北京郵電大學

Python 数据处理实验

姓名：江姝潼

班级：2019211314

学号：2019211653

学院：计算机学院

专业：网络工程

目录

一、 实验要求

1.爬取并存储链家的新房数据：

https://bj.fang.lianjia.com/loupan/

并对爬取的数据进行预处理，包括数据统计、异常值处理、离散化处理等操作。

2. 分析处理 2015 年北京市 PM2.5 指数数据集空值

从原始数据集中抽取 2015 年度数据，存储为新的 csv 文件，找出存在的空值列及相应的空值数量，对所有存在空值的列，给出空值的处理方法及理由，完成后给出新的空值列信息，并将处理后的数据存储为新的 csv 文件。

二、 链家

1、爬取链家新房数据与存储

本次爬虫实验使用 scrapy 的爬虫框架，具体流程包括“引擎”索要 URL、调度器调度完后将第一个 URL 出队列返回给引擎、引擎经由下载器中间件将该 URL 交给下载器去下载 response 对象、下载器得到响应对象后，将响应结果交给引擎、再经由蜘蛛中间件将响应结果交给爬虫文件、爬虫文件对响应结果进行处理分析，提取出所需要的数据。

爬虫实验将按照上述过程展开：

1.1 items.py

首先在 items.py 中定义一个 LianjiaspiderItem 类，用来存储爬取的数据，根据需求定义了房屋名称、地区、街道、路、面积、总价、均价等几个变量。

```
import scrapy

class LianjiaspiderItem(scrapy.Item):
    name = scrapy.Field()        #房屋名称
    zone = scrapy.Field()        #地区
    street = scrapy.Field()      #街道
    road = scrapy.Field()        #路
    Room_Type = scrapy.Field()   #房型
    area = scrapy.Field()        #面积
```

```
    tot_price = scrapy.Field()   #总价
    avg_price = scrapy.Field()   #均价
    pass
```

1.2 spider.py

下面编写爬虫过程的主要代码：

首先定义爬虫得三个强制属性：

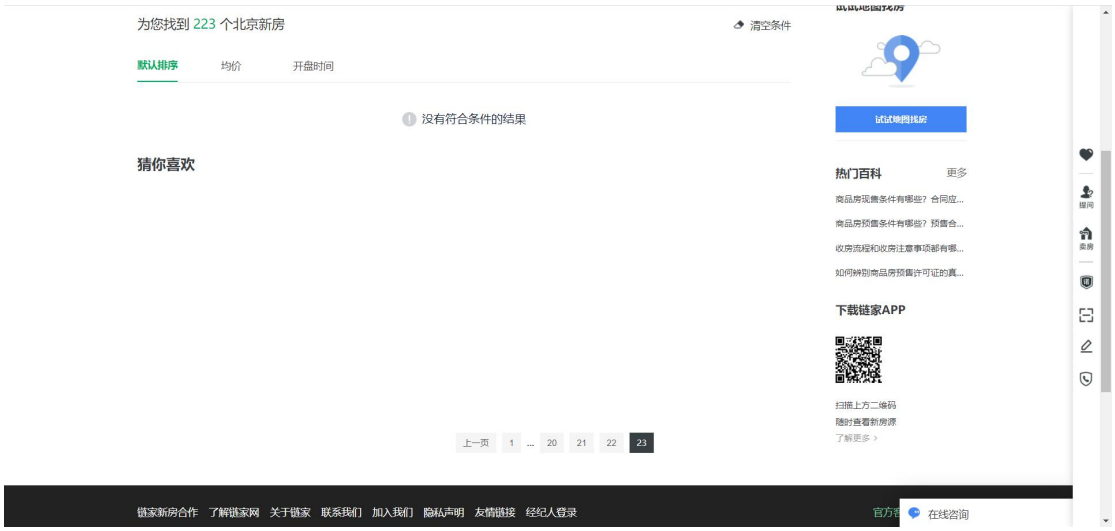
name = "" ：这个爬虫的识别名称，必须唯一。

allow_domains = [] 是搜索的域名范围，也就是爬虫的约束区域，规定爬虫只爬取这个域名下的网页，本题中为链家域名“bj.fang.lianjia.com”。

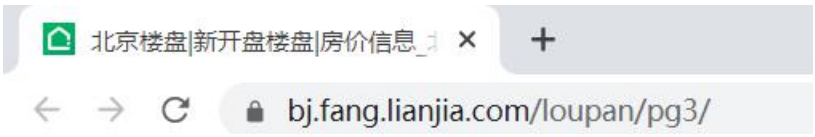
start_urls = () ：为要爬取的 URL 列表。爬虫从这里开始抓取数据。

```
class LianjiaSpdier(scrapy.Spider):
    name = 'lianjia'
    allowed_domains = ['bj.fang.lianjia.com']
    start_urls = ['https://bj.fang.lianjia.com/loupan/']
```

链家新房页面有 23 面，但是手动翻页到后面发现是空的，真实有效的有数据的页面仅有 19 面。



观察各个页面的 url 可知，每到对应的页面后面都会加上 pg+页号，可以用这种方式获取要爬的 19 页的 url。



此处编写了一个 start_requests 函数代替原本框架的 start_requests 函数，并在该函数中给 start_urls 中每一个 url 分别加上 pg1, pg2, …, pg5，并返回保存给 start_urls，并对每一个 url 提交一个 Request 爬虫。

```
    def start_requests(self):
        basic_urls = self.start_urls
        self.start_urls = []
        for urls in basic_urls:
            for i in range(1, 19):
                new_url = urls + "pg" + str(i) + '/'
                self.start_urls.append(new_url)
        for url in self.start_urls:
            yield Request(url, dont_filter=True)
```

下面编写对爬取的 response 进行信息获取操作，即编写 parse 函数。

先在初始网页中找到要爬取的数据的位置，并对应地复制它的 xpath，以同样的方法找到房屋名称、地区、街道、路、面积、总价、均价所在的位置，并把这些信息保存在之前定义的 LianjiaItem 类中，下一步提交给 pipeline 处理。

但是在看网络上楼盘信息的排布，可以发现均价和总价比较特殊，存在在同一个位置有时候是均价，有时却是总价的情况。所以在爬取这部分数据时候，需要特判，并给出不同的操作。

在存储数据的时候，需要将数据处理后再存储。如果红字部分是均价的话，则用面积*均价算出总价；如果是总价的话，需要总价/面积来得出均价。总价和均价转换的时候要注意单位的改变，同时总价要保留小数点后 4 位，具体代码如下：

```
  def parse(self, response):
        item = LianjiaspiderItem()
        for each in response.xpath('//ul[@class="resblock-list-
wrapper"]/*'):
            item.clear()
            item['name'] = each.xpath("div/div[1]/a/text()").get()
            if item['name'] != None:
                item['name'] = item['name'].strip()
            item['zone'] =
each.xpath("div/div[2]/span[1]/text()").get()
            if item['zone'] != None:
                item['zone'] = item['zone'].strip()
            item['street'] =
each.xpath("div/div[2]/span[2]/text()").get()
            if item['street'] != None:
                item['street'] = item['street'].strip()
            item['road'] = each.xpath("div/div[2]/a/text()").get()
            if item['road'] != None:
                item['road'] = item['road'].strip()
            #span[1]表示最小房型
            item['Room_Type'] =
each.xpath("div/a/span[1]/text()").get()
            if (item['Room_Type'] != None):
                item['Room_Type'] = item['Room_Type'].strip()
            #注意后面的情况可能有多种可能，注意特判处理
            item['area'] = each.xpath("div/div[3]/span/text()").get()
            if item['area'] != None:
                item['area'] = int(item['area'].split('
')[1].split('-')[0].strip())
            temp =
each.xpath("div/div[6]/div[1]/span[2]/text()").get()
            #如果那个位置是均价：
            if temp.find('均价') != -1:
                item['avg_price'] =
int(each.xpath("div/div[6]/div[1]/span[1]/text()").get().strip())
                item['tot_price'] = format(item['area'] *
item['avg_price']/10000, '4f')
            #如果那个位置是总价：
            else:
                item['tot_price'] =
each.xpath("div/div[6]/div[1]/span[1]/text()").get()
                item['tot_price'] = format(item['tot_price'].split('-
')[0].strip(), '.4f')
```

```
                item['avg_price'] = int(item['tot_price'] * 10000 /
item['area'])
            if item['name'] != None:
                yield (item)
```

