

**[数据预处理实验]**

**Python程序设计**

**学院：计算机学院**

**2020211376 马天成**

**2022年12月11日**

**北京邮电大学《Python程序设计》课程实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **实验**  **名称** | 数据预处理 | | **学 院** | 计算机 | **指导教师** | 王晶 |
| **班 级** | **班内序号** | **学 号** | | **学生姓名** | **成绩** | |
| **2020211305** | **2** | **2020211376** | | **马天成** |  | |
|  | 作业1：爬取并存储链家的新房数据，并进行预处理。  作业2：分析处理2015年北京市PM2.5指数数据集空值 | | | | | |
| **学生实验报告** | （详见“实验报告和源程序”册） | | | | | |
| **课**  **程**  **设**  **计**  **成**  **绩**  **评**  **定** | **评语**:  **成绩**:  指导教师签名：  年 月 日 | | | | | |

注：评语要体现每个学生的工作情况，可以加页。

目录

[1. 实验目的和要求 4](#_Toc121685637)

[2. 爬取房屋数据 4](#_Toc121685638)

[2.1 数据观测 4](#_Toc121685639)

[2.2 代码编写 5](#_Toc121685640)

[2.2.1 准备输出文件 5](#_Toc121685641)

[2.2.2 读取前25页数据到输出文件 5](#_Toc121685642)

[2.2.3 分析数据 6](#_Toc121685643)

[2.2.4 异常处理 6](#_Toc121685644)

[2.2.5 离散化处理 8](#_Toc121685645)

[3. 分析北京市数据 9](#_Toc121685646)

[3.1 数据观测 9](#_Toc121685647)

[3.2 代码编写 9](#_Toc121685648)

[3.2.1 数据抽取及存储 9](#_Toc121685649)

[3.2.2 找出空值 10](#_Toc121685650)

[3.2.3 空值处理方法 10](#_Toc121685651)

[4：代码部分 12](#_Toc121685652)

[4.1 houses.py 12](#_Toc121685653)

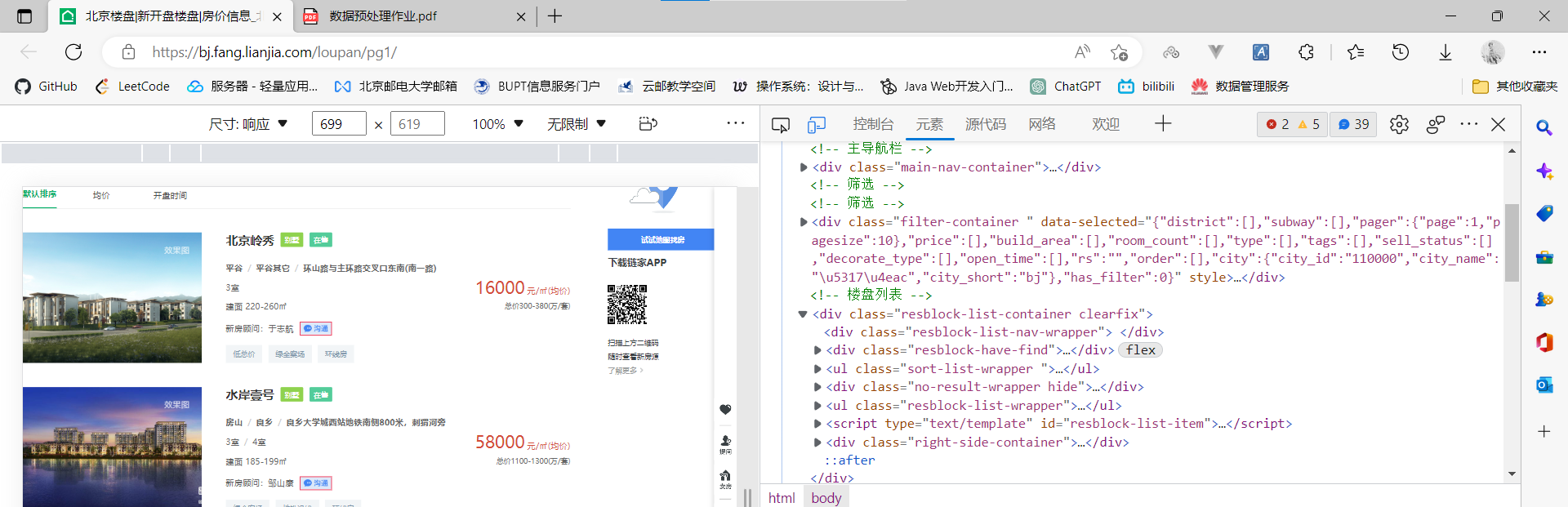
[4.2 BJPM.py 14](#_Toc121685654)

# 1. 实验目的和要求

|  |  |
| --- | --- |
| **实**  **验**  **内**  **容** | 作业1：爬取并存储链家的新房数据，并进行预处理。  （1）爬取起始网页：<https://bj.fang.lianjia.com/loupan/>  （2）爬取信息的提取及存储要求（单条数据示例在第3页）  （4）异常值处理 • 列出总价在均值三倍标准差以外的房屋，展示其基本信息（如果太 多可以只展示一部分）  （3）数据统计 • 找出总价最贵和最便宜的房子，以及总价的中位数 • 找出均价最贵和最便宜的房子，以及均价的中位数  （5）离散化处理 • 对房屋的均价进行离散化处理，自行设定每个区间的长度并给出设 置的理由，给出每个区间的房屋数量和所占比例  作业2：分析处理2015年北京市PM2.5指数数据集空值  （1）原始数据集：BeijingPM20100101\_20151231.csv（列信息见第5页 说明）  （2）数据抽取及存储：从原始数据集中抽取2015年度数据，存储为新 的csv文件  （3）找出空值：对新的csv文件，找出存在的空值列及相应的空值数量  （4）空值处理方法：对所有存在空值的列，给出空值的处理方法及理 由，要求处理方法必须可在本数据集范围内执行  （5）空值处理并存储：按照自己的处理方法，通过pandas、numpy或 python方法对空值进行处理，完成后给出新的空值列信息，并将处理后 的数据（不涉及空值的列应原样保留）存储为新的csv文件 |

