汽车价格分析与预测

案例背景

A公司是一家中国汽车生产公司。近年来，随着A公司不断的发展与壮大，管理层希望拓展海外市场，在B国建立汽车生产基地，并在当地生产与销售汽车。

为了能够在竞争中处于有利位置，A公司需要熟悉影响B国汽车定价的主要因素。这是因为，不同国家的汽车定价原则很可能是不同的，如果能够提前知道不同因素对汽车定价的影响，这对A公司是非常有利的。

任务说明

我们的任务是：

- 建立模型，确定影响汽车定价的主要因素。
- 在不同变量组合下，评估（预测）汽车的价格。

这样一来，就能够产生如下的价值：

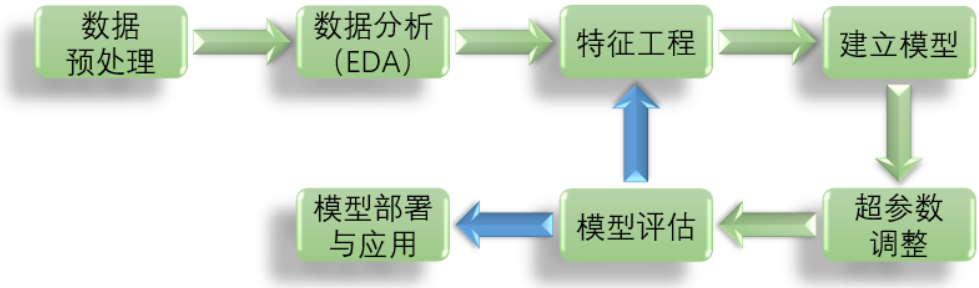
- 管理层借助于模型，熟悉海外新市场的定价动态。
- 根据特定的价格水平，进行汽车的设计，制定商业策略。

数据集描述

为了能够顺利完成任务，A公司联系了B国当地的一家汽车咨询公司（C公司）。根据市场调查，C公司收集了整个B国市场上各种类型汽车与价格的数据集。

属性	说明
car_ID	汽车编号。
symboling	保险风险等级。
CarName	汽车名称。
fueltype	燃料类型。
aspiration	增压机类型。
doornumber	车门数量。
carbody	车身类型。
drivewheel	车轮驱动。
enginelocation	发动机位置。
wheelbase	汽车的轴距。
carlength	汽车长度。
carwidth	汽车宽度。
carheight	汽车高度。
curbweight	汽车重量。
enginetype	发动机类型。
cylindernumber	气缸数量。
enginesize	发动机尺寸。
fuelsystem	燃料系统。
boreratio	口径比率。
stroke	冲程。
compressionratio	压缩比。
horsepower	马力。
peakrpm	最大每分钟转数。
citympg	城市路每加仑燃料所行英里数。
highwaympg	高速路每加仑燃料所行英里数。
price	汽车价格。

数据挖掘流程



数据预处理

加载数据集

导入所需要的库，并进行一些基本设置。

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

sns.set(style="darkgrid", font_scale=1.2)
plt.rcParams["font.family"] = "SimHei"
plt.rcParams["axes.unicode_minus"] = False
```

```
data = pd.read_csv("car.csv", header=0)
print(data.shape)
# 当列数较多时，会显示不完整，可以设置最大显示列数。
pd.set_option("display.max_columns", 100)
data.head()
```

```
(205, 26)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wh
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	front	88.
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	front	88.
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	front	94.
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	front	99.
4	5	2	audi 100ls	gas	std	four	sedan	4wd	front	99.

数据清洗

缺失值

我们可以调用DataFrame对象的info方法来显示数据的缺失情况，数据类型，占用空间等信息。

```
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
 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
```

重复值

```
data.duplicated().sum()
```

```
0
```

数据转换

CarName

对于CarName列，只需要保留前部分的厂商名称即可，对于厂商后面的内容，没有分析价值，可以去掉。

```
t = data["CarName"].str.split(" ", expand=True)
display(t.head())
data["CarName"] = t[0]
data["CarName"].value_counts()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	0	1	2	3	4
0	alfa-romero	giulia	None	None	None
1	alfa-romero	stelvio	None	None	None
2	alfa-romero	Quadrifoglio	None	None	None
3	audi	100	ls	None	None
4	audi	100ls	None	None	None

```
toyota         31
nissan         17
mazda          15
honda          13
mitsubishi     13
subaru         12
peugeot        11
volvo          11
volkswagen      9
dodge           9
buick           8
```

```
bmw             8
audi            7
plymouth        7
saab            6
isuzu           4
porsche         4
alfa-romero     3
chevrolet       3
jaguar          3
vw              2
maxda           2
renault         2
toyouta         1
vokswagen       1
Nissan          1
mercury         1
porcshce        1
Name: CarName, dtype: int64
```

我们成功将厂商名称提取出来，不过，名称中存在一些问题：

- 个别厂商名称存在一些拼写错误。
- 同一个厂商的大小写不统一。
- 同一个厂商，有的用全称，有的用简称。

因此，我们需要进行如下的转换：

- maxda =》 mazda
- Nissan =》 nissan
- porcshce =》 porsche
- toyouta =》 toyota
- vokswagen =》 volkswagen
- vw =》 volkswagen

```
data["CarName"] = data["CarName"].replace({"maxda": "mazda", "Nissan": "nissan", "porcshce": "porsche",
        "toyouta": "toyota", "vokswagen": "volkswagen", "vw": "volkswagen"})
data["CarName"].value_counts()
```

```
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: CarName, dtype: int64
```

探索性数据分析

探索性数据分析（EDA——Exploratory Data Analysis），是一种分析数据集以总结其主要特征的方法，在分析过程中，通常会通过可视化来进行辅助。探索性数据分析的作用为：

- 探索发现数据的一些特性。
- 探索发现变量与变量之间的关系。
- 探索发现变量与目标值之间的关系。
- 方便于特征工程的构建。

相关性分析（两个连续变量）

变量浏览

首先，我们可以输出所有的数值类型变量。

```
# 方案一。
# 定义列表，用来存放数值类型变量的名称。
number_var = []
for k, v in data.dtypes.items():
    if np.issubdtype(v, np.number):
        number_var.append(k)
print("数值类型变量个数：", len(number_var))
print(number_var)
```

```
数值类型变量个数： 16
['car_ID', 'symboling', 'wheelbase', 'carlength', 'carwidth', 'carheight', 'curbweight', 'enginesize', 'boreratio', 'stroke',
'compressionratio', 'horsepower', 'peakrpm', 'citympg', 'highwaympg', 'price']
```

```
# 方案二
# 也可以使用DataFrame对象的select_dtypes方法，该方法会返回DataFrame
# 对象，包含所有符合参数指定的类型列。
print(data.select_dtypes(np.number).columns)
```

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

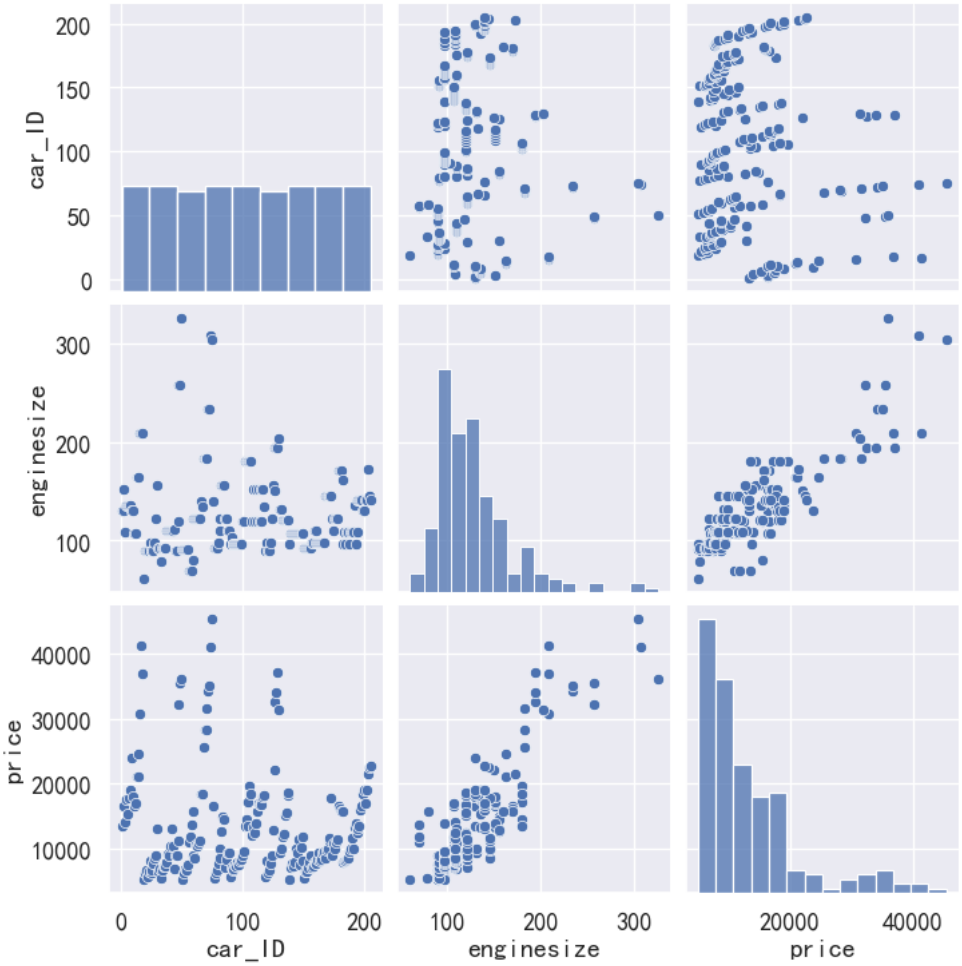
如果只是为了获取所有数值类型列的名称，优先使用哪种方案?
A 方案一。
B 方案二。
C 都可以。
D 根据具体情况而定。

散点图矩阵

对于两个连续类型的变量，可以绘制两个变量的散点图，来观察二者是否具有一定的关系。

