## 02-24~03-21

学习爬虫知识。

## 03-22~03-24

爬虫代码的编写。

## 03-25（周二）

爬虫代码的调试。

## 03-26（周三）

爬虫代码的调试。

## 03-27（周四）

第一次测试成功，代码如下：

import asyncio  
import aiohttp  
from lxml import etree  
import logging  
import datetime  
import openpyxl  
  
wb = openpyxl.Workbook()  
sheet = wb.active  
sheet.append(['房源', '房子信息', '所在区域', '单价', '关注人数和发布时间', '标签'])  
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s: %(message)s')  
start = datetime.datetime.now()  
  
class Spider(object):  
    def \_\_init\_\_(self):  
        self.semaphore = asyncio.Semaphore(6)  # 信号量，控制协程数，防止爬的过快被反爬  
        self.header = {  
            "Host": "bj.lianjia.com",  
            "Referer": "https://bj.lianjia.com/ershoufang/",  
            "Cookie":"lianjia\_uuid=170c4a34-bc23-4a94-8213-f979ed538ecd; Hm\_lvt\_46bf127ac9b856df503ec2dbf942b67e=1742878611; HMACCOUNT=DA3E61C8D3A04FDB; sajssdk\_2015\_cross\_new\_user=1; sensorsdata2015jssdkcross=%7B%22distinct\_id%22%3A%22195cba91792555-0e1d16db46adc9-26011d51-1821369-195cba91793a7a%22%2C%22%24device\_id%22%3A%22195cba91792555-0e1d16db46adc9-26011d51-1821369-195cba91793a7a%22%2C%22props%22%3A%7B%22%24latest\_traffic\_source\_type%22%3A%22%E7%9B%B4%E6%8E%A5%E6%B5%81%E9%87%8F%22%2C%22%24latest\_referrer%22%3A%22%22%2C%22%24latest\_referrer\_host%22%3A%22%22%2C%22%24latest\_search\_keyword%22%3A%22%E6%9C%AA%E5%8F%96%E5%88%B0%E5%80%BC\_%E7%9B%B4%E6%8E%A5%E6%89%93%E5%BC%80%22%7D%7D; \_jzqa=1.2109646134043611100.1742878612.1742878612.1742878612.1; \_jzqc=1; \_jzqckmp=1; \_ga=GA1.2.1122416408.1742878623; \_gid=GA1.2.1986202072.1742878623; \_ga\_XLL3Z3LPTW=GS1.2.1742878624.1.1.1742878687.0.0.0; \_ga\_NKBFZ7NGRV=GS1.2.1742878623.1.1.1742878687.0.0.0; \_jzqb=1.5.10.1742878612.1; Hm\_lpvt\_46bf127ac9b856df503ec2dbf942b67e=1742878847; select\_city=110000; Hm\_lvt\_28b65c68923b952bf94c102598920ce0=1742878859; session\_id=3f98c84b-5f3b-eaf1-a514-31ec875475ed; digData=%7B%22key%22%3A%22m\_pages\_ershoufangSearch%22%7D; \_gat=1; \_gat\_past=1; \_gat\_new=1; \_gat\_global=1; \_gat\_new\_global=1; srcid=; \_ga\_XRDEC2G0T9=GS1.2.1742878877.1.1.1742879397.0.0.0; \_ga\_XGP5EDPZTV=GS1.2.1742878877.1.1.1742879397.0.0.0; \_ga\_SNG6R1B3VY=GS1.2.1742878877.1.1.1742879397.0.0.0; lianjia\_ssid=f3f53cc4-0b38-4215-9901-078b0e8545ce; Hm\_lpvt\_28b65c68923b952bf94c102598920ce0=1742879408; beikeBaseData=%7B%22parentSceneId%22%3A%22484431939518451457%22%7D",  
            "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/134.0.0.0 Safari/537.36"  
        }  
  
    async def scrape(self, url):  
        async with self.semaphore:  
            session = aiohttp.ClientSession(headers=self.header)  
            response = await session.get(url)  
            result = await response.text()  
            await session.close()  
            return result  
  