# 2. 爬取房屋数据

## 2.1 数据观测

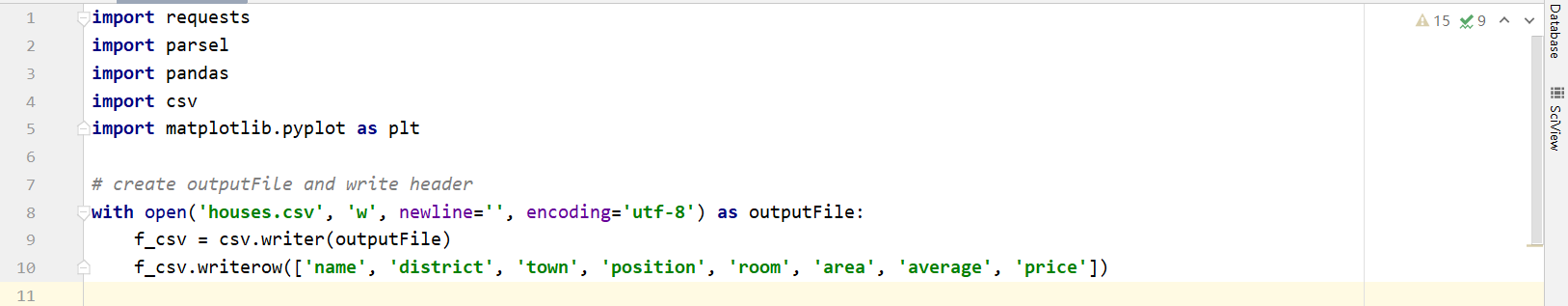


很显然可以观察到展示数据的模块。

我们需要做的就是根据库把所有div里的数值拿出来。

## 2.2 代码编写

### 2.2.1 准备输出文件

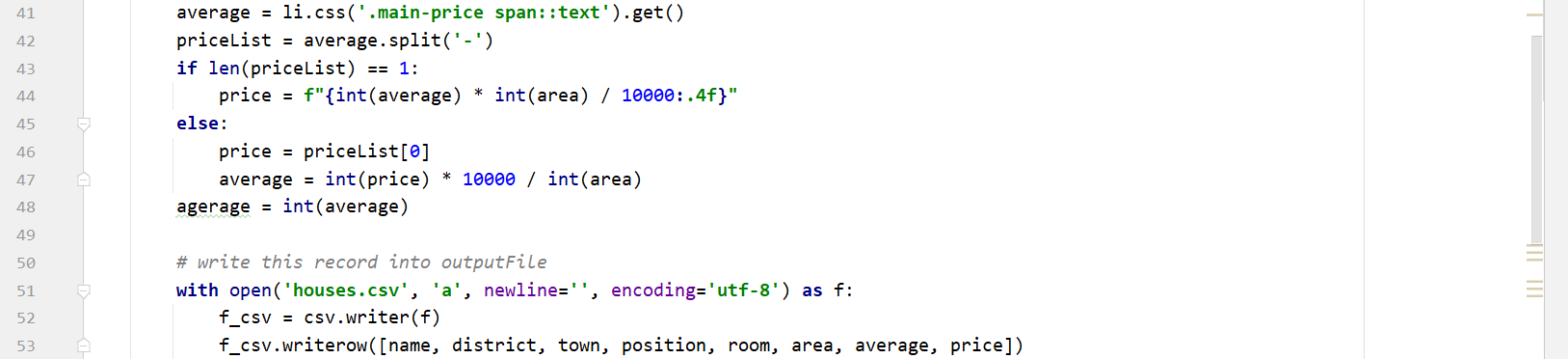


是准备output的文件。这里使用csv来存储。

