项目背景：

此项目为 Analiytics Vidhya 的竞赛题目，大型商场销售额预测，该项目提供了从不同城市的 10 家商店中收集的 1559 种产品的 2013 年销售数据。目的是通过 2013 年的销售数据，建立一个销售额预测模型，预测每个产品在特定商店的销售情况。

零售业是一个充分利用数据分析优化商业流程的行业。我们可以利用数据科学对商品的放置、库存管理、定制供应、商品捆绑等任务进行巧妙的处理。该数据集包含了商店的交易数据，是一个回归问题，共包含 8523 行 12 列个数据。

变量	描述	假设关联
Item_Identifier	唯一产品ID	无假设
Item_Weight	产品重量	无假设
Item_Fat_Content	产品是否低脂	与产品健康假设关联
Item_Visibility	该产品占商店总产品展示区的百分比	与商品展示假设关联
Item_Type	产品种类	与商品品类假设关联
Item_MRP	产品最高零售价（标价）	无假设
Outlet_Identifier	商店唯一ID	无假设
Outlet_Establishment_Year	商店成立的年份	与假设客户流量相关，成立越久在给区域可能知名度越高
Outlet_Size	商店的面积	与假设商店的大小相关
Outlet_Location_Type	商店所在城市类型	与假设城市类型相关
Outlet_Type	商店的类型	与假设商店大小相关，不同类型可能意味着商店的形态大小不一
Item_Outlet_Sales	预测变量，该产品在对应商店的销售额	

步骤：

1、问题：预测销量。

2、数据获取

1）导包

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib
matplotlib.rc("font", family='Microsoft YaHei')
plt.rcParams["font.sans-serif"]=['SimHei']
plt.rcParams["axes.unicode_minus"]=False
plt.rcParams.update({'font.size':13 }) #全局字体大小
plt.rcParams["font.family"]=['SimHei']
plt.rcParams['axes.unicode_minus']=False
```

2）数据读入

```
train = pd.read_csv('train.csv')
train.head()
```

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type	Outlet_Type	Item_Outlet_Sales
0	FDA15	9.30	Low Fat	0.016047	Dairy	249.8092	OUT049	1999	Medium	Tier 1	Supermarket Type1	3735.1380
1	DRC01	5.92	Regular	0.019278	Soft Drinks	48.2692	OUT018	2009	Medium	Tier 3	Supermarket Type2	443.4228
2	FDN15	17.50	Low Fat	0.016760	Meat	141.6180	OUT049	1999	Medium	Tier 1	Supermarket Type1	2097.2700
3	FDX07	19.20	Regular	0.000000	Fruits and Vegetables	182.0950	OUT010	1998	NaN	Tier 3	Grocery Store	732.3800
4	NCD19	8.93	Low Fat	0.000000	Household	53.8614	OUT013	1987	High	Tier 3	Supermarket Type1	994.7052

```
test = pd.read_csv('test.csv')
test.head()
```

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type	Outlet_Type
0	FDW58	20.750	Low Fat	0.007565	Snack Foods	107.8622	OUT049	1999	Medium	Tier 1	Supermarket Type1
1	FDW14	8.300	reg	0.038428	Dairy	87.3198	OUT017	2007	NaN	Tier 2	Supermarket Type1
2	NCN55	14.600	Low Fat	0.099575	Others	241.7538	OUT010	1998	NaN	Tier 3	Grocery Store
3	FDQ58	7.315	Low Fat	0.015388	Snack Foods	155.0340	OUT017	2007	NaN	Tier 2	Supermarket Type1
4	FDY38	NaN	Regular	0.118599	Dairy	234.2300	OUT027	1985	Medium	Tier 3	Supermarket Type3

3）合并数据集

```python
data = pd.concat([train,test],ignore_index=True)
data
```

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type	Outlet_Type	Item_Outlet_Sales
0	FDA15	9.30	Low Fat	0.016047	Dairy	249.8092	OUT049	1999	Medium	Tier 1	Supermarket Type1	3735.1380
1	DRC01	5.92	Regular	0.019278	Soft Drinks	48.2692	OUT018	2009	Medium	Tier 3	Supermarket Type2	443.4228
2	FDN15	17.50	Low Fat	0.016760	Meat	141.6180	OUT049	1999	Medium	Tier 1	Supermarket Type1	2097.2700
3	FDX07	19.20	Regular	0.000000	Fruits and Vegetables	182.0950	OUT010	1998	NaN	Tier 3	Grocery Store	732.3800
4	NCD19	8.93	Low Fat	0.000000	Household	53.8614	OUT013	1987	High	Tier 3	Supermarket Type1	994.7052

3、数据预处理处理

1）数据集信息

```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14204 entries, 0 to 14203
Data columns (total 12 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   Item_Identifier            14204 non-null  object 
 1   Item_Weight                11765 non-null  float64
 2   Item_Fat_Content           14204 non-null  object 
 3   Item_Visibility            14204 non-null  float64
 4   Item_Type                  14204 non-null  object 
 5   Item_MRP                   14204 non-null  float64
 6   Outlet_Identifier          14204 non-null  object 
 7   Outlet_Establishment_Year  14204 non-null  int64  
 8   Outlet_Size                10188 non-null  object 
 9   Outlet_Location_Type       14204 non-null  object 
 10  Outlet_Type                14204 non-null  object 
 11  Item_Outlet_Sales          8523 non-null   float64
dtypes: float64(4), int64(1), object(7)
memory usage: 1.3+ MB
```

2）缺失值检查

```python
data.isnull().sum()
```

```
Item_Identifier                 0
Item_Weight                  2439
Item_Fat_Content                0
Item_Visibility                 0
Item_Type                       0
Item_MRP                        0
Outlet_Identifier               0
Outlet_Establishment_Year       0
Outlet_Size                  4016
Outlet_Location_Type            0
Outlet_Type                     0
Item_Outlet_Sales            5681
dtype: int64
```

由上可知 ‘Item_Weight’、‘Outlet_Size’、‘Item_Outlet_Sales’ 存在缺失值

3）数据描述（数值型数据）

```
data.describe()
```

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Item_Outlet_Sales
count	11765.000000	14204.000000	14204.000000	14204.000000	8523.000000
mean	12.792854	0.065953	141.004977	1997.830681	2181.288914
std	4.652502	0.051459	62.086938	8.371664	1706.499616
min	4.555000	0.000000	31.290000	1985.000000	33.290000
25%	8.710000	0.027036	94.012000	1987.000000	834.247400
50%	12.600000	0.054021	142.247000	1999.000000	1794.331000
75%	16.750000	0.094037	185.855600	2004.000000	3101.296400
max	21.350000	0.328391	266.888400	2009.000000	13086.964800

4）变量'Item_Weight'的缺失值处理

```
# 获取变量'Item_Weight'为空的'Item_Identifier'列表
lt = np.unique(data[data.Item_Weight.isnull()].Item_Identifier)
lt = lt.tolist()
```

```
# 通过循环遍历上述 lt 列表，将对应'Item_Identifier'的缺失值'Item_Weight'用均值填充
for k in lt:
    mean_temp = round(data[data.Item_Identifier==k].Item_Weight.mean(),3)
    #print(mean_temp)
    data.loc[(data.Item_Identifier==k)&(data.Item_Weight.isnull()),'Item_Weight'] = mean_temp
```

```
# 检查'Item_Weight'的缺失值是否处理完毕
data.isnull().sum()
```

```
Item_Identifier                 0
Item_Weight                     0
Item_Fat_Content                0
Item_Visibility                 0
Item_Type                       0
Item_MRP                        0
Outlet_Identifier               0
Outlet_Establishment_Year       0
Outlet_Size                  4016
Outlet_Location_Type            0
Outlet_Type                     0
Item_Outlet_Sales            5681
Since_Establishment_Year        0
dtype: int64
```

5）变量'Outlet_Size'的缺失值处理

```
# 查看变量'Outlet_Size'缺失的数据情况
np.unique(data.loc[(data.Outlet_Size.isnull()),['Outlet_Identifier']])
```

```
array(['OUT010', 'OUT017', 'OUT045'], dtype=object)
```

```
#查看每个商店的'Outlet_Size'和'Outlet_Type'的情况
data.groupby(['Outlet_Identifier','Outlet_Size','Outlet_Type'])[['Outlet_Identifier']].count()
```

			Outlet_Identifier
Outlet_Identifier	Outlet_Size	Outlet_Type	
OUT013	High	Supermarket Type1	1553
OUT018	Medium	Supermarket Type2	1546
OUT019	Small	Grocery Store	880
OUT027	Medium	Supermarket Type3	1559
OUT035	Small	Supermarket Type1	1550
OUT046	Small	Supermarket Type1	1550
OUT049	Medium	Supermarket Type1	1550

分析：

'Grocery Store':Small
'Supermarket Type1':Small:1860,Medium:930,High:932
'Supermarket Type2':Medium
'Supermarket Type3':Medium

