plt.figure(figsize = (10,10))
ax1 = fig.add_subplot(2,2,1)
sns.histplot(data['price'], stat = 'density')
sns.kdeplot(data.price, color = 'red')
plt.title('Original: price')
ax2 = fig.add_subplot(2,2,2)
sns.histplot(bc_price, stat = 'density')
sns.kdeplot(bc_price, color = 'red')
plt.title('Box-Cox Transformed: price')
ax3 = fig.add_subplot(2,2,3)
stats.probplot(data.price, dist = "norm", plot = plt)
ax4 = fig.add_subplot(2,2,4)
stats.probplot(bc_price, dist = "norm", plot = plt)
```

```
price: -0.6280809555716815
```

Out[42]:
```
((array([-2.7088841 , -2.40021466, -2.22436022, -2.09847761, -1.99906119,
         -1.91619334, -1.84471334, -1.78157842, -1.72483748, -1.67316142,
         -1.62560232, -1.58145939, -1.54019924, -1.50140611, -1.46474936,
         -1.4299615 , -1.39682292, -1.36515103, -1.33479227, -1.30561622,
         -1.27751112, -1.25038041, -1.22414012, -1.19871667, -1.17404528,
         -1.15006853, -1.12673532, -1.10399994, -1.08182134, -1.0601625 ,
         -1.03898989, -1.01827309, -0.99798434, -0.9780983 , -0.95859171,
         -0.93944323, -0.92063316, -0.90214332, -0.88395686, -0.86605818,
         -0.84843274, -0.83106701, -0.81394837, -0.79706499, -0.78040581,
         -0.76396046, -0.74771918, -0.73167277, -0.71581259, -0.70013046,
         -0.68461865, -0.66926986, -0.65407714, -0.63903392, -0.62413394,
         -0.60937126, -0.59474021, -0.58023538, -0.56585161, -0.55158398,
         -0.53742776, -0.52337843, -0.50943166, -0.49558329, -0.48182931,
         -0.46816589, -0.45458932, -0.44109603, -0.42768258, -0.41434564,
         -0.401082  , -0.38788856, -0.37476229, -0.36170027, -0.34869968,
         -0.33575777, -0.32287185, -0.31003932, -0.29725765, -0.28452436,
         -0.27183703, -0.25919332, -0.24659091, -0.23402754, -0.221501  ,
         -0.20900912, -0.19654978, -0.18412087, -0.17172034, -0.15934617,
```

In [43]:
```
data['lprice'] = np.log(data.price)
```

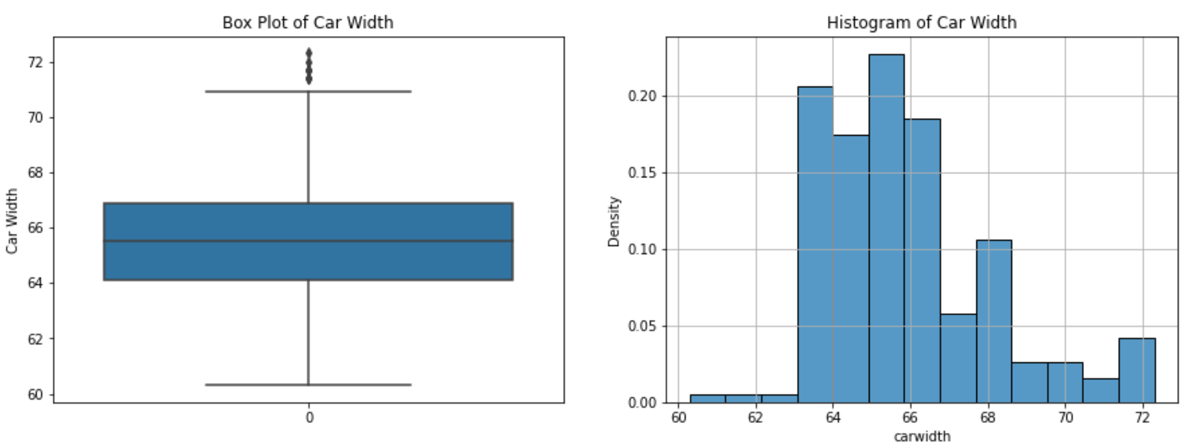
In [44]:
```python
# log(price)
fig = plt.figure(figsize = (15,10))
ax1 = fig.add_subplot(2,2,1)
plt.title("Box Plot of log(Price)")
sns.boxplot(data = data.lprice)
plt.ylabel("log(Price)")

ax2 = fig.add_subplot(2,2,2)
plt.title("Histogram of log(Price)")
sns.histplot(data.lprice, stat = "density")
plt.grid()
```



Since the number of my samples is not large, I don't want to delete the data directly, especially when I cannot judge whether it is unnormal theoretically. Though I got -0.628 from boxcox, I thought that would make the model difficult to explain, so I adopt log(price) and found no outlilers show.

carwidth

In [45]:
```python
fig = plt.figure(figsize = (15,5))
ax1 = fig.add_subplot(1,2,1)
plt.title("Box Plot of Car Width")
sns.boxplot(data = data.carwidth)
plt.ylabel("Car Width")
ax2 = fig.add_subplot(1,2,2)
plt.title("Histogram of Car Width")
sns.histplot(data.carwidth, stat = "density")
plt.grid()
```



In [46]:
```python
data.carwidth.describe()
```

Out[46]:
```
count    205.000000
mean      65.907805
std        2.145204
min       60.300000
25%       64.100000
50%       65.500000
75%       66.900000
max       72.300000
Name: carwidth, dtype: float64
```

In [47]:
```python
data[data.carwidth > up(data.carwidth)]
```

Out[47]:

symboling	doornumber	carbody	drivewheel	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	boreratio	stroke	compressionratio	horsep

In [48]:
```python
data[data.carwidth < down(data.carwidth)]
```

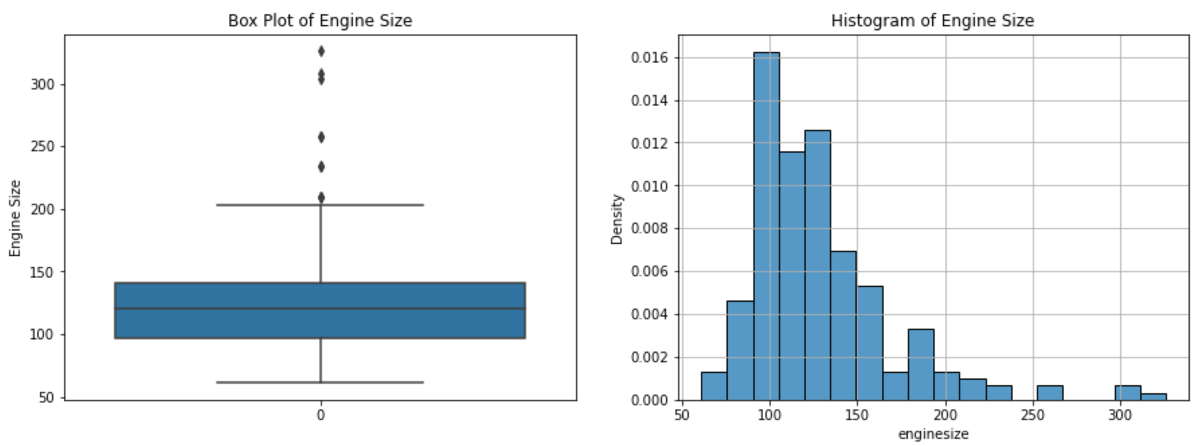
Out[48]:

symboling	doornumber	carbody	drivewheel	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	boreratio	stroke	compressionratio	horsep

While I can tell from boxplot that there are some high leverage points, when I look closely, the actual values are not that different much. Besides, under the three sigma principle, I don't see any value is defined as unusual, so I leave them alone.

enginesize

In [49]:
```python
fig = plt.figure(figsize = (15,5))
ax1 = fig.add_subplot(1,2,1)
plt.title("Box Plot of Engine Size")
sns.boxplot(data = data.enginesize)
plt.ylabel("Engine Size")
ax2 = fig.add_subplot(1,2,2)
plt.title("Histogram of Engine Size")
sns.histplot(data.enginesize, stat = "density")
plt.grid()
```



In [50]:
```python
data.enginesize.describe()
```

Out[50]:
```
count    205.000000
mean     126.907317
std       41.642693
min       61.000000
25%       97.000000
50%      120.000000
75%      141.000000
max      326.000000
Name: enginesize, dtype: float64
```

In [51]:
```python
data[data.enginesize > up(data.enginesize)]
```

Out[51]:

	symboling	doornumber	carbody	drivewheel	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	boreratio	stroke	compressionratio	hors
47	0	1	0	1	113.0	199.6	69.6	52.8	4066	258	3.63	4.17	8.1	
48	0	1	0	1	113.0	199.6	69.6	52.8	4066	258	3.63	4.17	8.1	
49	0	0	0	1	102.0	191.7	70.6	47.8	3950	326	3.54	2.76	11.5	
73	0	1	0	1	120.9	208.1	71.7	56.7	3900	308	3.80	3.35	8.0	
74	1	0	2	1	112.0	199.2	72.0	55.4	3715	304	3.80	3.35	8.0	

In [52]:
```python
data[data.enginesize < down(data.enginesize)]
```

Out[52]:

symboling	doornumber	carbody	drivewheel	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	boreratio	stroke	compressionratio	horsep

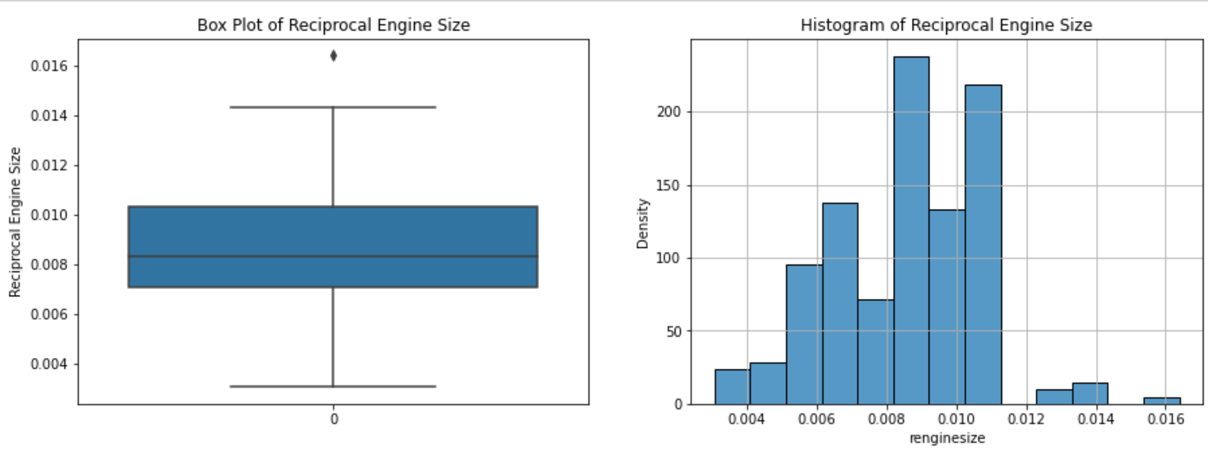
In [53]:
```python
# boxcox for enginesize
bc_enginesize, lambda_enginesize = stats.boxcox(data['enginesize'])
print('enginesize:' ,lambda_enginesize)
fig = plt.figure(figsize = (10,10))
ax1 = fig.add_subplot(2,2,1)
sns.histplot(data['enginesize'], stat = 'density')
sns.kdeplot(data.enginesize, color = 'red')
plt.title('Original: enginesize')
ax2 = fig.add_subplot(2,2,2)
sns.histplot(bc_enginesize, stat = 'density')
sns.kdeplot(bc_enginesize, color = 'red')
plt.title('Box-Cox Transformed: enginesize')
ax3 = fig.add_subplot(2,2,3)
stats.probplot(data.enginesize, dist = "norm", plot = plt)
ax4 = fig.add_subplot(2,2,4)
stats.probplot(bc_enginesize, dist = "norm", plot = plt)
```