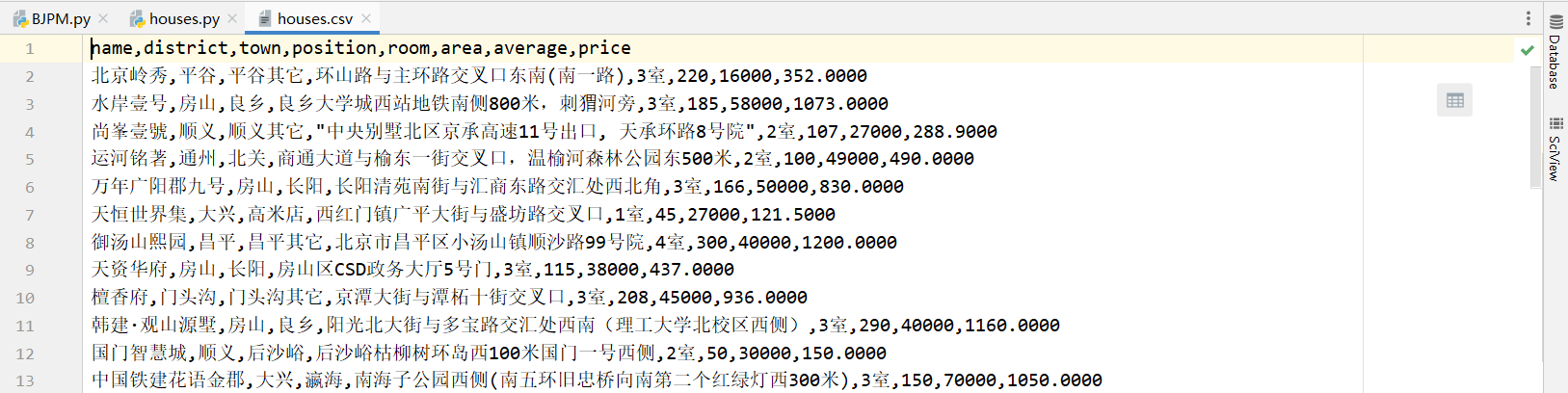
字段根据英文名就可以知道了吧。

### 2.2.2 读取前25页数据到输出文件

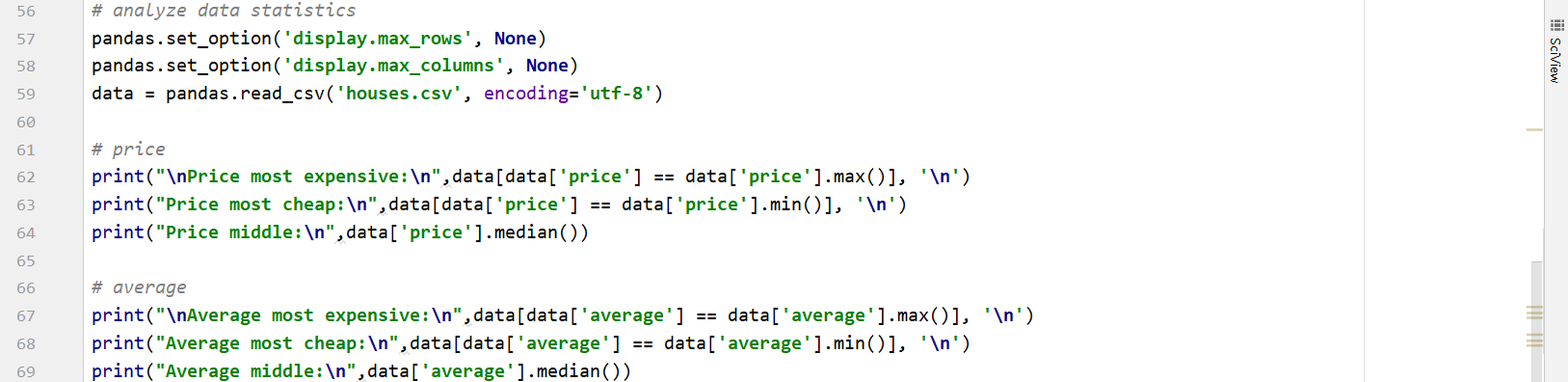




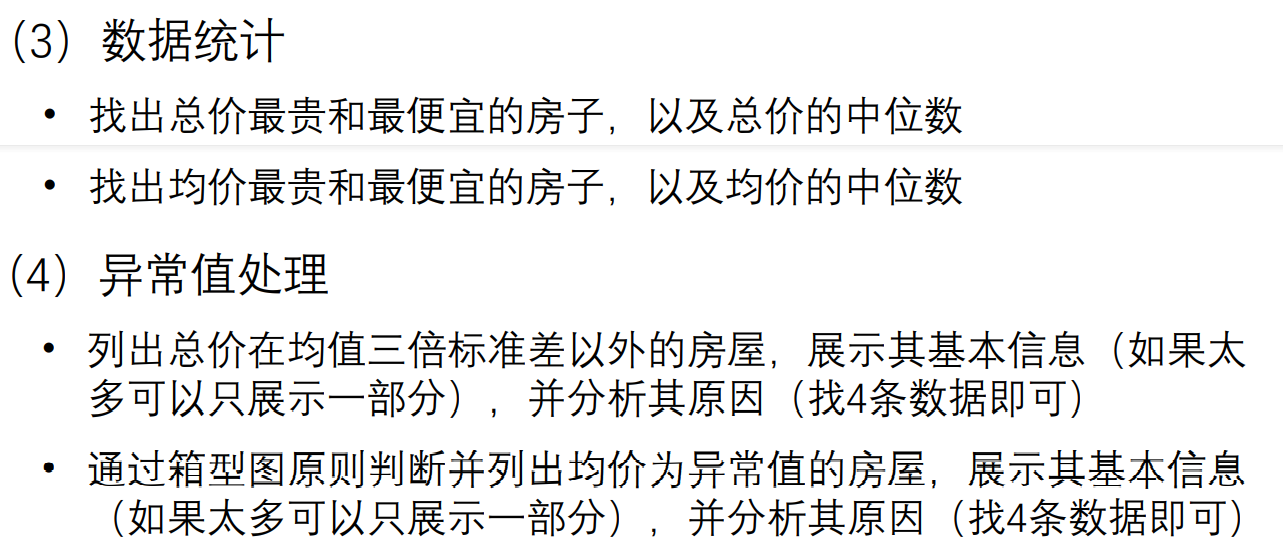
这里也非常简单，只不过点繁琐。根据div的属性把所有的值取出来，适当处理后形成需要的记录存储到csv。这就是数据预处理。输出文件长这样：



