



**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 2](#_Toc90625981)

[二、 链家 3](#_Toc90625982)

[1、 爬取链家新房数据与存储 3](#_Toc90625983)

[2、数据处理 9](#_Toc90625984)

[2、 异常值处理 11](#_Toc90625985)

[3、 离散化处理 14](#_Toc90625986)

[三、 分析2015年北京市PM2.5指数数据集空值 16](#_Toc90625987)

[1、 数据抽取 16](#_Toc90625988)

[2、 找到空值并处理 17](#_Toc90625989)

[四、 实验感想 23](#_Toc90625990)

[五、 附录：代码 24](#_Toc90625991)

[1、爬取链家代码 24](#_Toc90625992)

[2、处理PM2.5代码 31](#_Toc90625993)

# 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。

# 链家

## 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：

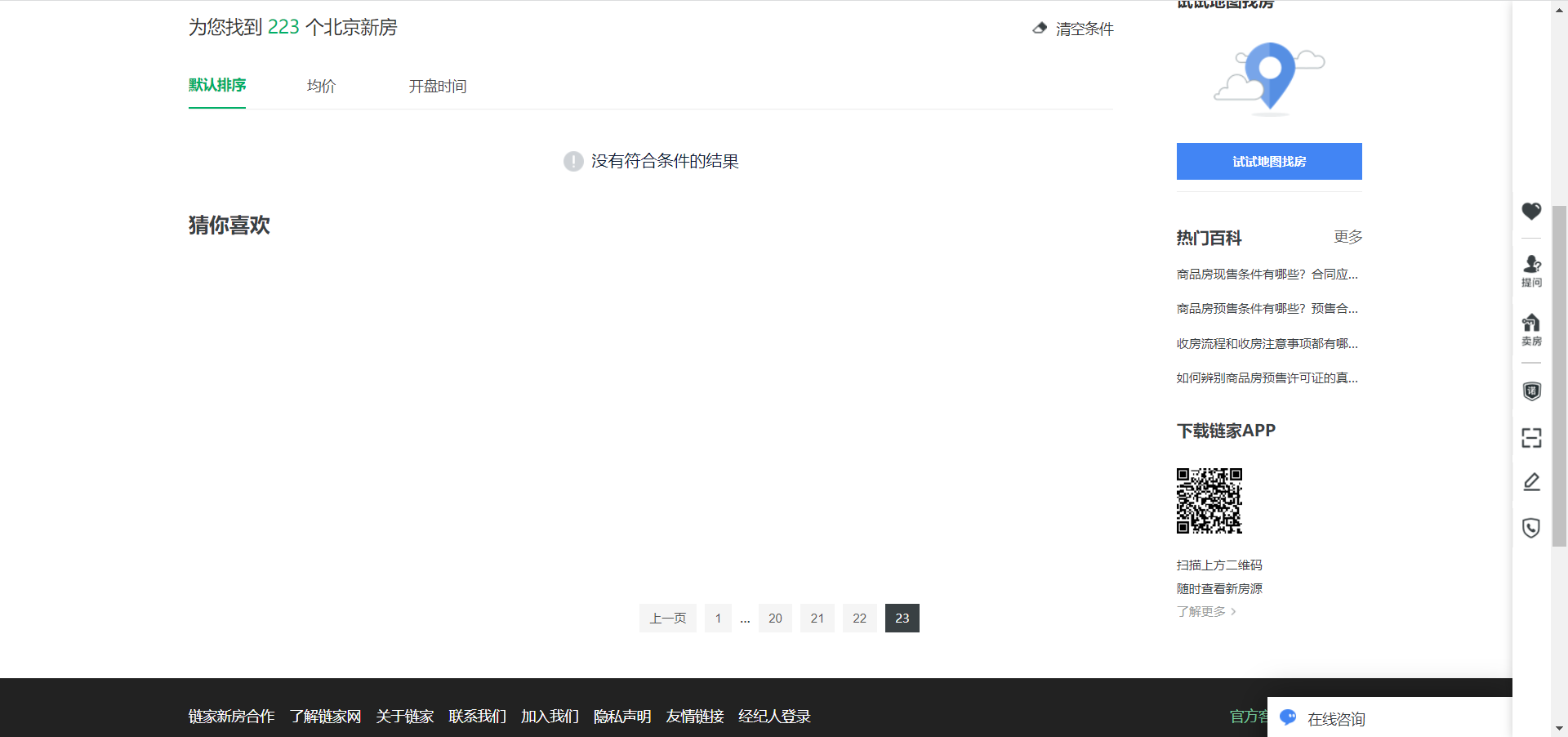
name = "" ：这个爬虫的识别名称，必须唯一。

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”。

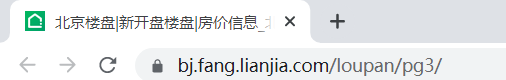
start\_urls = () ：为要爬取的URL列表。爬虫从这里开始抓取数据。