```
# 绘制变量之间的散点图矩阵。
# vars：指定使用哪些变量绘制，默认使用所有变量。
# 此处变量较多，故只选择了三个变量。
sns.pairplot(data, vars=["car_ID", "enginesize", "price"])
```

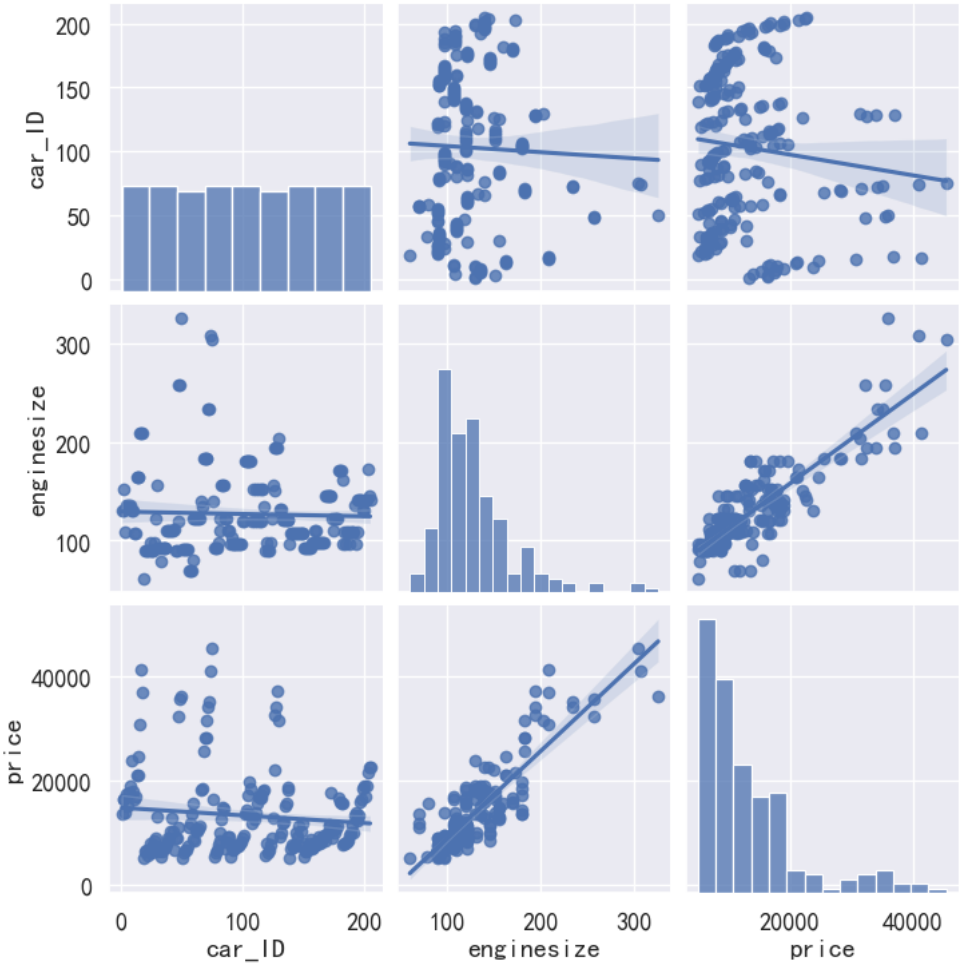
```
<seaborn.axisgrid.PairGrid at 0x26c60d8f070>
```



通过散点图矩阵，我们发现，enginesize与price存在较强的线性关系。另外，在绘制散点图矩阵的同时，也可以绘制回归线。

```
# kind：取值为scatter（默认）与reg。
#      scatter：绘制散点图。
#      reg：除了散点图外，同时绘制回归线与置信区间。
sns.pairplot(data, vars=["car_ID", "enginesize", "price"], kind="reg")
```

```
<seaborn.axisgrid.PairGrid at 0x26c61c69cf0>
```



协方差

协方差，体现的是两个变量之间的分散程度以及两个变量变化步调是否一致。设变量：

$$X = (x_1, x_2, \dots, x_n) \qquad Y = (y_1, y_2, \dots, y_n)$$

协方差的定义为：

$$Cov(X, Y) = E[(X - E(X))(Y - E(Y))]$$

在样本中，计算方式如下：

$$Cov(X, Y) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{n-1}$$

- $\bar{x}$：变量X的均值。

相关系数

相关系数，可以用来体现两个连续变量之间的相关性，是协方差的标准化版本，最为常用的为皮尔逊相关系数，衡量两个连续变量的线性相关性。两个变量的相关系数定义为：

$$r(X, Y) = \frac{Cov(X,Y)}{\sigma_X \sigma_Y}$$

在样本中，计算方式如下：

$$r(X, Y) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

- σ_X：变量X的标准差。

相关系数的取值范围为[-1, 1]，我们可以根据相关系数的取值（绝对值）来衡量两个变量的相关性：

- 0.8-1.0：极强相关

- 0.6-0.8：强相关
- 0.4-0.6：中等程度相关
- 0.2-0.4：弱相关
- 0.0-0.2：极弱相关或无相关

相关系数计算

我们以汽车长度（carlength）与汽车宽度（carwidth）为例，来求解两个变量的相关系数。

```
x = data["carlength"]
y = data["carwidth"]
# 计算carwidth与carheight的协方差。
a = (x - x.mean()) * (y - y.mean())
cov = np.sum(a) / (len(a) - 1)
print("协方差：", cov)
# 计算carwidth与carheight的相关系数。
corr = cov / (x.std() * y.std())
print("相关系数：", corr)
```

```
协方差： 22.261035150645633
相关系数： 0.8411182684818456
```

此外，Series对象也提供了cov与corr方法，来求解协方差与相关系数。

```
print("协方差：", x.cov(y))
print("相关系数：", x.corr(y))
```

```
协方差： 22.261035150645633
相关系数： 0.841118268481846
```

同样，DataFrame对象提供了计算了corr方法，可以计算任意两个数值变量的相关系数。

说明：

- 非数值类型不会在相关系数的结果中体现。

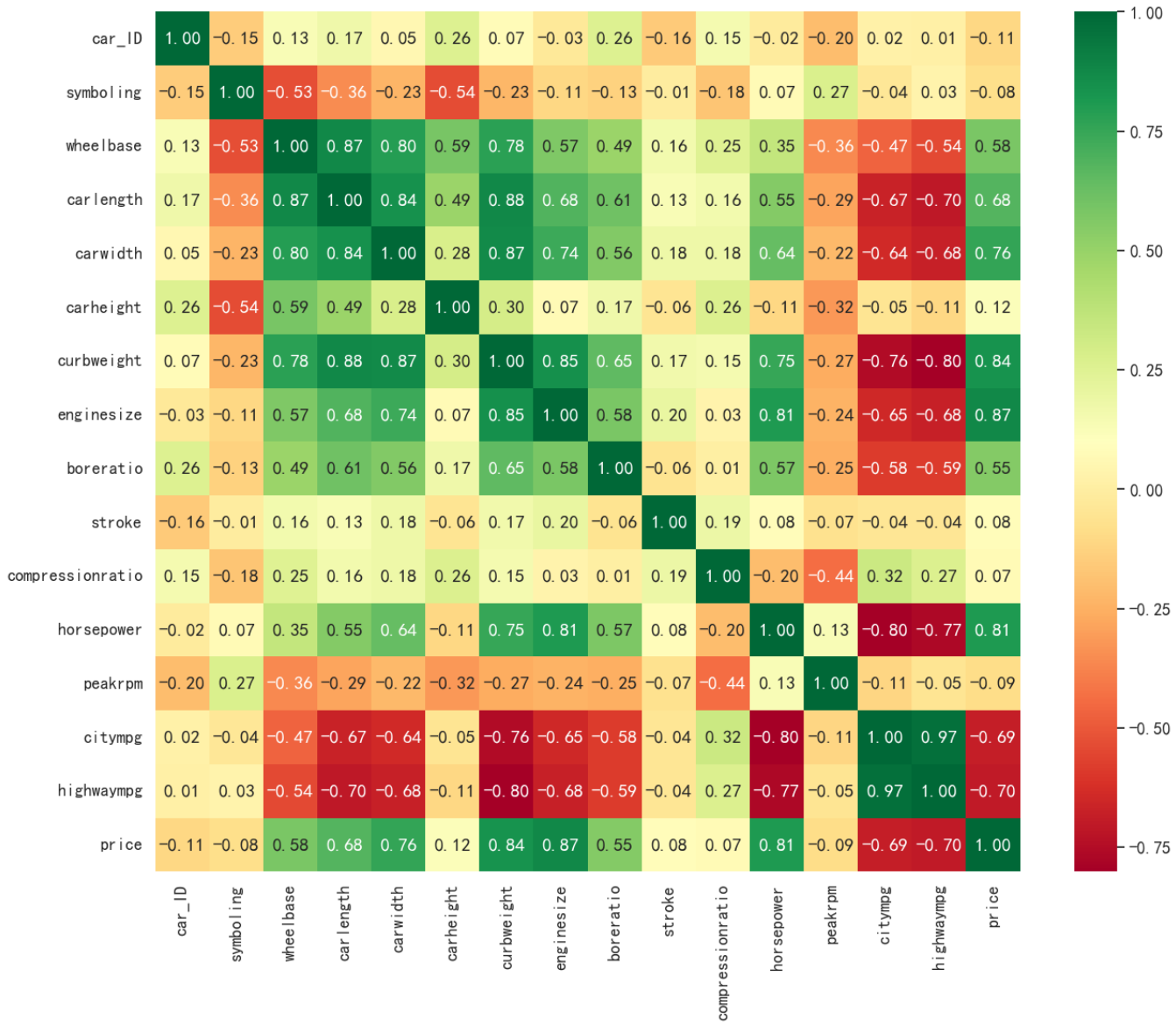
```
data.corr(numeric_only=True)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	car_ID	symboling	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	boreratio	stroke
car_ID	1.000000	-0.151621	0.129729	0.170636	0.052387	0.255960	0.071962	-0.033930	0.260064	-0.160824
symboling	-0.151621	1.000000	-0.531954	-0.357612	-0.232919	-0.541038	-0.227691	-0.105790	-0.130051	-0.008735
wheelbase	0.129729	-0.531954	1.000000	0.874587	0.795144	0.589435	0.776386	0.569329	0.488750	0.160959
carlength	0.170636	-0.357612	0.874587	1.000000	0.841118	0.491029	0.877728	0.683360	0.606454	0.129533
carwidth	0.052387	-0.232919	0.795144	0.841118	1.000000	0.279210	0.867032	0.735433	0.559150	0.182942
carheight	0.255960	-0.541038	0.589435	0.491029	0.279210	1.000000	0.295572	0.067149	0.171071	-0.055307
curbweight	0.071962	-0.227691	0.776386	0.877728	0.867032	0.295572	1.000000	0.850594	0.648480	0.168790
enginesize	-0.033930	-0.105790	0.569329	0.683360	0.735433	0.067149	0.850594	1.000000	0.583774	0.203129
boreratio	0.260064	-0.130051	0.488750	0.606454	0.559150	0.171071	0.648480	0.583774	1.000000	-0.055909
stroke	-0.160824	-0.008735	0.160959	0.129533	0.182942	-0.055307	0.168790	0.203129	-0.055909	1.000000
compressionratio	0.150276	-0.178515	0.249786	0.158414	0.181129	0.261214	0.151362	0.028971	0.005197	0.186110
horsepower	-0.015006	0.070873	0.353294	0.552623	0.640732	-0.108802	0.750739	0.809769	0.573677	0.080940
peakrpm	-0.203789	0.273606	-0.360469	-0.287242	-0.220012	-0.320411	-0.266243	-0.244660	-0.254976	-0.067964
citympg	0.015940	-0.035823	-0.470414	-0.670909	-0.642704	-0.048640	-0.757414	-0.653658	-0.584532	-0.042145
highwaympg	0.011255	0.034606	-0.544082	-0.704662	-0.677218	-0.107358	-0.797465	-0.677470	-0.587012	-0.043931
price	-0.109093	-0.079978	0.577816	0.682920	0.759325	0.119336	0.835305	0.874145	0.553173	0.079443