这里我还多做了一步处理，如果爬取的内容是空的话则不改变，如果不是空则用 strip 去除前后空格，方便后续的数据处理。

1.3 middlewares.py

为了防止被反爬并获取网页的动态内容，这里采用了使用 webdriver 动态打开网页的方式实现，打开选项增添—headless 属性来使得网页不弹出，并设置了一个 time.sleep，让每次切换页面时都等待 5 秒，给页面有充足的加载时间，但对应的爬取速度由于等待也变慢了。

```
    def process_request(self, request, spider):
        option = webdriver.ChromeOptions()
        option.add_argument('--headless')
        driver = webdriver.Chrome(chrome_options=option)
        driver.get(request.url)
        driver.implicitly_wait(5)
        time.sleep(5)
        html = driver.page_source
        driver.quit()
        return HtmlResponse(url=request.url, body=html,
request=request, encoding='utf-8')
```

1.4 pipelines.py

在 pipelines.py 文件中定义对数据的处理方法，此处定义 open_spider、process_item 和 close_spider 三个函数。

在本实验中，需要将爬取的数据存在 csv 文件中，故在 open_spider 函数中打开创建 newdata.csv 文件，如果打开失败则报错。在 process_item 函数中，使用 writerow 函数将每个 item 输出一行，全部输出完毕后关闭文件。

Pipelines.py 的具体代码如下：

```
class LianjiaspiderPipeline:
    def open_spider(self, spider):
        try:
            self.file = open('data.csv', 'w', encoding='utf_8_sig')
```

```
            self.result = csv.writer(self.file)
            self.result.writerow(['name', 'zone', 'street', 'road',
'Room_type', 'area', 'avg_price', 'tot_price'])
        except Exception as e:
            print(e)

    def process_item(self, item, spider):
        dict_item = ItemAdapter(item).asdict()
        self.result.writerow(dict_item.values())
        return item

    def close_spider(self, spider):
        self.file.close()
```

1.5 settings.py

在 settings 中启用 LianjiaspiderDownloaderMiddleware 和 LianjiaspiderPipeline，并设置优先级：

```
ROBOTSTXT_OBEY = True

DOWNLOADER_MIDDLEWARES = {
   'LianjiaSpider.middlewares.LianjiaspiderDownloaderMiddleware': 543,
}
ITEM_PIPELINES = {
   'LianjiaSpider.pipelines.LianjiaspiderPipeline': 300,
}
name,zone,street,road,Room_type,area,avg_price,tot_price
```

输入“scrapy crawl lianjia”运行爬虫，待运行完毕后，出现了一个 data.csv 文件，截取部分运行结果如下：

```
奥森春晓,昌平,回龙观,北京市昌平区文华路东侧约 150 米,2 室,96,63000,604.800000

水岸壹号,房山,良乡,良乡大学城西站地铁南侧 800 米，刺猬河旁,3
室,185,58000,1073.000000

檀香府,门头沟,门头沟其它,京潭大街与潭柘十街交叉口,3 室,208,45000,936.000000

韩建·观山源墅,房山,良乡,阳光北大街与多宝路交汇处西南（理工大学北校区西侧）,3
室,290,40000,1160.000000

都丽华府,平谷,平谷其它,新平南路与林荫南街交汇处向西 100 米,2
室,86,29000,249.400000
```

中粮京西祥云,房山,长阳,地铁稻田站北 800 米，西邻京深路,4
室,115,58000,667.000000

燕西华府,丰台,丰台其它,"王佐镇青龙湖公园东 1500 米,",4 室,60,42000,252.000000

水岸壹号,房山,良乡,良乡大学城西站地铁南侧 800 米，刺猬河旁,3
室,122,43000,524.600000

紫辰院,丰台,岳各庄,岳各庄北桥东北角 200 米处,5 室,266,128000,3404.800000

鲁能格拉斯小镇,通州,通州其它,北京市通州区宋庄镇格拉斯小镇营销中心,3
室,246,60000,1476.000000

兴创荣墅,大兴,大兴新机场洋房别墅区,北京市大兴区育胜街,3
室,240,23000,552.000000

温哥华森林,昌平,北七家,"北五环外紧邻立汤路，北七家建材城向北第一个路口 200 米路东,
枫树家园 6 区， 枫树家园五区",4 室,460,43478,1999.988000

润泽御府,朝阳,北苑,北京市朝阳区北五环顾家庄桥向北约 2.6 公里,4
室,540,110000,5940.000000

中骏西山天璟,门头沟,城子,西山永定楼北 300 米,4 室,117,65000,760.500000

凯德麓语,昌平,昌平其它,兴寿镇京承高速 G11 出口向西怀昌路北侧,3
室,280,35000,980.000000

京贸国际城·峰景,通州,武夷花园,芙蓉东路 1 号（通燕高速耿庄桥北出口向南 300 米）,1
室,69,68000,469.200000

观唐云鼎,密云,溪翁庄镇,溪翁庄镇密溪路 39 号院（云佛山度假村对面）,3
室,357,30000,1071.000000

尊悦光华,朝阳,CBD,北京市朝阳区光华东里甲 1 号院 3 号楼,3
室,133,130000,1729.000000

旭辉城,房山,房山其它,北京市房山区良锦街 6 号院旭辉城营销中心,2
室,75,28500,213.750000

檀香府,门头沟,门头沟其它,京潭大街与潭柘十街交叉口,3 室,124,43000,533.200000

中海丽春湖墅·合院,昌平,沙河,地铁昌平线沙河地铁站南 600 米,4
室,263,36000,946.800000

```
中粮天恒天悦壹号,丰台,西红门,南四环地铁新宫站南 500 米,4
室,220,85000,1870.000000

天竺悦府,顺义,天竺,北京市顺义区府前一街 38 号,3 室,125,58000,725.000000

新潮嘉园二期,通州,潞苑,潞苑南大街 185 号,1 室,65,58000,377.000000

泰禾金府大院,丰台,西红门,南四环地铁新宫站南 800 米,4 室,362,75000,2715.000000

电建地产溶悦苑,丰台,马家堡,南四环中路 115 号,3 室,89,63000,560.700000
```

2、数据处理

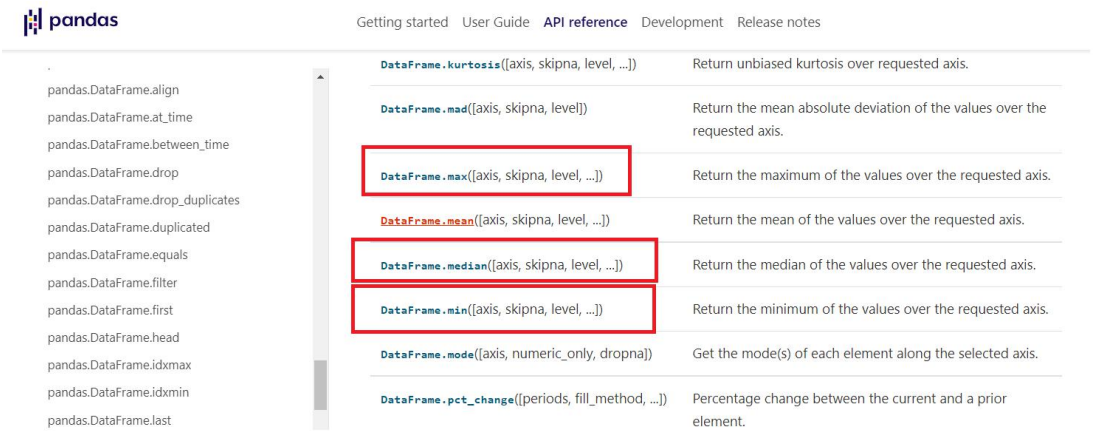
下面新建 dataProcessing.py 文件对爬取下来的 csv 文件里的数据处理:

首先打开新爬取的 csv 文件，并将其保存在 data 向量中，并设置对齐，方便后续操作。

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

data = pd.read_csv('data.csv', encoding='utf_8_sig')
pd.set_option('display.unicode.east_asian_width', True)
```

- 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，以及单价的中位数

第一步需要对某一特定的列的所有值统计计算，针对上述要求，查阅官方文档，获得信息如下：



可见，可以使用 max 函数找到最大值，用 min 函数找到最小值，并用 median 函数找到中位数，并对应输出。代码如下：

```
#找出总价最贵和最便宜的房子，以及总价的中位数
max_tot_price = pd.DataFrame.max(data["tot_price"])
min_tot_price = pd.DataFrame.min(data["tot_price"])
median_tot_price = pd.DataFrame.median(data["tot_price"])
print("for total price of houses:")
print("highest price:", max_tot_price)
print("lowest price:", min_tot_price)
print("median price:", median_tot_price)

#找出均价最贵和最便宜的房子，以及单价的中位数
max_avg_price = pd.DataFrame.max(data["avg_price"])
min_avg_price = pd.DataFrame.min(data["avg_price"])
median_avg_price = pd.DataFrame.median(data["avg_price"])
print("for average price of houses:")
print("highest price:", max_avg_price)
print("lowest price:", min_avg_price)
print("median price:", int(median_avg_price))
```

运行出的结果如下：

```
for total price of houses:
highest price: 5940.0
lowest price: 116.6
median price: 532.5
for average price of houses:
highest price: 130000
lowest price: 15000
median price: 43000
1024.4663060701018
```

2、异常值处理

2.1 列出总价在均值三倍标准差以外的房屋，展示其基本信息，并分析其原因

根据统计学知识可以知道，在正态分布中σ代表标准差，μ代表均值。数值分布在（μ-3σ，μ+3σ）中的概率为0.9974，超出这个范围的可能性仅占不到0.3%，如果一组测量数据中某个测量值的残余误差的绝对值＞3σ，则可以判定该测量值为异常值，应剔除。应用统计学原理可以筛除部分差异过大的数据，让数据处理后展现的结果更为直观。

首先需要找到求标准差的函数，查阅 pandas 官方文档：

DataFrame.std([axis, skipna, level, ddof, ...])　　Return sample standard deviation over requested axis.

然后按条件筛选出数据行，具体代码如下：

```
#找到总价在均值三倍标准差以外的房屋
standard_deviation = pd.DataFrame.std(data["tot_price"])
print(standard_deviation)

#列出总价在均值三倍标准差以外的房屋
standard = 3 * standard_deviation
a = data.loc[data['tot_price'] > standard]
print("Outliers of standard deviation:")
print(a)
```

运行出的结果如下：

```
1024.4663060701018
Outliers of standard deviation:
           name  zone      street                                  road Room_type  area  avg_price  tot_price
8        紫辰院   丰台        岳各庄                  岳各庄北桥东北角200米处        5室   266     128000     3404.8
12     润泽御府   朝阳          北苑  北京市朝阳区北五环顾家庄桥向北约2.6公里        4室   540     110000     5940.0
45     懋源·璟玺  朝阳  中央别墅区      孙河京密路与京平辅路交叉口西行1000米        4室   555      86000     4773.0
67  葛洲坝中国府   丰台        玉泉营                      丰台东路46号        4室   390     115000     4485.0
```

从上面可以看出，总价的标准差约为 1024，而上述房屋的总价都超过了 3000 万，观察各条目的详细信息，可以看出，这些房屋建筑面积都很大，尤其是朝阳区的两座别墅区，面积都超过了 500 平米，每平米的均价也不便宜，远超均价的中位数 43000。可以得知，上述值并不是因为异常而出现的，而是因为存在某些房屋面积大、均价高，进而导致总价远超其他项，但这些条目数量也是非常少的，仅为个别极端情况。

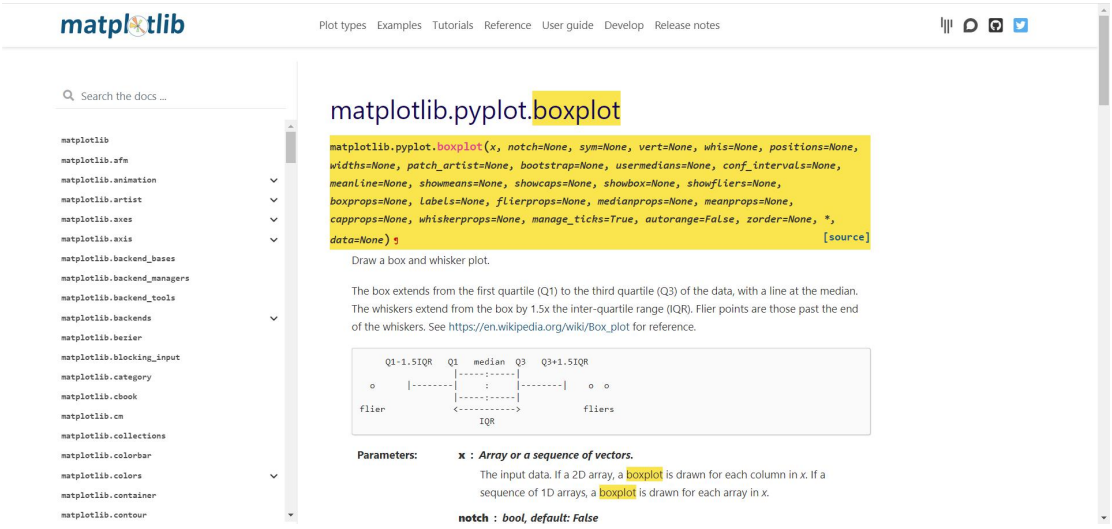
2.2 通过箱型图原则判断并列出均价为异常值的房屋，展示其基本信息，并分析其原因

箱线图也称箱须图，是利用数据中的五个统计量：最小值、第一四分位数、中位数、第三四分位数与最大值来描述数据的一种方法，它也可以粗略地看出数据是否具有有对称性，分布的分散程度等信息。具体含义如下：

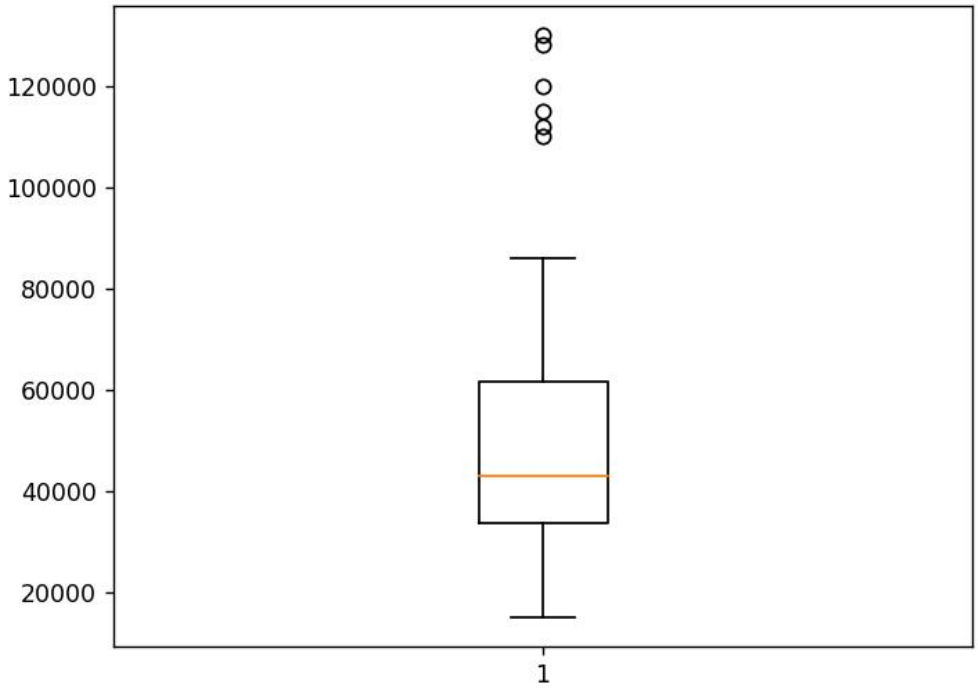
首先计算出第一四分位数（Q1）、中位数、第三四分位数（Q3）。 在 Q3＋1.5IQR 和 Q1-1.5IQR 处画两条与中位线一样的线段，这两条线段为异常值截断点，称其为内限；在 Q3＋3IQR 和 Q1－3IQR 处画两条线段，称其为外限。处于

内限以外位置的点表示的数据都是异常值，其中在内限与外限之间的异常值为温和的异常值，在外限以外的为极端的的异常值。

本次实验需要我们找出极端的异常值，通过查阅相关文档，可以发现，matplotlib 库中有很直观的 boxplot 的绘图函数：



绘制的 boxplot 图片如下：



可以看出，该项数据存在异常值，但是图片只能看出存在，无法找到有哪些数据，需要手动找出划分界限并筛选出异常数据，代码如下：