```
# 查看每个商店的 'Outlet_Type' 情况
data.groupby(['Outlet_Identifier','Outlet_Type'])[['Outlet_Type']].count()
```

		Outlet_Type
Outlet_Identifier	Outlet_Type	
OUT010	Grocery Store	925
OUT013	Supermarket Type1	1553
OUT017	Supermarket Type1	1543
OUT018	Supermarket Type2	1546
OUT019	Grocery Store	880
OUT027	Supermarket Type3	1559
OUT035	Supermarket Type1	1550
OUT045	Supermarket Type1	1548
OUT046	Supermarket Type1	1550
OUT049	Supermarket Type1	1550

分析：

商店'OUT010'的类型是'Grocery Store',和商店'OUT019'的一致，'Outlet_Size'为Small；
商店'OUT017'和'OUT045'的类型是'Supermarket Type1'，'Outlet_Size'取众数：Small；

```
# 处理
data.loc[(data.Outlet_Identifier=='OUT010')&(data.Outlet_Size.isnull()),'Outlet_Size'] = 'Small'
data.loc[(data.Outlet_Identifier=='OUT017')&(data.Outlet_Size.isnull()),'Outlet_Size'] = 'Small'
data.loc[(data.Outlet_Identifier=='OUT045')&(data.Outlet_Size.isnull()),'Outlet_Size'] = 'Small'
```

```
#再次检查数据集缺失值情况
data.isnull().sum()
```

```
Item_Identifier                 0
Item_Weight                     0
Item_Fat_Content                0
Item_Visibility                 0
Item_Type                       0
Item_MRP                        0
Outlet_Identifier               0
Outlet_Establishment_Year       0
Outlet_Size                     0
Outlet_Location_Type            0
Outlet_Type                     0
Item_Outlet_Sales            5681
dtype: int64
```

6）变量'Item_Visibility'的缺失值处理

此变量存在为0数据；
变量表示产品占总产品展示区的百分比，如若为0，并没有实际意义

```
# Item_Visibility 为 0 的无效数据
data[data.Item_Visibility==0]
```

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier
3	FDX07	19.200	Regular	0.0	Fruits and Vegetables	182.0950	OUT010
4	NCD19	8.930	Low Fat	0.0	Household	53.8614	OUT013
5	FDP36	10.395	Regular	0.0	Baking Goods	51.4008	OUT018
10	FDY07	11.800	Low Fat	0.0	Fruits and Vegetables	45.5402	OUT049
32	FDP33	18.700	Low Fat	0.0	Snack Foods	256.6672	OUT018
...	...	...	...	...	...	...	...
14166	FDQ19	7.350	Regular	0.0	Fruits and Vegetables	244.3512	OUT019

```
# 均值替代
data['Item_Visibility'] = data.loc[:,'Item_Visibility'].replace(0,data.Item_Visibility.mean())
```

```
# 检查替换情况
data[data.Item_Visibility==0]
```

Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year

变量'Item_Outlet_Sales'的缺失值是由于训练集和测试集合并导致目标变量的缺失。

7）变量'Item_Fat_Content'的重复值处理

```
# 'Item_Fat_Content'变量的具体情况
np.unique(data['Item_Fat_Content'])
```

```
array(['LF', 'Low Fat', 'Regular', 'low fat', 'reg'], dtype=object)
```

由上可知，变量'Item_Fat_Content'存在数据重复！

```
# 替换处理
data['Item_Fat_Content'].replace({'LF':'Low Fat','reg':'Regular','low fat':'Low Fat'},inplace=True)
# 查看情况
np.unique(data['Item_Fat_Content'])
```

```
array(['Low Fat', 'Regular'], dtype=object)
```

8）变量'Outlet_Establishment_Year'的转换处理

由于'Outlet_Establishment_Year'是表示该 Outlet 建立的年份，故而我们对此字段进行处理，通过 2013 年这一时间节点，算出该 Outlet 从成立至 2013 年的年份。

```
data['Since_Establishment_Year'] = 2013-data['Outlet_Establishment_Year']
data.iloc[0:5,7:].head()
```

	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type	Outlet_Type	Item_Outlet_Sales	Since_Establishment_Year
0	1999	Medium	Tier 1	Supermarket Type1	3735.1380	14
1	2009	Medium	Tier 3	Supermarket Type2	443.4228	4
2	1999	Medium	Tier 1	Supermarket Type1	2097.2700	14
3	1998	Small	Tier 3	Grocery Store	732.3800	15
4	1987	High	Tier 3	Supermarket Type1	994.7052	26

4、EDA

单变量分析

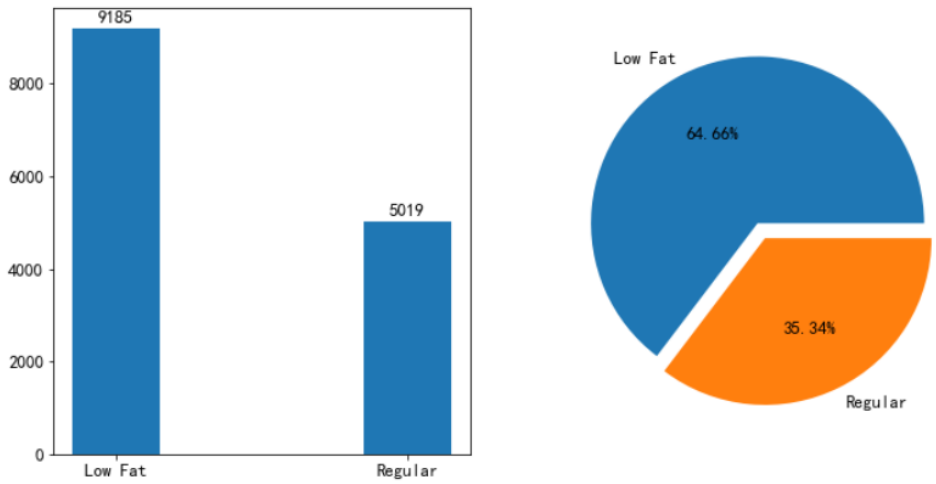
1）类别型变量

‘Item_Fat_Content’:

```
print('变量‘Item_Fat_Content’的分布情况:')
plt.figure(figsize=(12,6))
ax1 = plt.subplot(1,2,1)
rects = ax1.bar(x=data['Item_Fat_Content'].value_counts().index,
       height=data['Item_Fat_Content'].value_counts().values,
       width=0.3)
ax1.bar_label(rects,padding=3,size=13)
ax2 = plt.subplot(1,2,2)
ax2.pie(data['Item_Fat_Content'].value_counts().values,
        labels=data['Item_Fat_Content'].value_counts().index,
        autopct='%.2f%%',explode=[0.05,0.05])

plt.show()
```

变量‘Item_Fat_Content’的分布情况:

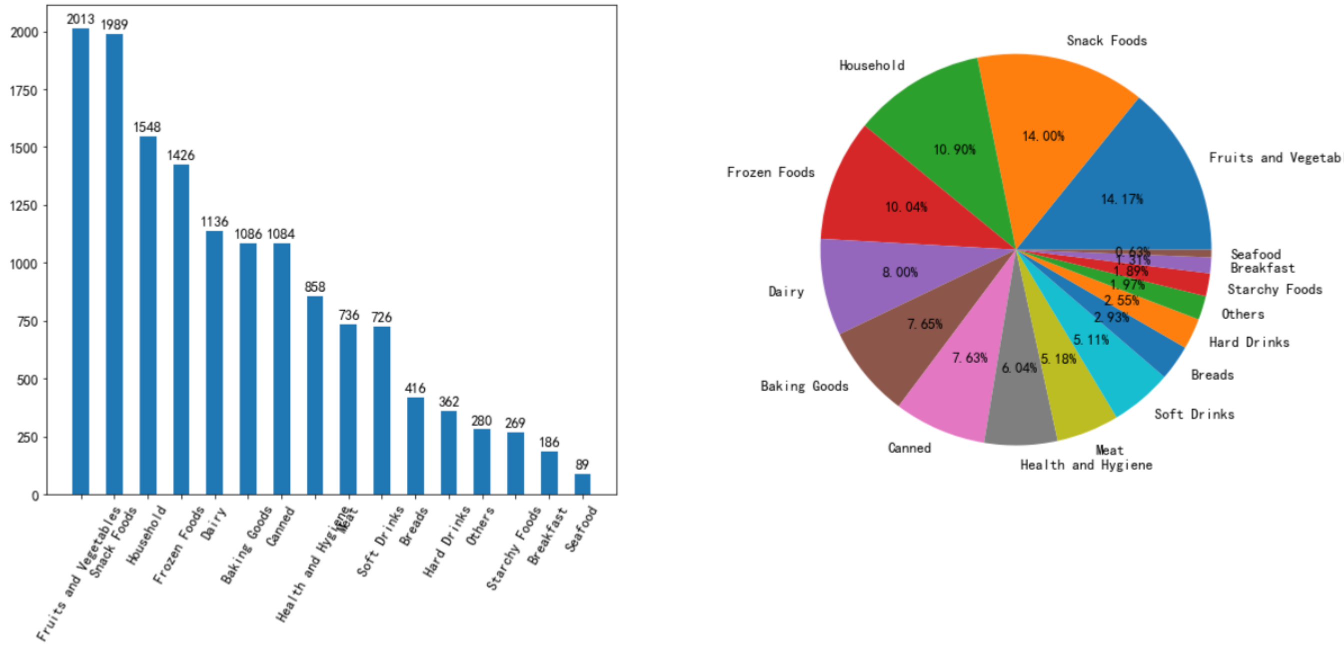


‘Item_Type’:

```
print('变量‘Item_Type’的分布情况:')
plt.figure(figsize=(20,8))
#直方图
ax1 = plt.subplot(1,2,1)
rects = ax1.bar(x=data['Item_Type'].value_counts().index,
               height=data['Item_Type'].value_counts().values,
               width=0.5)
ax1.bar_label(rects,padding=3,size=13)
plt.xticks(rotation=60)
#饼图
ax2 = plt.subplot(1,2,2)
ax2.pie(data['Item_Type'].value_counts().values,
        labels=data['Item_Type'].value_counts().index,
        autopct='%.2f%%')

plt.show()
```

变量‘Item_Type’的分布情况:



分析上述变量发布情况得出‘Item_Type’的类别众多，在后续选择将此变量进入模型时，需要考虑进行分箱处理。

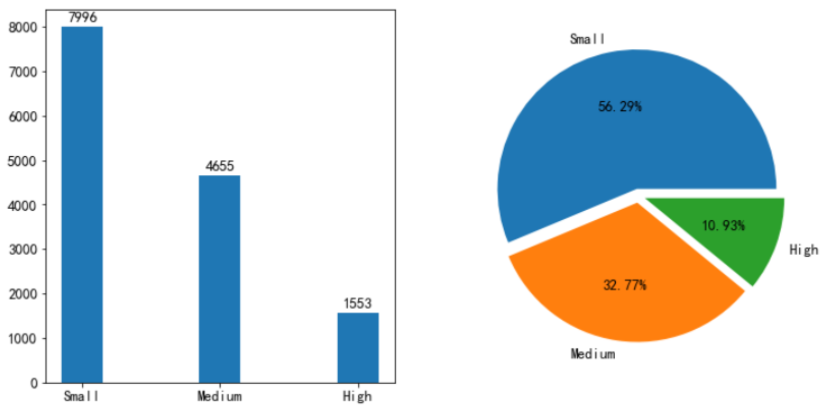
‘Outlet_Size’：

```
print('变量‘Outlet_Size’去除缺失值后的分布情况:')
plt.figure(figsize=(12,6))
#直方图
ax1 = plt.subplot(1,2,1)
rects = ax1.bar(x=data['Outlet_Size'].value_counts().index,height=data['Outlet_Size'].value_counts().values,width=0.3)
ax1.bar_label(rects,padding=3,size=13)

#plt.xticks(rotation=60)
#饼图
ax2 = plt.subplot(1,2,2)
ax2.pie(data['Outlet_Size'].value_counts().values,
        labels=data['Outlet_Size'].value_counts().index,
        autopct='%.2f%%',explode=[0.05,0.05,0.05])

plt.show()
```

变量‘Outlet_Size’去除缺失值后的分布情况:

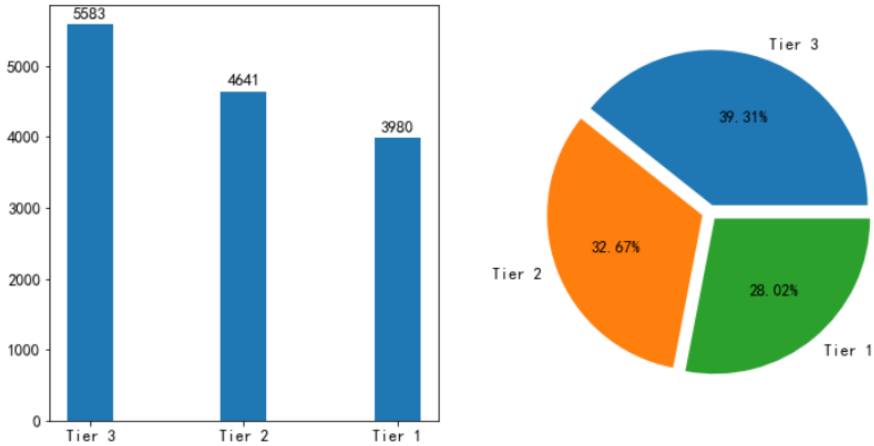


‘Outlet_Location_Type’:

```
print('变量‘Outlet_Location_Type’的分布情况:')
plt.figure(figsize=(12,6))
ax1 = plt.subplot(1,2,1)
rects = ax1.bar(x=data['Outlet_Location_Type'].value_counts().index,
   height=data['Outlet_Location_Type'].value_counts().values,
               width=0.3)
ax1.bar_label(rects,padding=3,size=13)
ax2 = plt.subplot(1,2,2)
ax2.pie(data['Outlet_Location_Type'].value_counts().values,
        labels=data['Outlet_Location_Type'].value_counts().index,
        autopct='%.2f%%',explode=[0.05,0.05,0.05])

plt.show()
```

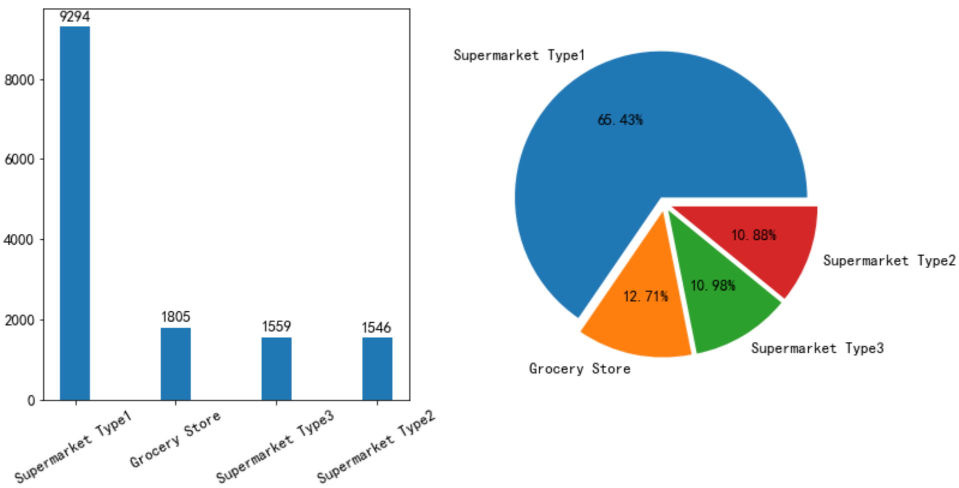
变量‘Outlet_Location_Type’的分布情况:



‘Outlet_Type’:

```
print('变量‘Outlet_Type’的分布情况:')
plt.figure(figsize=(12,6))
ax1 = plt.subplot(1,2,1)
rects = ax1.bar(x=data['Outlet_Type'].value_counts().index,
   height=data['Outlet_Type'].value_counts().values,
               width=0.3)
ax1.bar_label(rects,padding=3,size=13)
plt.xticks(rotation=30)
ax2 = plt.subplot(1,2,2)
ax2.pie(data['Outlet_Type'].value_counts().values,
        labels=data['Outlet_Type'].value_counts().index,
        autopct='%.2f%%',explode=[0.05,0.05,0.05,0.05])
plt.show()
```

变量‘Outlet_Type’的分布情况:



2）数值型变量

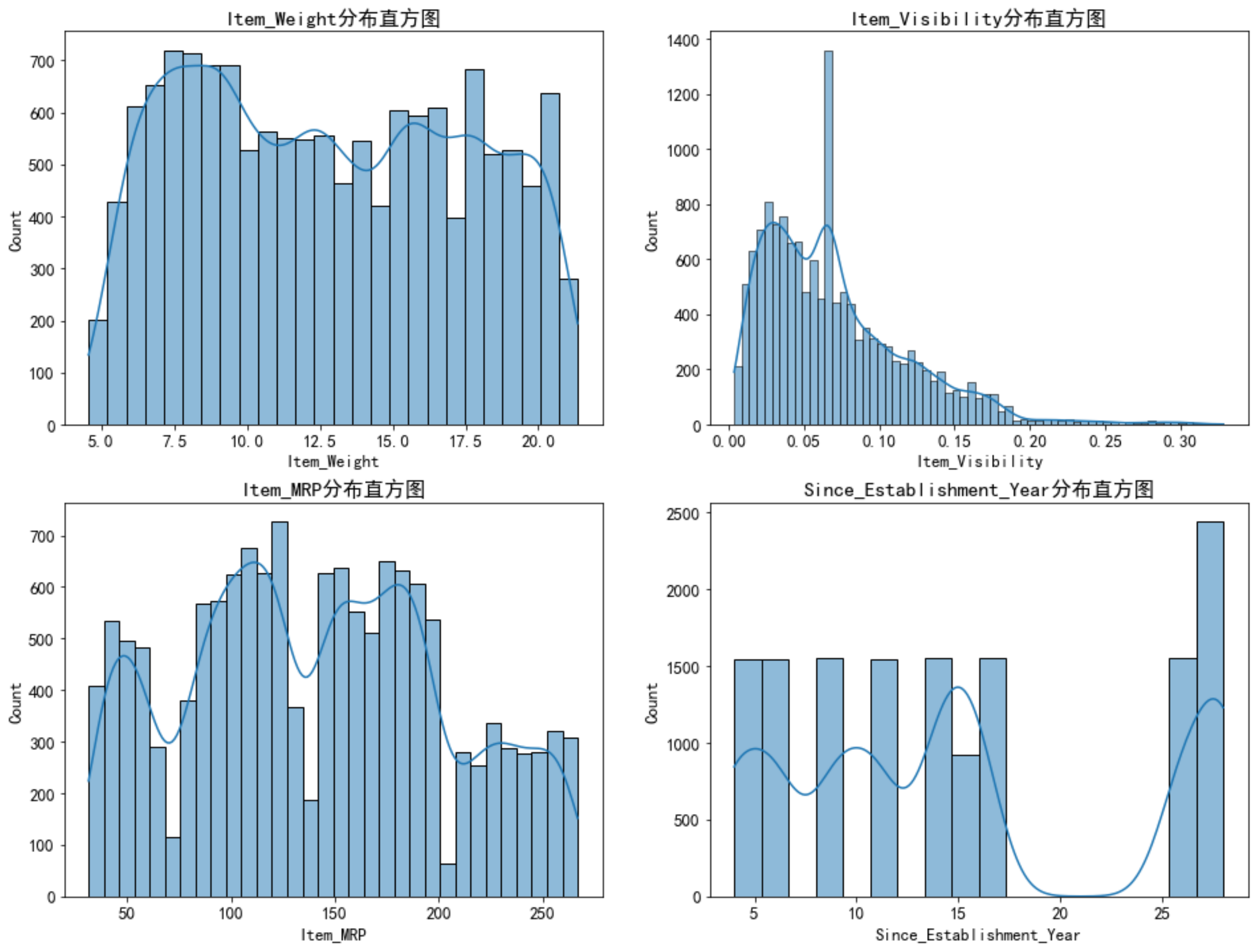
提取数值型变量