```
enginesize: -0.9617338982169858
```

Out[53]:
```
((array([-2.7088841 , -2.40021466, -2.22436022, -2.09847761, -1.99906119,
         -1.91619334, -1.84471334, -1.78157842, -1.72483748, -1.67316142,
         -1.62560232, -1.58145939, -1.54019924, -1.50140611, -1.46474936,
         -1.4299615 , -1.39682292, -1.36515103, -1.33479227, -1.30561622,
         -1.27751112, -1.25038041, -1.22414012, -1.19871667, -1.17404528,
         -1.15006853, -1.12673532, -1.10399994, -1.08182134, -1.0601625 ,
         -1.03898989, -1.01827309, -0.99798434, -0.9780983 , -0.95859171,
         -0.93944323, -0.92063316, -0.90214332, -0.88395686, -0.86605818,
         -0.84843274, -0.83106701, -0.81394837, -0.79706499, -0.78040581,
         -0.76396046, -0.74771918, -0.73167277, -0.71581259, -0.70013046,
         -0.68461865, -0.66926986, -0.65407714, -0.63903392, -0.62413394,
         -0.60937126, -0.59474021, -0.58023538, -0.56585161, -0.55158398,
         -0.53742776, -0.52337843, -0.50943166, -0.49558329, -0.48182931,
         -0.46816589, -0.45458932, -0.44109603, -0.42768258, -0.41434564,
         -0.401082  , -0.38788856, -0.37476229, -0.36170027, -0.34869968,
         -0.33575777, -0.32287185, -0.31003932, -0.29725765, -0.28452436,
         -0.27183703, -0.25919332, -0.24659091, -0.23402754, -0.221501  ,
         -0.20900912, -0.19654978, -0.18412087, -0.17172034, -0.15934617
```

In [54]:
```python
data['renginesize'] = 1/data['enginesize']
```

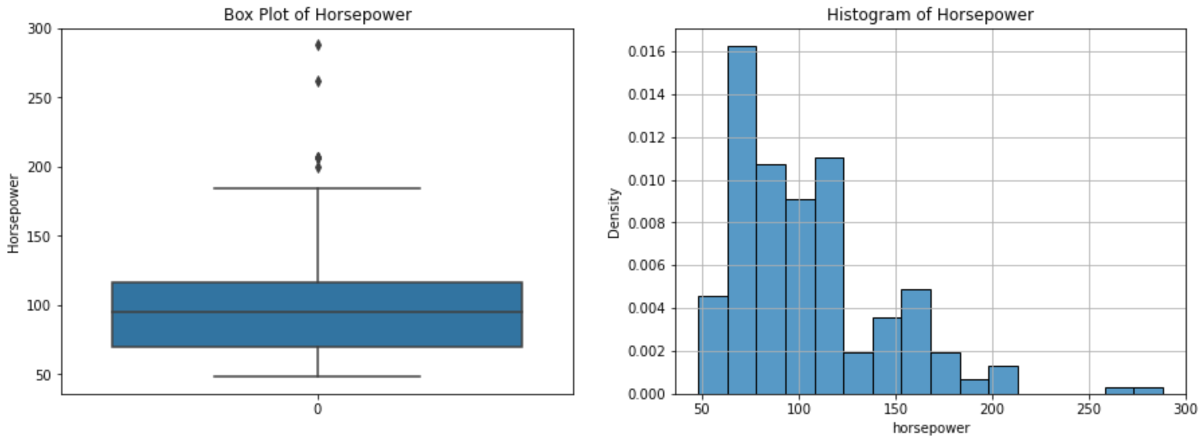
In [55]:
```python
# Reciprocal Engine Size
fig = plt.figure(figsize = (15,5))
ax1 = fig.add_subplot(1,2,1)
plt.title("Box Plot of Reciprocal Engine Size")
sns.boxplot(data = data.renginesize)
plt.ylabel("Reciprocal Engine Size")
ax2 = fig.add_subplot(1,2,2)
plt.title("Histogram of Reciprocal Engine Size")
sns.histplot(data.renginesize, stat = "density")
plt.grid()
```



Based on what I got from boxcox, I took reciprocal for enginesize. After the transformation, there is still high leverage point, I need to evaluate whether to delete later.

horsepower

In [56]:
```
fig = plt.figure(figsize = (15,5))
ax1 = fig.add_subplot(1,2,1)
plt.title("Box Plot of Horsepower")
sns.boxplot(data = data.horsepower)
plt.ylabel("Horsepower")
ax2 = fig.add_subplot(1,2,2)
plt.title("Histogram of Horsepower")
sns.histplot(data.horsepower, stat = "density")
plt.grid()
```



In [57]:
```
data.horsepower.describe()
```

Out[57]:
```
count    205.000000
mean     104.117073
std       39.544167
min       48.000000
25%       70.000000
50%       95.000000
75%      116.000000
max      288.000000
Name: horsepower, dtype: float64
```

In [58]:
```
data[data.horsepower > up(data.horsepower)]
```

Out[58]:

	symboling	doornumber	carbody	drivewheel	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	...	stroke	compressionratio	horsepowe
49	0	0	0	1	102.0	191.7	70.6	47.8	3950	326	...	2.76	11.5	26
129	1	0	1	1	98.4	175.7	72.3	50.5	3366	203	...	3.11	10.0	28

2 rows × 21 columns

In [59]:
```
data[data.horsepower < down(data.horsepower)]
```

Out[59]:

symboling	doornumber	carbody	drivewheel	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	...	stroke	compressionratio	horsepower

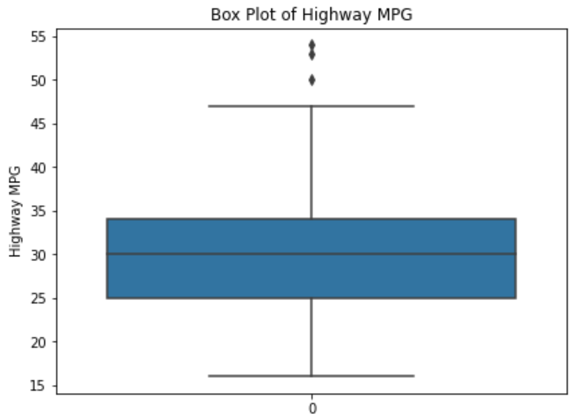
0 rows × 21 columns

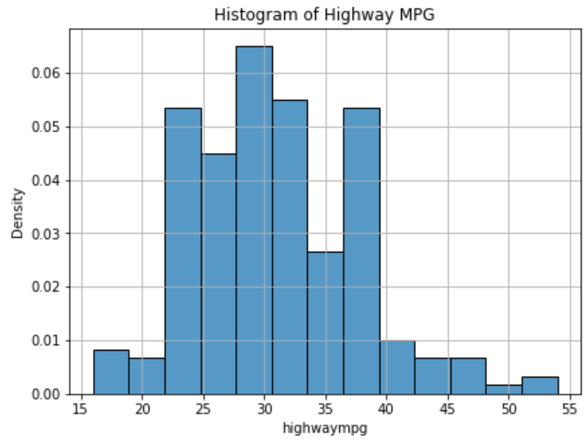
I identify these two high influential points and search for imformation online and find that both values are reasonable, so I keep them.

However, since there are only two large values and my sample size is small, if they are strong influential points, they may affect the accuracy of the model. Therefore, I will test whether to delete it in the future.

highwaympg

In [60]:
```
fig = plt.figure(figsize = (15,5))
ax1 = fig.add_subplot(1,2,1)
plt.title("Box Plot of Highway MPG")
sns.boxplot(data = data.highwaympg)
plt.ylabel("Highway MPG")
ax2 = fig.add_subplot(1,2,2)
plt.title("Histogram of Highway MPG")
sns.histplot(data.highwaympg, stat = "density")
plt.grid()
```





In [61]:
```
data.highwaympg.describe()
```

Out[61]:
```
count    205.000000
mean      30.751220
std        6.886443
min       16.000000
25%       25.000000
50%       30.000000
75%       34.000000
max       54.000000
Name: highwaympg, dtype: float64
```

In [62]:
```
data[data.highwaympg > up(data.highwaympg)]
```

Out[62]:

	symboling	doornumber	carbody	drivewheel	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	...	stroke	compressionratio	horsepower
18	2	0	1	0	88.4	141.1	60.3	53.2	1488	61	...	3.03	9.5	48
30	2	0	1	0	86.6	144.6	63.9	50.8	1713	92	...	3.41	9.6	58

2 rows × 21 columns

In [63]:
```
data[data.highwaympg < down(data.highwaympg)]
```

Out[63]:

symboling	doornumber	carbody	drivewheel	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	...	stroke	compressionratio	horsepower

0 rows × 21 columns

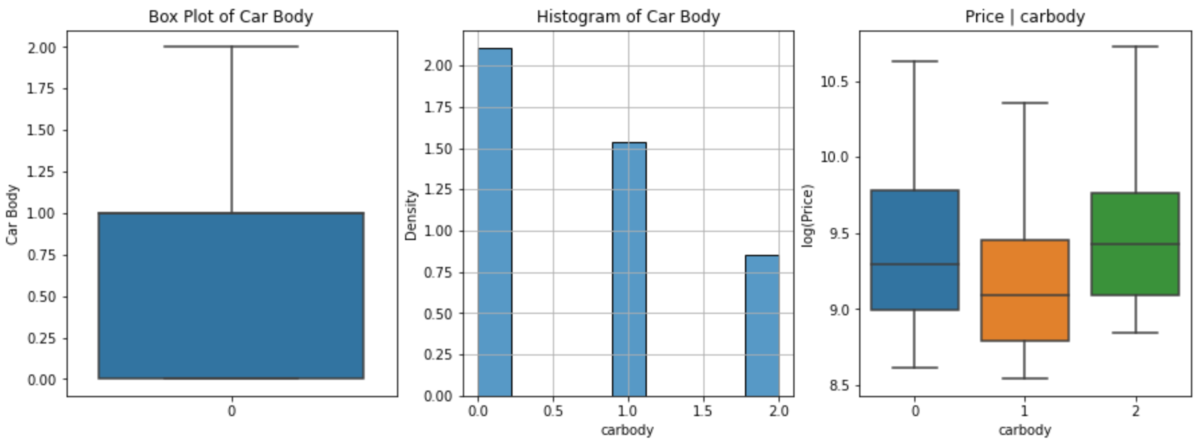
While I can tell from the boxplot that there are some high leverage points, when I look closely, the actual values are not that different much, so I leave them alone.

(2) Categorical Variables

carbody