为了能够更清晰的呈现相关系数值，我们可以使用热图来展示相关系数。

```python
plt.figure(figsize=(15, 12))
# cmap：颜色图，控制显式的颜色风格。
# annot：是否显示数值，默认为False。
# fmt：指定数值格式化方式。
ax = sns.heatmap(data.corr(numeric_only=True), cmap=plt.cm.RdYlGn, annot=True, fmt=".2f")
```



结果统计

从相关系数的结果，我们可以得出以下结论：

- car_ID，symboling，carheight，stroke，compressionratio，peakrpm与price相关度较低。
- wheelbase，carlength，carwidth，boreratio，citympg，highwaympg与price中度相关。
  - 正相关：wheelbase，carlength，carwidth，boreratio。
  - 负相关：citympg，highwaympg。
- curbweight，enginesize，horsepower与price高度相关。
  - 正相关：curbweight，enginesize，horsepower。
  - 负相关：无。
- symboling虽然与price不相关，但该变量也可以当做类别变量看待。

相关性分析（类别变量与连续变量）

变量浏览

我们输出所有的类别类型变量。

```python
category_var = []
for k, v in data.dtypes.items():
    if not np.issubdtype(v, np.number):
        category_var.append(k)
print("类别变量个数：", len(category_var))
print(category_var)
```

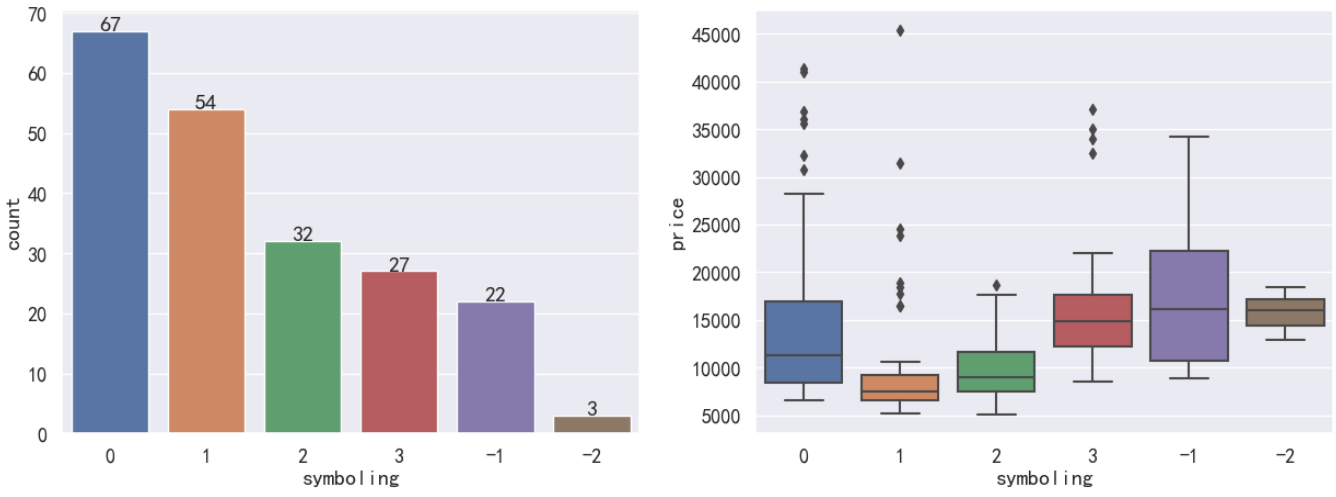
```
类别变量个数： 10
['CarName', 'fueltype', 'aspiration', 'doornumber', 'carbody', 'drivewheel', 'enginelocation', 'enginetype', 'cylindernumber',
'fuelsystem']
```

symboling

symboling（保险风险等级）在数据集中呈数值类型，但是，我们也可以将其视为类别类型，也许更加合理。

```python
from scipy import stats

def category_analysis(col_name, target_name="price"):
    """对类别变量进行分析。

    分析包括绘制柱形图与箱线图。并对类别变量进行假设检验。

    Parameters
    ----------
    col_name : str
        分类变量的名称。
    target_name : str，可选。
        目标变量的名称。

    """

    v = data[col_name].value_counts()
    # 如果分类数量超过10，则显示2行，否则显示1行。
    if len(v) > 10:
        row, col = 2, 1
    else:
        row, col = 1, 2
    fig, ax = plt.subplots(row, col)
    # 根据行数不同，设置不同的高度。
    fig.set_size_inches(15, row * 5)
    sns.countplot(x=col_name, data=data, order=v.index, ax=ax[0])
    # 在图像上绘制数值。
    for x, y in enumerate(v):
        text = ax[0].text(x, y, y)
        # 文本数值居中对齐。
        text.set_ha("center")
    sns.boxplot(x=col_name, y=target_name, order=v.index, data=data, ax=ax[1])
    if len(v) > 10:
        for a in ax:
            a.set_xticklabels(a.get_xticklabels(), rotation=30)
        plt.subplots_adjust(hspace=0.3)
    # 根据col_name进行分组。
    g = data.groupby(col_name)[target_name]
    group_data = []
    g.apply(lambda s: group_data.append(s.values))
    # Kruskal-Wallis H-test，用来检验各个分组的中位数是否相等。
    # 原假设：每个分组的中位数相等。
    # 备择假设：至少存在两个分组的中位数不等。
    print(stats.kruskal(*group_data))


category_analysis("symboling")
```

```
KruskalResult(statistic=55.05140268551899, pvalue=1.2739605541044965e-10)
```



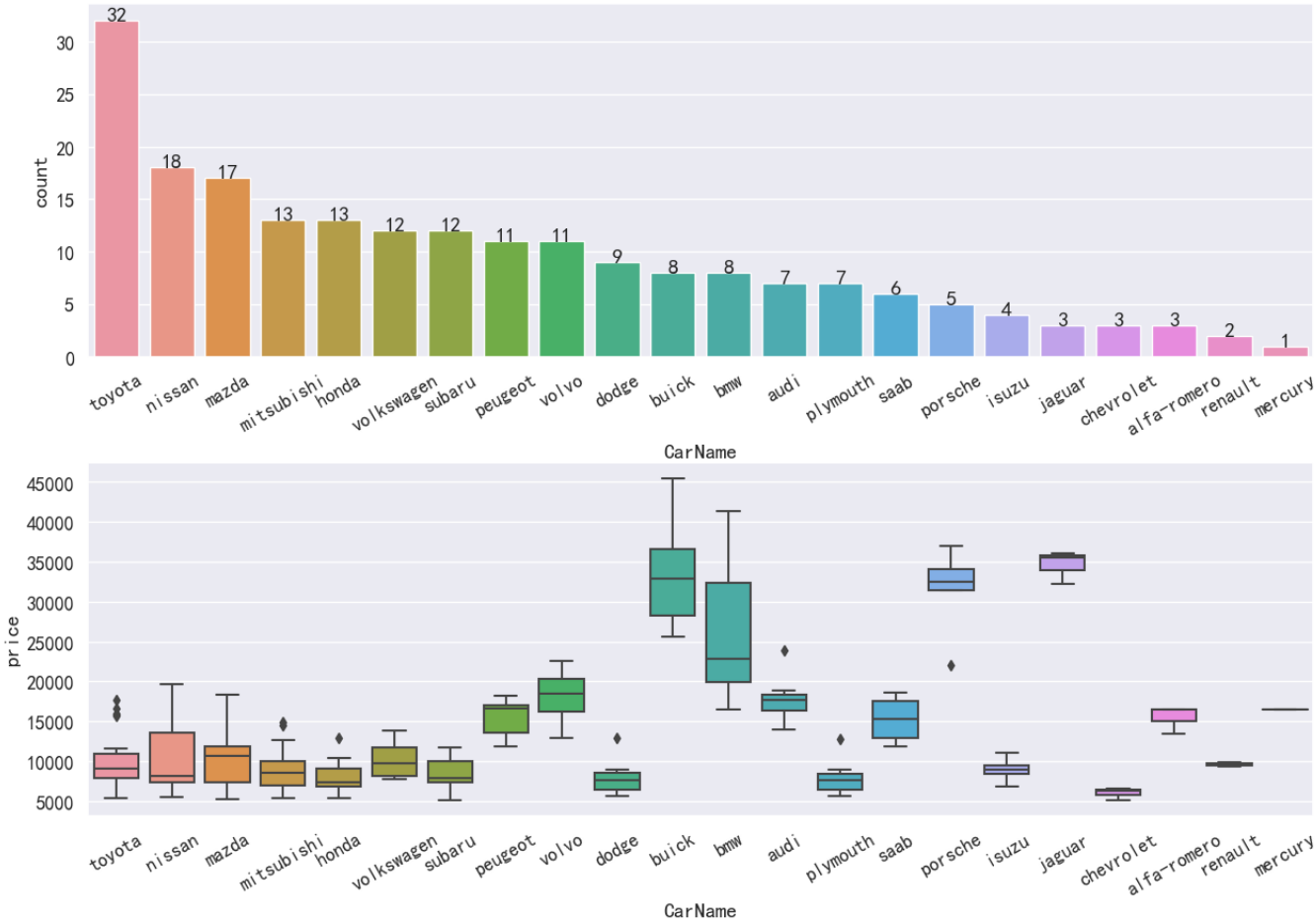
- 保险等级主要以0级与1级为主。
- Kruskal-Wallis H-test结果非常显著，说明不同风险等级，确实与汽车价格会有一定的关联。