```
#取出数值型变量
numeric = data.dtypes[data.dtypes.values!='object'].index
numeric = numeric.drop('Outlet_Establishment_Year')#去除替换前的变量
numeric = numeric.drop('Item_Outlet_Sales') #去除目标变量
numeric.tolist()
```

```
['Item_Weight', 'Item_Visibility', 'Item_MRP', 'Since_Establishment_Year']
```

循环画图

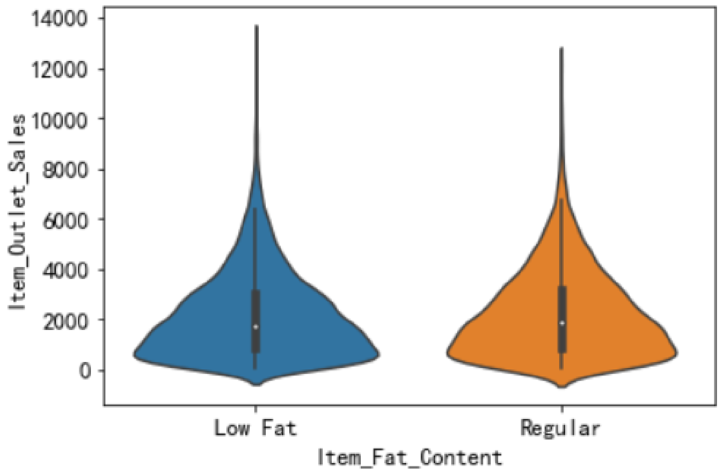
```
# 循环画图，展示数值型变量的分布情况
plt.figure(figsize=(16,12))
i = 1
for temp in numeric:
    plt.subplot(2,2,i)
    sns.histplot(data=data,x=temp,kde=True)
    plt.title(f'{temp}分布直方图')
    i += 1
plt.show()
```



特征变量和目标变量的相关性分析

1）‘Item_Fat_Content’与’Item_Outlet_Sales’：

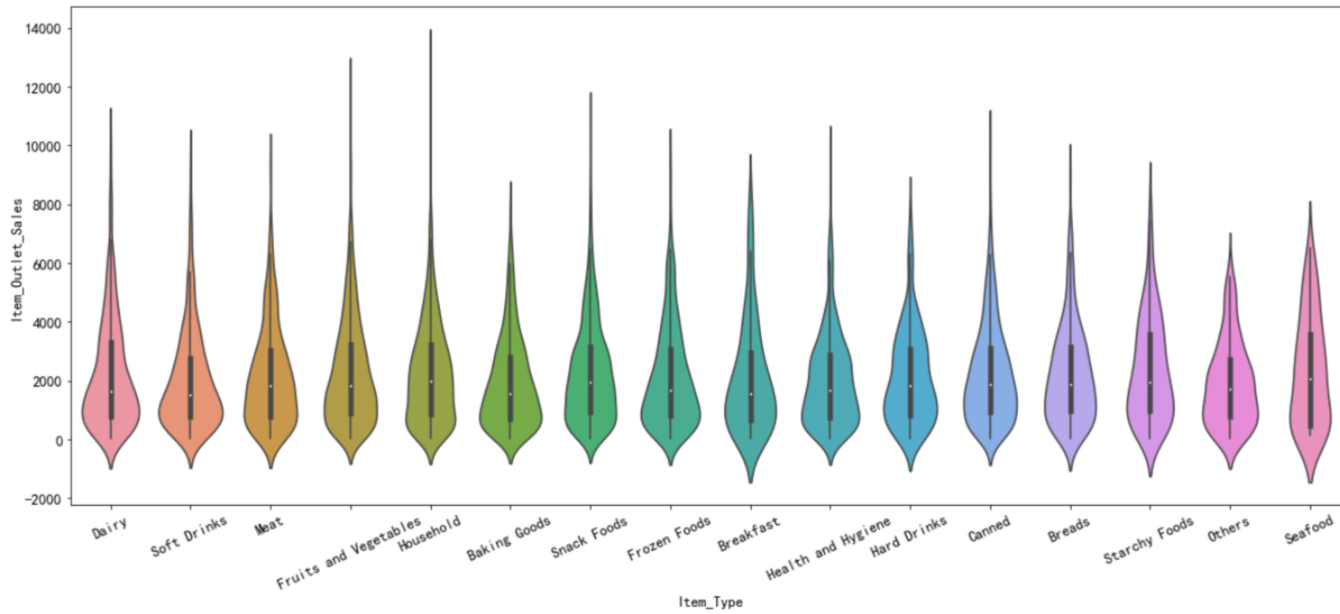
```
sns.violinplot(data=data,x='Item_Fat_Content',y='Item_Outlet_Sales')
plt.show()
```



分析：由上可知，‘Item_Fat_Content’ 的各类别与目标变量并没有明显的显著性差异。

2) ‘Item_Type’与’Item_Outlet_Sales’：

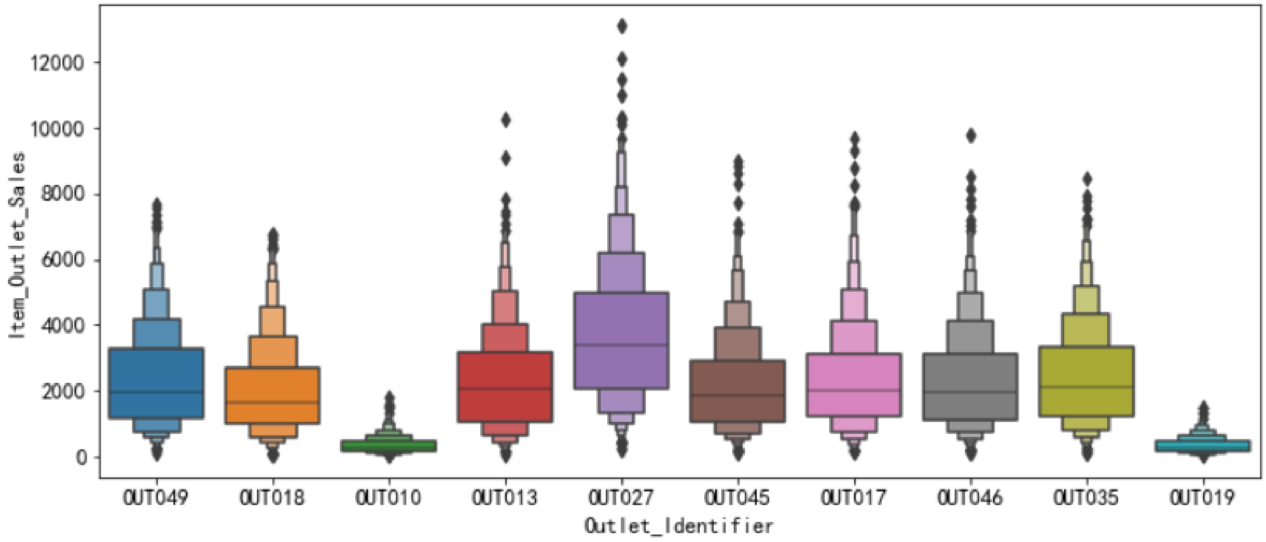
```
plt.figure(figsize=(20,8))
sns.violinplot(data=data,x='Item_Type',y='Item_Outlet_Sales')
plt.xticks(rotation=25)
plt.show()
```



分析：由上可知，‘Item_Type’的各类别与目标变量并没有明显的显著性差异。