```
In [64]: fig = plt.figure(figsize = (15,5))
ax1 = fig.add_subplot(1,3,1)
plt.title("Box Plot of Car Body")
sns.boxplot(data = data.carbody)
plt.ylabel("Car Body")
ax2 = fig.add_subplot(1,3,2)
plt.title("Histogram of Car Body")
sns.histplot(data.carbody, stat = "density")
plt.grid()
ax3 = fig.add_subplot(1,3,3)
plt.title("Price | carbody")
sns.boxplot(x='carbody',y='lprice', data = data)
plt.ylabel('log(Price)')
```

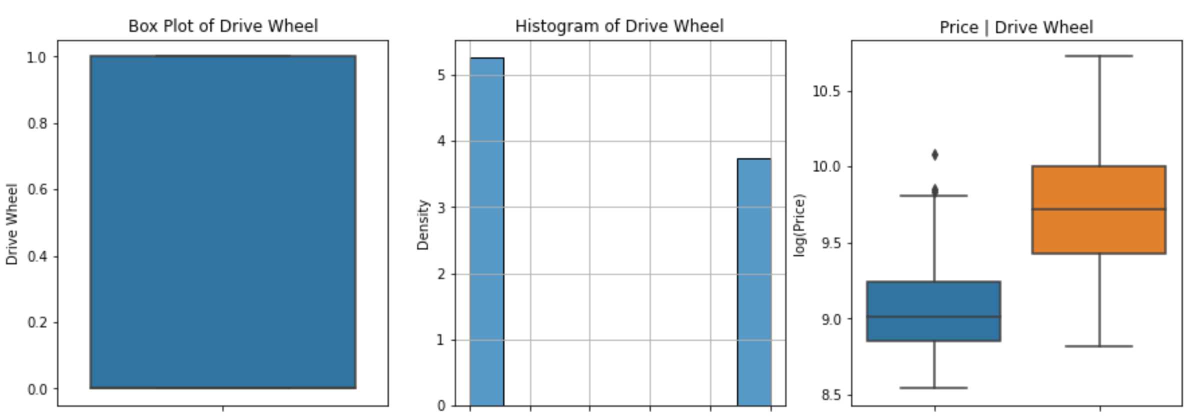
```
Out[64]: Text(0, 0.5, 'log(Price)')
```



drivewheel

```
In [65]: fig = plt.figure(figsize = (15,5))
ax1 = fig.add_subplot(1,3,1)
plt.title("Box Plot of Drive Wheel")
sns.boxplot(data = data.drivewheel)
plt.ylabel("Drive Wheel")
ax2 = fig.add_subplot(1,3,2)
plt.title("Histogram of Drive Wheel")
sns.histplot(data.drivewheel, stat = "density")
plt.grid()
ax3 = fig.add_subplot(1,3,3)
plt.title("Price | Drive Wheel")
sns.boxplot(x='drivewheel',y='lprice', data = data)
plt.ylabel('log(Price)')
```

```
Out[65]: Text(0, 0.5, 'log(Price)')
```



The respective box plots and histograms for both categorical variables, carbody and drivewheel, show a fairly balanced spread of observations between groups. Thus, I do not see any outliers or unusual features.

I are also able to draw some preliminary conclusions, such as cars with a hatchback body are cheaper compared to other body types. Besides, FWD cars will cost less than RWD and 4WD cars, which is consistent with my initial prediction.

5 Model Building and Evaluating

5.1 Model Building

In [66]:
```python
# Model1: original data
model1 = smf.ols(formula='price ~ carwidth + enginesize + horsepower + highwaympg + C(carbody) + C(drivewheel) ', data=
model1_fit = model1.fit()
# Type: dir(ols_fit) to look at other accessible attributes
print(model1_fit.summary2())
```

```
                        Results: Ordinary least squares
================================================================================
Model:                  OLS                   Adj. R-squared:         0.832
Dependent Variable:     price                 AIC:                    3907.5738
Date:                   2022-11-13 21:11      BIC:                    3934.1579
No. Observations:       205                   Log-Likelihood:         -1945.8
Df Model:               7                     F-statistic:            145.8
Df Residuals:           197                   Prob (F-statistic):     2.40e-74
R-squared:              0.838                 Scale:                  1.0694e+07
--------------------------------------------------------------------------------
                      Coef.     Std.Err.     t    P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
Intercept          -56148.2537 11788.0498 -4.7632 0.0000 -79395.2196 -32901.2879
C(carbody)[T.1]     -1533.9958   540.8803 -2.8361 0.0050  -2600.6546   -467.3371
C(carbody)[T.2]       -75.4528   642.4453 -0.1174 0.9066  -1342.4057   1191.5001
C(drivewheel)[T.1]   2302.3238   598.6220  3.8460 0.0002   1121.7939   3482.8536
carwidth              766.7896   172.2490  4.4516 0.0000    427.1009   1106.4783
enginesize             84.3690    11.0084  7.6641 0.0000     62.6595    106.0784
horsepower             56.3632    11.9746  4.7069 0.0000     32.7484     79.9780
highwaympg             61.6452    59.7833  1.0311 0.3037    -56.2521    179.5425
--------------------------------------------------------------------------------
Omnibus:                  20.236            Durbin-Watson:               0.837
Prob(Omnibus):            0.000             Jarque-Bera (JB):            41.657
Skew:                     0.466             Prob(JB):                    0.000
Kurtosis:                 5.002             Condition No.:               9618
================================================================================
* The condition number is large (1e+04). This might indicate             strong
multicollinearity or other numerical problems.
```

It can be seen from the residual QQ plot that it is not very normal. Meanwhile, JBtest shows that the null hypothesis that the normal distribution of residual is rejected.

In [67]:
```python
# Model2: transformed data : lprice
model2 = smf.ols(formula='lprice ~ carwidth + enginesize + horsepower + highwaympg + C(carbody) + C(drivewheel) ', data
model2_fit = model2.fit()
print(model2_fit.summary2())
```

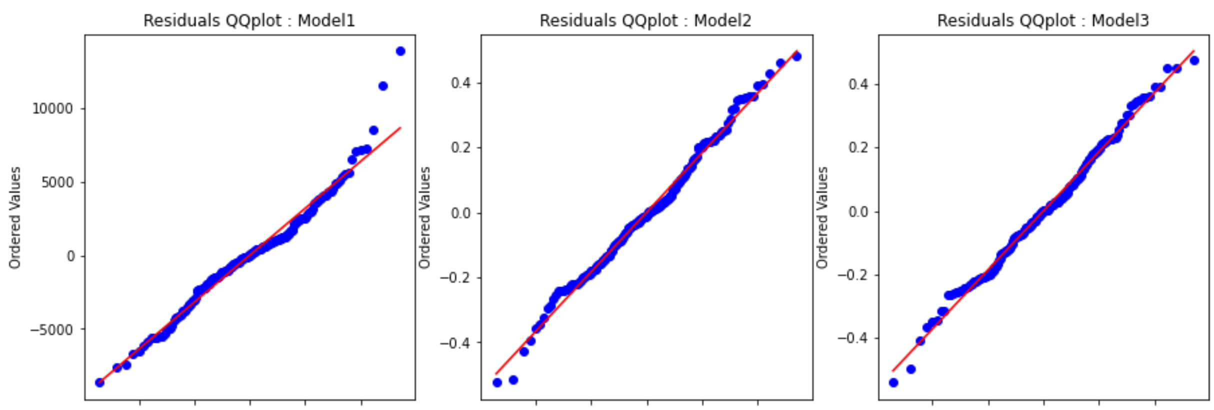
```
                 Results: Ordinary least squares
==================================================================
Model:              OLS              Adj. R-squared:     0.864
Dependent Variable: lprice           AIC:                -100.6196
Date:               2022-11-13 21:11 BIC:                -74.0355
No. Observations:   205              Log-Likelihood:     58.310
Df Model:           7                F-statistic:        186.3
Df Residuals:       197              Prob (F-statistic): 2.88e-83
R-squared:          0.869            Scale:              0.034495
------------------------------------------------------------------
                    Coef.  Std.Err.    t    P>|t|   [0.025  0.975]
------------------------------------------------------------------
Intercept           4.1582   0.6695  6.2110 0.0000  2.8379  5.4785
C(carbody)[T.1]    -0.1364   0.0307 -4.4409 0.0000 -0.1970 -0.0758
C(carbody)[T.2]    -0.0186   0.0365 -0.5102 0.6105 -0.0906  0.0533
C(drivewheel)[T.1]  0.1821   0.0340  5.3555 0.0000  0.1150  0.2491
carwidth            0.0703   0.0098  7.1819 0.0000  0.0510  0.0896
enginesize          0.0021   0.0006  3.3373 0.0010  0.0009  0.0033
horsepower          0.0043   0.0007  6.2634 0.0000  0.0029  0.0056
```

In [68]:
```python
# Model3: transformed data lprice, renginesize
model3 = smf.ols(formula='lprice ~ carwidth + renginesize + horsepower + highwaympg + C(carbody) + C(drivewheel)', dat
model3_fit = model3.fit()
print(model3_fit.summary2())
```