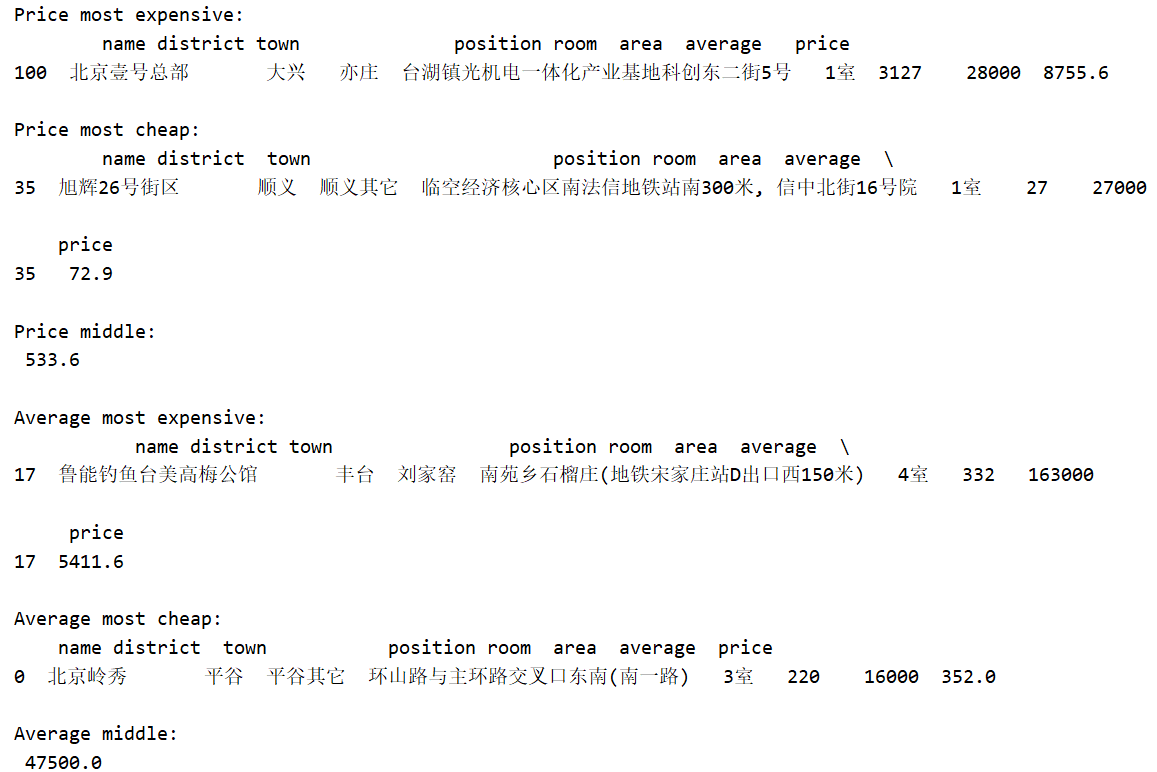
### 2.2.3 分析数据



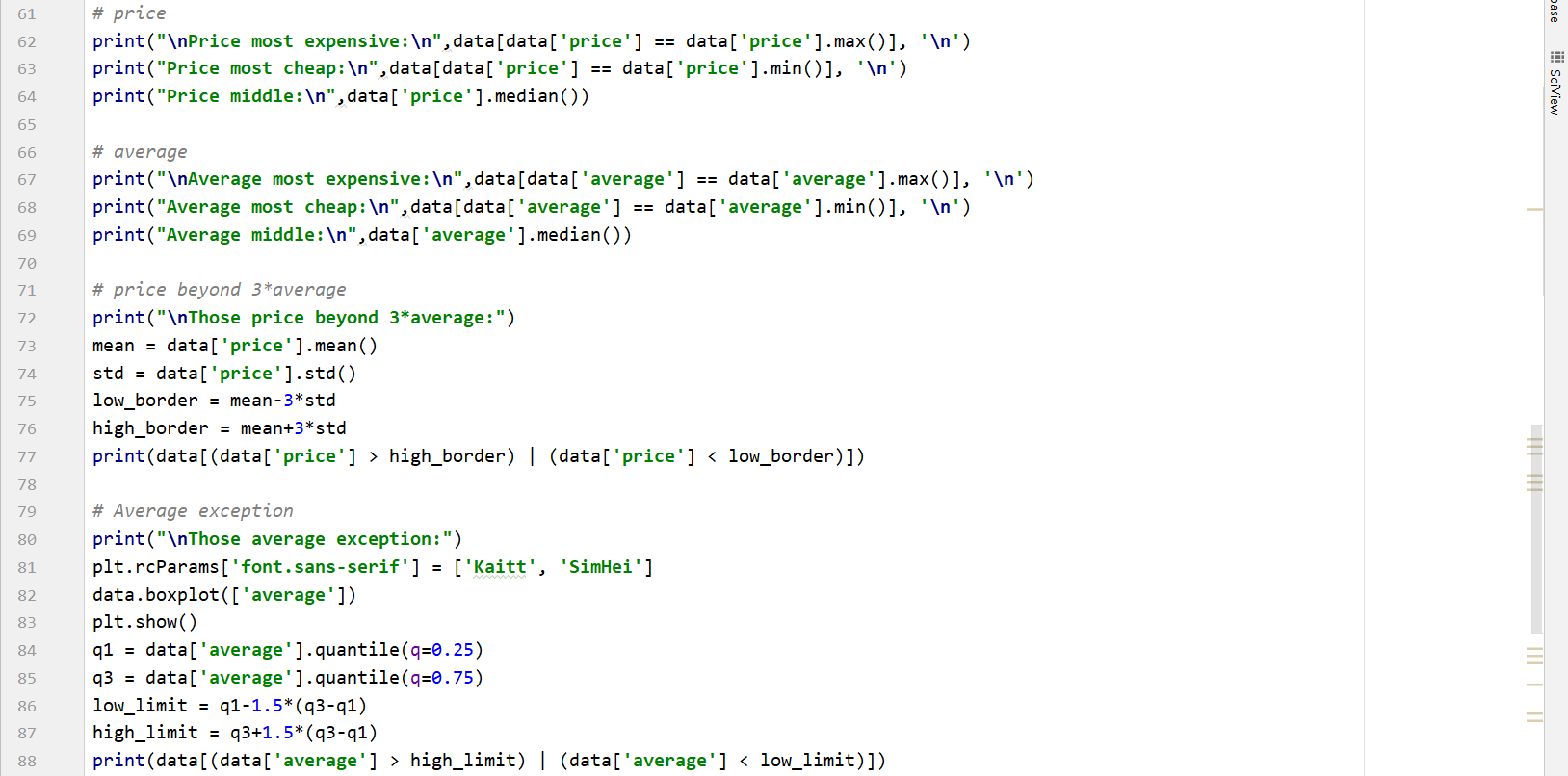
这里是寻找一些特殊的记录展示出来。这里分别是找到：



所以6个记录分别是：

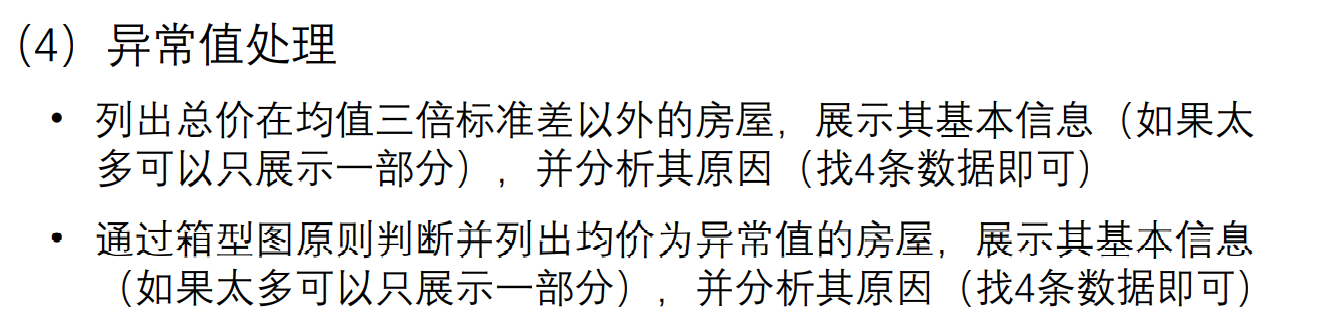


### 2.2.4 异常处理

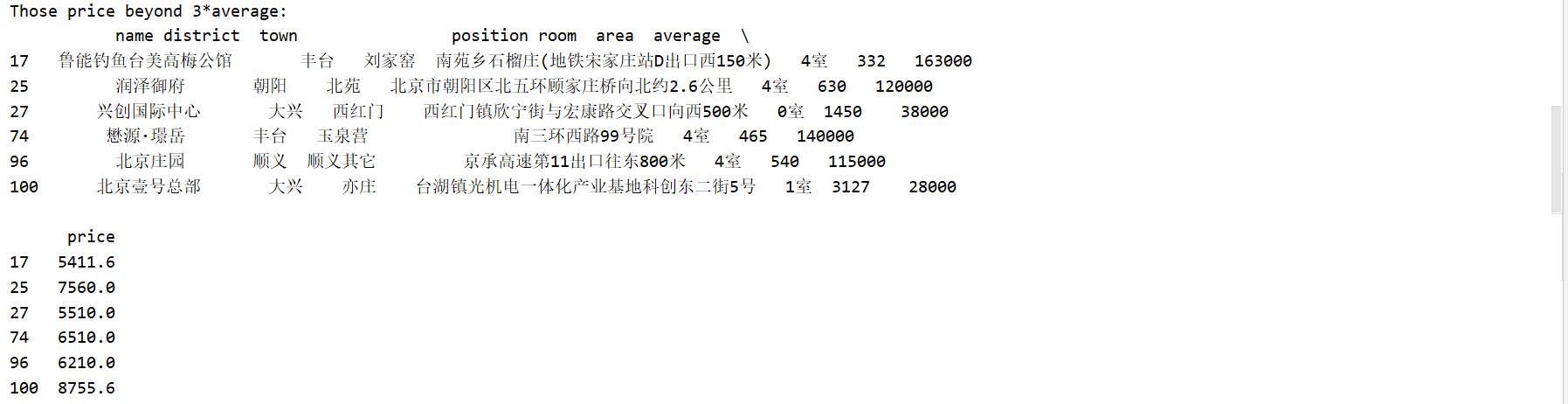


这里就是很显然的数据处理了。

找出的是以下属性的记录：