3) ‘Outlet_Identifier’与’Item_Outlet_Sales’：

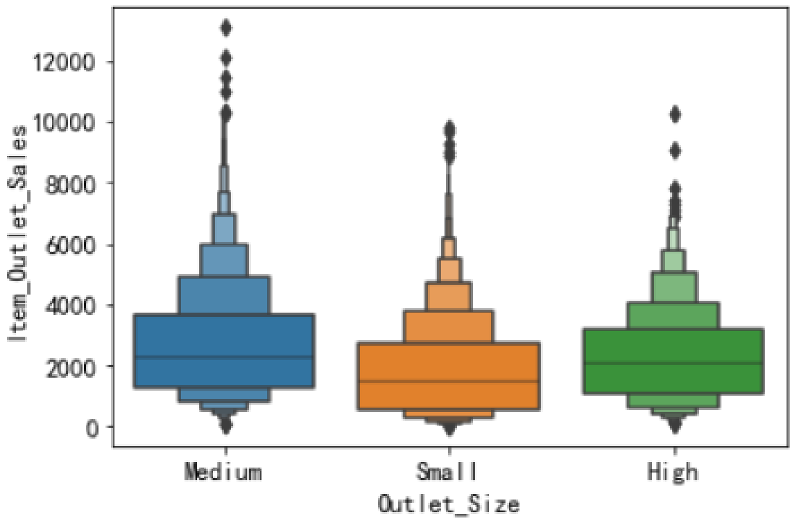
```
plt.figure(figsize=(12,5))
sns.boxenplot(data=data,x='Outlet_Identifier',y='Item_Outlet_Sales')
plt.show()
```



分析：由上可知，OUT010 和 OUT019 与其他商店编号相比，商品销售数量明显较少。

4) ‘Outlet_Size’与’Item_Outlet_Sales’：

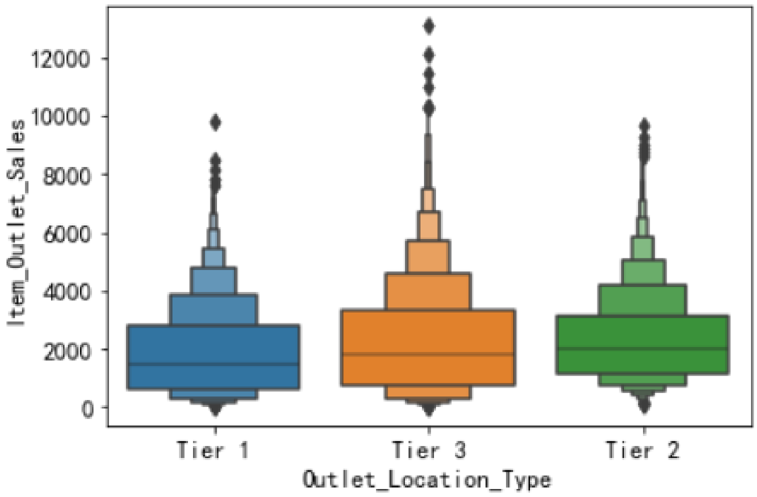
```
sns.boxenplot(data=data,x='Outlet_Size',y='Item_Outlet_Sales')
plt.show()
```



分析：由上可知，变量‘Outlet_Size’与目标变量并没有明显的显著性关系。

5) **‘Outlet_Location_Type’与’Item_Outlet_Sales’：**

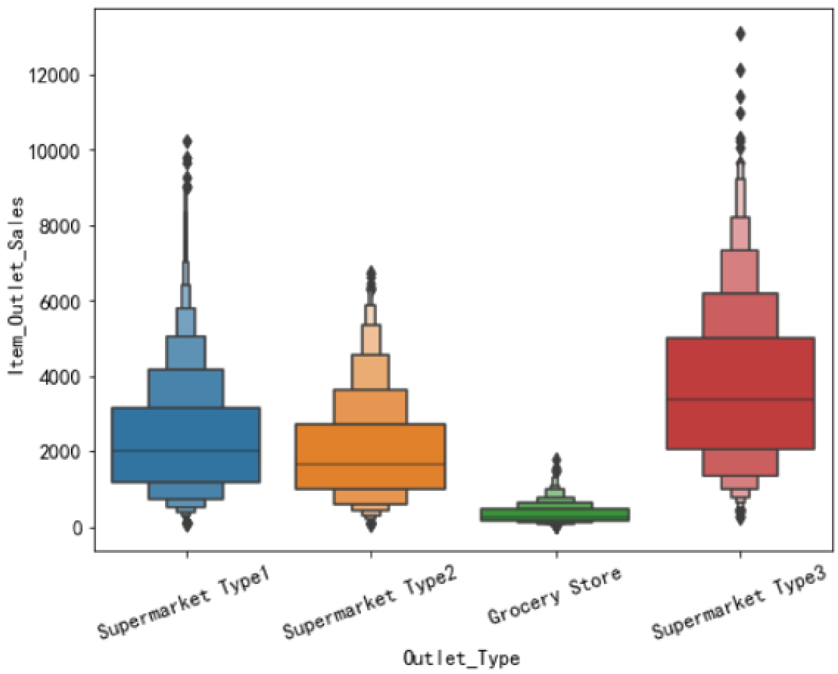
```
sns.boxenplot(data=data,x='Outlet_Location_Type',y='Item_Outlet_Sales')
plt.show()
```



分析：由上可知，变量‘Outlet_Location_Type’的各类别与目标变量并没有明显的显著性关系。

6) **‘Outlet _Type’与’Item_Outlet_Sales’：**

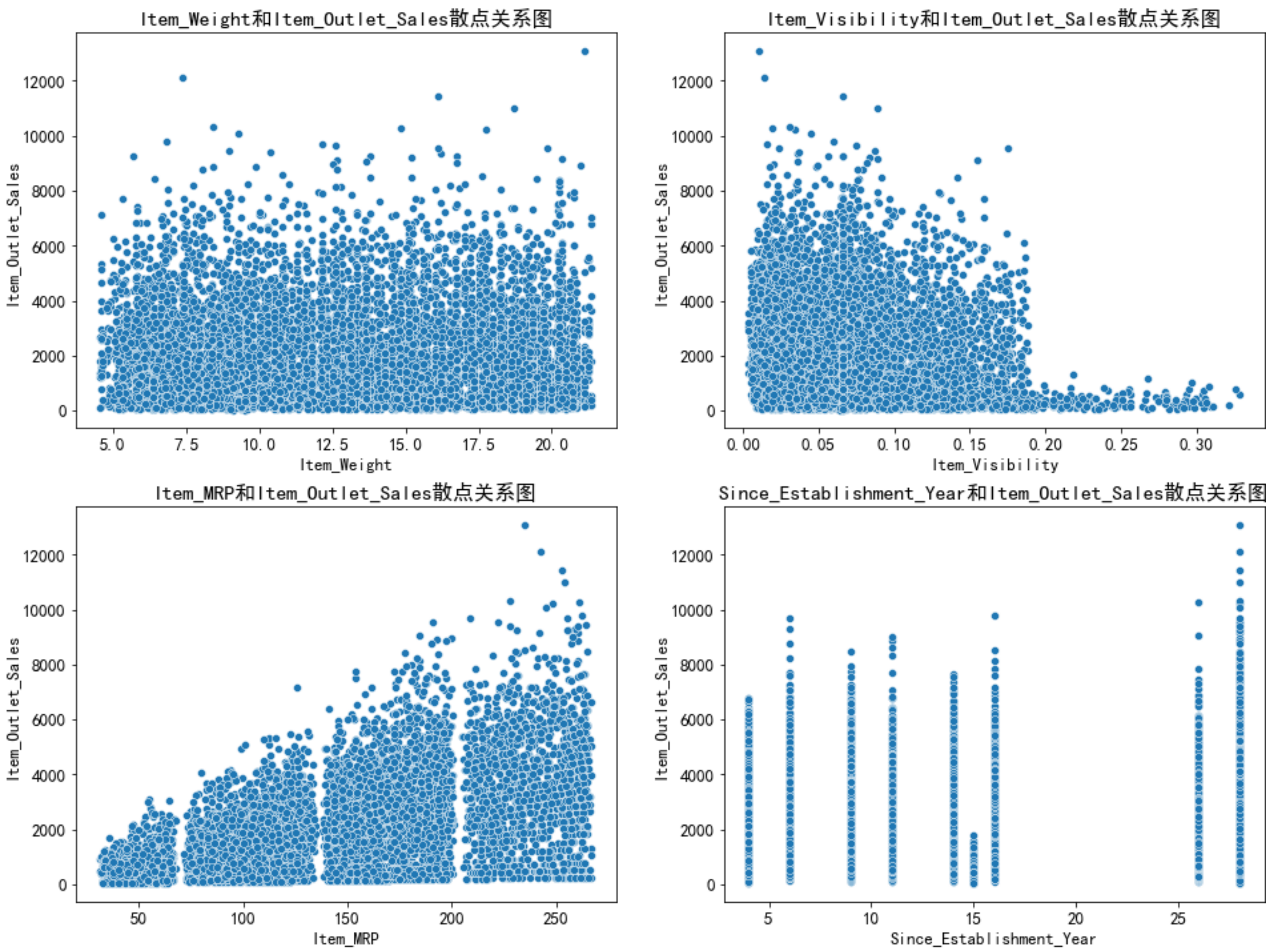
```
plt.figure(figsize=(8,6))
sns.boxenplot(data=data,x='Outlet_Type',y='Item_Outlet_Sales')
plt.xticks(rotation=20)
plt.show()
```



分析：由上可知，变量‘Outlet_Type’的 Grocery Store 的商品销售数量明显低于其他商店类型。

数值型变量和目标变量'Item_Outlet_Sales'