```
                 Results: Ordinary least squares
==================================================================
Model:              OLS              Adj. R-squared:     0.861
Dependent Variable: lprice           AIC:                -95.7552
Date:               2022-11-13 21:11 BIC:                -69.1711
No. Observations:   205              Log-Likelihood:     55.878
Df Model:           7                F-statistic:        181.3
Df Residuals:       197              Prob (F-statistic): 2.95e-82
R-squared:          0.866            Scale:              0.035323
------------------------------------------------------------------
                    Coef.   Std.Err.    t    P>|t|   [0.025  0.975]
------------------------------------------------------------------
Intercept            4.3297   0.7478  5.7899 0.0000   2.8550  5.8044
C(carbody)[T.1]     -0.1369   0.0314 -4.3620 0.0000  -0.1988 -0.0750
C(carbody)[T.2]     -0.0185   0.0370 -0.4995 0.6180  -0.0914  0.0545
C(drivewheel)[T.1]   0.1876   0.0343  5.4622 0.0000   0.1199  0.2553
carwidth             0.0738   0.0099  7.4755 0.0000   0.0544  0.0933
renginesize        -28.2850  11.3078 -2.5014 0.0132 -50.5849 -5.9852
horsepower           0.0048   0.0006  7.6050 0.0000   0.0036  0.0061
```

In [69]:
```python
fig = plt.figure(figsize = (15,5))
ax1 = fig.add_subplot(1,3,1)
stats.probplot(model1_fit.resid, dist = "norm", plot = plt)
plt.title("Residuals QQplot : Model1")
ax2 = fig.add_subplot(1,3,2)
stats.probplot(model2_fit.resid, dist = "norm", plot = plt)
plt.title("Residuals QQplot : Model2")
ax3 = fig.add_subplot(1,3,3)
stats.probplot(model3_fit.resid, dist = "norm", plot = plt)
plt.title("Residuals QQplot : Model3")
```

Out[69]:
```
Text(0.5, 1.0, 'Residuals QQplot : Model3')
```



According to the regression results and the residuals QQ plots, Model 2 and 3 is better.

5.2 Multicollinearity

In [70]:
```python
import patsy as pt
import statsmodels.stats.outliers_influence as smo

y, X = pt.dmatrices('lprice ~ carwidth + enginesize + horsepower + highwaympg', data = data,
                   return_type = 'dataframe')
X.head()
k = X.shape[1]
VIF = np.empty(k)
for i in range(k):
    VIF[i] = smo.variance_inflation_factor(X.values, i)

print('VIF:', VIF)
```

```
VIF: [2.54911515e+03 2.52028303e+00 3.78590321e+00 3.98038831e+00
 2.87023950e+00]
```

I uses 4 as the threshold, as predicted in the heatmap, since the VIF of the 'horsepower' is close to 4, I consider dropping it. But when I transform the variable, I find that the multicollinearity problem disappeare.

In [71]:
```
import patsy as pt
import statsmodels.stats.outliers_influence as smo

# Get the design matrix
y, X = pt.dmatrices('lprice ~ carwidth + renginesize + horsepower + highwaympg', data = data,
                   return_type = 'dataframe')
X.head()
k = X.shape[1]
VIF = np.empty(k)
for i in range(k):
    VIF[i] = smo.variance_inflation_factor(X.values, i)

print('VIF:', VIF)
```

```
VIF: [3.11864795e+03 2.50730894e+00 3.29532750e+00 3.34180765e+00
 2.87612712e+00]
```

So the aim now is to compare model3 and model 4.

In [72]:
```
# Model3: transformed data lprice, renginesize
model3 = smf.ols(formula='lprice ~ carwidth + renginesize + horsepower + highwaympg + C(carbody) + C(drivewheel)', dat
model3_fit = model3.fit()
```

In [73]:
```
# Model4:
model4 = smf.ols(formula='lprice ~ carwidth + enginesize + highwaympg  + C(carbody) + C(drivewheel)', data=data)
model4_fit = model4.fit()
```

5.3 Model Misspecification

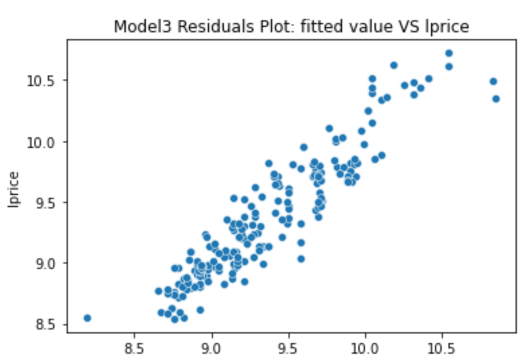
MODEL3

In [74]:
```
data["model3_fitted2"] = model3_fit.fittedvalues**2
ramseyreg = smf.ols('lprice ~ X + model3_fitted2', data).fit()
hypotheses = ['model3_fitted2 = 0']
ramseyreg.f_test(hypotheses)
```

Out[74]:
```
<class 'statsmodels.stats.contrast.ContrastResults'>
<F test: F=array([[54.06984466]]), p=4.9311844744667276e-12, df_denom=199, df_num=1>
```

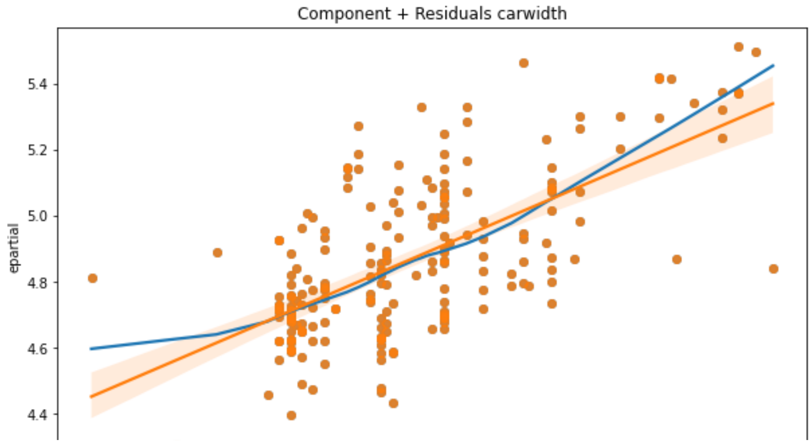
In [75]:
```
plt.title('Model3 Residuals Plot: fitted value VS lprice')
sns.scatterplot(model3_fit.fittedvalues, data['lprice'], )
```

Out[75]:
```
<AxesSubplot:title={'center':'Model3 Residuals Plot: fitted value VS lprice'}, ylabel='lprice'>
```



In [76]:
```
def ccpr_plot(model, data, variable):
    df_copy = data.copy()
    df_copy["epartial"] = model.resid + model.params[variable]*data[variable]
    plt.figure(figsize = (10, 6))
    sns.regplot(x = variable, y = "epartial", data =df_copy, lowess = True)
    sns.regplot(x = variable, y = "epartial", data =df_copy)
    plt.title("Component + Residuals "+variable)
```

In [77]:
```
ccpr_plot(model3_fit, data, "carwidth")
ccpr_plot(model3_fit, data, "renginesize")
ccpr_plot(model3_fit, data, "highwaympg")
ccpr_plot(model3_fit, data, "horsepower")
```



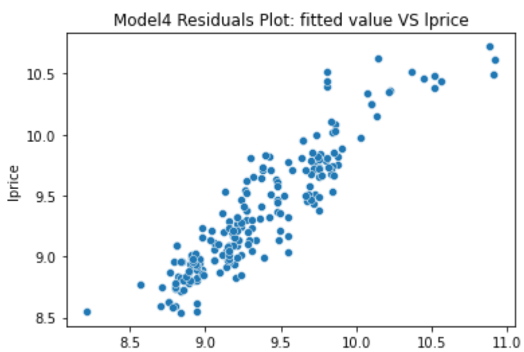
MODEL 4

In [78]:
```
data["model4_fitted2"] = model4_fit.fittedvalues**2
ramseyreg = smf.ols('lprice ~ X + model4_fitted2', data).fit()
hypotheses = ['model4_fitted2 = 0']
ramseyreg.f_test(hypotheses)
```

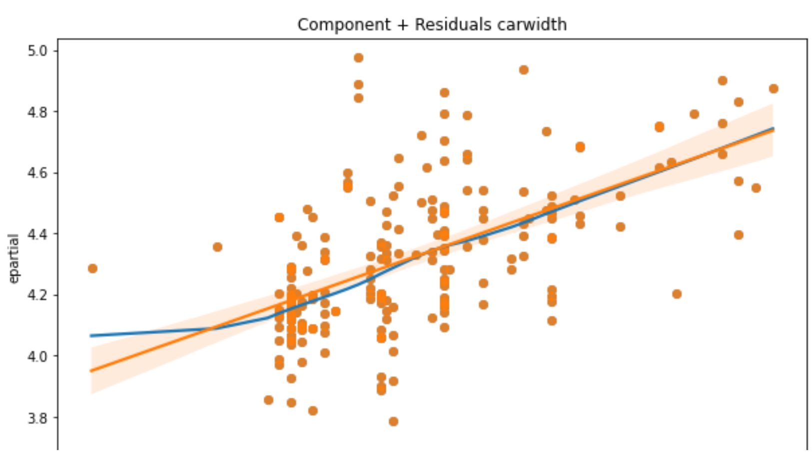
Out[78]:
```
<class 'statsmodels.stats.contrast.ContrastResults'>
<F test: F=array([[45.0980529]]), p=1.9179534071562455e-10, df_denom=199, df_num=1>
```

In [79]:
```
plt.title('Model4 Residuals Plot: fitted value VS lprice')
sns.scatterplot(model4_fit.fittedvalues, data['lprice'], )
```

Out[79]:
```
<AxesSubplot:title={'center':'Model4 Residuals Plot: fitted value VS lprice'}, ylabel='lprice'>
```



In [80]:
```
ccpr_plot(model4_fit, data, "carwidth")
ccpr_plot(model4_fit, data, "enginesize")
ccpr_plot(model4_fit, data, "highwaympg")
```



From the CPR Plot and the residuals plot, I think the model4 performs better, so I include quadratic terms and all interaction terms in the model to see if I can alleviate the problem of missing variables.

In [81]:
```
# Model41: Model4 + quadratic terms + interaction terms
model41 = smf.ols(formula='lprice ~ carwidth + enginesize + highwaympg + I(carwidth**2) + I(enginesize**2) + I(highway
model41_fit = model41.fit()
print(model41_fit.summary2())
```