    async def scrape\_index(self, page):  
        url = f'https://bj.lianjia.com/ershoufang/pg{page}/'  
        text = await self.scrape(url)  
        await self.parse(text)  
  
    async def parse(self, text):  
        html = etree.HTML(text)  
        lis = html.xpath('//\*[@id="content"]/div[1]/ul/li')  
        for li in lis:  
            house\_data = li.xpath('.//div[@class="title"]/a/text()')[0]  # 房源  
            house\_info = li.xpath('.//div[@class="houseInfo"]/text()')[0]  # 房子信息  
            address = ' '.join(li.xpath('.//div[@class="positionInfo"]/a/text()'))  # 位置信息  
            price = li.xpath('.//div[@class="priceInfo"]/div[2]/span/text()')[0]  # 单价 元/平米  
            attention\_num = li.xpath('.//div[@class="followInfo"]/text()')[0]  # 关注人数和发布时间  
            tag = ' '.join(li.xpath('.//div[@class="tag"]/span/text()'))  # 标签  
            sheet.append([house\_data, house\_info, address, price, attention\_num, tag])  
            logging.info([house\_data, house\_info, address, price, attention\_num, tag])  
  
    def main(self):  
        # 100页的数据  
        scrape\_index\_tasks = [asyncio.ensure\_future(self.scrape\_index(page)) for page in range(1, 101)]  
        loop = asyncio.get\_event\_loop()  
        tasks = asyncio.gather(\*scrape\_index\_tasks)  
        loop.run\_until\_complete(tasks)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
    spider = Spider()  
    spider.main()  
    wb.save('house.xlsx')  
    delta = (datetime.datetime.now() - start).total\_seconds()  
    print("用时：{:.3f}s".format(delta))

遇到的问题：因为后续机器学习进行房价预测需要上千条数据，所以我尝试将爬取的页数从1~101改成了从1~201，但是发现无论怎么更改参数，始终只能爬取到约150条数据，并且页数改动后，python文件输出终端报错。