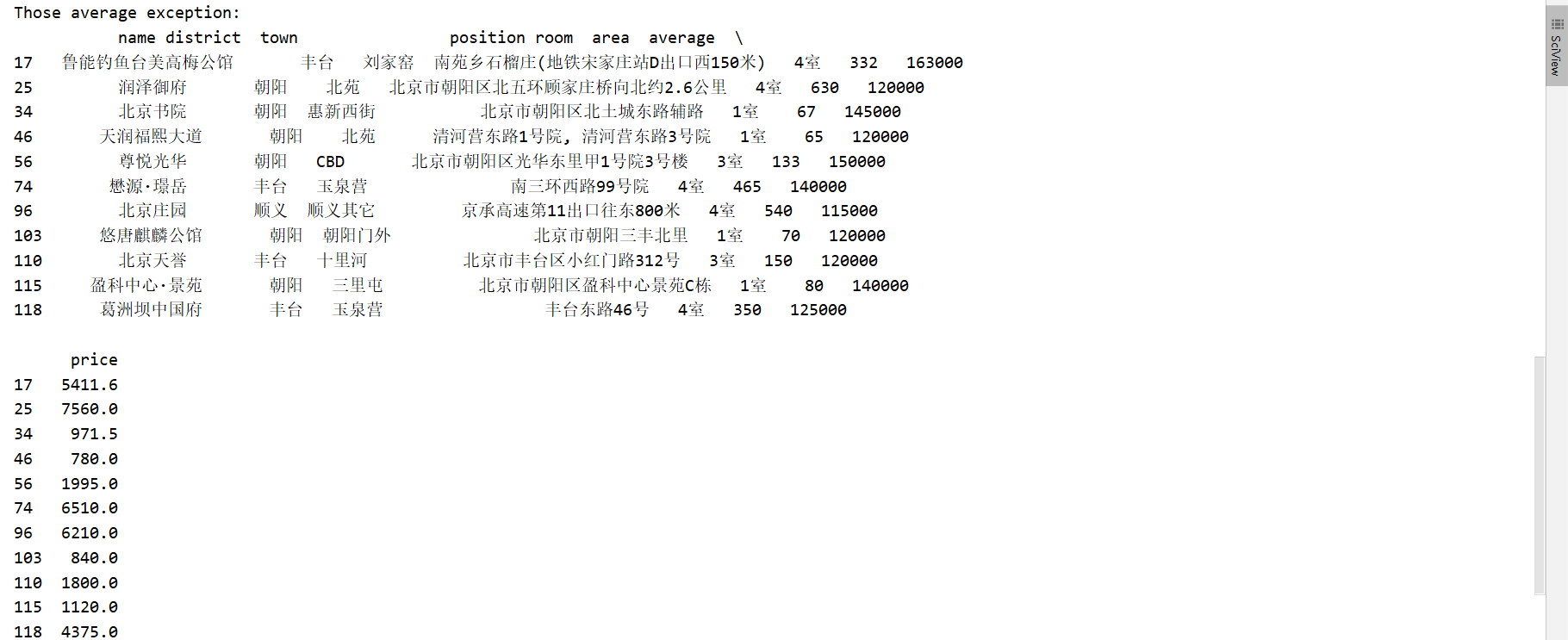


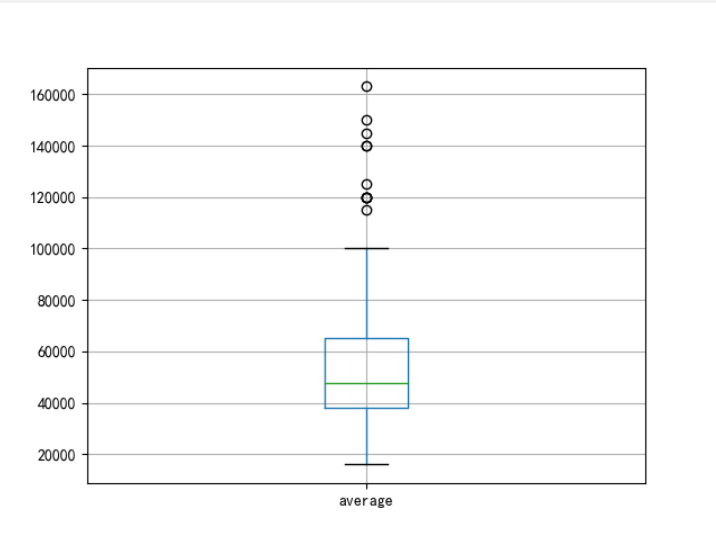
1. 超三倍的异常记录如下：



在这个纪录中，可看到这些圆点。这些数据的共同点为面积大，均价高，房屋所占面积越大总价越高。

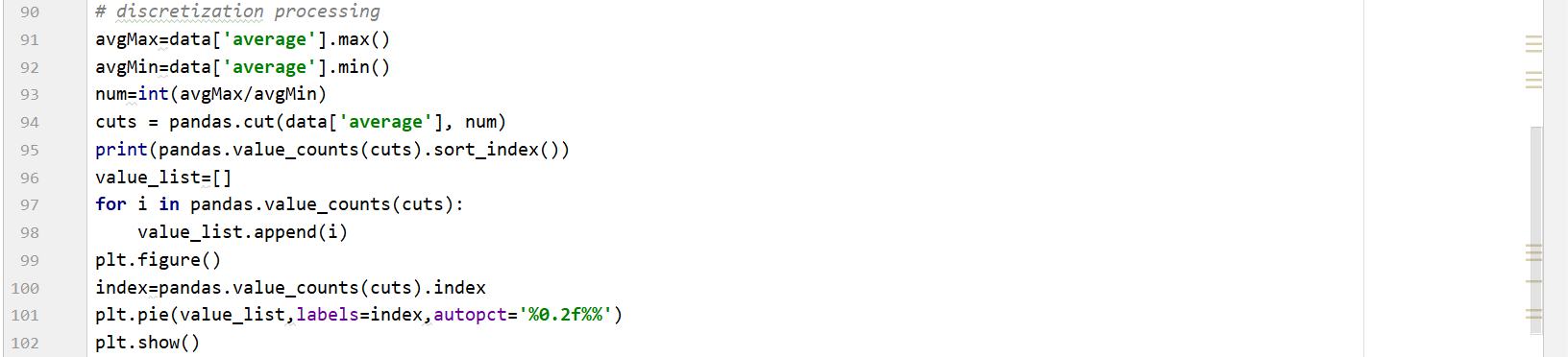
2. 均价异常的房屋如下：



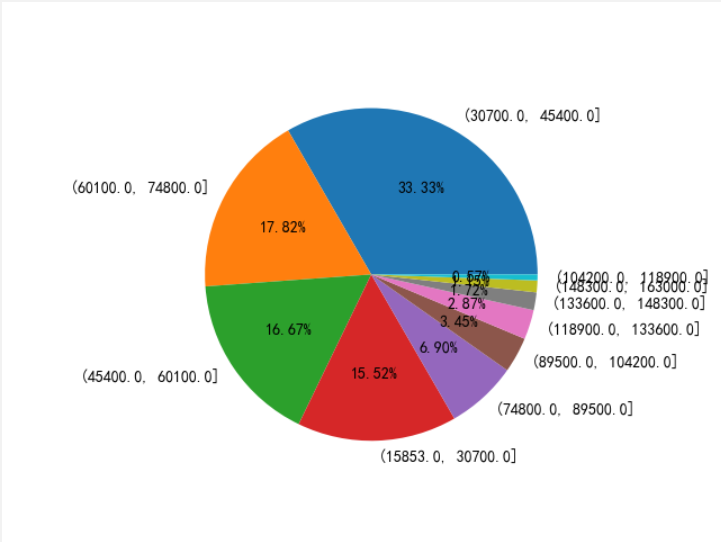


这些都是因为单价很高或price很高的房屋，导致最终的average计算出来高。