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

电建地产洺悦苑,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

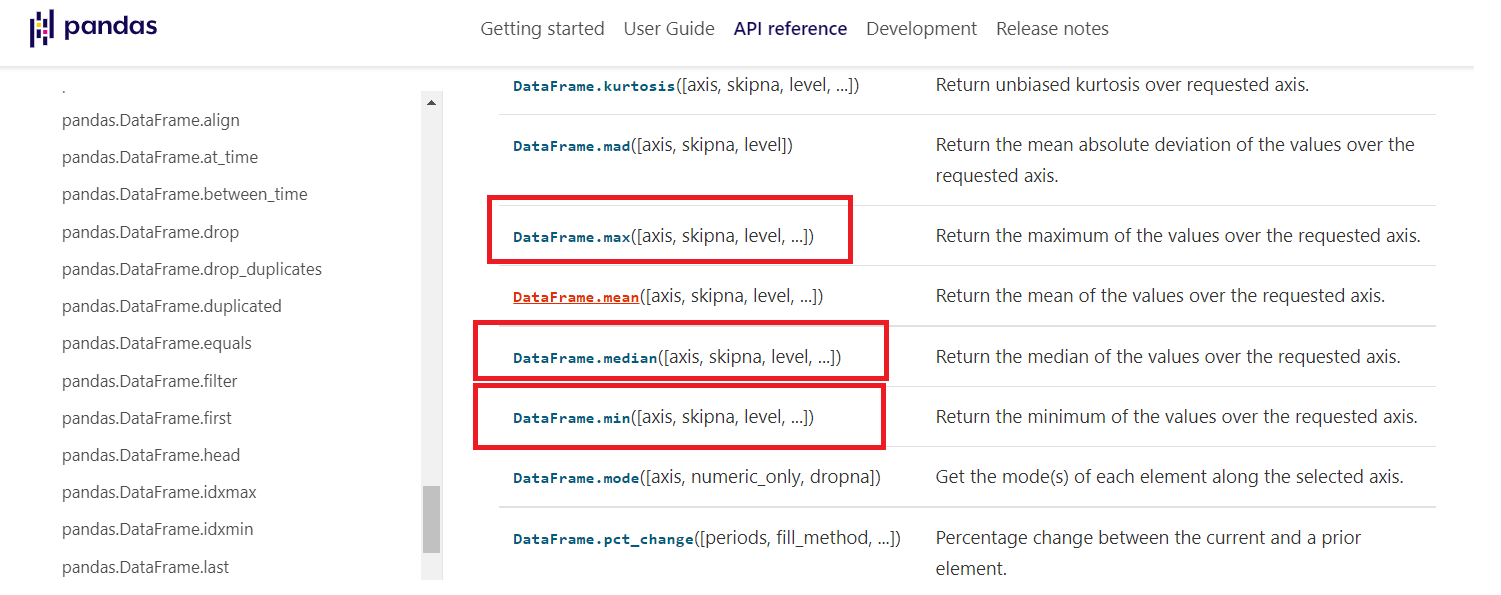
data = pd.read\_csv('data.csv', encoding='utf\_8\_sig')

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"])

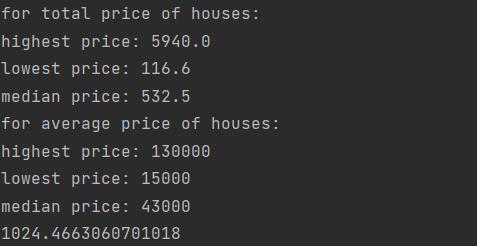
print("for average price of houses:")

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：



## 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：



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