- -2等级的数量很少。

CarName

```
category_analysis("CarName")
```

```
KruskalResult(statistic=128.0293000475285, pvalue=2.3580012078648455e-17)
```

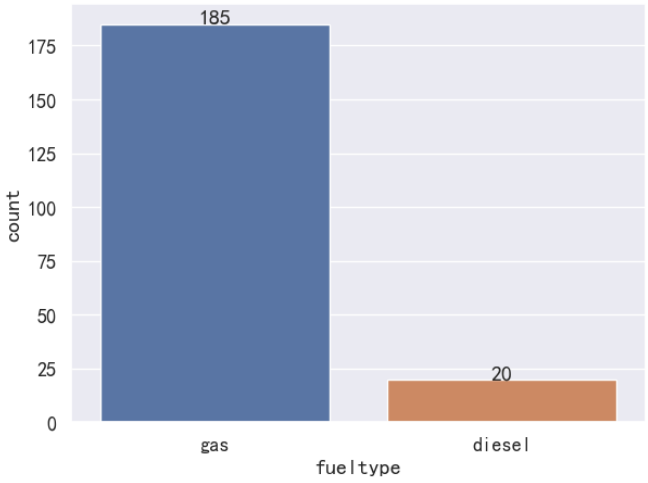


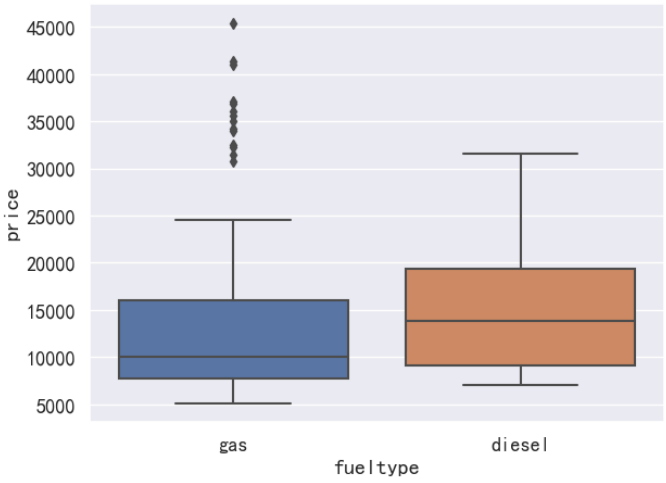
- 不同品牌的汽车，车型数量差异较大，车型数量以toyota为主。
- 不同品牌的汽车，在价格上有着很大的差别。
  - bmw，buick，porsche与jaguar的价格较高。
  - bmw与buick的价格浮动区间较大。

fueltype

```
category_analysis("fueltype")
```

```
KruskalResult(statistic=3.9833925913400314, pvalue=0.0459509243643051)
```



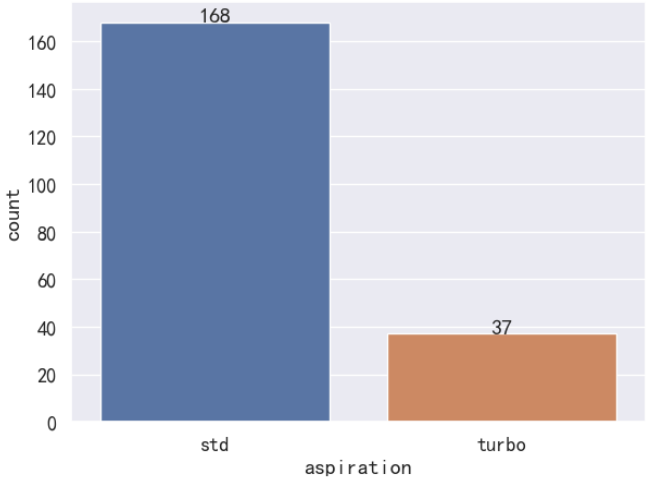


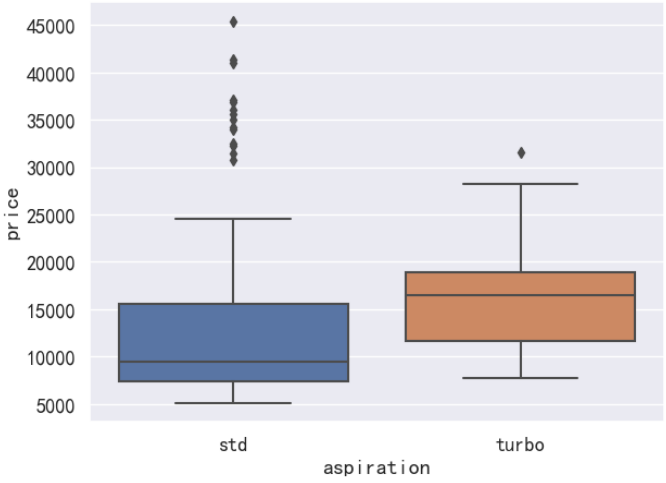
- 汽车主要以汽油为主，柴油的汽车数量较少。
- 汽油与柴油两种类型在价格上有所差异（但并不是非常显著）。
  - 柴油汽车价格普遍高于汽油汽车价格。

aspiration

```
category_analysis("aspiration")
```

```
KruskalResult(statistic=19.47319231451774, pvalue=1.020215018682171e-05)
```



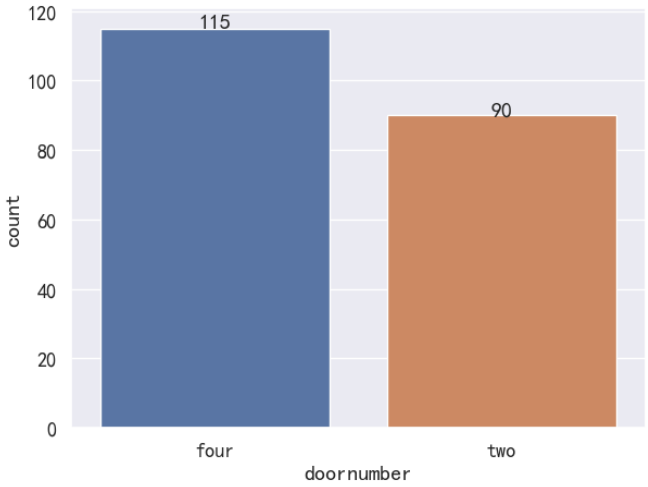


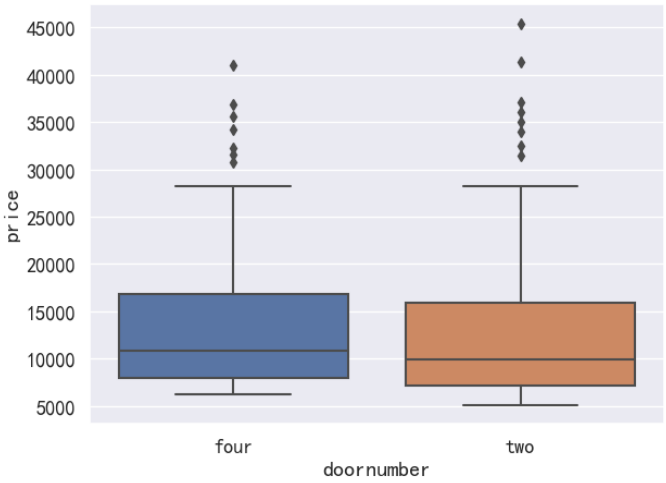
- 增压机类型以标准类型（std）为主。涡轮增压（turbo）的数量较少。
- 两种类型价格差异显著。
  - 标准类型价格低于涡轮增压类型价格。

doornumber

```
category_analysis("doornumber")
```

```
KruskalResult(statistic=2.8612616578123022, pvalue=0.09073630460630733)
```



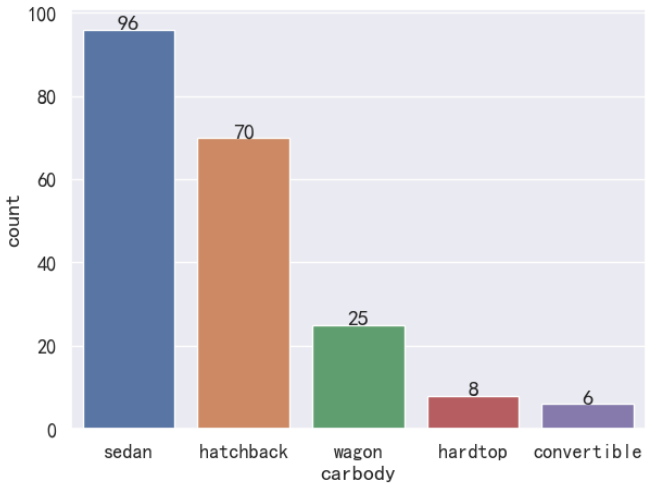


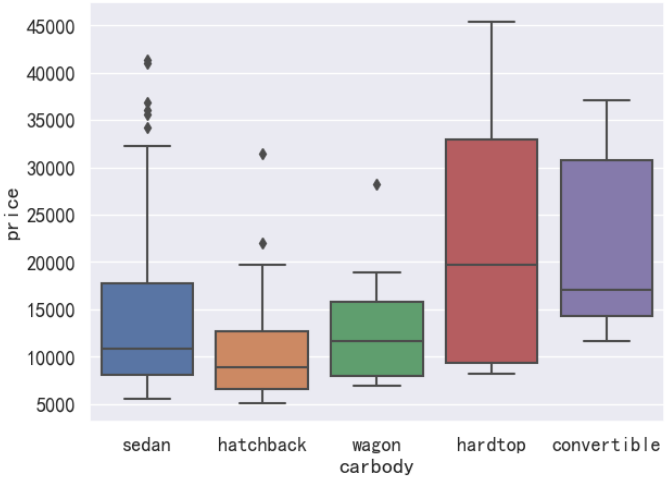
- 双门与四门汽车类型接近。
- 两个类型对价格影响不显著。

carbody

```
category_analysis("carbody")
```

```
KruskalResult(statistic=22.183437941390444, pvalue=0.0001842534808688567)
```