### 2.2.5 离散化处理



直接把东西贴出来完事。就是简单把这些归类分组然后展示成饼状图。



分组依据：用均价最高值除以均价最低值获取划分个数。

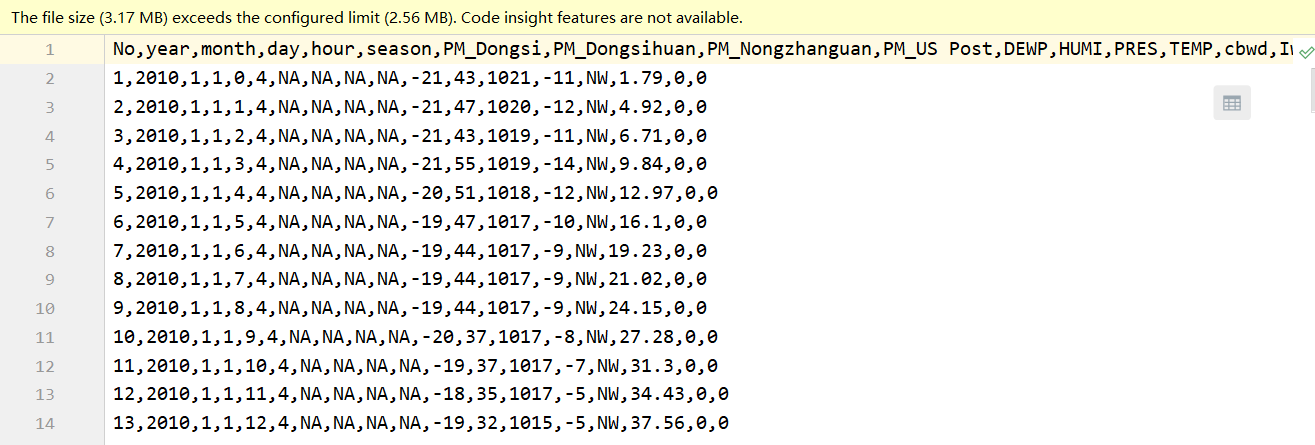
分组结果：



由上图可知，呈现一个类似正态分布的形状。中间值大约在20700到45400中间偏大的位置。

# 3. 分析北京市数据

## 3.1 数据观测



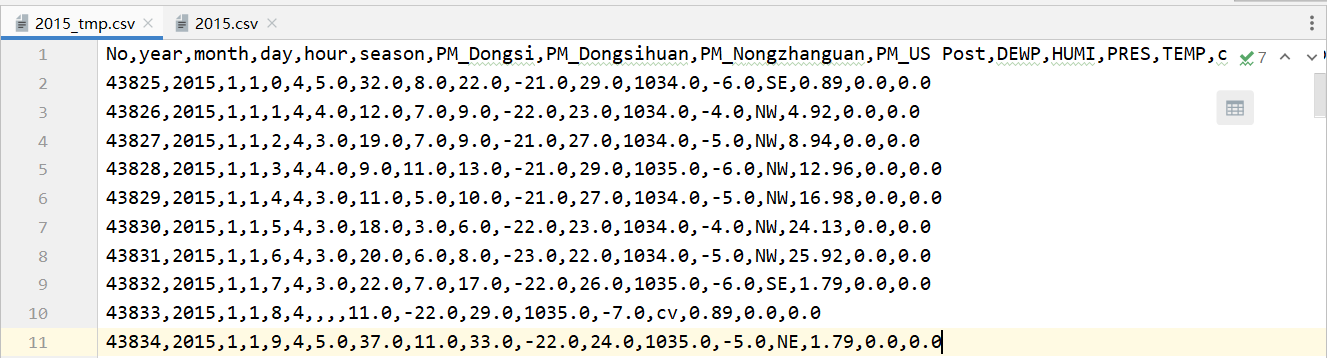