```
plt.figure(figsize=(16,12))
i = 1
for temp in numeric:
    plt.subplot(2,2,i)
    sns.scatterplot(data=data,x=temp,y='Item_Outlet_Sales')
    plt.title(f"{temp}和Item_Outlet_Sales散点关系图")
    i += 1
plt.show()
```



分析：由上可知，’Item_Weight’与目标变量并没有显著性关系；’Item_Visibility’和’Since_Estalishment_Year’与目标变量存在一定的相关关

系，但是关系程度并不明显；'Item_MRP'与目标变量存在较强的的正相关关系。

5、特征工程

1）对变量'Item_Type'，减少商品类别，分成'DR','FD','NC'三种

```
# 对变量'Item_Type'，减少商品类别，分成'DR','FD','NC'三种
DR = ['Hard Drinks','Soft Drinks']
FD = ['Breads','Breakfast','Canned','Dairy','Frozen Foods','Fruits and Vegetables','Meat','Seafood','Snack Foods','Starchy Foods']
NC = ['Baking Goods','Health and Hygiene','Household','Others']

#循环遍历data['Item_Type'],获取新的类型列表
Item_Type_new = []
for i in data['Item_Type']:
    if i in DR:
        Item_Type_new.append('DR')
    elif i in FD:
        Item_Type_new.append('FD')
    else:
        Item_Type_new.append('NC')

#新建data['Item_Type_new']列，存入上面 Item_Type_new 列表
data['Item_Type_new'] = Item_Type_new
data.loc[:,['Item_Type','Item_Type_new']]
```

	Item_Type	Item_Type_new
0	Dairy	FD
1	Soft Drinks	DR
2	Meat	FD
3	Fruits and Vegetables	FD
4	Household	NC
...	...	...
14199	Snack Foods	FD
14200	Starchy Foods	FD
14201	Health and Hygiene	NC
14202	Canned	FD
14203	Canned	FD

2）删除'Item_Identifier','Outlet_Identifier','Item_Type'和'Outlet_Establishment_Year',并将新的数据集存到 data_

```
# 删除'Item_Identifier','Outlet_Identifier','Item_Type'和'Outlet_Establishment_Year',并将新的数据集存到 data_
data_ = data.drop(['Item_Type','Outlet_Identifier','Outlet_Establishment_Year','Item_Identifier'],axis=1)
data_.head()
```

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_MRP	Outlet_Size	Outlet_Location_Type	Outlet_Type	Item_Outlet_Sales	Since_Establishment_Year	Item_Type_new
0	9.30	Low Fat	0.016047	249.8092	Medium	Tier 1	Supermarket Type1	3735.1380	14	FD
1	5.92	Regular	0.019278	48.2692	Medium	Tier 3	Supermarket Type2	443.4228	4	DR
2	17.50	Low Fat	0.016760	141.6180	Medium	Tier 1	Supermarket Type1	2097.2700	14	FD
3	19.20	Regular	0.000000	182.0950	Small	Tier 3	Grocery Store	732.3800	15	FD
4	8.93	Low Fat	0.000000	53.8614	High	Tier 3	Supermarket Type1	994.7052	26	NC

3）标签编码转换

```
# 针对变量存在特征值大小关系的采用标签编码转换
# 获取类别型变量列表
label_name = ['Item_Fat_Content','Outlet_Size','Outlet_Type']

# 标签编码转换
from sklearn.preprocessing import LabelEncoder
for i in label_name:
    data_[i] = LabelEncoder().fit_transform(data_[i])
```

4）独热编码转换

```
# 独热编码转换
#data_['Outlet_Location_Type'] = data_['Outlet_Location_Type'].astype('category',categories=['Tier 1','Tier 2','Tier 3'])
data_['Outlet_Location_Type'] = pd.Categorical(data_['Outlet_Location_Type'], categories=['Tier 1','Tier 2','Tier 3'], ordered=True)
one_hot = ['Outlet_Location_Type','Item_Type_new']#
data_ = pd.get_dummies(data_,columns=one_hot,prefix_sep='_')
data_.head()
```

m_Fat_Content	Item_Visibility	Item_MRP	Outlet_Size	Outlet_Type	Item_Outlet_Sales	Since_Establishment_Year	Outlet_Location_Type_Tier 1	Outlet_Location_Type_Tier 2	Outlet_Location_Type_Tier 3	Item_Type_new_DR	Item_Type_new_FD	Item_Type_new_
0	0.016047	249.8092	1	1	3735.1380	14	1	0	0	0	1	
1	0.019278	48.2692	1	2	443.4228	4	0	0	1	1	0	
0	0.016760	141.6180	1	1	2097.2700	14	1	0	0	0	1	
1	0.000000	182.0950	2	0	732.3800	15	0	0	1	0	1	
0	0.000000	53.8614	0	1	994.7052	26	0	0	1	0	0	

5）查看数据集类型情况：

```
# 查看数据集类型情况
data_.dtypes
```

```
Item_Weight                    float64
Item_Fat_Content                 int32
Item_Visibility                float64
Item_MRP                       float64
Outlet_Size                      int32
Outlet_Type                      int32
Item_Outlet_Sales              float64
Since_Establishment_Year         int64
Outlet_Location_Type_Tier 1      uint8
Outlet_Location_Type_Tier 2      uint8
Outlet_Location_Type_Tier 3      uint8
Item_Type_new_DR                 uint8
Item_Type_new_FD                 uint8
Item_Type_new_NC                 uint8
dtype: object
```

6）数值型变量归一化处理

由于上述数值型变量并非服从正态分布，因此选择的标准化方式为：归一化

```python
# 归一化处理
from sklearn.preprocessing import MinMaxScaler
num_lt = ['Item_Weight','Item_MRP','Item_Visibility','Since_Establishment_Year']
for n in num_lt:
    data_[n] = MinMaxScaler().fit_transform(data_[[n]])
```

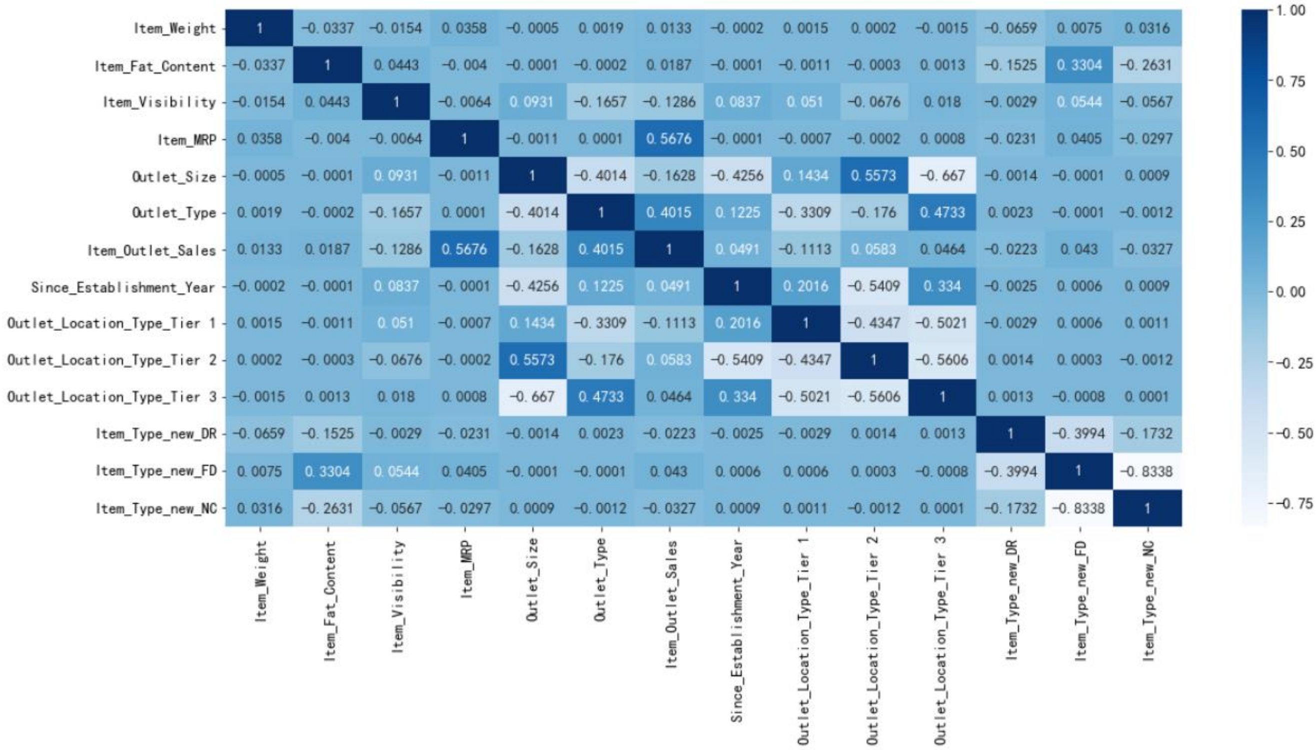
7）变量描述