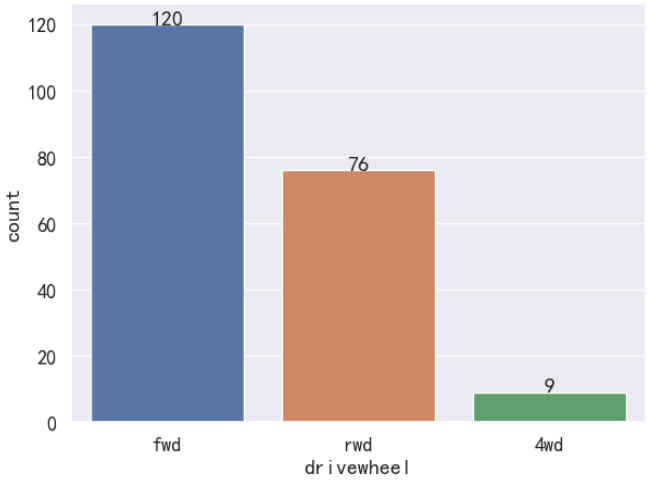


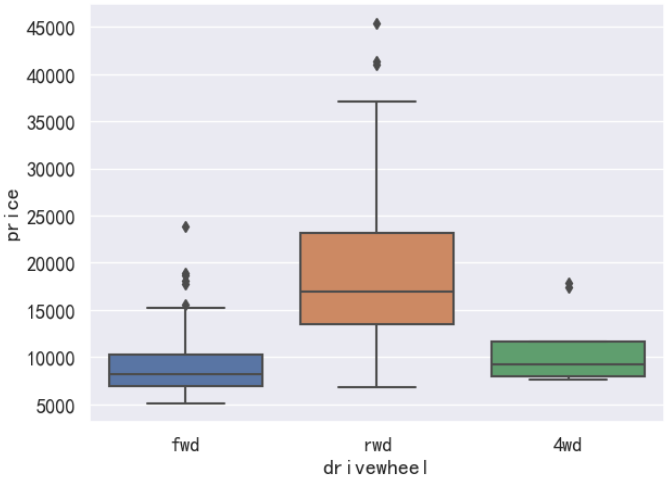
- 汽车类型以轿车（sedan）与掀背车（hatchback）为主。
- 汽车类型对价格影响显著。
- 其中轿车，硬顶轿车（hardtop）的价格浮动区间较大。掀背车与货车（wagon carbody）价格浮动区间较小。

drivewheel

```
category_analysis("drivewheel")
```

```
KruskalResult(statistic=95.02387066880598, pvalue=2.3218205980565562e-21)
```



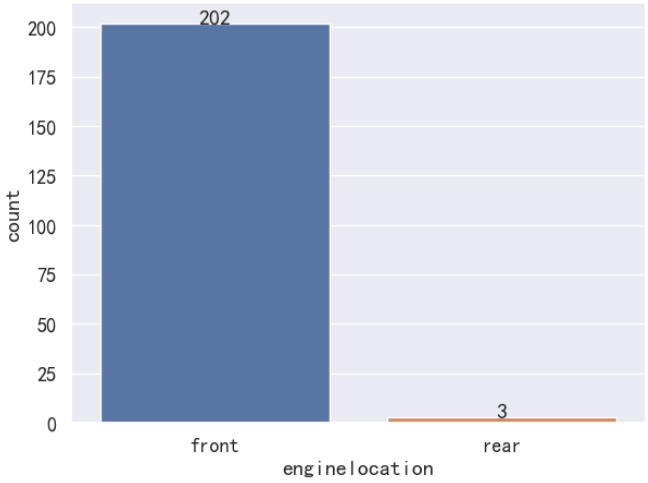


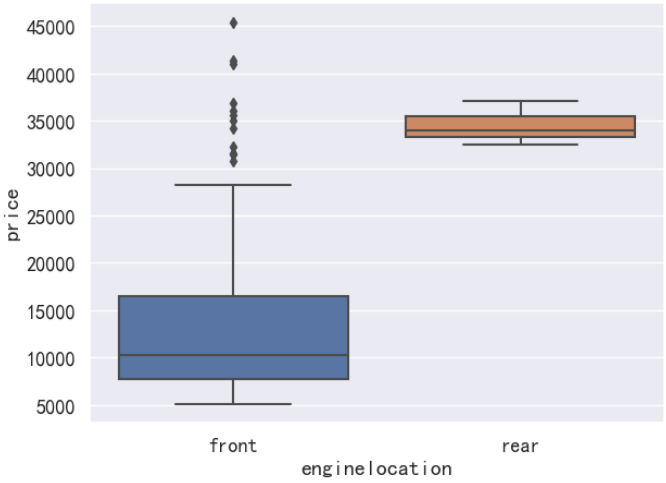
- 4轮驱动车型（4wd）很少，前轮驱动（fwd）最多，后轮驱动（rwd）其次。
- 不同类型对价格影响显著。
- 后轮驱动的价格最高。

enginelocation

```
category_analysis("enginelocation")
```

```
KruskalResult(statistic=7.753234528191141, pvalue=0.005361642811118273)
```



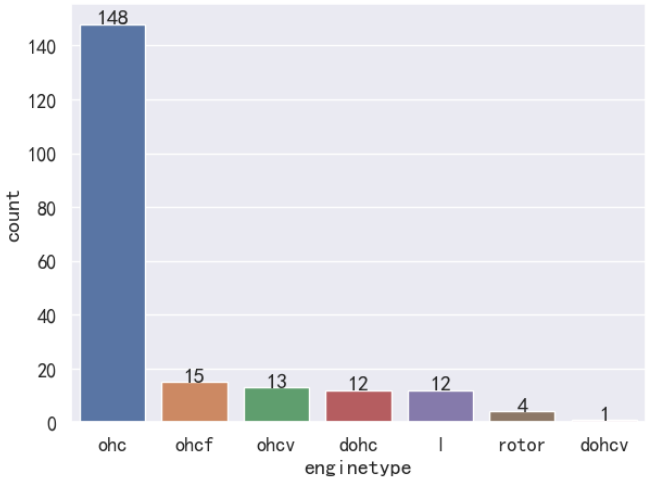


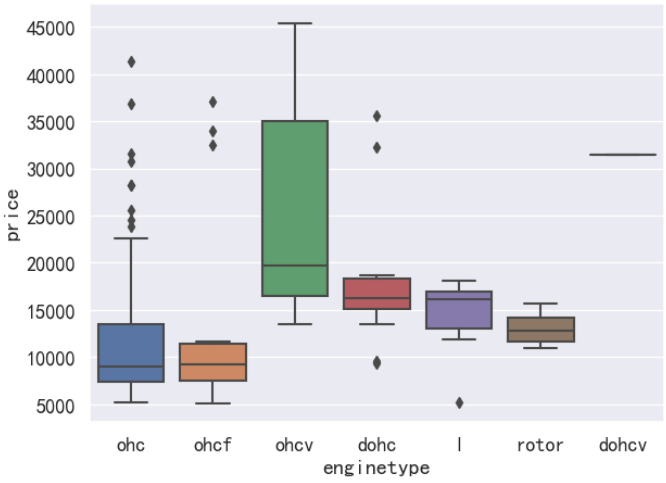
- 发动机位置几乎都是前置。
- 虽然发动机位置对价格影响显著，但是后置发动机样本数量过少，可能不足以说明问题。
- 前置与后置发动机的价格区间差距很大。

enginetype

```
category_analysis("enginetype")
```

```
KruskalResult(statistic=40.133921438590676, pvalue=4.287270740376635e-07)
```



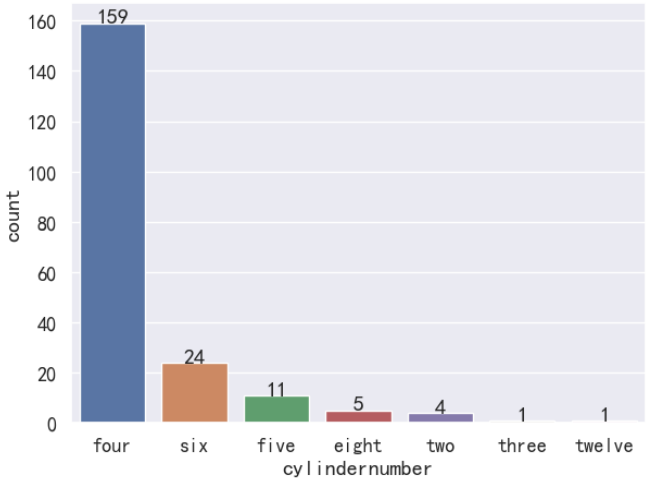


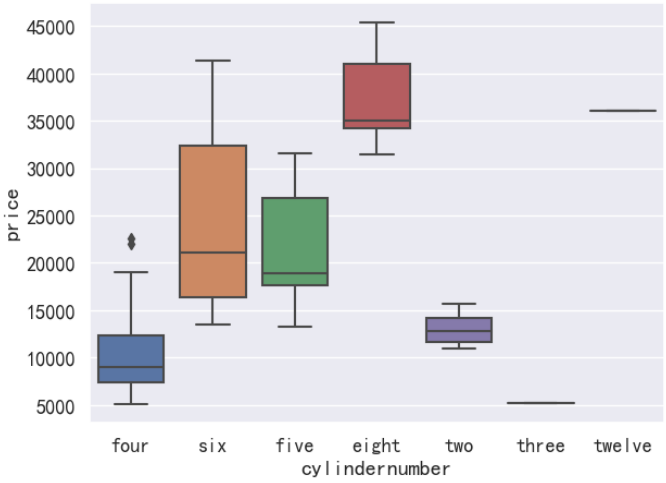
- 发动机几乎都是ohc类型，其余6个类型很少。
- 不同类型发动机，价格差异显著。

cylindernumber

```
category_analysis("cylindernumber")
```

```
KruskalResult(statistic=84.76297575868743, pvalue=3.696266502144558e-16)
```