```
                     Results: Ordinary least squares
======================================================================
Model:                OLS                Adj. R-squared:      0.861
Dependent Variable:   lprice             AIC:                 -87.7804
Date:                 2022-11-13 21:11   BIC:                 -34.6122
No. Observations:     205                Log-Likelihood:      59.890
Df Model:             15                 F-statistic:         84.90
Df Residuals:         189                Prob (F-statistic):  2.07e-75
R-squared:            0.871              Scale:               0.035405
----------------------------------------------------------------------
                      Coef.  Std.Err.    t    P>|t|   [0.025   0.975]
----------------------------------------------------------------------
Intercept             2.7326  12.2585  0.2229 0.8238 -21.4485 26.9137
carwidth              0.1527   0.3750  0.4072 0.6843  -0.5871  0.8925
enginesize            0.0056   0.0024  2.3375 0.0205   0.0009  0.0104
highwaympg           -0.1125   0.0274 -4.1045 0.0001  -0.1666 -0.0584
I(carwidth ** 2)     -0.0004   0.0028 -0.1409 0.8881  -0.0059  0.0051
I(enginesize ** 2)   -0.0000   0.0000 -4.0184 0.0001  -0.0000 -0.0000
I(highwaympg ** 2)    0.0014   0.0003  3.8912 0.0001   0.0007  0.0021
```

In [82]:
```
# I do the Ramsey Test again
data["model41_fitted2"] = model41_fit.fittedvalues**2
ramseyreg = smf.ols('lprice ~ X + model41_fitted2', data).fit()
print(ramseyreg.summary())
hypotheses = ['model41_fitted2 = 0']
ramseyreg.f_test(hypotheses)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 lprice   R-squared:                       0.879
Model:                            OLS   Adj. R-squared:                  0.876
Method:                 Least Squares   F-statistic:                     288.2
Date:                Sun, 13 Nov 2022   Prob (F-statistic):           4.48e-89
Time:                        21:11:05   Log-Likelihood:                 66.361
No. Observations:                 205   AIC:                            -120.7
Df Residuals:                     199   BIC:                            -100.8
Df Model:                           5
Covariance Type:            nonrobust
===================================================================================
                      coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------
Intercept           2.0675      0.347      5.963      0.000       1.384       2.751
X[0]                2.0675      0.347      5.963      0.000       1.384       2.751
X[1]                0.0120      0.012      1.013      0.312      -0.011       0.035
X[2]               13.8844     12.116      1.146      0.253     -10.008      37.777
X[3]                0.0024      0.001      3.874      0.000       0.001       0.004
X[4]                0.0022      0.003      0.661      0.509      -0.004       0.009
model41_fitted2     0.0455      0.005      9.065      0.000       0.036       0.055
==============================================================================
Omnibus:                        3.384   Durbin-Watson:                   1.050
Prob(Omnibus):                  0.184   Jarque-Bera (JB):                3.450
Skew:                           0.303   Prob(JB):                        0.178
Kurtosis:                       2.808   Cond. No.                     2.26e+17
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 9.96e-29. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

Out[82]:
```
<class 'statsmodels.stats.contrast.ContrastResults'>
<F test: F=array([[82.17421421]]), p=1.1935006028978514e-16, df_denom=199, df_num=1>
```

I find that the coefficient of 'model5_fitted2' is still significant, indicating that I may need to include higher-order terms, but I do not want to exceed quadratic. It's also possible that certain variables are ignored at the beginning when I select variables manually. However, I can't make changes based on the current information, so we'll move on.

I remove the least significant variables from the last regression in turn to get a new model with the most significant variables.

In [83]:

```python
# Model42: Model4 + significant quadratic terms + significant interaction terms
model42 = smf.ols(formula='lprice ~ carwidth + enginesize + highwaympg + I(enginesize**2) + I(highwaympg**2) + drivewhe
model42_fit = model42.fit()
print(model42_fit.summary2())
```

```
                  Results: Ordinary least squares
======================================================================
Model:                OLS              Adj. R-squared:     0.853
Dependent Variable:   lprice           AIC:                -84.0143
Date:                 2022-11-13 21:11 BIC:                -54.1073
No. Observations:     205              Log-Likelihood:     51.007
Df Model:             8                F-statistic:        149.3
Df Residuals:         196              Prob (F-statistic): 4.25e-79
R-squared:            0.859            Scale:              0.037231
----------------------------------------------------------------------
                       Coef.  Std.Err.    t    P>|t|   [0.025  0.975]
----------------------------------------------------------------------
Intercept              4.4752   1.0548  4.2429 0.0000  2.3951  6.5553
carwidth               0.0903   0.0158  5.7210 0.0000  0.0592  0.1214
enginesize             0.0083   0.0019  4.3753 0.0000  0.0045  0.0120
highwaympg            -0.0915   0.0185 -4.9347 0.0000 -0.1280 -0.0549
I(enginesize ** 2)    -0.0000   0.0000 -4.5206 0.0000 -0.0000 -0.0000
I(highwaympg ** 2)     0.0011   0.0003  4.1374 0.0001  0.0006  0.0016
drivewheel             3.2259   1.2212  2.6415 0.0089  0.8174  5.6343
```

In [84]:

```python
# Model42: check multicollinearity again
y, X = pt.dmatrices('lprice ~ carwidth + enginesize + highwaympg + I(enginesize**2) + I(highwaympg**2)', data = data,
                   return_type = 'dataframe')
X.head()
k = X.shape[1]
VIF = np.empty(k)
for i in range(k):
    VIF[i] = smo.variance_inflation_factor(X.values, i)
print('VIF:', VIF)
```

```
VIF: [3.32018450e+03 2.68845051e+00 2.91274250e+01 7.79495296e+01
 2.58864513e+01 6.84725445e+01]
```

In [85]:

```python
# Model42: drop some variables
y, X = pt.dmatrices('lprice ~ carwidth + highwaympg + I(enginesize**2)', data = data,
                   return_type = 'dataframe')
X.head()
k = X.shape[1]
VIF = np.empty(k)
for i in range(k):
    VIF[i] = smo.variance_inflation_factor(X.values, i)
print('VIF:', VIF)
```

```
VIF: [2.44152165e+03 2.31799494e+00 2.01714257e+00 2.02404210e+00]
```

In [86]:

```python
# Model43: Model4 + significant quadratic terms + interaction terms + mitigate multicollinearity
model43 = smf.ols(formula='lprice ~ carwidth + I(enginesize**2) + highwaympg + drivewheel*carwidth + drivewheel*engine
model43_fit = model43.fit()
print(model43_fit.summary2())
```

```
                  Results: Ordinary least squares
======================================================================
Model:                OLS              Adj. R-squared:     0.841
Dependent Variable:   lprice           AIC:                -68.8492
Date:                 2022-11-13 21:11 BIC:                -42.2651
No. Observations:     205              Log-Likelihood:     42.425
Df Model:             7                F-statistic:        155.5
Df Residuals:         197              Prob (F-statistic): 1.15e-76
R-squared:            0.847            Scale:              0.040277
----------------------------------------------------------------------
                       Coef.  Std.Err.    t    P>|t|   [0.025  0.975]
----------------------------------------------------------------------
Intercept              3.0544   1.0373  2.9446 0.0036  1.0088  5.1000
carwidth               0.0928   0.0164  5.6566 0.0000  0.0604  0.1251
I(enginesize ** 2)    -0.0000   0.0000 -2.9560 0.0035 -0.0000 -0.0000
highwaympg            -0.0162   0.0038 -4.3248 0.0000 -0.0237 -0.0088
drivewheel             2.6955   1.2632  2.1338 0.0341  0.2043  5.1866
drivewheel:carwidth   -0.0459   0.0206 -2.2327 0.0267 -0.0864 -0.0054
enginesize             0.0068   0.0019  3.5274 0.0005  0.0030  0.0106
drivewheel:enginesize  0.0042   0.0016  2.5682 0.0110  0.0010  0.0075
----------------------------------------------------------------------
Omnibus:                4.896        Durbin-Watson:           1.068
Prob(Omnibus):          0.086        Jarque-Bera (JB):        4.499
Skew:                   0.325        Prob(JB):                0.105
Kurtosis:               3.324        Condition No.:           2491492
======================================================================
* The condition number is large (2e+06). This might indicate
strong multicollinearity or other numerical problems.
```

In [87]:

```python
# Model31: Model3 + quadratic terms + interaction terms
model31 = smf.ols(formula='lprice ~ carwidth + renginesize + highwaympg + horsepower + I(carwidth**2) + I(renginesize*
model31_fit = model31.fit()
print(model31_fit.summary2())
```

```
                          Results: Ordinary least squares
==============================================================================
Model:                   OLS                  Adj. R-squared:        0.871
Dependent Variable:      lprice               AIC:                   -99.5499
Date:                    2022-11-13 21:11     BIC:                   -29.7667
No. Observations:        205                  Log-Likelihood:        70.775
Df Model:                20                   F-statistic:           69.97
Df Residuals:            184                  Prob (F-statistic):    7.00e-75
R-squared:               0.884                Scale:                 0.032703
------------------------------------------------------------------------------
                          Coef.     Std.Err.    t    P>|t|    [0.025    0.975]
------------------------------------------------------------------------------
Intercept                 5.4265    12.7588  0.4253 0.6711  -19.7459    30.5989
carwidth                  0.1222     0.3910  0.3126 0.7549   -0.6491     0.8936
renginesize            -347.8232   126.6149 -2.7471 0.0066 -597.6269   -98.0195
highwaympg               -0.0461     0.0343 -1.3439 0.1806   -0.1137     0.0216
horsepower                0.0045     0.0029  1.5535 0.1220   -0.0012     0.0102
I(carwidth ** 2)         -0.0001     0.0029 -0.0462 0.9632   -0.0059     0.0056
I(renginesize ** 2)   12116.9225 5023.2916  2.4121 0.0168 2206.2668 22027.5781
```

In [88]:

```python
# I do the Ramsey Test again
data["model31_fitted2"] = model31_fit.fittedvalues**2
ramseyreg = smf.ols('lprice ~ X + model31_fitted2', data).fit()
print(ramseyreg.summary())
hypotheses = ['model31_fitted2 = 0']
ramseyreg.f_test(hypotheses)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 lprice   R-squared:                       0.884
Model:                            OLS   Adj. R-squared:                  0.881
Method:                 Least Squares   F-statistic:                     379.8
Date:                Sun, 13 Nov 2022   Prob (F-statistic):           3.35e-92
Time:                        21:11:05   Log-Likelihood:                 70.658
No. Observations:                 205   AIC:                            -131.3
Df Residuals:                     200   BIC:                            -114.7
Df Model:                           4
Covariance Type:            nonrobust
===================================================================================
                      coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------
Intercept           2.3550      0.300      7.854      0.000       1.764       2.946
X[0]                2.3550      0.300      7.854      0.000       1.764       2.946
X[1]               -0.0004      0.011     -0.039      0.969      -0.022       0.021
X[2]               -0.0008      0.003     -0.239      0.811      -0.007       0.005
X[3]            -1.293e-06   1.57e-06     -0.825      0.411   -4.39e-06      1.8e-06
```

I get this model like what I did for Model4.