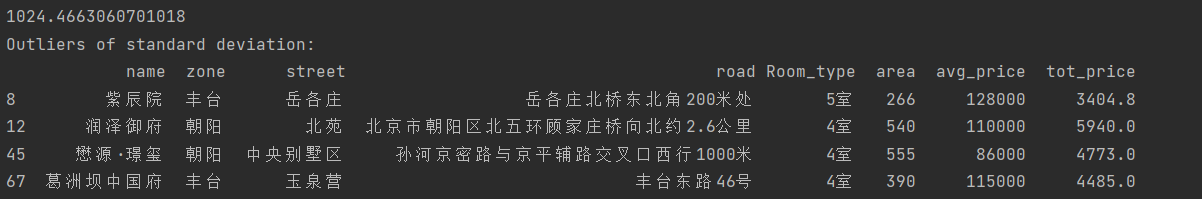
standard = 3 \* standard\_deviation

a = data.loc[data['tot\_price'] > standard]

print("Outliers of standard deviation:")

print(a)

运行出的结果如下：



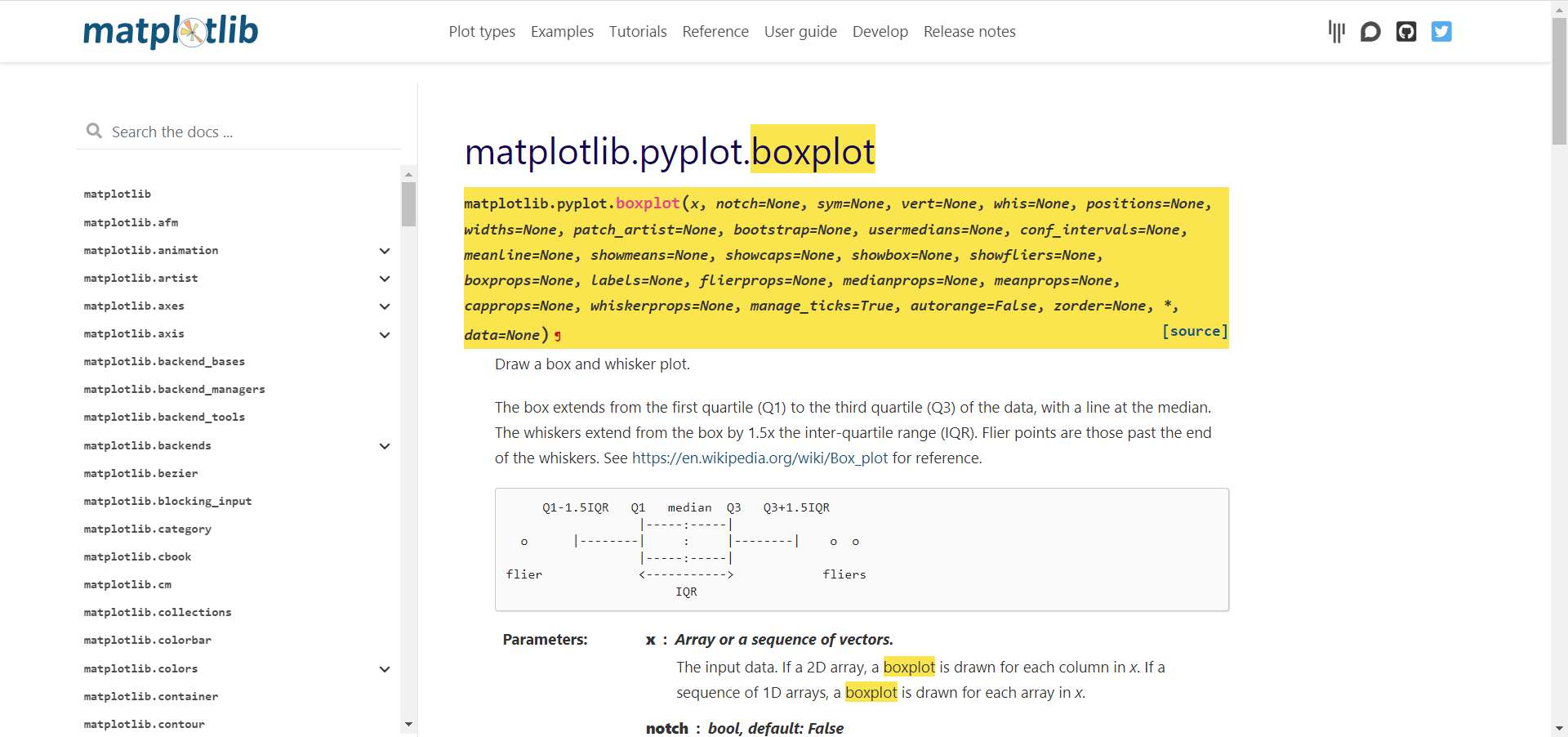
从上面可以看出，总价的标准差约为1024，而上述房屋的总价都超过了3000万，观察各条目的详细信息，可以看出，这些房屋建筑面积都很大，尤其是朝阳区的两座别墅区，面积都超过了500平米，每平米的均价也不便宜，远超均价的中位数43000。可以得知，上述值并不是因为异常而出现的，而是因为存在某些房屋面积大、均价高，进而导致总价远超其他项，但这些条目数量也是非常少的，仅为个别极端情况。