显然NA啥的呀NW肯定要处理一下。

## 3.2 代码编写

### 3.2.1 数据抽取及存储



就是把所有的2015存到2015\_tmp.csv中。输出文件如下：

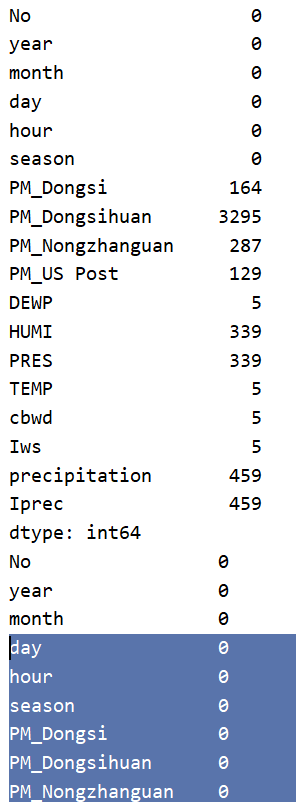




### 3.2.2 找出空值



就是把所有空值记录取出输出。



### 3.2.3 空值处理方法



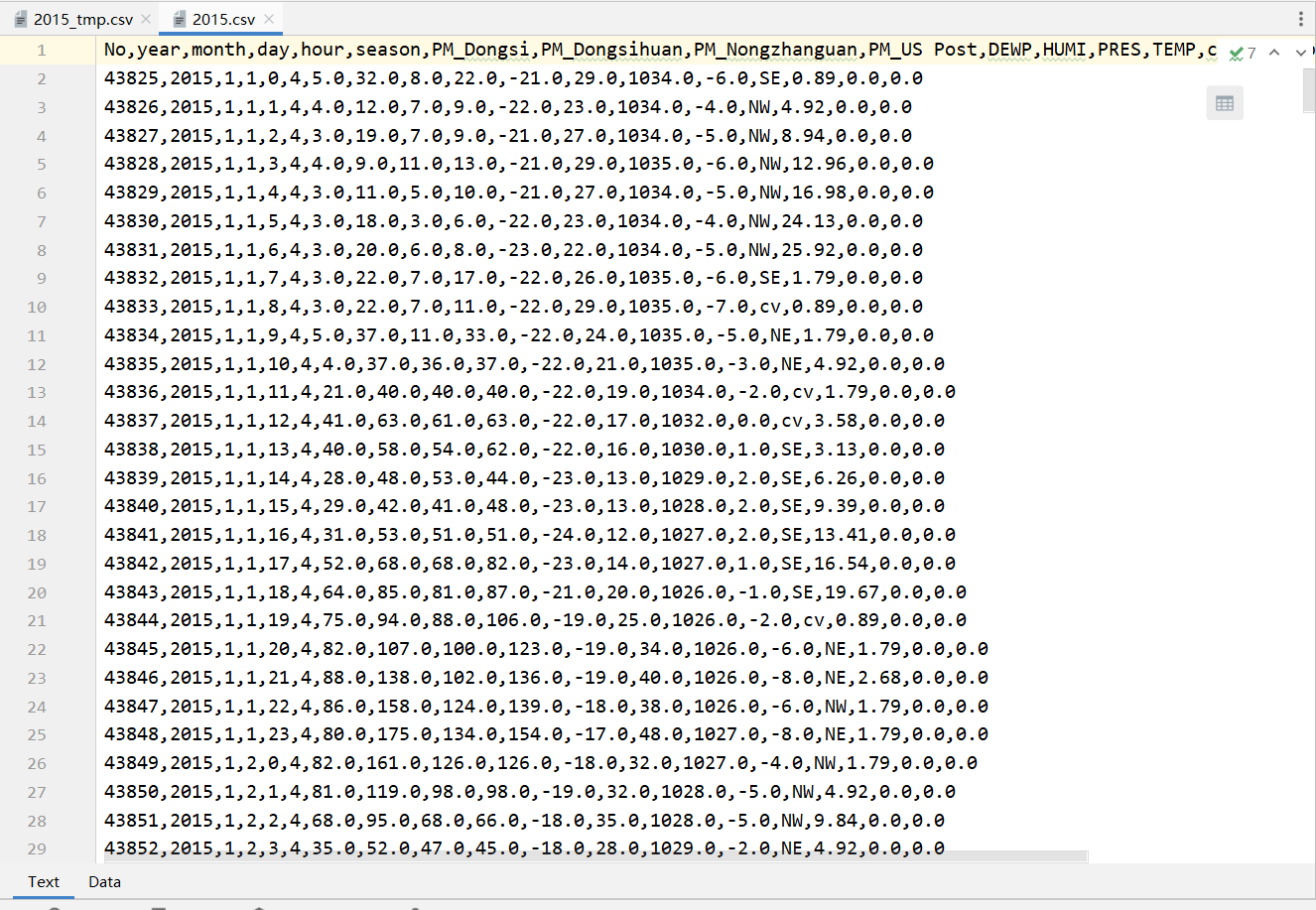
对所有存在空值的列，给出空值的处理方法及理 由，要求处理方法必须可在本数据集范围内执行。