In [89]:

```python
# Model32: Model3 + quadratic terms + interaction terms
model32 = smf.ols(formula='lprice ~ carwidth + renginesize + highwaympg + horsepower + I(renginesize**2) + I(highwaymp
model32_fit = model32.fit()
print(model32_fit.summary2())
```

```
                          Results: Ordinary least squares
==============================================================================
Model:                   OLS                  Adj. R-squared:        0.863
Dependent Variable:      lprice               AIC:                   -96.7204
Date:                    2022-11-13 21:11     BIC:                   -60.1673
No. Observations:        205                  Log-Likelihood:        59.360
Df Model:                10                   F-statistic:           129.9
Df Residuals:            194                  Prob (F-statistic):    2.47e-80
R-squared:               0.870                Scale:                 0.034671
------------------------------------------------------------------------------
                          Coef.     Std.Err.    t    P>|t|    [0.025    0.975]
------------------------------------------------------------------------------
Intercept                 6.7041     1.4861  4.5111 0.0000    3.7730     9.6351
carwidth                  0.0979     0.0156  6.2568 0.0000    0.0670     0.1287
renginesize            -358.4004   108.9231 -3.2904 0.0012 -573.2259  -143.5749
highwaympg               -0.0544     0.0200 -2.7185 0.0072   -0.0939    -0.0149
horsepower                0.0027     0.0007  3.8475 0.0002    0.0013     0.0040
I(renginesize ** 2)   12849.3533 4422.9085  2.9052 0.0041 4126.1945 21572.5121
I(highwaympg ** 2)        0.0006     0.0003  2.3010 0.0225    0.0001     0.0012
```

In [90]:
```
# Model32: check multicollinearity again
y, X = pt.dmatrices('lprice ~ carwidth + renginesize + highwaympg + horsepower + I(renginesize**2) + I(highwaympg**2)'
                    return_type = 'dataframe')
X.head()
k = X.shape[1]
VIF = np.empty(k)
for i in range(k):
    VIF[i] = smo.variance_inflation_factor(X.values, i)

print('VIF:', VIF)
```

```
VIF: [4.72947818e+03 2.79399777e+00 7.14570616e+01 1.00404038e+02
 4.29678252e+00 5.69529667e+01 8.54823694e+01]
```

In [91]:
```
# Model22: drop some variables
y, X = pt.dmatrices('lprice ~ carwidth + horsepower + I(renginesize**2) + I(highwaympg**2)', data = data,
                    return_type = 'dataframe')
X.head()
k = X.shape[1]
VIF = np.empty(k)
for i in range(k):
    VIF[i] = smo.variance_inflation_factor(X.values, i)

print('VIF:', VIF)
```

```
VIF: [2.32004729e+03 2.25102225e+00 2.67588678e+00 2.48866174e+00
 2.38705462e+00]
```

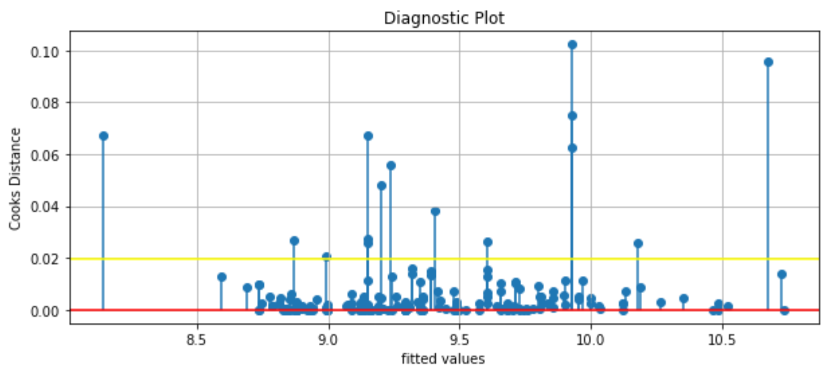
In [92]:
```
# Model33: Model3 + quadratic terms + interaction terms + mitigate multicollinearity
model33 = smf.ols(formula='lprice ~ carwidth + horsepower', data=data)
model33_fit = model33.fit()
print(model33_fit.summary2())
```

```
                 Results: Ordinary least squares
=================================================================
Model:              OLS              Adj. R-squared:     0.807
Dependent Variable: lprice           AIC:                -33.4949
Date:               2022-11-13 21:11 BIC:                -23.5259
No. Observations:   205              Log-Likelihood:     19.747
Df Model:           2                F-statistic:        427.3
Df Residuals:       202              Prob (F-statistic): 2.67e-73
R-squared:          0.809            Scale:              0.049007
-------------------------------------------------------------------
             Coef.    Std.Err.      t      P>|t|     [0.025   0.975]
-------------------------------------------------------------------
Intercept    1.4741     0.5878    2.5079   0.0129    0.3151   2.6331
carwidth     0.1089     0.0094   11.5752   0.0000    0.0904   0.1275
horsepower   0.0067     0.0005   13.1930   0.0000    0.0057   0.0077
-------------------------------------------------------------------
Omnibus:              14.482       Durbin-Watson:          0.994
Prob(Omnibus):        0.001        Jarque-Bera (JB):       25.908
Skew:                 0.361        Prob(JB):               0.000
Kurtosis:             4.585        Condition No.:          4865
=================================================================
* The condition number is large (5e+03). This might indicate
strong multicollinearity or other numerical problems.
```

After I removed the insignificant variables again, I got Model33. Compared to Model43, which has a higher Adjusted R-squared with lower AIC and BIC, the residuals of Model43 conform to the normal distribution, so I chose Model43 to move on.

5.4 Cook's Distance Plot

In [93]:
```
influence = model43_fit.get_influence()
cooks = influence.cooks_distance

plt.figure(figsize = (10, 4))
plt.scatter(model43_fit.fittedvalues, cooks[0])
plt.axhline(0, color = 'red')
plt.vlines(x = model43_fit.fittedvalues, ymin = 0, ymax = cooks[0])
threshold_cooks = 4/len(cooks[0])
plt.axhline(threshold_cooks, color = 'yellow')
plt.xlabel('fitted values')
plt.ylabel('Cooks Distance')
plt.title("Diagnostic Plot")
plt.grid()
```



I find that when I take 4/n as the threshold, there are some strong influential points, which are consistent with my original judgment based on the box and bar graphs, so I choose to delete these variables.

I rebuild the model with new dataset and get the Model44 while retaining Model43 as a contrast.

In [94]:
```
drop_indices = [i for i, v in enumerate(cooks[0]) if v > threshold_cooks]
data_new = data.drop(drop_indices)
```

In [95]:
```
# Model44: Model4 + significant quadratic terms + interaction terms + mitigate multicollinearity + drop influential po
model44 = smf.ols(formula='lprice ~ carwidth + I(enginesize**2) + highwaympg + drivewheel*carwidth + drivewheel*engine
model44_fit = model44.fit()
print(model44_fit.summary2())
```

```
                 Results: Ordinary least squares
====================================================================
Model:              OLS                  Adj. R-squared:     0.877
Dependent Variable: lprice               AIC:                -131.9982
Date:               2022-11-13 21:11     BIC:                -106.0220
No. Observations:   190                  Log-Likelihood:     73.999
Df Model:           7                    F-statistic:        193.6
Df Residuals:       182                  Prob (F-statistic): 8.56e-81
R-squared:          0.882                Scale:              0.028049
--------------------------------------------------------------------
                       Coef.  Std.Err.    t    P>|t|   [0.025  0.975]
--------------------------------------------------------------------
Intercept              2.6343   0.9226  2.8551 0.0048  0.8138  4.4547
carwidth               0.0995   0.0146  6.8317 0.0000  0.0708  0.1283
I(enginesize ** 2)    -0.0000   0.0000 -2.6550 0.0086 -0.0000 -0.0000
highwaympg            -0.0167   0.0033 -4.9909 0.0000 -0.0233 -0.0101
drivewheel             2.1470   1.1648  1.8432 0.0669 -0.1512  4.4452
drivewheel:carwidth   -0.0373   0.0191 -1.9600 0.0515 -0.0749  0.0002
enginesize             0.0067   0.0020  3.4190 0.0008  0.0028  0.0105
drivewheel:enginesize  0.0038   0.0015  2.4936 0.0135  0.0008  0.0068
--------------------------------------------------------------------
Omnibus:               4.078         Durbin-Watson:          1.175
Prob(Omnibus):         0.130         Jarque-Bera (JB):       3.762
Skew:                  0.339         Prob(JB):               0.152
Kurtosis:              3.128         Condition No.:          2449384
====================================================================
* The condition number is large (2e+06). This might indicate
strong multicollinearity or other numerical problems.
```