**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**

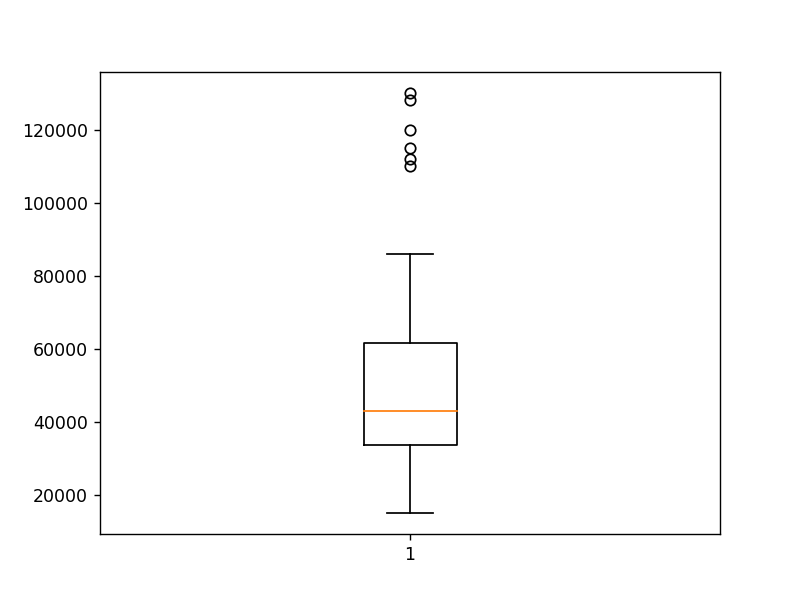
箱线图也称箱须图，是利用数据中的五个统计量：最小值、第一四分位数、中位数、第三四分位数与最大值来描述数据的一种方法，它也可以粗略地看出数据是否具有有对称性，分布的分散程度等信息。具体含义如下：

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)

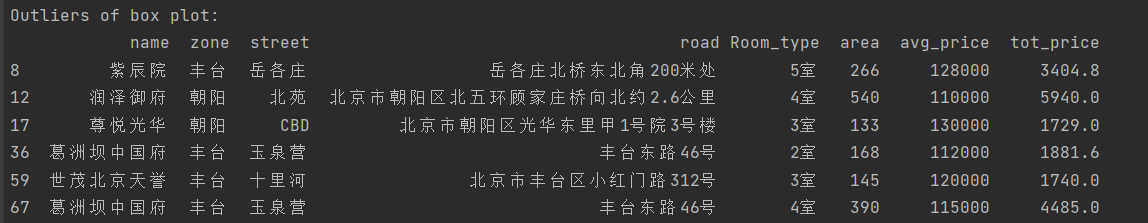
high\_limit = q3 + 1.5 \* (q3-q1)