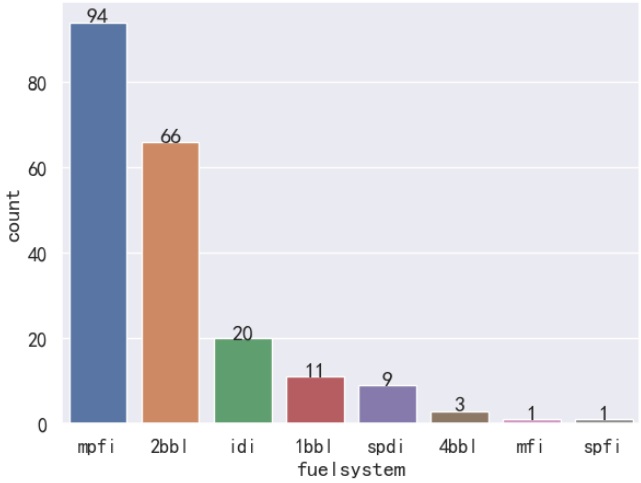


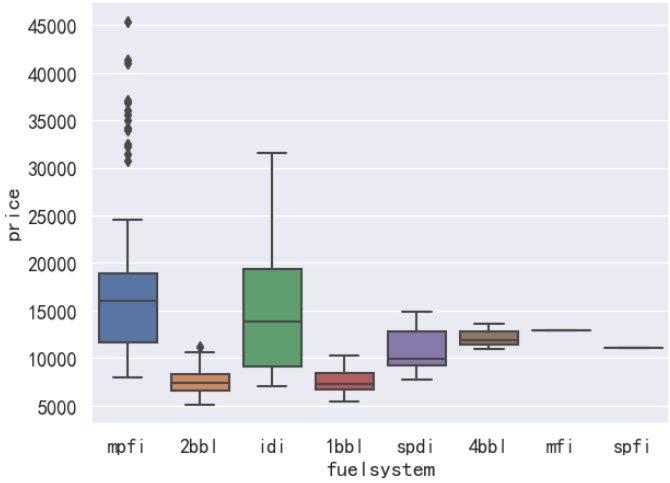
- 气缸基本都是4气缸，其余气缸车型很少。
- 三气缸车型价格较低，十二气缸与八气缸类型价格较高。
- 气缸数量对汽车价格影响显著。

fuelsystem

```
category_analysis("fuelsystem")
```

```
KruskalResult(statistic=125.0821070976894, pvalue=6.683871002354369e-24)
```





- 燃料系统以mpfi与2bbl为主。
- 不同燃料系统，差异显著。

特征工程

特征工程是数据挖掘 / 机器学习中非常重要的一个环节。所谓特征工程，就是从原始数据中提取有用的信息，以便于更好的建立模型，提高模型的质量。

删除无用特征

如果变量与目标（都是连续变量）之间的相关系数很低，则通常情况下，这些变量对预测因变量不会起到太大作用，我们可以考虑将相关系数很低的变量删除。

- 很明显，car_ID仅是一个汽车编号，与汽车的价格毫无关联，我们可以将其删除。
- 对于其他相关性很低的变量，我们暂时保留，待定。

```
data.drop("car_ID", inplace=True, axis=1)
```

建立模型

基模型（baseline）

在进行特征工程之前，我们可以事先建立一个相对简单的基模型，作为我们的基准，并且在后面的操作中，进行改进。这里，我们仅使用enginesize一个特征来建立基模型。

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

SEED = 0
# 将Series转换为DataFrame。
X = data["enginesize"].to_frame()
y = data["price"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=SEED)
lr = LinearRegression()
lr.fit(X_train, y_train)
print(lr.score(X_train, y_train))
print(lr.score(X_test, y_test))
```

```
0.7557897278380871
0.7797940080893313
```

使用所有数值变量

然后，我们使用所有的数值变量，来进行拟合。

```
# 选择所有数值（连续）类型的变量。
X = data.select_dtypes(include=np.number)
y = X.pop("price")
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=SEED)
lr = LinearRegression()
lr.fit(X_train, y_train)
print(lr.score(X_train, y_train))
print(lr.score(X_test, y_test))
```

```
0.858252048258165
0.8214693378253439
```

类别变量处理

在使用所有数值类型变量后，我们也应该要考虑类别类型的变量。只不过，类别类型变量不能直接放到模型中使用，我们需要对其进行处理。这里，我们对每个类别变量进行分组，然后使用该组价格的均值（或中位数）来作为该变量的值。

```
# symboling当成类别变量看待。
for name in category_var + ["symboling"]:
    # 对每一个类别变量，根据其取值进行分组，使用每组价格的均值，
    # 来作为新的特征值。
    v = data.groupby(name)["price"].mean()
    data[name] = data[name].map(v)

data.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase
0	17221.296296	15498.333333	12999.7982	12611.270833	12989.924078	21890.500000	19910.809211	12961.097361	88.6
1	17221.296296	15498.333333	12999.7982	12611.270833	12989.924078	21890.500000	19910.809211	12961.097361	88.6
2	10037.907407	15498.333333	12999.7982	12611.270833	12989.924078	10376.652386	19910.809211	12961.097361	94.5
3	10109.281250	17859.166714	12999.7982	12611.270833	13501.152174	14344.270833	9239.308333	12961.097361	99.8
4	10109.281250	17859.166714	12999.7982	12611.270833	13501.152174	14344.270833	11087.463000	12961.097361	99.4

```
y = data["price"]
X = data.iloc[:, :-1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=SEED)
lr = LinearRegression()
lr.fit(X_train, y_train)
print(lr.score(X_train, y_train))
print(lr.score(X_test, y_test))
```

```
0.9497131369526582
0.886804982188443
```

不显著的变量

在之前类别变量分析中，我们通过假设检验（Kruskal-Wallis H-test）发现如下的信息：

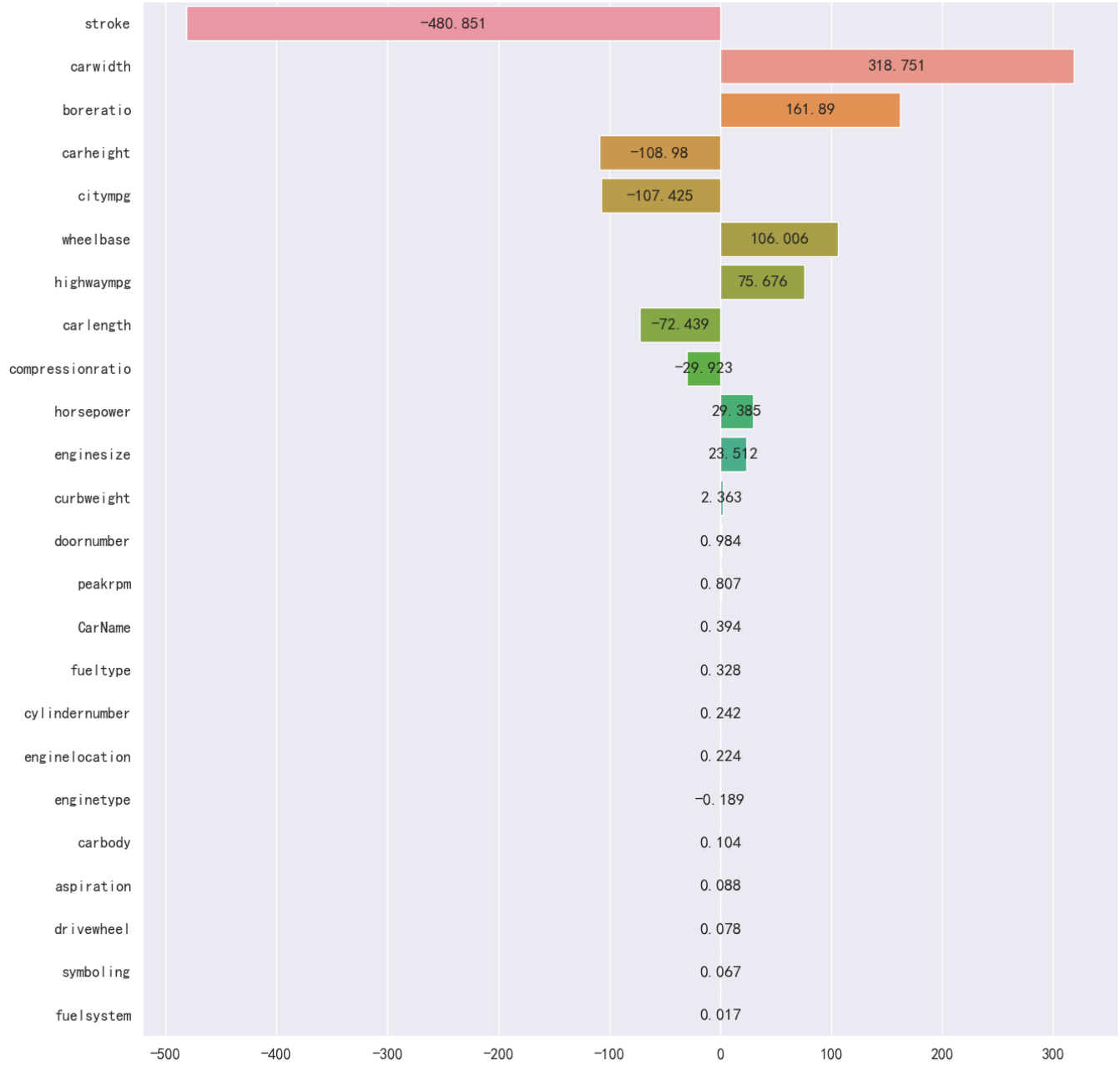
- doornumber与price之间不显著。
- fueltype与price之间显著性不太高，处于临界状态。
- 其他变量与price之间显著。

这里，我们可以删除doornumber变量，而对于fueltype，暂时保留。不过，为了能够在后面证明这点，我们将两个变量全部保留。

权重可视化

我们可以绘制模型的权重。

```
def plot_weight(model, names, figsize=None):
    """绘制模型的权重。

    会根据权重的绝对值进行降序排列。

    Parameters
    ----------
    model : object
        模型对象。
    names : array-like
        权重对应的名称。
    figsize : tuple
        图像尺寸。

    """
    s = pd.Series(index=names, data=model.coef_)
    s.sort_values(ascending=False, inplace=True, key=lambda a: a.abs())
    ax = sns.barplot(y=s.index, x=s.values)
    if not figsize:
        figsize = (15, len(model.coef_) // 1.5)
    ax.get_figure().set_size_inches(figsize)
    for y, x in enumerate(s):
        ax.text(x / 2, y, round(x, 3), va="center", ha="center")


plot_weight(lr, X_train.columns)
```



课堂练习

以上权重值能否客观反映对价格的解释（影响）力度?
A 能。
B 不能。
C 暂时还不确定。

数据标准化

很多算法对特征的数量级都是敏感的，因此，在使用算法之前，我们最好将数据集中的特征转换成相同的量纲，从而消除不同量纲对算法造成的负面影响，我们将这个过程称为**数据标准化**。实际上，即使特征量纲相同，标准化也不会产生负面影响。