```
#通过箱型图原则判断并列出均价为异常值的房屋
plt.boxplot(x=data['avg_price'])
plt.show()
q1 = data['avg_price'].quantile(q=0.25)
q3 = data['avg_price'].quantile(q=0.75)
q2 = data['avg_price'].quantile(q=0.5)
low_limit = q1 - 1.5 * (q3-q1)
high_limit = q3 + 1.5 * (q3-q1)
val = data.loc[(data['avg_price'] > high_limit)|(data['avg_price'] <
low_limit)]
print("Outliers of box plot: ")
print(val)
```

运行出的结果如下：

```
Outliers of box plot:
           name  zone  street                                road Room_type  area  avg_price  tot_price
8          紫辰院   丰台   岳各庄                岳各庄北桥东北角200米处        5室   266     128000     3404.8
12      润泽御府   朝阳    北苑  北京市朝阳区北五环顾家庄桥向北约2.6公里        4室   540     110000     5940.0
17      尊悦光华   朝阳     CBD        北京市朝阳区光华东里甲1号院3号楼        3室   133     130000     1729.0
36  葛洲坝中国府   丰台   玉泉营                      丰台东路46号        2室   168     112000     1881.6
59  世茂北京天誉   丰台   十里河               北京市丰台区小红门路312号        3室   145     120000     1740.0
67  葛洲坝中国府   丰台   玉泉营                      丰台东路46号        4室   390     115000     4485.0
```

为了判断异常原因，输出 q1,q2,q3,low_limit,high_limit 如下：

```
33500.0 43000.0 61500.0 -8500.0 103500.0
```

可以看出，每平米的均值约为 43000 元，外限的上界为 103500 元，但是上述房屋的均价都超出了 110000 元。这些数据之所以异常，是因为存在部分房屋处于热门地段，价格较高。

3、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

要给数据实现离散化，查阅官方文档，由下可知，可以使用 cut 函数来实现，其中 Bin 值表示为离散的间隔，该函数可以预先指定的存储箱阵列进行存储。

首先需要确定划分区间，但是数据条目较多，怎么划分更合理并不直观，所以先用最简单的方式划分，以每一万为界限，查看分布情况：

```
#离散化处理
bins = [0, 10000, 20000, 30000, 40000, 50000, 60000, 70000, 80000, 90000, 100000]
```

运行出的结果如下：

```
(30000, 40000]     0.220779
(20000, 30000]     0.194805
(50000, 60000]     0.194805
(40000, 50000]     0.155844
(60000, 70000]     0.103896
(70000, 80000]     0.051948
(10000, 20000]     0.038961
(80000, 90000]     0.038961
(0, 10000]         0.000000
(90000, 100000]    0.000000
Name: avg_price, dtype: float64
```

可以看出，一万以下和九万以上都没有对应的房屋，可以确定所有房屋的均价都分布在一万到九万之间。其中，均价超过五万的房屋在每一万的去年内都分布较少，不超过 10%，可以考虑将区间合并。

最终将划分标准确定为[0, 20000, 40000, 60000, 90000]，每个区间都有代表含义，[0, 20000]代表低档房屋，[20000, 40000]代表中档房屋，[40000, 60000]代表中高档房屋，[60000, 90000]代表高档房屋。具体代码如下：

```
#离散化处理
```

```
#bins = [0, 10000, 20000, 30000, 40000, 50000, 60000, 70000, 80000,
90000, 100000]
bins = [0,20000,40000,60000,90000]
cuts = pd.cut(data['avg_price'], bins)
print(pd.value_counts(cuts))
print(pd.value_counts(cuts, normalize=True))
```

运行出的每个区间的房屋数量和所占比例结果如下：

```
(20000, 40000]    32
(40000, 60000]    27
(60000, 90000]    15
(0, 20000]         3
Name: avg_price, dtype: int64
(20000, 40000]    0.415584
(40000, 60000]    0.350649
(60000, 90000]    0.194805
(0, 20000]        0.038961
Name: avg_price, dtype: float64
```

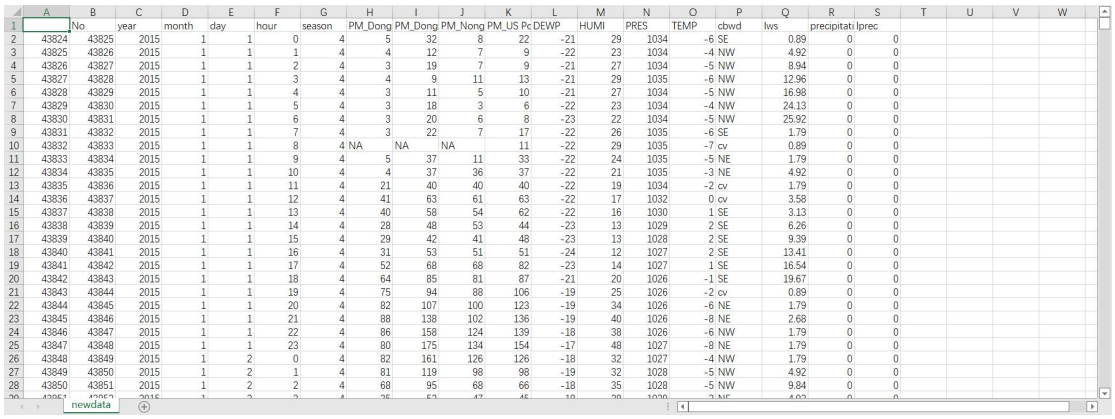
三、 分析 2015 年北京市 PM2.5 指数数据集空值

1、 数据抽取

首先需要从原始数据集中抽取 2015 年度数据，存储为新的 csv 文件，该项操作较为简单，只需要用 index 选择年份，筛选出符合条件的对应条目即可，并保存在新的 csv 文件中：

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

#找出 2015 年的数据，保存在新的 csv 文件中
data = pd.read_csv('BeijingPM20100101_20151231.csv',
encoding='utf_8_sig')
pd.set_option('display.unicode.east_asian_width', True)

data2015 = data.loc[data['year'] == 2015]
data2015.to_csv("newdata.csv", na_rep='NA')
```