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：



为了判断异常原因，输出q1,q2,q3,low\_limit,high\_limit如下：

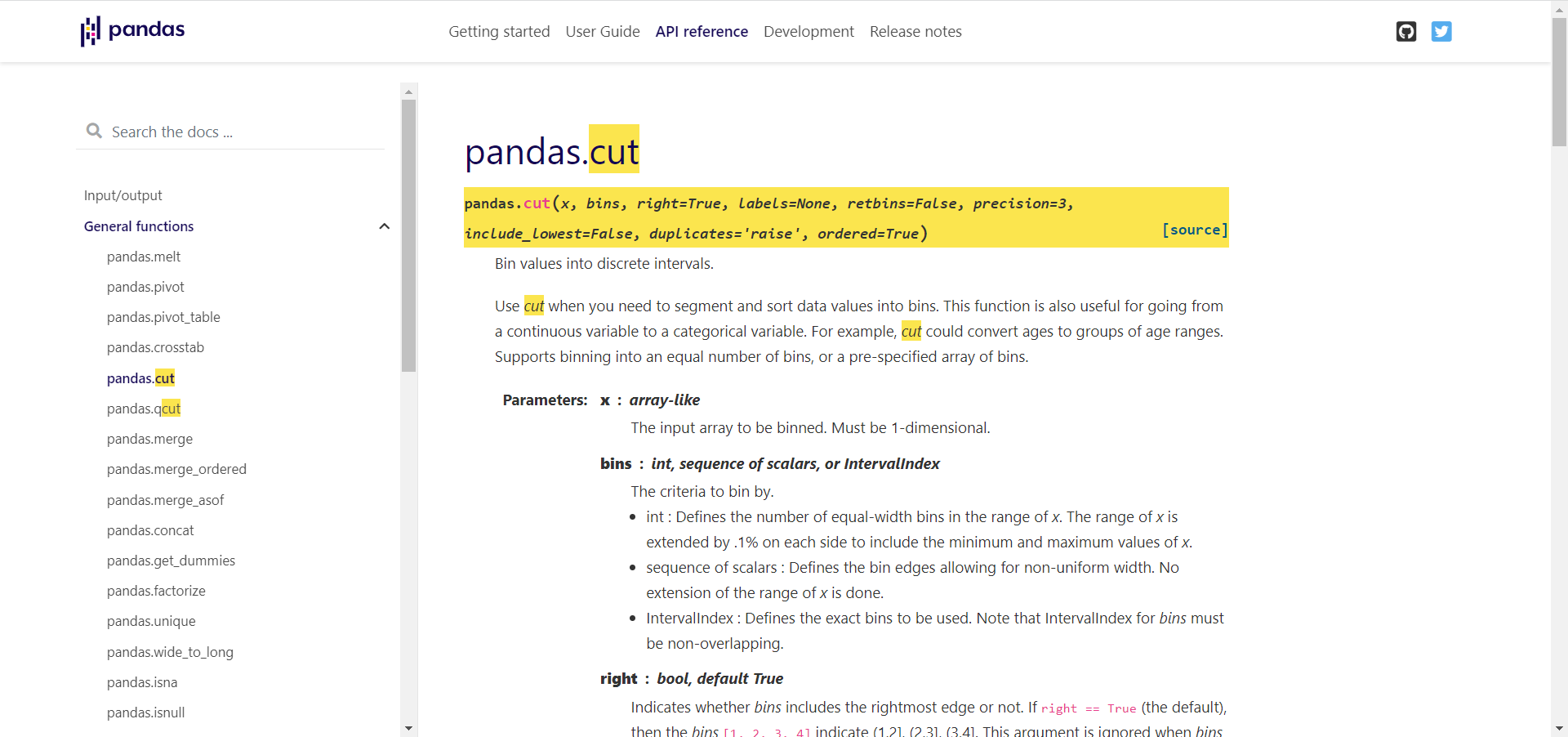


可以看出，每平米的均值约为43000元，外限的上界为103500元，但是上述房屋的均价都超出了110000元。这些数据之所以异常，是因为存在部分房屋处于热门地段，价格较高。

## 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

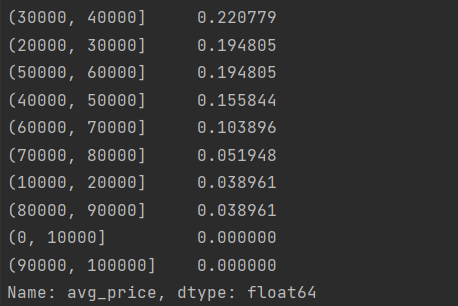
要给数据实现离散化，查阅官方文档，由下可知，可以使用cut函数来实现，其中Bin值表示为离散的间隔，该函数可以预先指定的存储箱阵列进行存储。



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：



运行出的结果如下：



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]

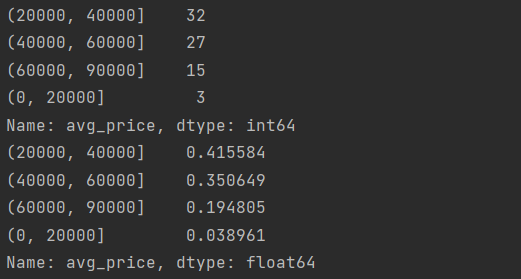
bins = [0,20000,40000,60000,90000]

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：



# 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

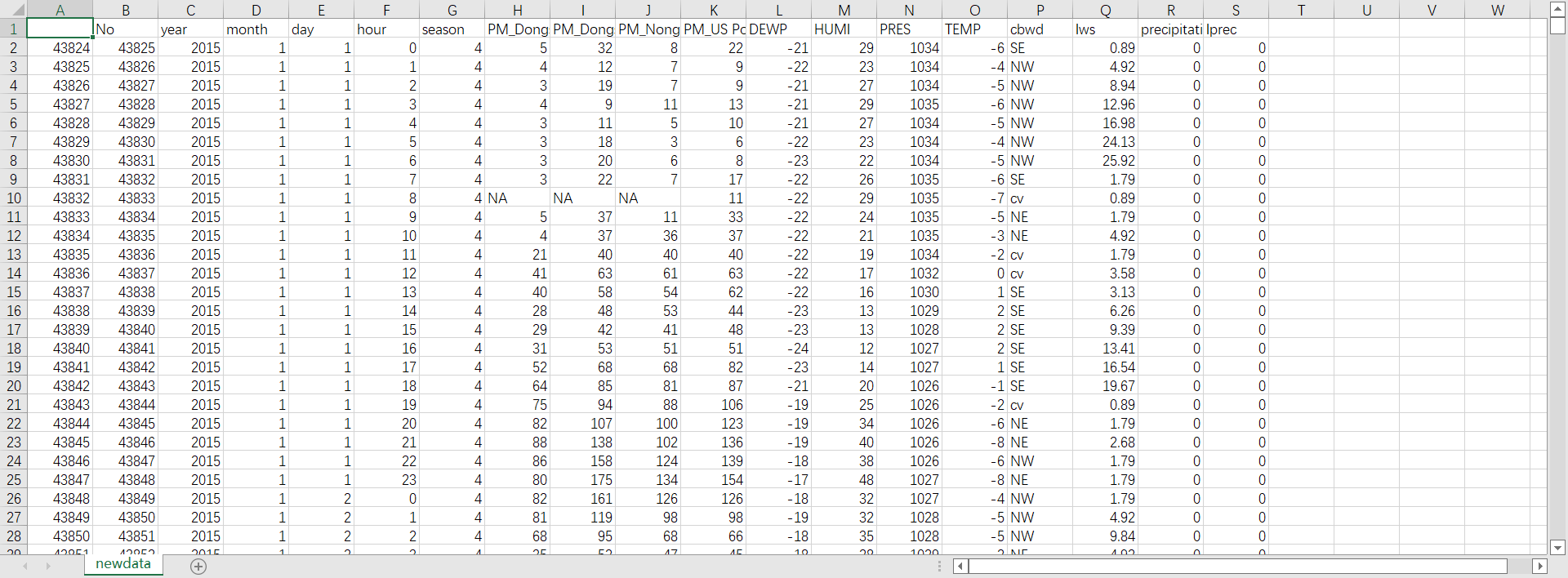
data = pd.read\_csv('BeijingPM20100101\_20151231.csv', encoding='utf\_8\_sig')