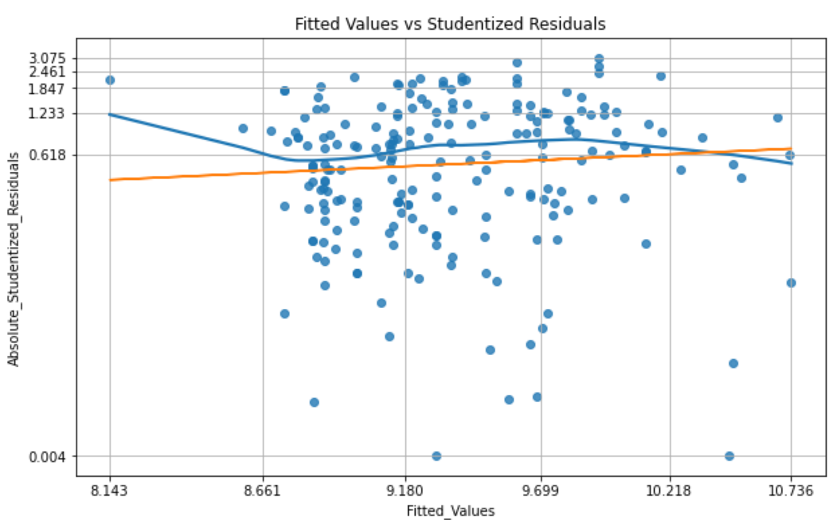
5.5 Heteroskedasticity

(1) Spread Level Plot

In [96]:
```
from matplotlib.ticker import ScalarFormatter

def spread_level(model, data):
    df_copy = data.copy()

    df_copy["Absolute_Studentized_Residuals"] = (np.abs(model.get_influence().resid_studentized))
    df_copy["Fitted_Values"] = (model.fittedvalues)

    slreg = smf.rlm("np.log(Absolute_Studentized_Residuals) ~ np.log(Fitted_Values)", df_copy).fit()
    slope = slreg.params[1]

    fig, ax = plt.subplots(figsize = (10, 6))
    ax.set_title("Fitted Values vs Studentized Residuals")
    sns.regplot(x = "Fitted_Values", y = "Absolute_Studentized_Residuals", data = df_copy, lowess = True, ax = ax)
    ax.plot(df_copy.Fitted_Values.values, np.exp(slreg.fittedvalues).values)

    ax.set_yscale('log')
    ax.set_xscale('log')

    ax.yaxis.set_major_formatter(ScalarFormatter())
    ax.xaxis.set_major_formatter(ScalarFormatter())

    ax.minorticks_off()

    ax.set_xticks(np.linspace(df_copy["Fitted_Values"].min(),df_copy["Fitted_Values"].max(), 6))
    ax.set_yticks(np.linspace(df_copy["Absolute_Studentized_Residuals"].min(),
                              df_copy["Absolute_Studentized_Residuals"].max(), 6))

    ax.grid()

    print("Suggested Power Transformation:", 1-slope)
```

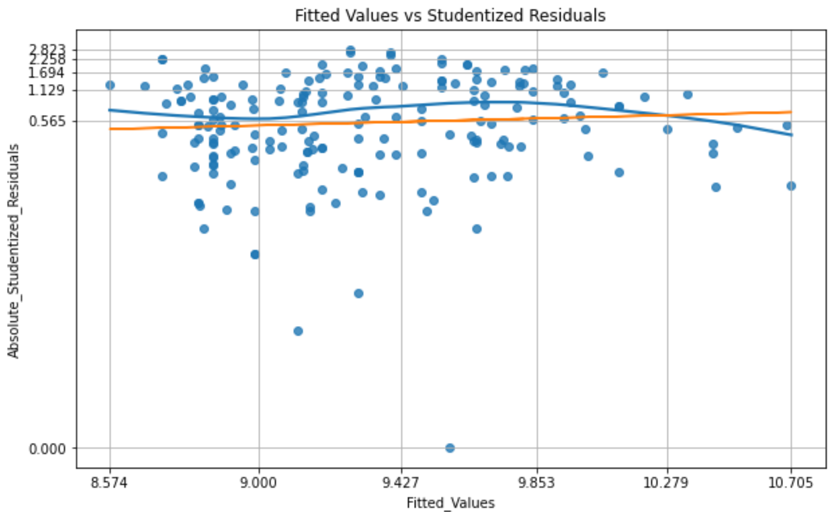
In [97]:
```
spread_level(model43_fit, data)
```

```
Suggested Power Transformation: -0.871198411204094
```



In [98]:
```
spread_level(model44_fit, data_new)
```

```
Suggested Power Transformation: -0.7231067118136434
```



(2) Bruesch-Pagan Test

In [99]:
```
# BP Test for Model43
data["res43"] = model43_fit.resid**2
aux_reg43 = smf.ols('res43 ~ carwidth + I(enginesize**2) + highwaympg ', data = data).fit()
f43 = aux_reg43.fvalue
fp43 = aux_reg43.f_pvalue

print("The F-Statistic for the Auxiliary Regression is: "+ str(f43) +" and the P-Value is: "+ str(fp43))
```

```
The F-Statistic for the Auxiliary Regression is: 4.383973007698641 and the P-Value is: 0.005158959596740866
```

In [100]:
```
# BP Test for Model44
data_new["res44"] = model44_fit.resid**2
aux_reg44 = smf.ols('res44 ~ carwidth + I(enginesize**2) + highwaympg', data = data_new).fit()
f44 = aux_reg44.fvalue
fp44 = aux_reg44.f_pvalue

print("The F-Statistic for the Auxiliary Regression is: "+ str(f44) +" and the P-Value is: "+ str(fp44))
```

```
The F-Statistic for the Auxiliary Regression is: 0.807428456432709 and the P-Value is: 0.491201410206189
```

(3) White Test

In [101]:
```
# White Test for Model43
y, X = pt.dmatrices('lprice ~ carwidth + I(enginesize**2) + highwaympg', data,
                   return_type = 'dataframe')
print(sm.stats.diagnostic.het_white(model43_fit.resid, X))
print("The F-Statistic for the Auxiliary Regression is: "+ '5.785203259243668' +" and the P-Value is: "+ '3.9981811207
```

```
(43.20167155628193, 1.9794236449470636e-06, 5.785203259243668, 3.998181120789298e-07)
The F-Statistic for the Auxiliary Regression is: 5.785203259243668 and the P-Value is: 3.998181120789298e-07
```

In [102]:
```
# White Test for Model44
y, X = pt.dmatrices('lprice ~ carwidth + I(enginesize**2) + highwaympg ', data_new,
                   return_type = 'dataframe')
print(sm.stats.diagnostic.het_white(model44_fit.resid, X))
print("The F-Statistic for the Auxiliary Regression is: "+ '1.6550572583428078' +" and the P-Value is: "+ '0.1029459820
```

```
(14.521359852973118, 0.10495058728794583, 1.6550572583428078, 0.10294598207221041)
The F-Statistic for the Auxiliary Regression is: 1.6550572583428078 and the P-Value is: 0.10294598207221041
```

Based on spread level plot and two test results I reject the null hypothesis that $\delta 1=\delta 2=\ldots=\delta k=0$ and conclude heteroscedasticity is present in the Model43 while I can safely conclude that I don't see the problem of heteroscedasticity in Model44.

Now I arrive my final model:

Model44

lprice = 2.6343 + 0.0995 * carwidth + 0.0067 * enginesize - 1.587e-05 * enginesize^2 - 0.0167*highwaympg

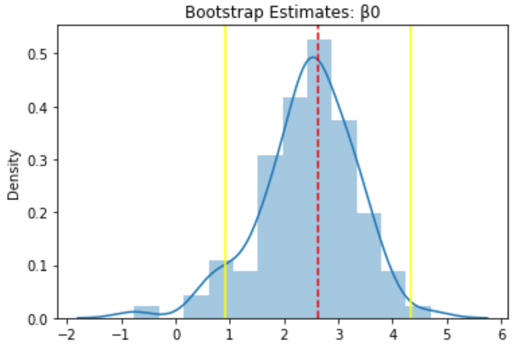
+ 2.1470 * drivewheel - 0.0373 * (drivewheel x carwidth) + 0.0038 * (drivewheel x enginesize)

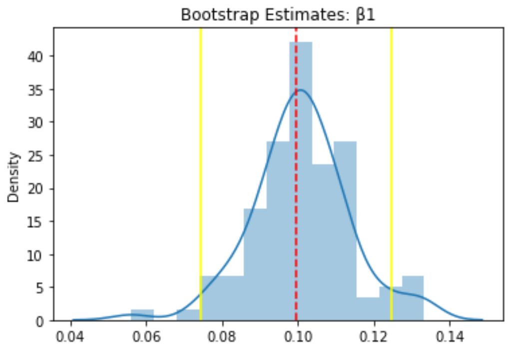
5.6 Bootstrapping

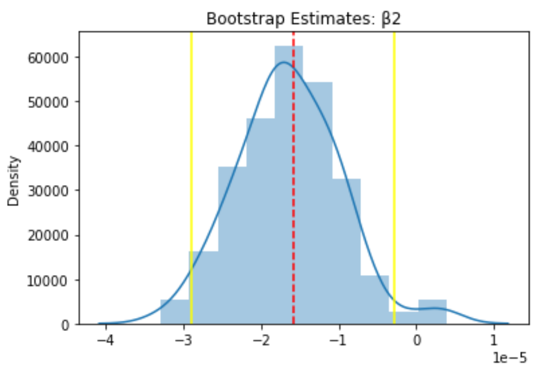
In [103]:
```
from scipy.stats import bootstrap
from sklearn.linear_model import LinearRegression
```

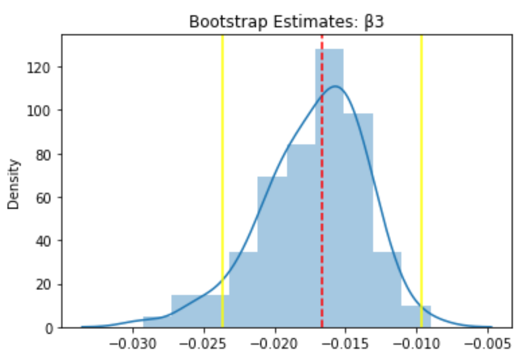
In [104]:
```
def bootstrap(k):
    boot_beta = []
    n_boots = 100
    n_points = data_new.shape[0]
    plt.figure()
    for _ in range(n_boots):
        sample_df = data_new.sample(n = n_points, replace = True)
        ols_model_temp = smf.ols(formula = 'lprice ~ carwidth + I(enginesize**2) + highwaympg + drivewheel*carwidth + d
        results_temp = ols_model_temp.fit()

        boot_beta.append(results_temp.params[k])
    se_bt = np.array(boot_beta).std(ddof = 1)
    CI_l, CI_h = [model44_fit.params[k] - 1.96*se_bt, model44_fit.params[k] + 1.96*se_bt]
    sns.distplot(boot_beta)
    plt.axvline(x=model44_fit.params[k],color='red', linestyle='--')
    plt.axvline(x = CI_l, color = 'yellow')
    plt.axvline(x = CI_h, color = 'yellow')
```