打开新的 csv 文件如下，可以看出，下列数据只含 2015 年的数据：



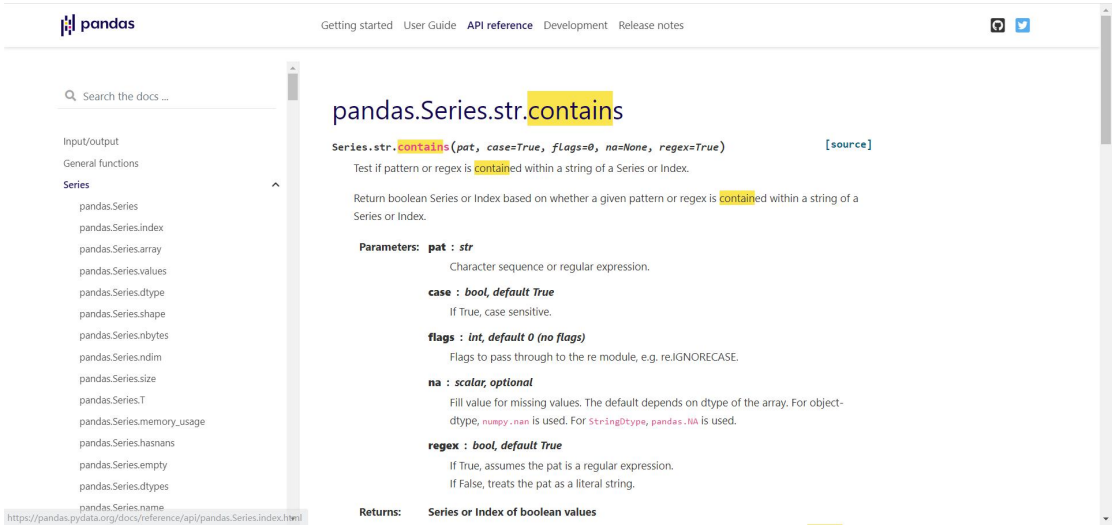
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1		No	year	month	day	hour	season	PM_Dong	PM_Dong	PM_Nong	PM_US Pc	DEWP	HUMI	PRES	TEMP	cbwd	lws	precipitati	Iprec				
2	43824	43825	2015	1	1	0	4	5	32	8	22	-21	29	1034	-6	SE	0.89	0	0				
3	43825	43826	2015	1	1	1	4	4	12	7	9	-22	23	1034	-4	NW	4.92	0	0				
4	43826	43827	2015	1	1	2	4	3	19	7	9	-21	27	1034	-5	NW	8.94	0	0				
5	43827	43828	2015	1	1	3	4	4	9	11	13	-21	29	1035	-6	NW	12.96	0	0				
6	43828	43829	2015	1	1	4	4	3	11	5	10	-21	27	1034	-5	NW	16.98	0	0				
7	43829	43830	2015	1	1	5	4	3	18	3	6	-22	23	1034	-4	NW	24.13	0	0				
8	43830	43831	2015	1	1	6	4	3	20	6	8	-23	22	1034	-5	NW	25.92	0	0				
9	43831	43832	2015	1	1	7	4	3	22	7	17	-22	26	1035	-6	SE	1.79	0	0				
10	43832	43833	2015	1	1	8	4	NA	NA	NA	11	-22	29	1035	-7	cv	0.89	0	0				
11	43833	43834	2015	1	1	9	4	5	37	11	33	-22	24	1035	-5	NE	1.79	0	0				
12	43834	43835	2015	1	1	10	4	4	37	36	37	-22	21	1035	-3	NE	4.92	0	0				
13	43835	43836	2015	1	1	11	4	21	40	40	40	-22	19	1034	-2	cv	1.79	0	0				
14	43836	43837	2015	1	1	12	4	41	63	61	63	-22	17	1032	0	cv	3.58	0	0				
15	43837	43838	2015	1	1	13	4	40	58	54	62	-22	16	1030	1	SE	3.13	0	0				
16	43838	43839	2015	1	1	14	4	28	48	53	44	-23	13	1029	2	SE	6.26	0	0				
17	43839	43840	2015	1	1	15	4	29	42	41	48	-23	13	1028	2	SE	9.39	0	0				
18	43840	43841	2015	1	1	16	4	31	53	51	51	-24	12	1027	2	SE	13.41	0	0				
19	43841	43842	2015	1	1	17	4	52	68	68	82	-23	14	1027	1	SE	16.54	0	0				
20	43842	43843	2015	1	1	18	4	64	85	81	87	-21	20	1026	-1	SE	19.67	0	0				
21	43843	43844	2015	1	1	19	4	75	94	88	106	-19	25	1026	-2	cv	0.89	0	0				
22	43844	43845	2015	1	1	20	4	82	107	100	123	-19	34	1026	-6	NE	1.79	0	0				
23	43845	43846	2015	1	1	21	4	88	138	102	136	-19	40	1026	-8	NE	2.68	0	0				
24	43846	43847	2015	1	1	22	4	86	158	124	139	-18	38	1026	-6	NW	1.79	0	0				
25	43847	43848	2015	1	1	23	4	80	175	134	154	-17	48	1027	-8	NE	1.79	0	0				
26	43848	43849	2015	1	2	0	4	82	161	126	126	-18	32	1027	-4	NW	1.79	0	0				
27	43849	43850	2015	1	2	1	4	81	119	98	98	-19	32	1028	-5	NW	4.92	0	0				
28	43850	43851	2015	1	2	2	4	68	95	68	66	-18	35	1028	-5	NW	9.84	0	0				

newdata

2、 找到空值并处理

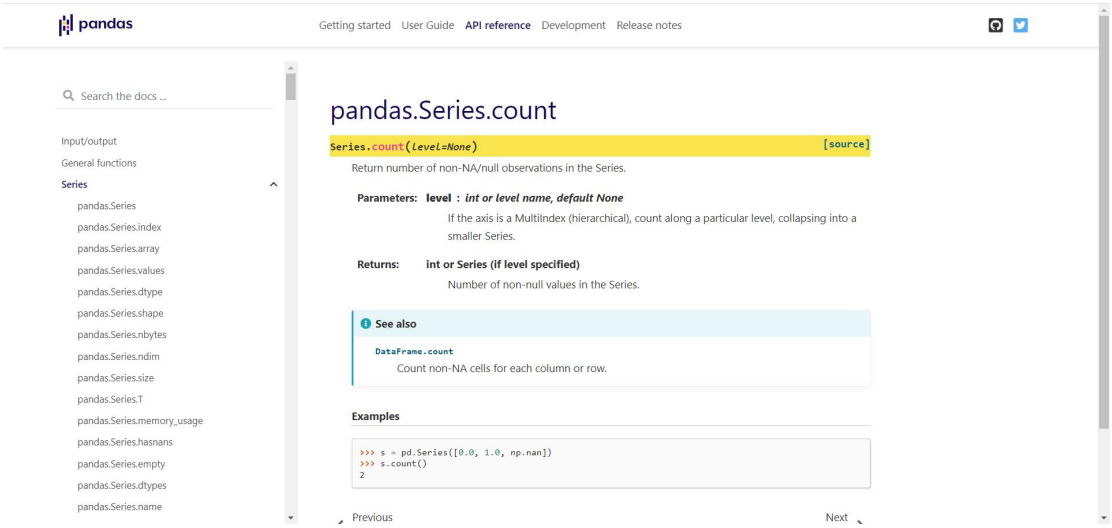
- 对新的 csv 文件，找出存在的空值列及相应的空值数量

该问题让我思考了很久，原因是这个要求不是对某一列的数据来判断，而是对所有列都查找是否存在空的情况。通过查阅相关资料，可以用 str.contains 函数来筛选出含“NA”的项目，文档说明如下：