pd.set\_option('display.unicode.east\_asian\_width', True)

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

data2015.to\_csv("newdata.csv", na\_rep='NA')

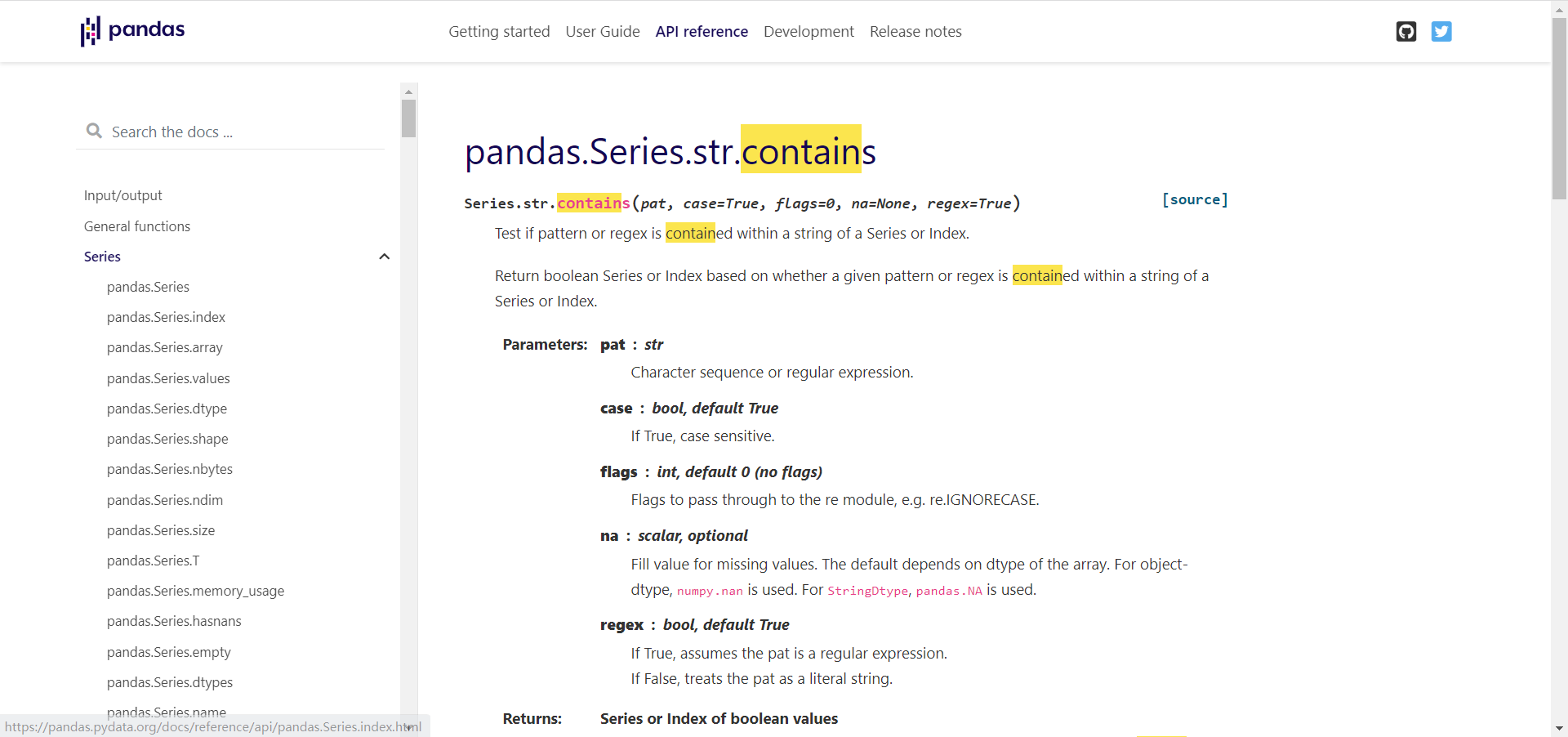
打开新的csv文件如下，可以看出，下列数据只含2015年的数据：



## 2、 找到空值并处理

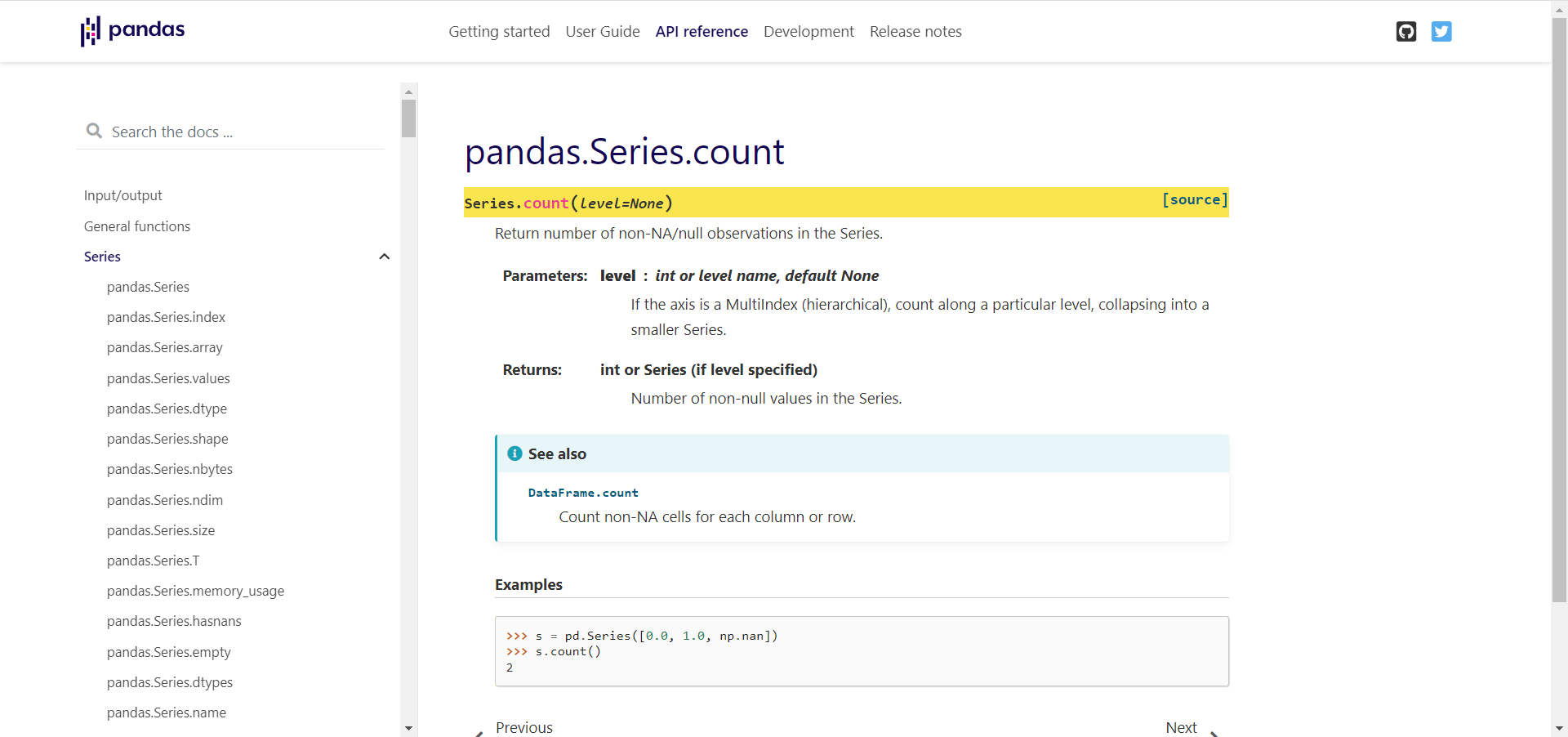
* 对新的csv文件，找出存在的空值列及相应的空值数量

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：

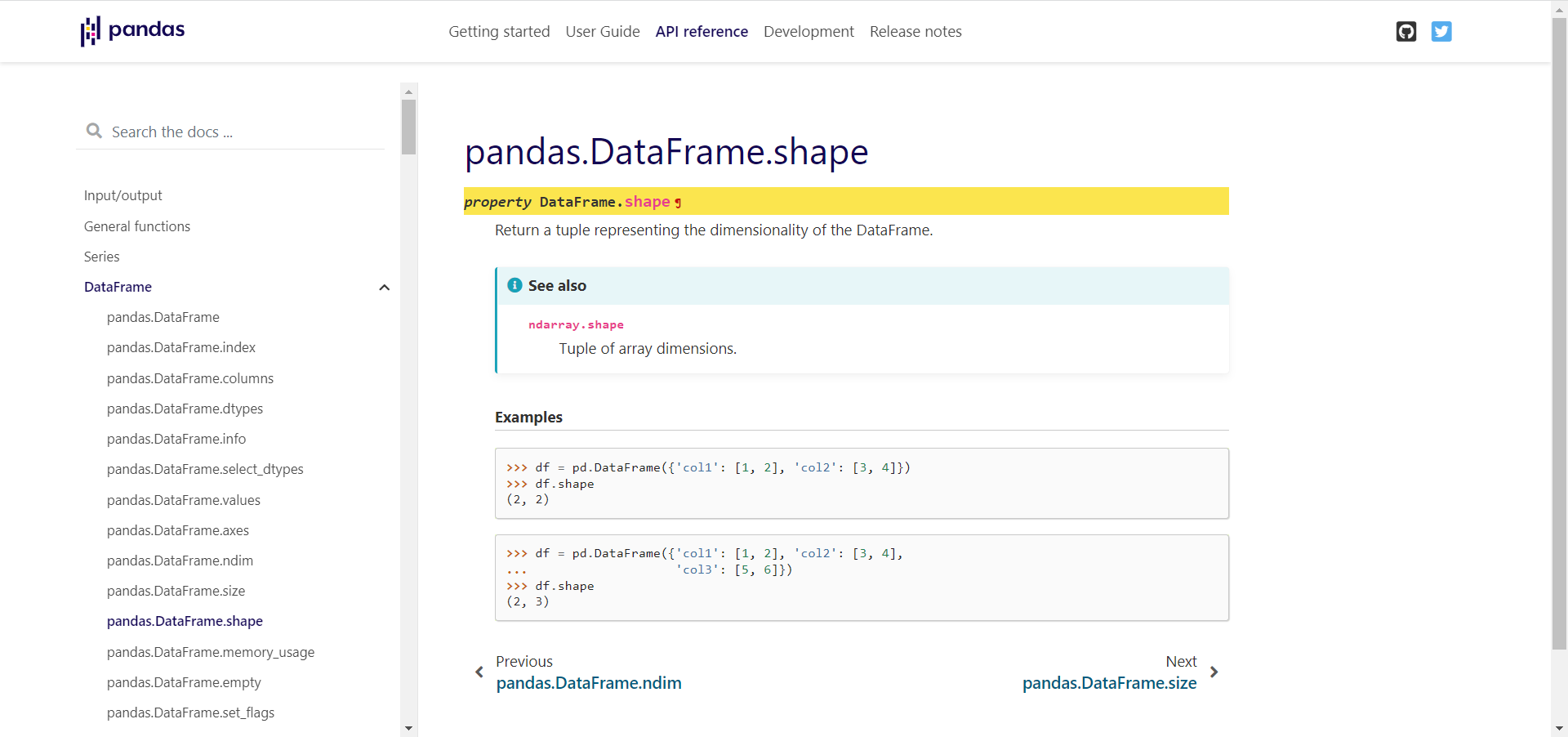


但是该函数只能对series对象操作，可以选择遍历每列并类型转换为series来判断，但是该操作比较繁琐，而且涉及到遍历，在列数较少的时候还比较方便操作，但是若列数较多，该方法的缺陷就愈加明显，于是我想用更加优美简单的方法去实现。

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：