在scikit-learn中，常用的标准化方式为：

- 均值标准差标准化（StandardScaler）。
- 最小最大值标准化（MinMaxScaler）。

标准化与否，对线性回归会造成怎样的影响？
A 影响w与R^2。
B 影响w，不影响R^2。
C 不影响w，影响R^2。
D 不会影响w与R^2。

执行标准化

```
from sklearn.preprocessing import MinMaxScaler

x_scaler = MinMaxScaler()
y_scaler = MinMaxScaler()

X_train_scale = x_scaler.fit_transform(X_train)
y_train_scale = y_scaler.fit_transform(y_train.values.reshape(-1, 1)).ravel()
X_test_scale = x_scaler.transform(X_test)
y_test_scale = y_scaler.transform(y_test.values.reshape(-1, 1)).ravel()

# 重新将ndarray数组对象包装成DataFrame对象，方便查看。
X_train_scale = pd.DataFrame(X_train_scale, columns=X_train.columns)
X_test_scale = pd.DataFrame(X_test_scale, columns=X_test.columns)
display(X_train_scale.head(3))
display(X_test_scale.head(3))
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	carlength	carwi
0	0.593609	0.076162	0.0	0.0	1.0	0.335334	0.0	0.0	0.288630	0.485039	0.070
1	0.593609	0.162483	0.0	0.0	1.0	0.335334	0.0	0.0	0.355685	0.522835	0.474
2	0.985001	0.162483	0.0	0.0	0.0	0.000000	1.0	0.0	0.253644	0.384252	0.393

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	carlength	carwi
0	0.000000	0.162483	0.0	0.0	0.0	0.000000	0.0	0.0	0.189504	0.228346	0.242
1	1.000000	0.135656	0.0	0.0	1.0	0.168639	1.0	0.0	0.521866	0.680315	0.474
2	0.009787	0.414513	0.0	0.0	0.0	0.335334	0.0	0.0	0.384840	0.514961	0.454

拟合优度对比

对线性回归来讲，标准化不会影响到模型的拟合优度。

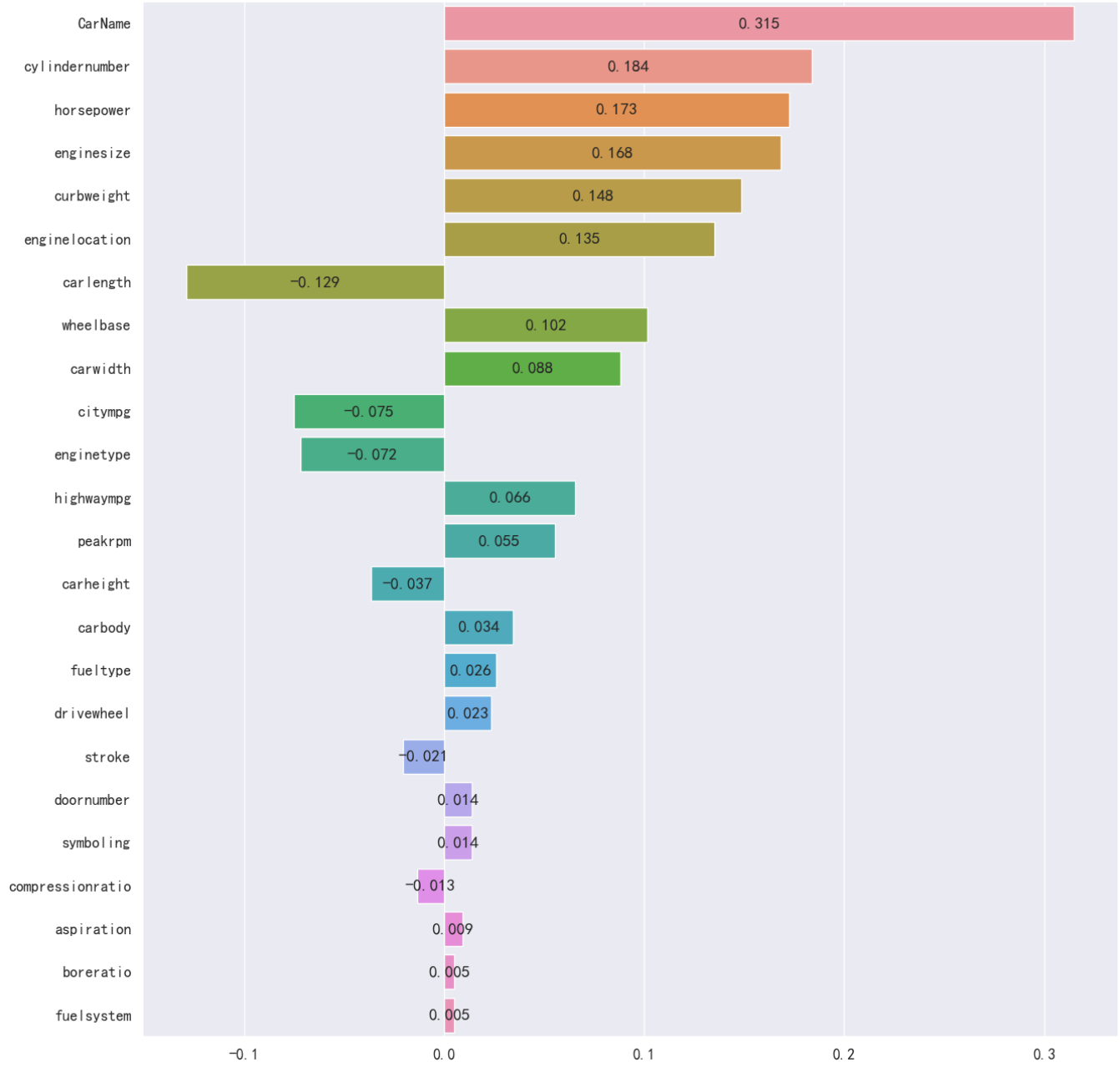
```
lr.fit(X_train_scale, y_train_scale)
print(lr.score(X_train_scale, y_train_scale))
print(lr.score(X_test_scale, y_test_scale))
```

```
0.9497131369526582
0.8868049821884444
```

权重系数对比

不过，标准化会对权重系数造成影响，使得权重更加客观。

```
plot_weight(lr, X_train.columns)
```



多重共线性分析

权重的疑惑

从刚才的运行结果中，我们发现了一个非常严重的问题，citympg与highwaympg对目标price的影响起到了完全相反的结果！这是怎么回事呢?

概念

在多元线性回归中，自变量与因变量应该要存在线性关系。但是，如果自变量之间存在线性关系，我们称这种现象为**多重共线性**。多重共线性会造成权重的不确定性，甚至可能会出现拟合后的权重与相关系数呈现相反的结果。

不过，多重共线性并不会影响拟合优度（模型的效果），如果我们仅追求模型的拟合优度，而不考虑权重系数的大小，则可以不处理多重共线性。

方差膨胀系数

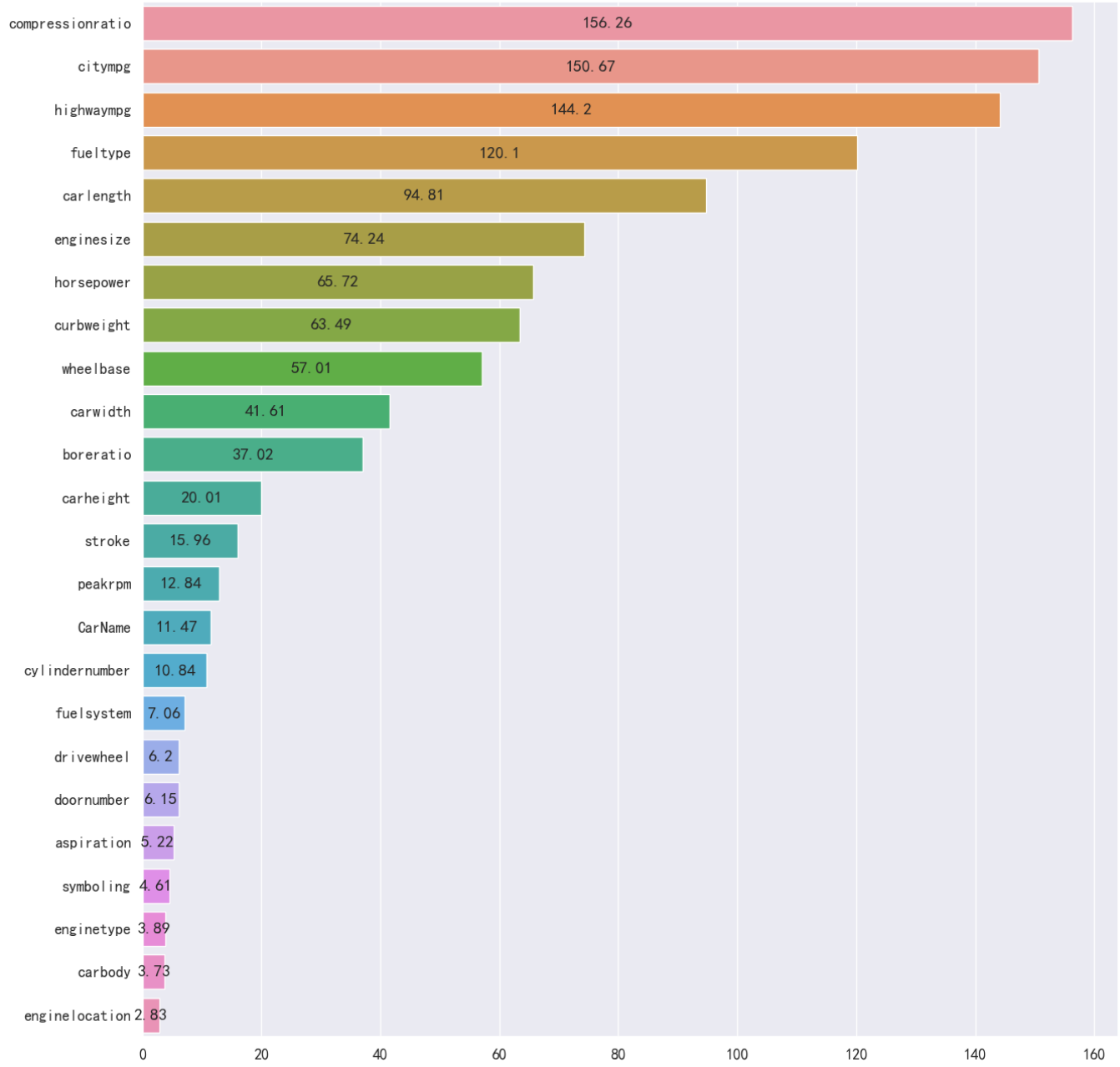
我们常使用方差膨胀系数（VIF——Variance Inflation Factor）来检测多元线性回归中的多重共线性。方差膨胀系数的计算公式如下：

$$VIF_i = \frac{1}{1-R_i^2}$$

- VIF_i：第i个变量的方差膨胀系数。
- R_i^2：第i个变量作为因变量，其他变量作为自变量进行回归时的R^2。

由公式可知，如果变量之间存在多重共线性，则得到的VIF值就会较高。当VIF值大于10时，说明存在严重的多重共线性。