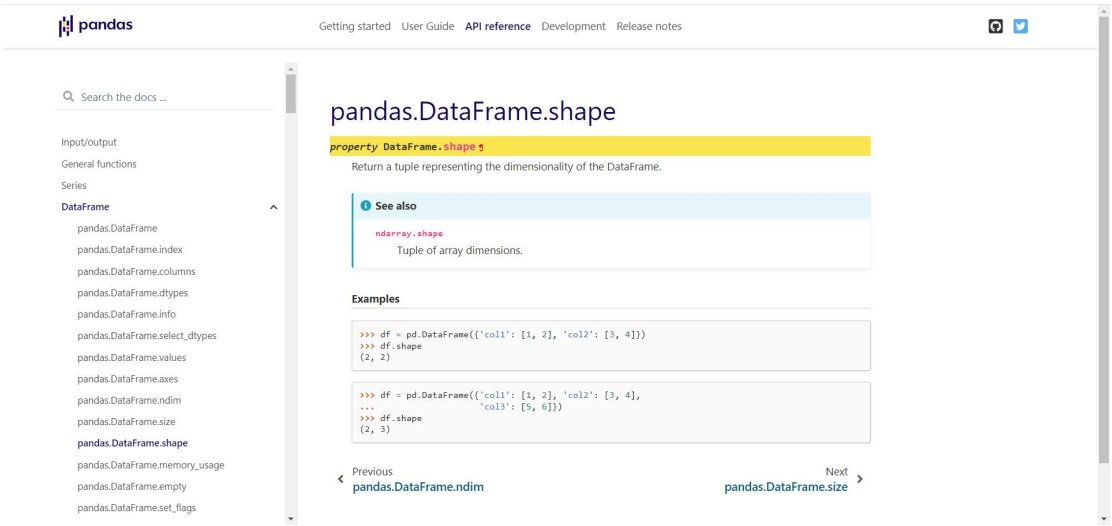


但是该函数只能对 series 对象操作，可以选择遍历每列并类型转换为 series 来判断，但是该操作比较繁琐，而且涉及到遍历，在列数较少的时候还比较方便操作，但是若列数较多，该方法的缺陷就愈加明显，于是我想用更加优美简单的方法去实现。

通过查阅相关文档发现，pandas 存在 count 函数来数出某一列的非空项目，如果知道一列所有项目的数量，将所有项目的数量减去非空项目，若为 0，则不含空项。



以此为思路，首先要获取列表的大小，发现该功能可以用 shape 函数实现。



运行出的结果如下：

```
(8760, 19)
```

实现代码如下，分别输出 count 非空项的数量，和相减后含空项的数量。

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

data = pd.read_csv('newdata.csv', encoding='utf_8_sig')
pd.set_option('display.unicode.east_asian_width', True)

#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
print(data.shape)
print(data.count(axis=0))
print([data.shape[0]] - data.count(axis=0))
```

```
emptylines = [data.shape[0]] - data.count(axis=0)
```

运行出的结果如下：

```
Unnamed: 0          8760
No                  8760
year                8760
month               8760
day                 8760
hour                8760
season              8760
PM_Dongsi           8596
PM_Dongsihuan       5465
PM_Nongzhanguan     8473
PM_US Post          8631
DEWP                8755
HUMI                8421
PRES                8421
TEMP                8755
cbwd                8755
Iws                 8755
precipitation       8301
Iprec               8301
dtype: int64
```

```
dtype: int64
Unnamed: 0            0
No                    0
year                  0
month                 0
day                   0
hour                  0
season                0
PM_Dongsi           164
PM_Dongsihuan      3295
PM_Nongzhanguan     287
PM_US Post          129
DEWP                  5
HUMI                339
PRES                339
TEMP                  5
cbwd                  5
Iws                   5
precipitation       459
Iprec               459
dtype: int64
```

通过结果可以看出，除了年月日季节这些基本的时间信息，其他列均存在含有空项的情况，其中 DEWP、TEMP、cbwd、Iws 这四项缺失项目较少，仅有 5 个空项；PM_Dongsi,PM_Nongzhanguan,OM_USPost,HUMI,PRES,precipitation,Iprec,这些项目有部分缺失，在有 8760 项的一列数据里，这些项目缺失的数量仅几百，并不会有太大的影响。但是 PM_Dongsihuan 这一列缺失数据过多，缺失了将近一半的数据。

方法二：

经查阅资料发现，还有更简单直观的方法，pandas 直接提供了 isnull()函数来判断是否为空，再用 sum（）计数即可，需要对每列遍历输出一下即可：

```
for item in data.columns:
    print(item, data[item].isnull().sum())
```

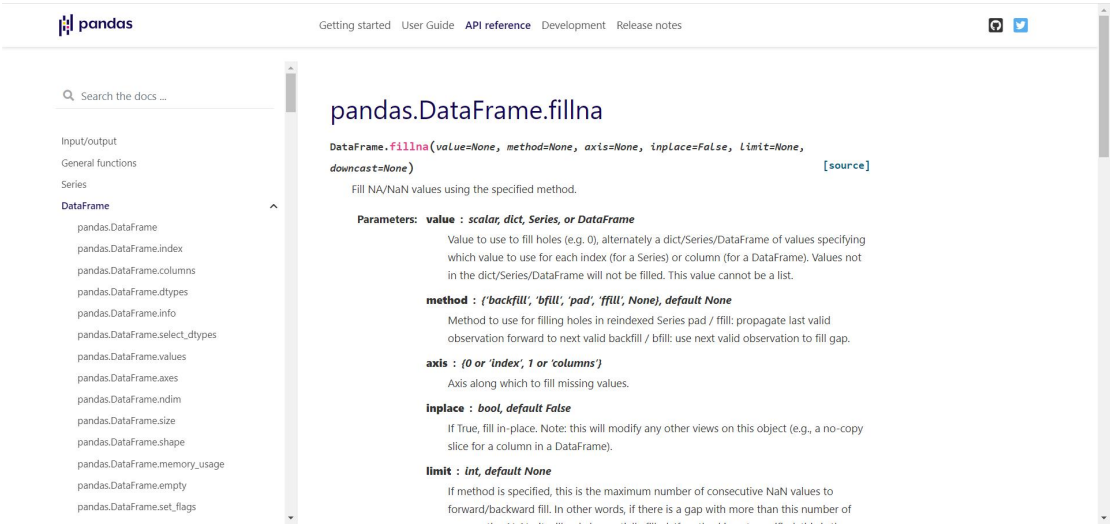
```
Unnamed: 0 0
No 0
year 0
month 0
day 0
hour 0
season 0
PM_Dongsi 164
PM_Dongsihuan 3295
PM_Nongzhanguan 287
PM_US Post 129
DEWP 5
HUMI 339
PRES 339
TEMP 5
cbwd 5
Iws 5
precipitation 459
Iprec 459
```

运行出的结果是一样的。

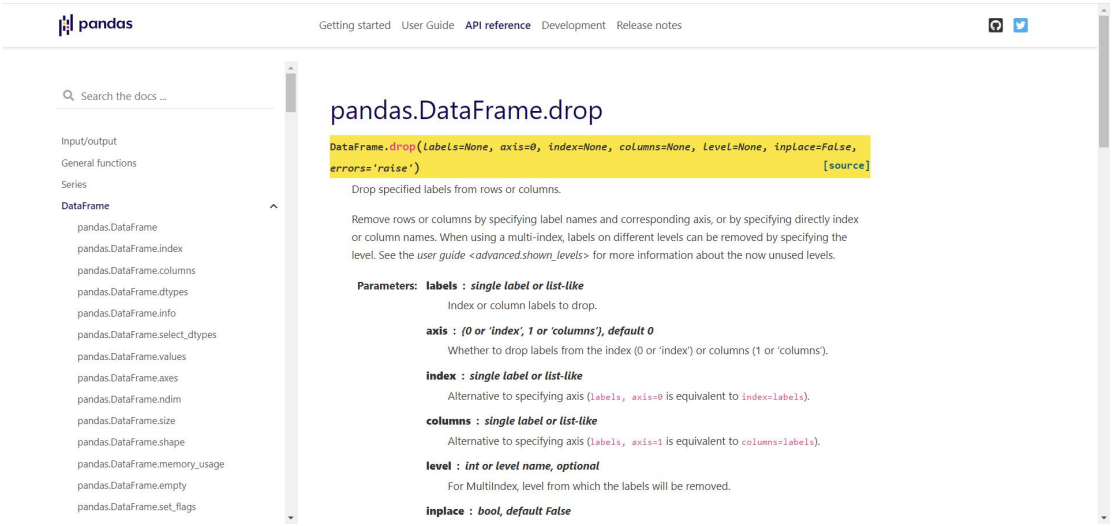
- 空值处理方法：对所有存在空值的列，给出空值的处理方法及理由，要求处理方法必须可在本数据集范围内执行

对于含有缺项的列，需要根据情况来分类讨论如何处理。

PM_Dongsi,PM_Nongzhanguan,OM_USPost,HUMI,PRES,precipitation,Iprec 等项目虽然有缺失数据，但是数量相对总量来说较少。若把空值全部置为 0，在后续统一做数据处理时会有较大的影响，比如影响平均数和方差等；再纵观整个数据表，这些列上下相差的数量并不大，往往是同一个地区前一天还有数据，但是后一天就缺失了。在这种情况下，可以考虑用前面的数据补充在后面的一项中，两天时间相近，数据也应该相差不大。这种填充的方式，用较小的误差实现了对空值的消除，查阅文档发现可以用 fillna()函数实现。



但是 PM_Dongsihuan 这一列缺失数据过多，缺失了将近一半的数据，再用填充的方法会非常不准确，造成有很多缺失数据，所以选择用 drop 函数直接删去该列。



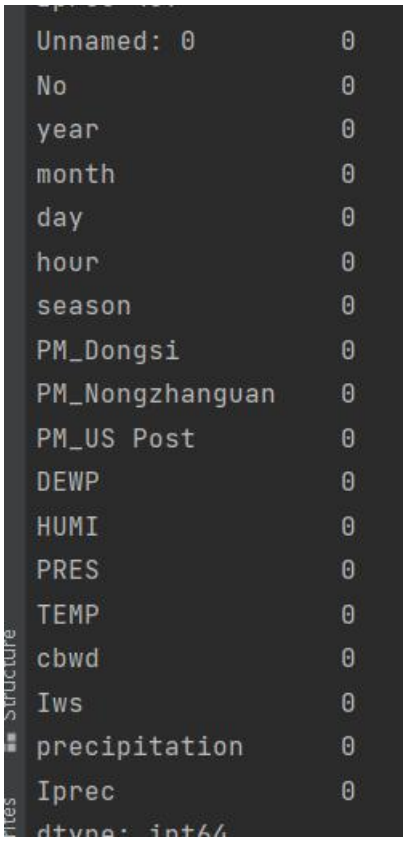
空值处理的代码如下：