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

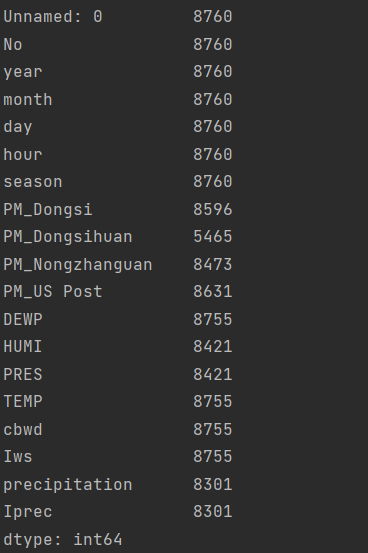
print(data.shape)

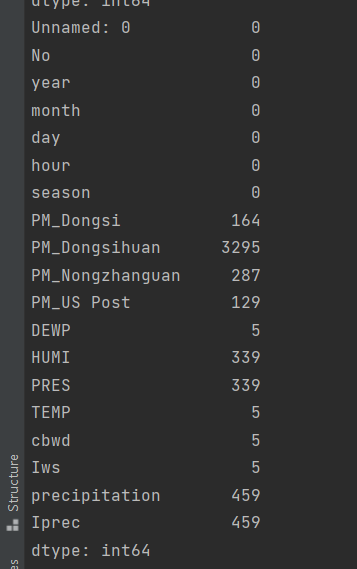
print(data.count(axis=0))

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

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

运行出的结果如下：



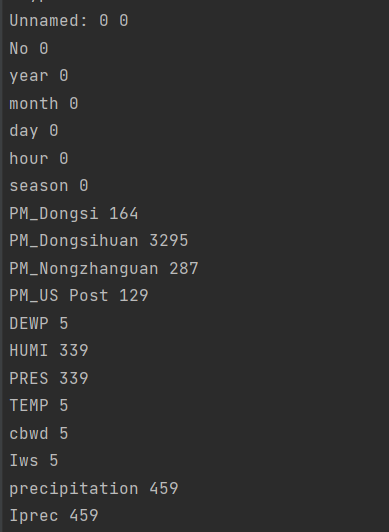


通过结果可以看出，除了年月日季节这些基本的时间信息，其他列均存在含有空项的情况，其中DEWP、TEMP、cbwd、Iws这四项缺失项目较少，仅有5个空项；PM\_Dongsi,PM\_Nongzhanguan,OM\_USPost,HUMI,PRES,precipitation,Iprec,这些项目有部分缺失，在有8760项的一列数据里，这些项目缺失的数量仅几百，并不会有太大的影响。但是 PM\_Dongsihuan这一列缺失数据过多，缺失了将近一半的数据。

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())