```python
data_.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Item_Weight	14204.0	0.490526	0.276970	0.00	0.247395	0.479012	0.726109	1.0000
Item_Fat_Content	14204.0	0.353351	0.478027	0.00	0.000000	0.000000	1.000000	1.0000
Item_Visibility	14204.0	0.200836	0.156699	0.00	0.082328	0.164501	0.286358	1.0000
Item_MRP	14204.0	0.465686	0.263529	0.00	0.266224	0.470958	0.656055	1.0000
Outlet_Size	14204.0	1.453605	0.683045	0.00	1.000000	2.000000	2.000000	2.0000
Outlet_Type	14204.0	1.201281	0.796543	0.00	1.000000	1.000000	1.000000	3.0000
Item_Outlet_Sales	8523.0	2181.288914	1706.499616	33.29	834.247400	1794.331000	3101.296400	13086.9648
Since_Establishment_Year	14204.0	0.465388	0.348819	0.00	0.208333	0.416667	0.916667	1.0000
Outlet_Location_Type_Tier 1	14204.0	0.280203	0.449114	0.00	0.000000	0.000000	1.000000	1.0000
Outlet_Location_Type_Tier 2	14204.0	0.326739	0.469037	0.00	0.000000	0.000000	1.000000	1.0000
Outlet_Location_Type_Tier 3	14204.0	0.393058	0.488447	0.00	0.000000	0.000000	1.000000	1.0000
Item_Type_new_DR	14204.0	0.076598	0.265962	0.00	0.000000	0.000000	0.000000	1.0000
Item_Type_new_FD	14204.0	0.657843	0.474449	0.00	0.000000	1.000000	1.000000	1.0000
Item_Type_new_NC	14204.0	0.265559	0.441646	0.00	0.000000	0.000000	1.000000	1.0000

8）变量的相关性分析

```python
# 变量相关性分析
corr = data_.corr()
corr = round(corr,4)
plt.figure(figsize=(18,8))
sns.heatmap(corr,annot=True,cmap=plt.cm.Blues,fmt='g')
plt.show()
```



9）划分处理后的数据集

```
# 划分处理后的数据集
train_ = data_.loc[data_.Item_Outlet_Sales.isnull()==False]
test_ = data_.loc[data_.Item_Outlet_Sales.isnull()==True]
test_ = test_.drop(['Item_Outlet_Sales'],axis=1)
```

```
# 模型选取是的数据切分对象是 train_
X_ = train_.drop(['Item_Outlet_Sales'],axis=1)
Y_ = train_['Item_Outlet_Sales']
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(X_,Y_,test_size=0.3)
```

6、建立模型

1）模型选取

```
from sklearn.metrics import mean_squared_error
```

```
# 线性回归
from sklearn.linear_model import LinearRegression
LR = LinearRegression()
LR.fit(x_train,y_train)
y_pred = LR.predict(x_test)
LR_RMSE = np.sqrt(mean_squared_error(y_test,y_pred))
LR_RMSE
```

1146.3431559263283

```
# 随机森林回归
from sklearn.ensemble import RandomForestRegressor
RF = RandomForestRegressor()
RF.fit(x_train,y_train)
y_pred = RF.predict(x_test)
RF_RMSE = np.sqrt(mean_squared_error(y_test,y_pred))
RF_RMSE
```

1143.3699936853388

```
# K邻近回归
from sklearn.neighbors import KNeighborsRegressor
KNN = KNeighborsRegressor()
KNN.fit(x_train,y_train)
y_pred = KNN.predict(x_test)
KNN_RMSE = np.sqrt(mean_squared_error(y_test,y_pred))
KNN_RMSE
```

1156.4055562746805

```
# xgboost回归
from xgboost import XGBRegressor
XGBoost = XGBRegressor()
XGBoost.fit(x_train,y_train)
y_pred = XGBoost.predict(x_test)
XGBoost_RMSE = np.sqrt(mean_squared_error(y_test,y_pred))
XGBoost_RMSE
```

1177.6679419718419

```
# 梯度提升回归
from sklearn.ensemble import GradientBoostingRegressor
GB = GradientBoostingRegressor()
GB.fit(x_train,y_train)
y_pred = GB.predict(x_test)
GB_RMSE = np.sqrt(mean_squared_error(y_test,y_pred))
GB_RMSE
```

1067.794211014586

最优模型:

```
model_result = pd.DataFrame({'model':['LinearRegression',
                                      'RandomForestRegressor',
                                      'KNeighborsRegressor',
                                      'XGBRegressor',
                                      'GradientBoostingRegressor'],
                           'RMSE':[LR_RMSE,RF_RMSE,KNN_RMSE,XGBoost_RMSE,GB_RMSE]})
```

```
print('最优的模型(均方根误差最小): \n',model_result.sort_values(by='RMSE').iloc[[0],:])
```

```
最优的模型(均方根误差最小):
                        model         RMSE
4  GradientBoostingRegressor  1067.794211
```

2）模型参数搜索

```
# 对train_划分特征变量和目标变量
X_train = train_.drop(['Item_Outlet_Sales'],axis=1)
Y_train = train_['Item_Outlet_Sales']
```

网格搜索:

```
from sklearn.model_selection import GridSearchCV,StratifiedKFold
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import KFold,StratifiedKFold
GB_model = GradientBoostingRegressor()
# 调优参数
parameters = {'learning_rate':[0.01,0.1,0.3],
             'max_depth':[1,3,5],'n_estimators':[300,500,800],
              'min_samples_split':[2,4,6,10],
             'min_samples_leaf':[3,5,7,10,12],'alpha':[0.01,0.1,0.9]
             }
#5折交叉验证
kfold = KFold(n_splits=5, shuffle=True, random_state=10)
#网格搜索
grid_search = GridSearchCV(GB_model, parameters, scoring="r2",n_jobs=-1,cv=kfold)
grid_result = grid_search.fit(X_train,Y_train)
#输出最优结果
print(f"Best:{grid_result.best_score_};\nusing:{grid_result.best_params_}")
```

```
Best:0.5985534339222054;
using:{'alpha': 0.01, 'learning_rate': 0.01, 'max_depth': 3, 'min_samples_leaf': 12, 'min_samples_split': 2, 'n_estimators': 50
```

3）建模

GradientBoostingRegressor模型

```
from sklearn.ensemble import GradientBoostingRegressor
GB_model = GradientBoostingRegressor(alpha=0.01,learning_rate=0.01,max_depth=3,n_estimators=500,
                                     min_samples_leaf=12, min_samples_split=2,random_state=10)
```

```
# 模型训练
GB_model.fit(X_train,Y_train)
```

```
GradientBoostingRegressor
GradientBoostingRegressor(alpha=0.01, learning_rate=0.01, min_samples_leaf=12,
                          n_estimators=500, random_state=10)
```

```
# 模型预测
Y_pred = GB_model.predict(test_)
```

7、模型的评估

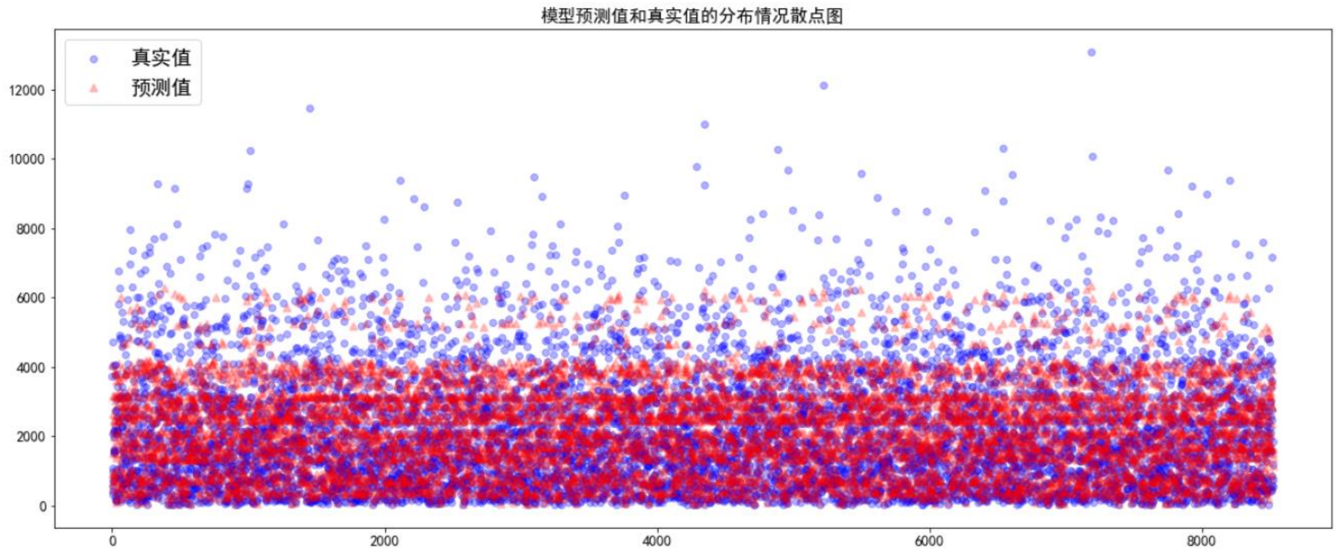
模型的各评价指标情况：

```
# 模型评价
from sklearn.metrics import mean_squared_error,r2_score,mean_absolute_error
Y_train_pred = round(pd.Series(data=GB_model.predict(X_train)),4)
print("Score:",GB_model.score(X_train,Y_train))
print("RMSE:",np.sqrt(mean_squared_error(Y_train,Y_train_pred)))
print("R Squared is:" ,r2_score(Y_train,Y_train_pred))
print("MAE:",mean_absolute_error(Y_train,Y_train_pred))
```

```
Score: 0.6156870930085013
RMSE: 1057.8478335652023
R Squared is: 0.6156870925027944
MAE: 743.2906075912238
```