```
#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
#东四环空值较多，填充数据则不准确，选择删去该列
newdata = data.drop(columns="PM_Dongsihuan")

#对于其他含空的数据，使用前面一个值进行填充
newdata = newdata.fillna(method="ffill")

#输出成新的 csv 文件
newdata.to_csv("finalresult.csv")
```

```
print([data.shape[0]] - newdata.count(axis=0))
```

处理完后在对空值的项目数量统计，可以发现，原来 PM_Dongsihuan 这一列已经没有了，所有列都无空值。

结果如下：



四、 实验感想

本次实验在基于爬虫的基础上对爬下来的数据做了进一步的处理，包括数据统计、空值处理等内容。

本次实验主要使用了 pandas、matplotlib 等库。方便之处是这些库提供了很多封装好的函数，直接使用起来非常方便，但是麻烦之处也同样在此，函数过多过杂，要使用时不知道要选取哪个。所幸是老师在 ppt 里插入了官方文档的链接，直接在文档里查阅就可以知道所需的信息，文档都是全英文的，一开始阅读起来有些困难，但是慢慢理解再结合文档提供的例子，就能够很好地完成了。

除此之外，本次实验也提升了我数据处理的能力。爬取下来的数据一开始总不是很全面的，故首先要对空值做好处理，可以采用填充、删除的方式，先让数据统一化。之后再对数据做统计展示，包括计算均值、中位数、方差等数据，并用可视化的方式更为直观地展示出来，可以更直观地得到结论。此次数据处理也帮助我更好地体会了从数据到结论的过程，整体收获颇丰。

五、 附录：代码

1、爬取链家代码

items.py

```
# Define here the models for your scraped items
#
# See documentation in:
# https://docs.scrapy.org/en/latest/topics/items.html

import scrapy

class LianjiaspiderItem(scrapy.Item):
    name = scrapy.Field()        #房屋名称
    zone = scrapy.Field()        #地区
    street = scrapy.Field()      #街道
    road = scrapy.Field()        #路
    Room_Type = scrapy.Field()   #房型
    area = scrapy.Field()        #面积
    tot_price = scrapy.Field()   #总价
    avg_price = scrapy.Field()   #均价
    pass
```

spider.py

```
import scrapy
from LianjiaSpider.items import LianjiaspiderItem
from scrapy.http import Request

class LianjiaSpdier(scrapy.Spider):
    name = 'lianjia'
    allowed_domains = ['bj.fang.lianjia.com']
    start_urls = ['https://bj.fang.lianjia.com/loupan/']
```

```
    def start_requests(self):
        basic_urls = self.start_urls
        self.start_urls = []
        for urls in basic_urls:
            for i in range(1, 19):
                new_url = urls + "pg" + str(i) + '/'
                self.start_urls.append(new_url)
        for url in self.start_urls:
            yield Request(url, dont_filter=True)

    def parse(self, response):
        item = LianjiaspiderItem()
        for each in response.xpath('//ul[@class="resblock-list-
wrapper"]/*'):
            item.clear()
            item['name'] = each.xpath("div/div[1]/a/text()").get()
            if item['name'] != None:
                item['name'] = item['name'].strip()
            item['zone'] =
each.xpath("div/div[2]/span[1]/text()").get()
            if item['zone'] != None:
                item['zone'] = item['zone'].strip()
            item['street'] =
each.xpath("div/div[2]/span[2]/text()").get()
            if item['street'] != None:
                item['street'] = item['street'].strip()
            item['road'] = each.xpath("div/div[2]/a/text()").get()
            if item['road'] != None:
                item['road'] = item['road'].strip()
            #span[1]表示最小房型
            item['Room_Type'] =
each.xpath("div/a/span[1]/text()").get()
            if (item['Room_Type'] != None):
                item['Room_Type'] = item['Room_Type'].strip()
            #注意后面的情况可能有多种可能，注意特判处理
            item['area'] = each.xpath("div/div[3]/span/text()").get()
            if item['area'] != None:
                item['area'] = int(item['area'].split('
')[1].split('-')[0].strip())
            temp =
each.xpath("div/div[6]/div[1]/span[2]/text()").get()
            #如果那个位置是均价：
            if temp.find('均价') != -1:
```

```
                item['avg_price'] =
int(each.xpath("div/div[6]/div[1]/span[1]/text()").get().strip())
                item['tot_price'] = format(item['area'] *
item['avg_price']/10000, '4f')
            #如果那个位置是总价:
            else:
                item['tot_price'] =
each.xpath("div/div[6]/div[1]/span[1]/text()").get()
                item['tot_price'] = format(item['tot_price'].split('-
')[0].strip(), '.4f')
                item['avg_price'] = int(item['tot_price'] * 10000 /
item['area'])
            if item['name'] != None:
                yield (item)
```

middlewares.py

```
# Define here the models for your spider middleware
#
# See documentation in:
# https://docs.scrapy.org/en/latest/topics/spider-middleware.html

from scrapy import signals

# useful for handling different item types with a single interface
from itemadapter import is_item, ItemAdapter
from scrapy.http import HtmlResponse
from selenium import webdriver
import time

class LianjiaspiderSpiderMiddleware:
    # Not all methods need to be defined. If a method is not defined,
    # scrapy acts as if the spider middleware does not modify the
    # passed objects.