## 03-28（周五）

尝试解决该问题，并尝试使用编写其他方法爬取，但效果相同。

## 03-29（周六）

尝试解决该问题，并尝试使用编写其他方法爬取，但效果相同。

## 03-30（周日）

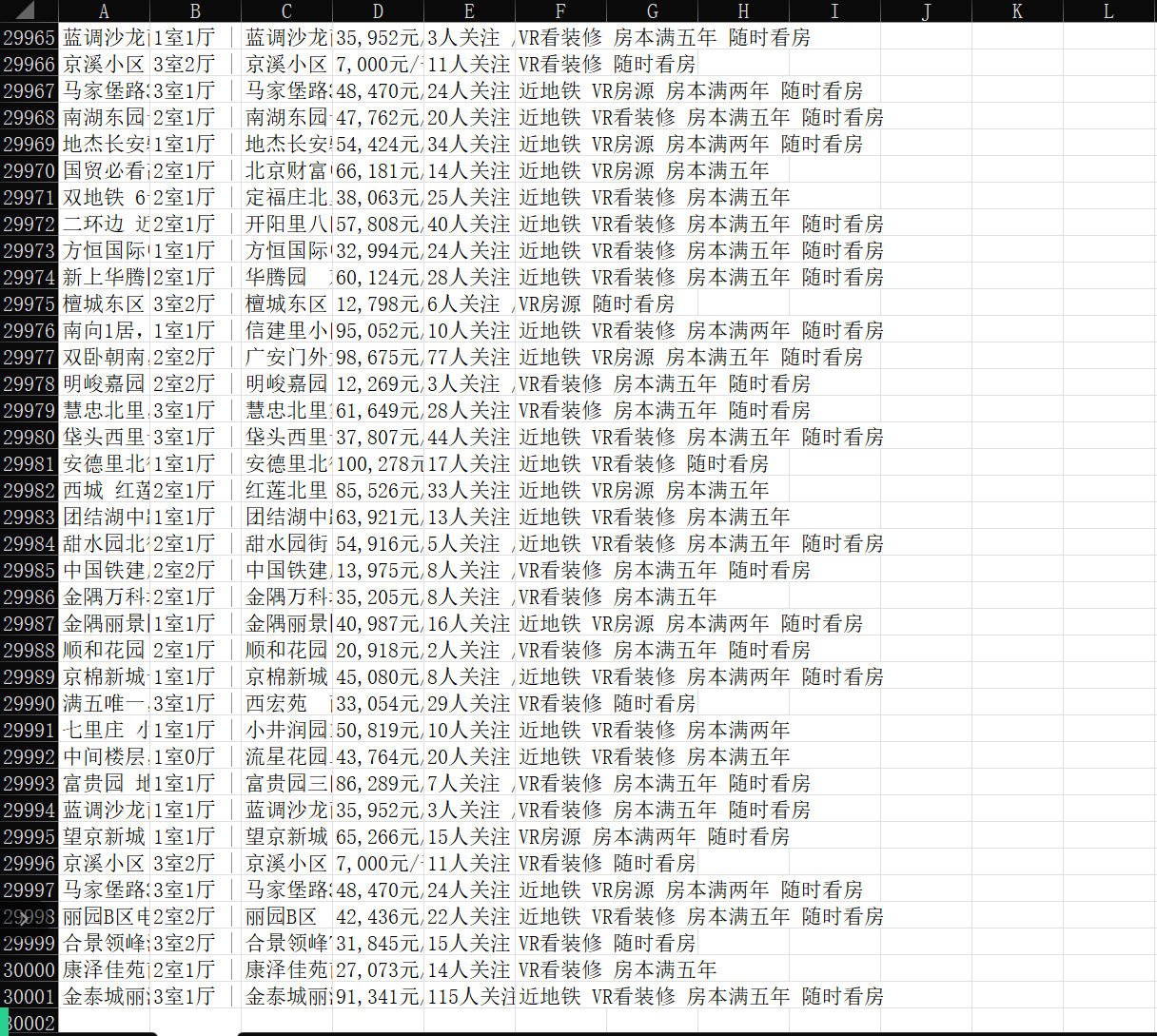
问题解决，原因是没有实名登录链家网。对于游客模式，链家网只展示少量房源数据，所以只能爬取到page1~page100，大约150条房源信息（声明：这里的page和真实的page不同， css对鼠标滚动动作动态渲染加载出新的房源信息，每加载一定数量的房源信息就更新一次page，所以即使程序里写的是page1~page100，也不代表是爬取了真实的100页数据，可能只有十余页数据）。如果将page1~page100改成page1~page200或更多，则python文件执行后会报错，因为游客模式访问不到更多的信息。

解决的方式就是实名认证登录。但是这个地方后续也可以进一步优化，比如设置代理池。但是因为代理池涉及资金和成本问题，所以此处选择了实名认证方式。如果本项目将商业化，则团队应分配给爬虫部分固定资本以维护数据库和代理池的稳定。