观察原数据，DEWP、TEMP等空值数量较少的属性的空值都是来自同一组数据，且数量只有5组，因此直接将此5组数据进行删除。

剩余空值较多的列，考虑到数据采集过程，前后差距不大，默认采用上一条记录数据进行复制填充。

然后把新数据存储到2015.csv中。输出文件如下：



# 4：代码部分

## 4.1 houses.py

**import** requests  
**import** parsel  
**import** pandas  
**import** csv  
**import** matplotlib.pyplot **as** plt  
  
*# create outputFile and write header***with** open(**'houses.csv'**, **'w'**, newline=**''**, encoding=**'utf-8'**) **as** outputFile:  
 f\_csv = csv.writer(outputFile)  
 f\_csv.writerow([**'name'**, **'district'**, **'town'**, **'position'**, **'room'**, **'area'**, **'average'**, **'price'**])  
  
*# crawl data from front 25 pages***for** i **in** range(1, 25):  
 *# get data and store into result* selector = parsel.Selector(requests.get(**f"https://bj.fang.lianjia.com/loupan/pg{**i**}/"**).text)  
 result = selector.css(**'.resblock-list.post\_ulog\_exposure\_scroll.has-results'**)  
  
 **for** li **in** result:  
 *# name* name = li.css(**'.resblock-name a::text'**).get()  
 *# location* location = li.css(**'.resblock-location span::text'**).getall()  
 district = location[0]  
 town = location[1]  
 position = li.css(**'.resblock-location a::text'**).get()  
 *# size* room = li.css(**'.resblock-room span::text'**).get()  
 *# area* area = li.css(**'.resblock-area span::text'**).get()  
 **if** area == **None**:  
 **continue** area = area.split(**' '**)  
 area = area[1]  
 area = area.split(**'-'**)  
 **if** len(area) != 1:  
 area = area[0]  
 **else**:  
 area = **""**.join(list(filter(str.isdigit, **""**.join(area))))  
  
 *# average & price* average = li.css(**'.main-price span::text'**).get()  
 priceList = average.split(**'-'**)  
 **if** len(priceList) == 1:  
 price = **f"{**int(average) \* int(area) / 10000**:.4f}"  
 else**:  
 price = priceList[0]  
 average = int(price) \* 10000 / int(area)  
 agerage = int(average)  
  
 *# write this record into outputFile* **with** open(**'houses.csv'**, **'a'**, newline=**''**, encoding=**'utf-8'**) **as** f:  
 f\_csv = csv.writer(f)  
 f\_csv.writerow([name, district, town, position, room, area, average, price])  
  
  
*# analyze data statistics*pandas.set\_option(**'display.max\_rows'**, **None**)  
pandas.set\_option(**'display.max\_columns'**, **None**)  
data = pandas.read\_csv(**'houses.csv'**, encoding=**'utf-8'**)  
  
*# price*print(**"\nPrice most expensive:\n"**,data[data[**'price'**] == data[**'price'**].max()], **'\n'**)  
print(**"Price most cheap:\n"**,data[data[**'price'**] == data[**'price'**].min()], **'\n'**)  
print(**"Price middle:\n"**,data[**'price'**].median())  
  
*# average*print(**"\nAverage most expensive:\n"**,data[data[**'average'**] == data[**'average'**].max()], **'\n'**)  
print(**"Average most cheap:\n"**,data[data[**'average'**] == data[**'average'**].min()], **'\n'**)  
print(**"Average middle:\n"**,data[**'average'**].median())  
  
*# price beyond 3\*average*print(**"\nThose price beyond 3\*average:"**)  
mean = data[**'price'**].mean()  
std = data[**'price'**].std()  
low\_border = mean-3\*std  
high\_border = mean+3\*std  
print(data[(data[**'price'**] > high\_border) | (data[**'price'**] < low\_border)])  
  
*# Average exception*print(**"\nThose average exception:"**)  
plt.rcParams[**'font.sans-serif'**] = [**'Kaitt'**, **'SimHei'**]  
data.boxplot([**'average'**])  
plt.show()  
q1 = data[**'average'**].quantile(q=0.25)  
q3 = data[**'average'**].quantile(q=0.75)  
low\_limit = q1-1.5\*(q3-q1)  
high\_limit = q3+1.5\*(q3-q1)  
print(data[(data[**'average'**] > high\_limit) | (data[**'average'**] < low\_limit)])  
  
*# discretization processing*avgMax=data[**'average'**].max()  
avgMin=data[**'average'**].min()  
num=int(avgMax/avgMin)  
cuts = pandas.cut(data[**'average'**], num)  
print(pandas.value\_counts(cuts).sort\_index())  
value\_list=[]  
**for** i **in** pandas.value\_counts(cuts):  
 value\_list.append(i)  
plt.figure()  
index=pandas.value\_counts(cuts).index  
plt.pie(value\_list,labels=index,autopct=**'%0.2f%%'**)  
plt.show()

## 4.2 BJPM.py

**import** pandas  
  
*# read data*pandas.set\_option(**'display.max\_rows'**,**None**)  
pandas.set\_option(**'display.max\_columns'**,**None**)  
data = pandas.read\_csv(**'BeijingPM20100101\_20151231.csv'**)  
  
*# select 2015 into 2015.csv*data\_2015 = data[data[**'year'**]==2015]  
data\_2015.to\_csv(**'2015\_tmp.csv'**, index=**None**)  
  
*# select null record*data\_2015 = pandas.read\_csv(**'2015\_tmp.csv'**)  
print(data\_2015.isnull().sum())  
  
*# solve null*new\_data=data\_2015.dropna(subset=[**'Iws'**]).fillna(method=**'pad'**, axis=0)  
new\_data.to\_csv(**'2015.csv'**, index=**None**)  
print(new\_data.isnull().sum())