可视化呈现预测值和真实值的分布:

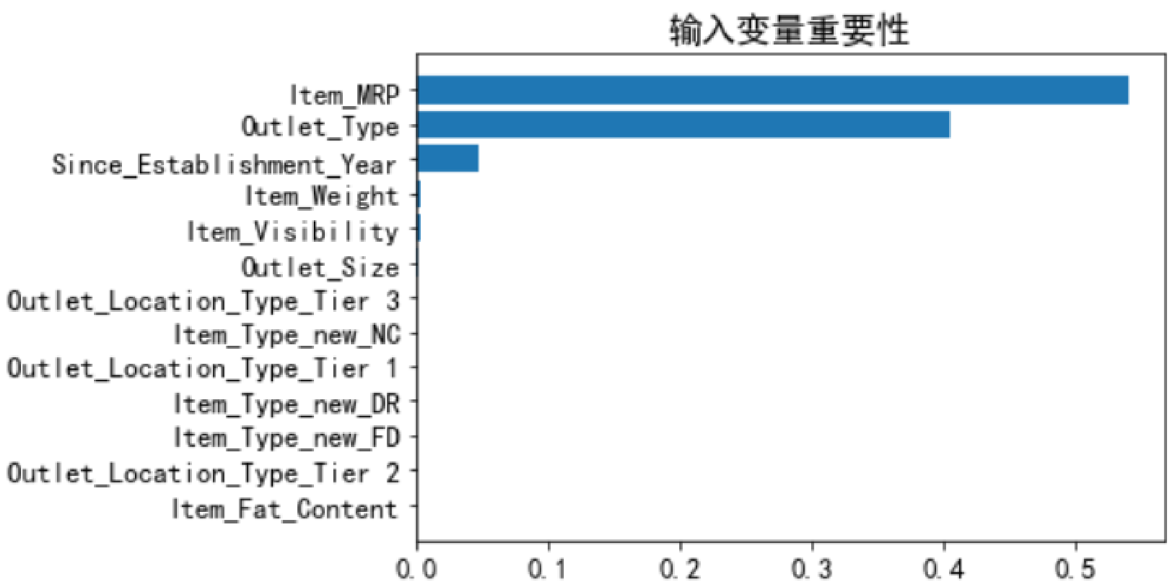
```
plt.figure(figsize=(20,8))
plt.scatter(Y_train.index,Y_train.values, c='b', alpha=0.3, linewidth=1,label='真实值')
plt.scatter(Y_train_pred.index,Y_train_pred.values, c='r',marker='^', alpha=0.2,linewidth=2,label='预测值')
plt.title('模型预测值和真实值的分布情况散点图')
plt.legend(loc=2,fontsize=18)
plt.show()
```



输入变量重要性:

```
feature_importances = pd.DataFrame({'feature_name':GB_model.feature_names_in_,
                                    'feature_importance':GB_model.feature_importances_})
feature_importances.sort_values(by='feature_importance',inplace=True)
```

```
plt.barh(y=feature_importances.feature_name.to_list(),width=feature_importances.feature_importance.to_list())
plt.title('输入变量重要性')
plt.show()
```



```
display('输入变量重要性前五个：',feature_importances.iloc[-5:,:])
```

'输入变量重要性前五个：'

	feature_name	feature_importance
0	Item_Weight	0.002483
2	Item_Visibility	0.003061
6	Since_Establishment_Year	0.049784
5	Outlet_Type	0.402470
3	Item_MRP	0.540958

8、模型应用

```
# 运用模型对测试集test_进行商品销量预测
Y_pred_test = GB_model.predict(test_)
```

```
# 预测结果
Y_pred_test
```

```
array([1674.45111491, 1406.9321468 ,  626.23112538, ..., 1877.18881615,
       3546.77104798, 1305.06394184])
```

```
# 结果输出
test_pred = test_  #构建新的dataframe
#对于test_pred新增'Item_Outlet_Sales_pred'列，存放模型预测结果
test_pred['Item_Outlet_Sales_pred'] = Y_pred_test
# 对预测值保留4位小数
test_pred['Item_Outlet_Sales_pred']=test_pred['Item_Outlet_Sales_pred'].map(lambda x:round(x,4)
# 结果写入excel文档
test_pred.to_excel('对测试集进行预测商品销售额.xlsx')
```

对测试集进行预测商品销售额.xlsx - Excel

	Item_Weigh	_Fat_Con	_Visibil	Item_MRP	tlet_Siz	tlet_Typ	tablishm	ation_Ty	ation_Ty	ation_Ty	_Type_ne	_Type_ne	_Type_ne	Item_Outlet_Sales_pred
8523	0.964275	0	0.012284	0.325012	1	1	0.416667	1	0	0	0	1	0	1674.4511
8524	0.222983	1	0.107301	0.237819	2	1	0.083333	0	1	0	0	1	0	1406.9321
8525	0.598095	0	0.295552	0.893316	2	0	0.458333	0	0	1	0	0	1	626.2311
8526	0.164335	0	0.03637	0.525233	2	1	0.083333	0	1	0	0	1	0	2488.4225
8527	0.538553	1	0.354122	0.861381	1	3	1	0	0	1	0	1	0	6065.8321
8528	0.312295	1	0.185466	0.36443	2	1	0.5	1	0	0	0	1	0	1877.1888
8529	0.880917	1	0.243297	0.079854	1	2	0	0	0	1	0	0	1	675.6665
8530	0.276273	0	0.037584	0.211246	1	3	1	0	0	1	0	0	1	2215.6282
8531	0.104198	1	0.368795	0.273574	2	1	0.291667	0	1	0	0	1	0	1534.2954
8532	0.085144	0	0.006538	0.660456	2	1	0.083333	0	1	0	0	0	1	3066.3534
8533	0.717178	0	0.307849	0.369513	2	1	0.083333	0	1	0	0	1	0	1898.2158
8534	0.121167	0	0.314753	0.229631	2	1	0.291667	0	1	0	0	1	0	1382.9003
8535	0.78565	0	0.51569	0.887653	2	0	1	1	0	0	0	0	1	601.4293
8536	0.013695	0	0.274503	0.386335	1	1	0.416667	1	0	0	0	1	0	2072.3044
8537	0.726109	0	0.054282	0.08803	0	1	0.916667	0	0	1	1	0	0	839.812
8538	0.094076	1	0.233597	0.510812	1	1	0.416667	1	0	0	0	0	1	2480.5731
8539	0.910688	0	0.155659	0.7109	2	1	0.291667	0	1	0	0	1	0	3099.0194
8540	0.791605	0	0.105925	0.683064	1	2	0	0	0	1	1	0	0	2875.1429
8541	0.093778	0	0.075765	0.332775	1	3	1	0	0	1	0	0	1	2739.6821
8542	0.538553	0	0.595176	0.689409	2	0	0.458333	0	0	1	0	1	0	551.2091
8543	0.151533	0	0.327402	0.610013	0	1	0.916667	0	0	1	0	1	0	2871.966
8544	0.871986	0	0.551218	0.885531	2	1	0.208333	0	1	0	0	0	1	3869.8165
8545	0.54153	0	0.19105	0.069823	2	1	0.5	1	0	0	0	1	0	798.3498
8546	0.642751	0	0.073495	0.471383	2	0	1	1	0	0	0	0	1	399.5401
8547	0.871986	0	0.097299	0.634307	2	1	0.208333	0	1	0	0	1	0	2962.6366
8548	0.132778	1	0.107385	0.756384	2	0	0.458333	0	0	1	0	1	0	629.5618
8549	0.502828	0	0.097062	0.069242	2	1	0.208333	0	1	0	0	0	1	827.5134
8550	0.075618	0	0.270647	0.552208	2	1	0.5	1	0	0	0	0	1	2620.4969
8551	0.550461	0	0.167873	0.905737	2	1	0.5	1	0	0	0	1	0	3889.4998
8552	0.49092	0	0.05962	0.361884	2	1	0.208333	0	1	0	0	1	0	1908.8507
8553	0.270914	0	0.196425	0.003396	2	0	1	1	0	0	0	0	1	83.8183
8554	0.294433	1	0.057183	0.694636	1	2	0	0	0	1	0	1	0	2864.1001
8555	0.473057	0	0.009907	0.047204	1	2	0	0	0	1	0	0	1	560.8826

五、实验问题:

对于回归模型而言，进入模型的数据集的转换相当重要，即特征工程，其中对于存在明显大小关系的类别型变量可以采取标签编码转换，对于其余的可采用独热编码转换。从上述结果看来，利用 GradientBoostingRegressor 模型的预测值与真实值比较，预测效果偏保守。