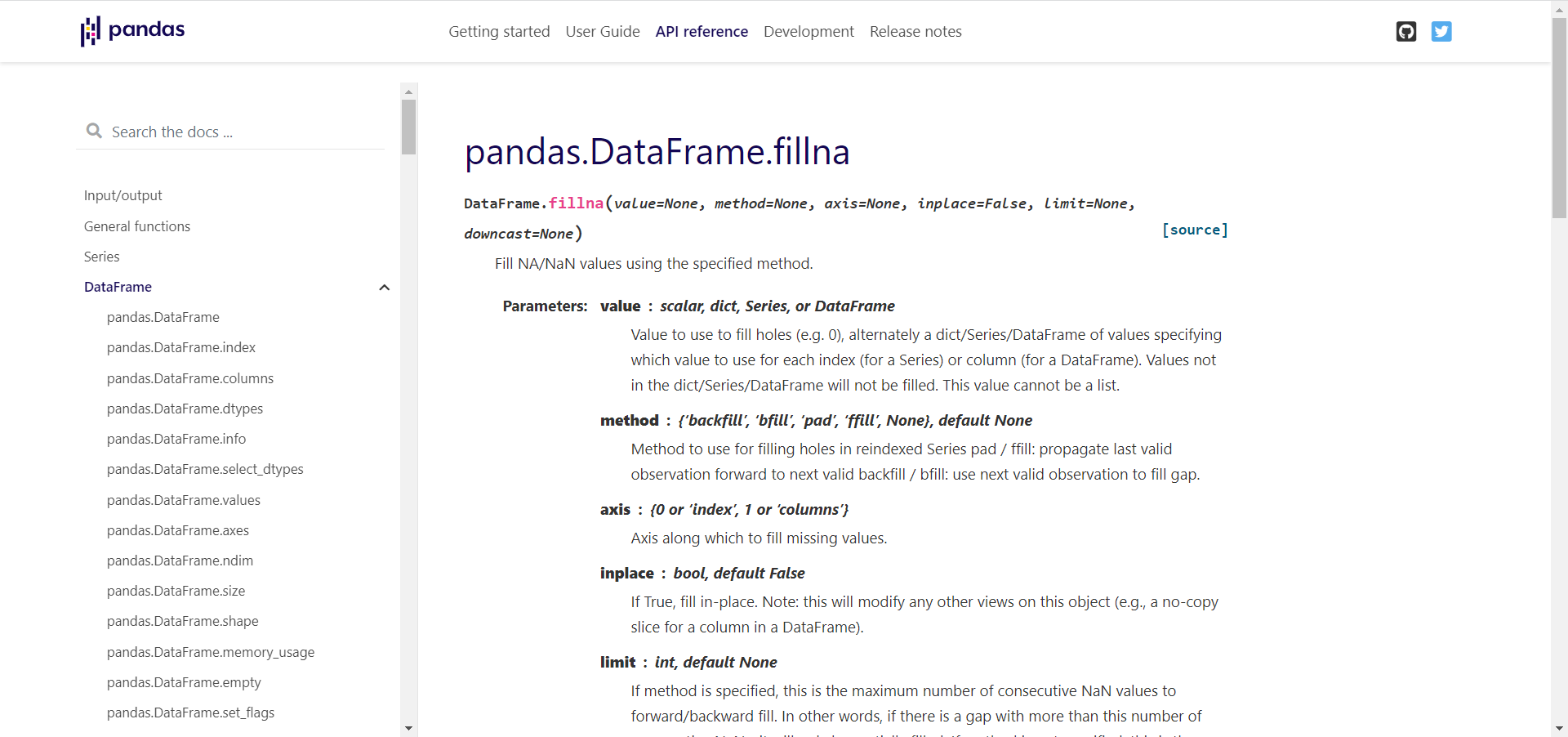


运行出的结果是一样的。

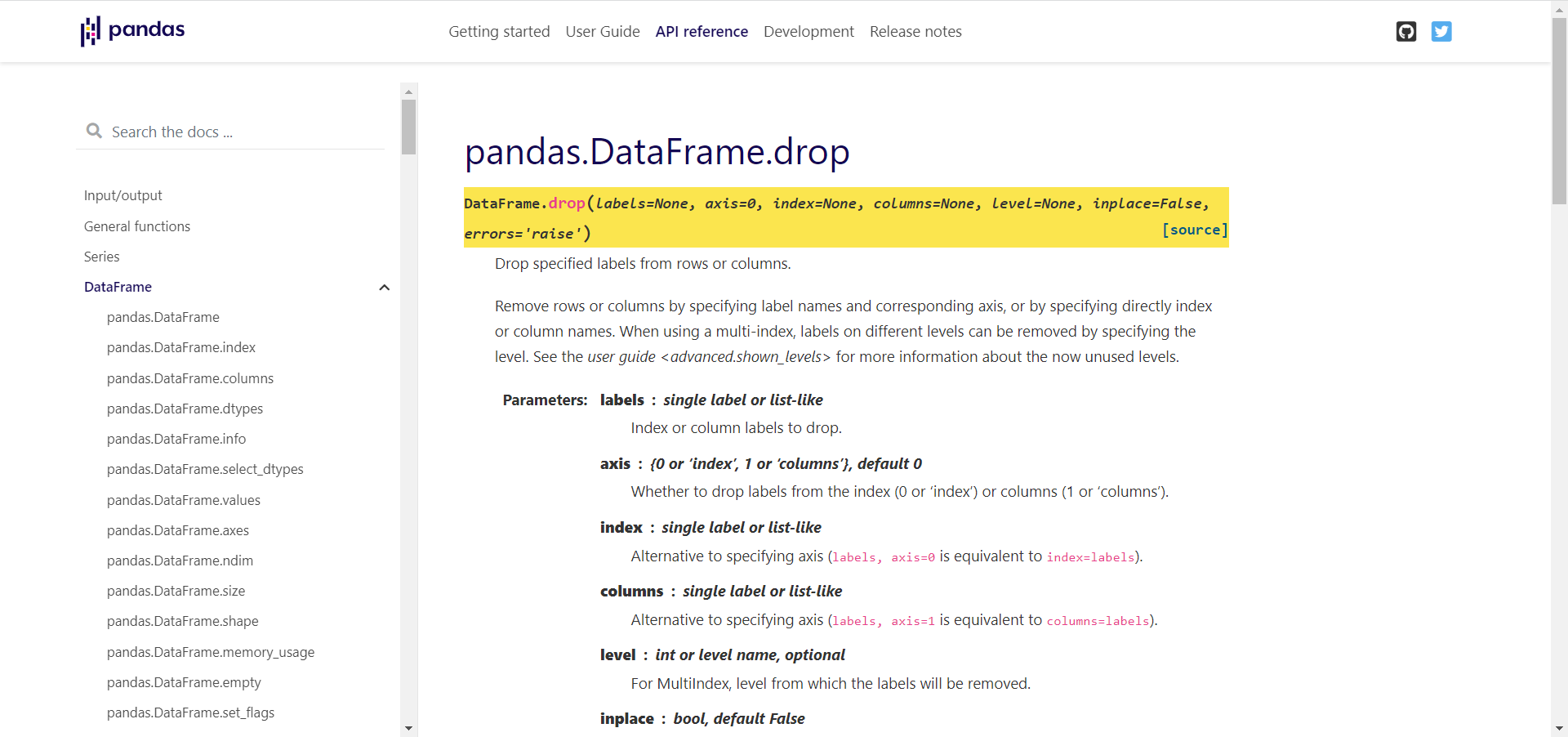
* 空值处理方法：对所有存在空值的列，给出空值的处理方法及理由，要求处理方法必须可在本数据集范围内执行

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")

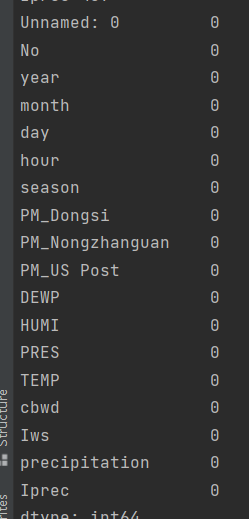
#输出成新的csv文件

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：



# 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。

# 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))