```
from statsmodels.stats.outliers_influence import variance_inflation_factor

def plot_VIF(X, figsize=None):
    """绘制特征变量的方差膨胀系数（VIF）。

    将每个特征变量以条形图的方式绘制，并进行降序排列。

    Parameters
    ----------
    X : DataFrame，形状为（样本数量， 特征数量）。
        需要计算VIF的所有变量数据。
    figsize : tuple，形状为（图像宽度，图像高度），可选。
        设置图像的宽度与高度。

    """

    # 对每一个特征变量，求解VIF值。
    vif = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
    # 组成Series对象，方便对值排序时，可以与对应的列名相对应。
    s = pd.Series(vif, index=X.columns)
    s.sort_values(ascending=False, inplace=True)
    s = s.round(2)
    ax = sns.barplot(y=s.index, x=s.values)
    ax.get_figure().set_size_inches(15, X.shape[1] // 1.5)
    for y, x in enumerate(s):
        ax.text(x / 2, y, x, va="center", ha="center")


plot_VIF(X_train_scale)
```



课堂练习

我们发现了一些VIF较大的特征，应该如何处理?

A 将所有大于阈值的特征删除。

B 将VIF最大的特征删除。

C 将VIF最小的特征删除。

D 不删除任何特征。

```
def feature_eliminate_by_VIF(X, max_vif=10, verbose=1):
    """根据特征变量的方差膨胀系数（VIF），删除特征。

    该方法每次会删除VIF最大的特征，直到所有的特征都不超过
    指定的阈值为止。

    Parameters
    ----------
    X : DataFrame，形状为（样本数量， 特征数量）。
        需要删除特征变量的数据。
    max_vif : float，可选。
        指定特征的最大VIF值，超过该值的特征将会删除。
    verbose : int，可选（1或0）。
```

```
        是否显示详细的删除信息，默认为1。

    Returns
    -------
    eli_col_names : list
        所有因为VIF大于指定阈值而被删除的特征名。

    """

    # 对参数复制，避免修改影响到参数。
    X = X.copy()
    # 用来保存所有因为VIF过大而被删除的特征名。
    eli_col_names = []
    while X.shape[1] > 0:
        vif = pd.DataFrame()
        vif["features"] = X.columns
        vif["vif"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
        vif = vif.sort_values("vif", ascending=False)
        # 获取vif最大的行。
        s = vif.iloc[0]
        if s["vif"] > max_vif:
            X.drop(s["features"], axis=1, inplace=True)
            eli_col_names.append(s["features"])
            if verbose:
                print(f'{s["features"]}的vif值过大（{s["vif"]:.2f}），删除！')
        else:
            # 如果已经没有VIF过大的特征，则显示最终的结果，退出循环。
            display(vif)
            break
    return eli_col_names


drop_columns = feature_eliminate_by_VIF(X_train_scale, max_vif=10)
print(drop_columns)
```

```
compressionratio的vif值过大（156.26），删除!
highwaympg的vif值过大（143.99），删除!
carlength的vif值过大（89.89），删除!
enginesize的vif值过大（67.84），删除!
curbweight的vif值过大（53.34），删除!
wheelbase的vif值过大（43.58），删除!
horsepower的vif值过大（35.28），删除!
carwidth的vif值过大（24.59），删除!
boreratio的vif值过大（15.21），删除!
carheight的vif值过大（10.67），删除!
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	features	vif
1	CarName	7.579864
13	citympg	6.661770
12	peakrpm	6.626759
11	stroke	6.533504
10	fuelsystem	5.156966
9	cylindernumber	5.105943
6	drivewheel	4.017209
0	symboling	3.434676
4	doornumber	3.270645
5	carbody	3.032210
2	fueltype	2.889652
8	enginetype	2.808392
3	aspiration	2.168334
7	enginelocation	1.744288

```
['compressionratio', 'highwaympg', 'carlength', 'enginesize', 'curbweight', 'wheelbase', 'horsepower', 'carwidth', 'boreratio',
'carheight']
```

删除共线性特征

结果与之前的表现也是一致的，在数据集中，citympg与highwaympg的相关性很高，而curbweight与carlength，carwidth，enginesize等特征的相关性很高。这里，我们将删除多重共线性的特征。

```
X_train_scale.drop(drop_columns, axis=1, inplace=True)
X_test_scale.drop(drop_columns, axis=1, inplace=True)
lr.fit(X_train_scale, y_train_scale)
print(lr.score(X_train_scale, y_train_scale))
print(lr.score(X_test_scale, y_test_scale))
```

```
0.9339202978016744
0.852217003937153
```

```
plot_weight(lr, X_train_scale.columns)
```

特征选择

在建立模型时，特征并非越多越好，有些特征可能对模型质量并没有什么改善，我们可以进行删除。

RFECV

特征选择的方式有很多，这里，我们使用RFECV方法来实现特征选择。RFECV分成两个部分，分别是：

- RFE（Recursive feature elimination）：递归特征消除，用来对特征进行重要性评级。
- CV（Cross Validation）：交叉验证，在特征评级后，通过交叉验证，选择最佳数量的特征。

具体过程如下：

- RFE阶段。
    1. 初始的特征集为所有可用的特征。
    2. 使用当前特征集进行建模，然后计算每个特征的重要性。
    3. 删除最不重要的一个（或多个）特征，更新特征集。
    4. 跳转到步骤2，直到完成所有特征的重要性评级。
- CV阶段。
    1. 根据RFE阶段确定的特征重要性，依次选择不同数量的特征。
    2. 对选定的特征集进行交叉验证。
    3. 确定平均分最高的特征数量，完成特征选择。

```
from sklearn.feature_selection import RFECV

# estimator： 要操作的模型。
```

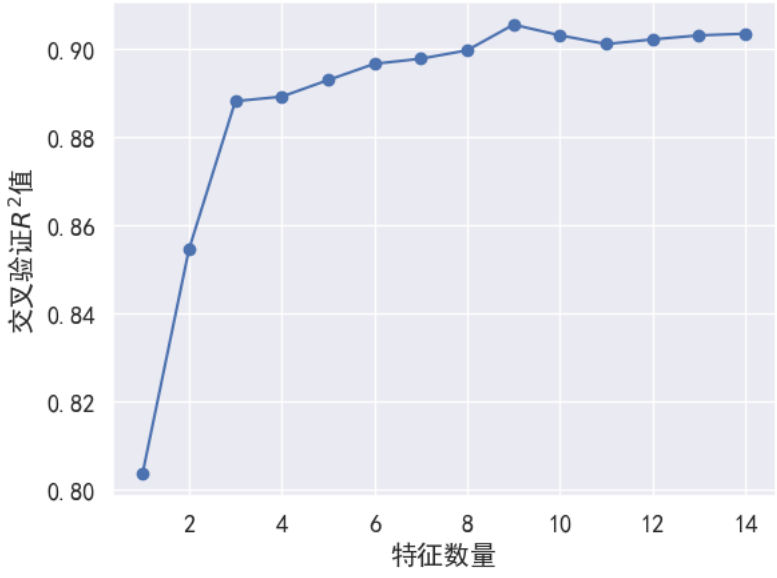
```
# step:  每次删除的变量数。
# cv:  使用的交叉验证折数。
# n_jobs:  并发的数量。
# scoring: 评估的方式。
rfecv = RFECV(estimator=lr, step=1, cv=5, n_jobs=-1, scoring="r2")
rfecv.fit(X_train_scale, y_train_scale)
# 返回经过选择之后，剩余的特征数量。
print(rfecv.n_features_)
# 返回经过特征选择后，使用缩减特征训练后的模型。
print(rfecv.estimator_)
# 返回每个特征的等级，数值越小，特征越重要。
print(rfecv.ranking_)
# 返回布尔数组，用来表示特征是否被选择。
print(rfecv.support_)
# 返回对应数量特征时，模型交叉验证的评分。
print(rfecv.cv_results_["mean_test_score"])
```

```
9
LinearRegression()
[1 1 3 1 4 1 1 1 5 1 2 1 6 1]
[ True  True False  True False  True  True  True False  True False  True
 False  True]
[0.80375872 0.85457418 0.88819254 0.88920672 0.89296809 0.89665574
 0.89781439 0.89971934 0.90554417 0.90312207 0.90114276 0.90220023
 0.90311225 0.90348766]
```

我们也可以绘制下，在特征选择过程中，使用交叉验证获取的R^2值。

```
score = rfecv.cv_results_["mean_test_score"]
plt.plot(range(1, len(score) + 1), score, marker="o")
plt.xlabel("特征数量")
plt.ylabel("交叉验证$R^2$值")
```

```
Text(0, 0.5, '交叉验证$R^2$值')
```



特征选择后，我们可以对现有的数据集进行转换。

```
print("剔除的变量：", X_train_scale.columns[~rfecv.support_])
X_train_eli = rfecv.transform(X_train_scale)
X_test_eli = rfecv.transform(X_test_scale)
print(X_train_scale.shape, X_train_eli.shape)
# 说明：经过transform转换之后，返回ndarray数组类型。
# 这里X_train_scale是DataFrame类型，如果依然希望返回DataFrame类型，
# 需要我们自己去调整。
col = X_train_scale.columns[rfecv.support_]
# X_train_eli = pd.DataFrame(X_train_eli, columns=col)
# X_test_eli = pd.DataFrame(X_test_eli, columns=col)
print(rfecv.estimator_.score(X_train_eli, y_train_scale))
print(rfecv.estimator_.score(X_test_eli, y_test_scale))
```

```
剔除的变量： Index(['fueltype', 'doornumber', 'enginetype', 'fuelsystem', 'peakrpm'], dtype='object')
(143, 14) (143, 9)
0.9318950398900173
0.8521250777462107
```

删除变量分析

- doornumber，fueltype为不显著变量。
- peakrpm为相关性很低的变量。