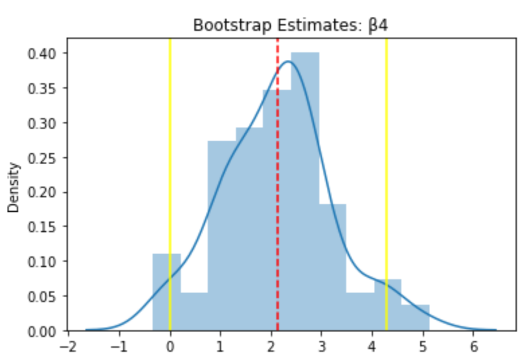
In [105]:
```
for i in range(8):
    bootstrap(i)
    plt.title('Bootstrap Estimates: β%i' %i)
```

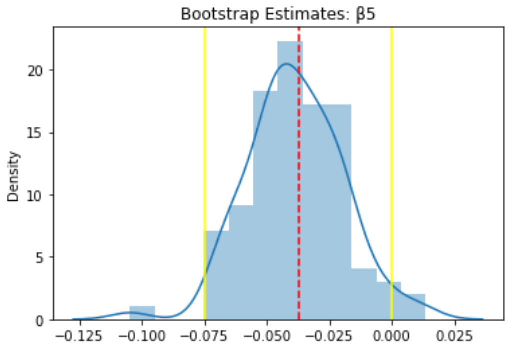


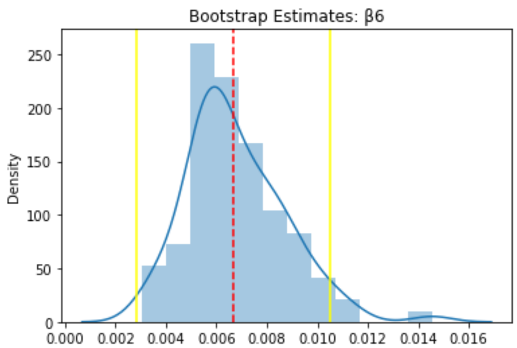


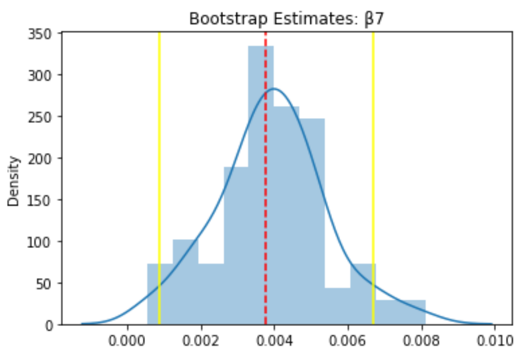












Based on these histograms, I find that the estimated parameter histograms realized by bootstrapping are close to normally distributed, and the initial estimated parameters represented by the red line locate in the center of the histograms, and the 0's are basically not within the yellow line, indicating that the estimated parameters I got initially are good.

5.7 Cross-Validation

5-Fold Cross-Validation

In [106]:
```
from sklearn.model_selection import cross_val_score
```

In [107]:
```
data_new['enginesize_sq'] = data_new.enginesize**2
data_new['drv_carw_int'] = data_new.drivewheel*data_new.carwidth
data_new['drv_eng_int'] = data_new.drivewheel*data_new.enginesize

x = data_new[['carwidth', 'enginesize', 'enginesize_sq', 'highwaympg', 'drivewheel', 'drv_carw_int', 'drv_eng_int']]
y = data_new[['lprice']]

regr = LinearRegression()
scores = cross_val_score(regr, x, y, cv=5, scoring='neg_mean_squared_error')
scores2 = cross_val_score(regr, x, y, cv=5, scoring = 'neg_root_mean_squared_error')
print('5-Fold CV MSE Scores:', scores)
print('5-Fold CV RMSE Scores:', scores2)
```

```
5-Fold CV MSE Scores: [-0.06193005 -0.0276839  -0.01977327 -0.04316925 -0.03530001]
5-Fold CV RMSE Scores: [-0.24885749 -0.16638479 -0.14061745 -0.20777212 -0.18788296]
```

In [108]:
```
avg_MSE = -scores.mean()
print("Average MSE:", avg_MSE)

avg_RMSE = -scores2.mean()
print("Average RMSE:", avg_RMSE)
```

```
Average MSE: 0.03757129457642022
Average RMSE: 0.19030296007051714
```

In [109]:
```
data_new.lprice.mean()
```

Out[109]:
```
9.332248509375736
```

In [110]:
```
avg_RMSE/data_new.lprice.mean()
```

Out[110]:
```
0.02039197304693798
```

Based on the 5-Fold CV MSE and RMSE scores, my model produced relatively consistent out of sample predictions for each fold.

I can gauge the overall peformance of my model across all the data by looking at the average MSE of my model across all five folds. my model's average MSE of 0.0376 indicates a favorable performance evaluation.

Next, I looked at the RMSE in order to further evaluate how well my model fit the data.

The mean of the RMSE scores generated by 5-Fold CV is 0.1903 (in absolute value). The mean log price is about 9.3322. Thus, I are approximately 0.0204, or 2.04% off in my predictions. I conclude that my model can predict the data with a fairly high level of accuracy.

Testing and Training

In [111]:
```
from sklearn.model_selection import train_test_split
from sklearn import linear_model
from sklearn import metrics
```

In [112]:
```
x = data_new[['carwidth', 'enginesize', 'enginesize_sq', 'highwaympg', 'drivewheel', 'drv_carw_int', 'drv_eng_int']]
y = data_new[['lprice']]

regr = LinearRegression()
model = regr.fit(x,y)
regr.coef_
regr.intercept_

# Split the data into train  (70%)/test(30%) samples:
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=0)

# Train the model:
regr = LinearRegression()
regr.fit(x_train, y_train)

# Make predictions based on the test sample
y_pred = regr.predict(x_test)

# Evaluate Performance
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
MAE: 0.12766043231233107
MSE: 0.028173947023643856
RMSE: 0.16785096670452587
```

The out of sample prediction accuracy of my model displayed limited variation between samples, as evidenced by the similar MSE and RMSE values I obtained from 5-fold cross validation and the Holdout Method. The consistency of these evaluation metrics indicates a positive assessment of my model's predictive performance.

6 Conclusion

After sufficient data cleaning and exploratory analysis on the dataset ‘Car_Price’, I have removed ‘car_ID’ and categorical variables with extremely unbalanced sample sizes. Using box-cox transformations, I deal with outliers/ unusual observations. I then transform ‘lprice’ and ‘rengineprice’ to fit the best model and check for multicollinearity using ‘VIF’. I then check for model misspecification using different models and with the inclusion of quadratic and interaction terms to finalize a model with the normal distribution of residuals. Cook’s distance plot is used to identify outliers and influential observations and I rework the dataset. Heteroskedasticity is tested using various means and I arrive at my final model which with homoskedasticity. After conducting a 5-fold cross-validation of estimates and looking at the MSE and RMSE scores, I concluded that the model can predict data with a high level of accuracy.

My final model is:
$lprice = 2.6343 + 0.0995 * carwidth + 0.0067 * enginesize - (1.587e-05) * enginesize^2 - 0.0167*highwaympg + 2.1470 * drivewheel - 0.0373 * (drivewheel x carwidth) + 0.0038 * (drivewheel x enginesize)$

Interpretation

Interpretation of regression coefficients for predictor variables:

Firstly, I think the interception is meaningless because the variables cannot take a value of 0 except for the dummy variables.

Holding other variables constant:

For a hundred unit increase in carwidth, price is expected to increase by 9.95% for FWD cars (i.e. drivewheel = 0). If the car is RWD or 4WD (i.e. drivewheel = 1), then price is expected to decrease by an additional 3.73%.

For a hundred unit increase in highway mpg, price is expected to decrease by 1.67%.

With regard to the effect of enginesize on price, the coefficient of quadratic term of enginesize is very low, however, I ultimately traded off economic interpretability for enhanced predictive power because it's statistically significant. But for interpretation, I will ignore it. For a hundred unit increase in enginesize, price is expected to increase by 0.67% for FWD cars (i.e. drivewheel = 0). If the car is RWD or 4WD (i.e. drivewheel = 1), then price is expected to increase by an additional .38%.

Marginal effects estimated

The marginal effects come from the interaction terms included in my final model: drivewheel x carwidth and drivewheel x enginesize, and a quadratic term. The coefficient of the drivewheel x carwidth variable measures the effect that carwidth has on price, given which type of drive wheel the car has. Because I converted drivewheel into a dummy variable where fwd = 0 and all other types of drivewheel take on the value 1, the coefficient measures the additional effect that carwidth has on price if the drive wheel is not fwd. Similarly, the coefficient of drivewheel x enginesize measures the additional effect of enginesize on car price if the drive wheel time is not fwd.

The inclusion of the quadratic term for enginesize also indicates a marginal effect on price. The negative sign of the quadratic term for engine size shows there are diminishing returns to increasing engine size. An increase of engine size increases car price up to a certain point, and then price starts to decrease with engine size.

7 Reference

Dronax. “Car Prices Dataset.” Kaggle, Kaggle, 20 June 2019, https://www.kaggle.com/code/dronax/car-prices-dataset/notebook#Cars-Prices-with-Multiple-Linear-Regression-and-RFE (https://www.kaggle.com/code/dronax/car-prices-dataset/notebook#Cars-Prices-with-Multiple-Linear-Regression-and-RFE). Accessed 12 November 2022.

Markus, Frank. “AWD, FWD, or RWD-Which Is Best, and Which Should You Buy?” MotorTrend, MotorTrend, 29 Sept. 2020, https://www.motortrend.com/features/awd-vs-fwd-vs-rwd-which-wheel-drive-is-best/#:~:text=What%20Is%20the%20Difference%20Between%20AWD%2C%20FWD%2C%20and%20RWD%3F&text=These%20acronyms%20refer%20to% (https://www.motortrend.com/features/awd-vs-fwd-vs-rwd-which-wheel-drive-is-best/#:~:text=What%20Is%20the%20Difference%20Between%20AWD%2C%20FWD%2C%20and%20RWD%3F&text=These%20acronyms%20refer%20to% Accessed 13 November 2022.