    @classmethod
    def from_crawler(cls, crawler):
        # This method is used by Scrapy to create your spiders.
        s = cls()
        crawler.signals.connect(s.spider_opened,
signal=signals.spider_opened)
        return s

    def process_spider_input(self, response, spider):
        # Called for each response that goes through the spider
```

```
        # middleware and into the spider.

        # Should return None or raise an exception.
        return None

    def process_spider_output(self, response, result, spider):
        # Called with the results returned from the Spider, after
        # it has processed the response.

        # Must return an iterable of Request, or item objects.
        for i in result:
            yield i

    def process_spider_exception(self, response, exception, spider):
        # Called when a spider or process_spider_input() method
        # (from other spider middleware) raises an exception.

        # Should return either None or an iterable of Request or item
objects.
        pass

    def process_start_requests(self, start_requests, spider):
        # Called with the start requests of the spider, and works
        # similarly to the process_spider_output() method, except
        # that it doesn’t have a response associated.

        # Must return only requests (not items).
        for r in start_requests:
            yield r

    def spider_opened(self, spider):
        spider.logger.info('Spider opened: %s' % spider.name)


class LianjiaspiderDownloaderMiddleware:
    # Not all methods need to be defined. If a method is not defined,
    # scrapy acts as if the downloader middleware does not modify the
    # passed objects.

    @classmethod
    def from_crawler(cls, crawler):
        # This method is used by Scrapy to create your spiders.
        s = cls()
```

```
        crawler.signals.connect(s.spider_opened,
signal=signals.spider_opened)
        return s

    def process_request(self, request, spider):
        option = webdriver.ChromeOptions()
        option.add_argument('--headless')
        driver = webdriver.Chrome(chrome_options=option)
        driver.get(request.url)
        driver.implicitly_wait(5)
        time.sleep(5)
        html = driver.page_source
        driver.quit()
        return HtmlResponse(url=request.url, body=html,
request=request, encoding='utf-8')

    def process_response(self, request, response, spider):
        # Called with the response returned from the downloader.

        # Must either;
        # - return a Response object
        # - return a Request object
        # - or raise IgnoreRequest
        return response

    def process_exception(self, request, exception, spider):
        # Called when a download handler or a process_request()
        # (from other downloader middleware) raises an exception.

        # Must either:
        # - return None: continue processing this exception
        # - return a Response object: stops process_exception() chain
        # - return a Request object: stops process_exception() chain
        pass

    def spider_opened(self, spider):
        spider.logger.info('Spider opened: %s' % spider.name)
```

pipelines.py

```
# Define your item pipelines here
#
# Don't forget to add your pipeline to the ITEM_PIPELINES setting
# See: https://docs.scrapy.org/en/latest/topics/item-pipeline.html
```

```
# useful for handling different item types with a single interface
from itemadapter import ItemAdapter

from itemadapter import ItemAdapter
import csv


class LianjiaspiderPipeline:
    def open_spider(self, spider):
        try:
            self.file = open('data.csv', 'w', encoding='utf_8_sig')
            self.result = csv.writer(self.file)
            self.result.writerow(['name', 'zone', 'street', 'road',
'Room_type', 'area', 'avg_price', 'tot_price'])
        except Exception as e:
            print(e)

    def process_item(self, item, spider):
        dict_item = ItemAdapter(item).asdict()
        self.result.writerow(dict_item.values())
        return item

    def close_spider(self, spider):
        self.file.close()
```

settings.py

```
# Scrapy settings for LianjiaSpider project
#
# For simplicity, this file contains only settings considered
important or
# commonly used. You can find more settings consulting the
documentation:
#
#     https://docs.scrapy.org/en/latest/topics/settings.html
#     https://docs.scrapy.org/en/latest/topics/downloader-
middleware.html
#     https://docs.scrapy.org/en/latest/topics/spider-middleware.html

BOT_NAME = 'LianjiaSpider'

SPIDER_MODULES = ['LianjiaSpider.spiders']
NEWSPIDER_MODULE = 'LianjiaSpider.spiders'
```

```
# Crawl responsibly by identifying yourself (and your website) on the
user-agent
#USER_AGENT = 'LianjiaSpider (+http://www.yourdomain.com)'

# Obey robots.txt rules
ROBOTSTXT_OBEY = True

DOWNLOADER_MIDDLEWARES = {
   'LianjiaSpider.middlewares.LianjiaspiderDownloaderMiddleware': 543,
}



ITEM_PIPELINES = {
   'LianjiaSpider.pipelines.LianjiaspiderPipeline': 300,
}
```

dataProcessing.py

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

data = pd.read_csv('data.csv', encoding='utf_8_sig')
pd.set_option('display.unicode.east_asian_width', True)

#找出总价最贵和最便宜的房子，以及总价的中位数
max_tot_price = pd.DataFrame.max(data["tot_price"])
min_tot_price = pd.DataFrame.min(data["tot_price"])
median_tot_price = pd.DataFrame.median(data["tot_price"])
print("for total price of houses:")
print("highest price:", max_tot_price)
print("lowest price:", min_tot_price)
print("median price:", median_tot_price)

#找出均价最贵和最便宜的房子，以及单价的中位数
max_avg_price = pd.DataFrame.max(data["avg_price"])
min_avg_price = pd.DataFrame.min(data["avg_price"])
median_avg_price = pd.DataFrame.median(data["avg_price"])
print("for average price of houses:")
print("highest price:", max_avg_price)
print("lowest price:", min_avg_price)
print("median price:", int(median_avg_price))

#找到总价在均值三倍标准差以外的房屋
standard_deviation = pd.DataFrame.std(data["tot_price"])
```

```
print(standard_deviation)

#列出总价在均值三倍标准差以外的房屋
standard = 3 * standard_deviation
a = data.loc[data['tot_price'] > standard]
print("Outliers of standard deviation:")
print(a)

#通过箱型图原则判断并列出均价为异常值的房屋
plt.boxplot(x=data['avg_price'])
plt.show()
q1 = data['avg_price'].quantile(q=0.25)
q3 = data['avg_price'].quantile(q=0.75)
q2 = data['avg_price'].quantile(q=0.5)
low_limit = q1 - 1.5 * (q3-q1)
high_limit = q3 + 1.5 * (q3-q1)
print(q1,q2,q3,low_limit,high_limit)
val = data.loc[(data['avg_price'] > high_limit)|(data['avg_price'] <
low_limit)]
print("Outliers of box plot: ")
print(val)

#离散化处理
#bins = [0, 10000, 20000, 30000, 40000, 50000, 60000, 70000, 80000,
90000, 100000]
bins = [0,20000,40000,60000,90000]
cuts = pd.cut(data['avg_price'], bins)
print(pd.value_counts(cuts))
print(pd.value_counts(cuts, normalize=True))
```

2、处理 PM2.5 代码

step1.py

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

#找出 2015 年的数据，保存在新的 csv 文件中
data = pd.read_csv('BeijingPM20100101_20151231.csv',
encoding='utf_8_sig')
pd.set_option('display.unicode.east_asian_width', True)

data2015 = data.loc[data['year'] == 2015]
```

```
data2015.to_csv("newdata.csv", na_rep='NA')
```

step2.py

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

data = pd.read_csv('newdata.csv', encoding='utf_8_sig')
pd.set_option('display.unicode.east_asian_width', True)

#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
print(data.shape)
print(data.count(axis=0))
print([data.shape[0]] - data.count(axis=0))
emptylines = [data.shape[0]] - data.count(axis=0)

#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
#东四环空值较多，填充数据则不准确，选择删去该列
data.drop(columns="PM_Dongsihuan")

#对于其他含空的数据，使用前面一个值进行填充
newdata = data.fillna(method="ffill")

#输出成新的 csv 文件
newdata.to_csv("finalresult.csv")
print([data.shape[0]] - newdata.count(axis=0))
```