解决问题后，当晚进行爬取测试，约30min内能爬取到约1w数据，并成功保存进excel中。对数据进行去重清洗，得到约2100条可用数据。产率相对很高。

代码部分：同03-27。

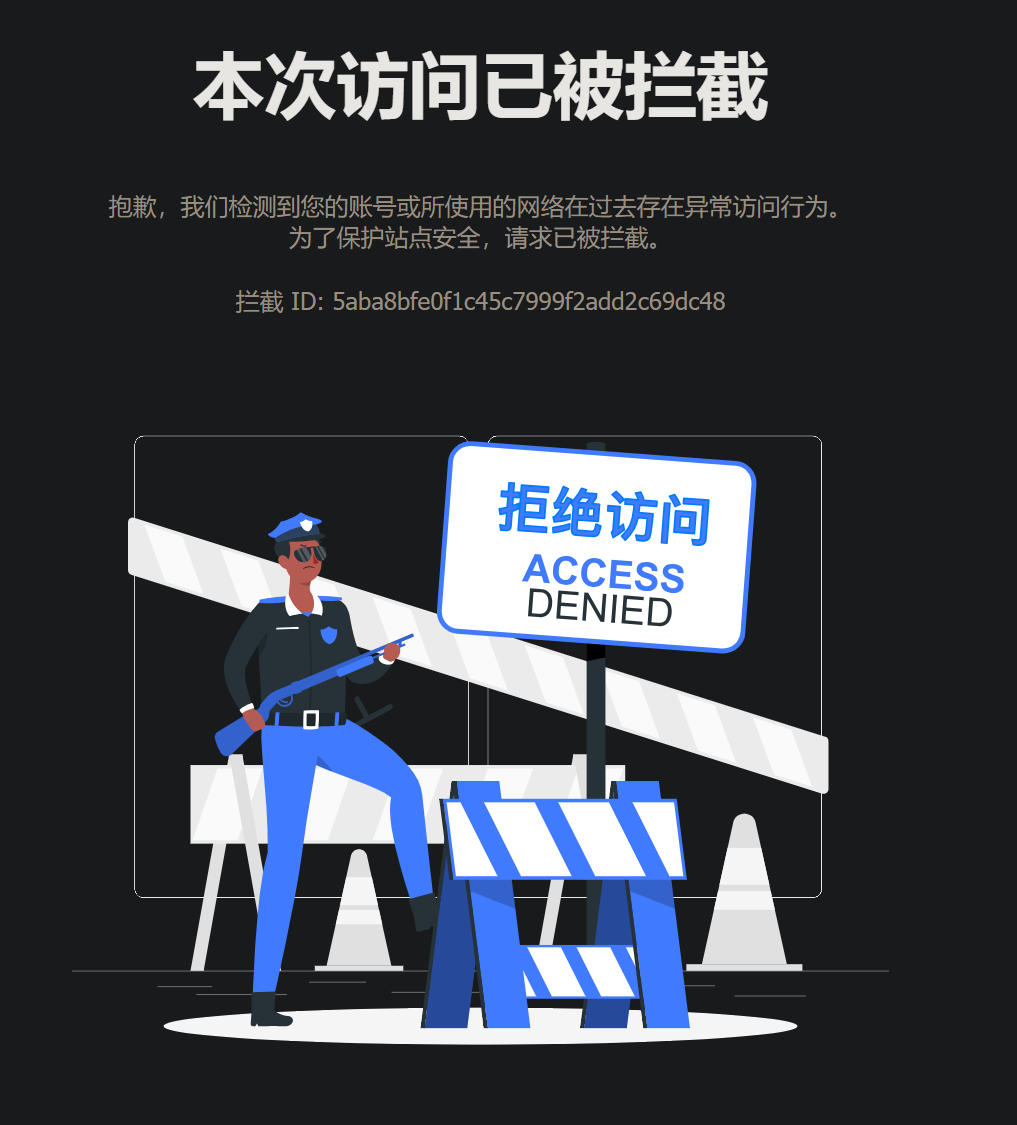
爬取了3w数据，清洗后剩2100条可用数据。



## 03-31（周一）

遇到问题：隔天IP被封。该爬虫是异步爬虫，初始时设置的并发数为6，即同一时间6条爬虫并发，爬取速度过快，被链家网后台判定为恶意爬虫而非正常访问，IP被封。

由此猜想链家网的恶意代码检测机制是统计同一个IP在过去12h内访问网站的次数和行为，如果访问频率过高远超真人访问速率极限，则判定为恶意代码，并对该IP进行封禁。



解决方法：尝试和链家客服、技术部门沟通协商解除封禁无效。于是更换了另一个账号重新注册登录，修改爬虫代码中的请求头中的用户数据，将异步爬虫并发数降至最低：

# self.semaphore = asyncio.Semaphore(6) # 信号量，控制协程数，防止爬得过快被反爬  
self.semaphore = asyncio.Semaphore(1) # 信号量，控制协程数，防止爬得过快被反爬

并增加了很多延时措施，最终代码如下：

import asyncio

import aiohttp

from lxml import etree

import pandas as pd

import logging

import datetime

import openpyxl

import random

wb = openpyxl.Workbook()

sheet = wb.active

sheet.append(['房源', '房子信息', '所在区域', '单价', '关注人数和发布时间', '标签'])

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s: %(message)s')

start = datetime.datetime.now()

class Spider(object):

    def \_\_init\_\_(self):

*# self.semaphore = asyncio.Semaphore(6)  # 信号量，控制协程数，防止爬得过快被反爬*

        self.semaphore = asyncio.Semaphore(1)  *# 信号量，控制协程数，防止爬得过快被反爬*

        self.delay\_range = (0.5, 3)

        self.header = {

            "Host": "bj.lianjia.com",

            "Referer": "https://bj.lianjia.com/ershoufang/",

            'Cookie': 'lianjia\_uuid=0a245c01-0249-4548-a9de-66e6367fdd82; \_ga=GA1.2.1314715887.1743086781; \_ga\_EYZV9X59TQ=GS1.2.1743086782.1.1.1743086797.0.0.0; \_ga\_DX18CJBZRT=GS1.2.1743086782.1.1.1743086797.0.0.0; \_ga\_BKB2RJCQXZ=GS1.2.1743088349.1.0.1743088349.0.0.0; Qs\_lvt\_200116=1743090960; Qs\_pv\_200116=3209232641914898400%2C2610558853758770700%2C2860350900287569000%2C2313803583990493000; \_ga\_E91JCCJY3Z=GS1.2.1743090925.1.1.1743091086.0.0.0; \_ga\_MFYNHLJT0H=GS1.2.1743090925.1.1.1743091086.0.0.0; Hm\_lvt\_46bf127ac9b856df503ec2dbf942b67e=1743086769,1743327950; HMACCOUNT=37989191F901F592; \_jzqc=1; \_qzjc=1; \_gid=GA1.2.1869468630.1743327970; crosSdkDT2019DeviceId=-2gp41z-o3lzyu-63pkn8jey4ausm7-e52x58qto; sensorsdata2015jssdkcross=%7B%22distinct\_id%22%3A%22195d81156b0581-08cc2e949a02e-4c657b58-1821369-195d81156b125a3%22%2C%22%24device\_id%22%3A%22195d81156b0581-08cc2e949a02e-4c657b58-1821369-195d81156b125a3%22%2C%22props%22%3A%7B%22%24latest\_traffic\_source\_type%22%3A%22%E8%87%AA%E7%84%B6%E6%90%9C%E7%B4%A2%E6%B5%81%E9%87%8F%22%2C%22%24latest\_referrer%22%3A%22https%3A%2F%2Fcn.bing.com%2F%22%2C%22%24latest\_referrer\_host%22%3A%22cn.bing.com%22%2C%22%24latest\_search\_keyword%22%3A%22%E6%9C%AA%E5%8F%96%E5%88%B0%E5%80%BC%22%7D%7D; select\_city=110000; \_jzqckmp=1; lfrc\_=7ac40290-a05f-4af6-a343-5cf68223992f; lianjia\_ssid=cef6d138-2a67-4902-825c-acca81cd9b18; hip=ZqJNHesc-q8M-cLKVXS1DczLesgAfOukX5k-TW536DddyCxdjzFacxTuoT6OFIL9zPCvPoLBTCWdqch3wo14TobVAlrneXVkD5LWjb5-nAG\_Okvr3CEe251SMxD8pkD4Vk97E8DHYOWUUedxHu-\_FUOV3EytI2WFqr0wcAcr75y\_KMth0Pay2WVBDA%3D%3D; \_jzqa=1.2936646379323121700.1743086770.1743413095.1743417476.5; \_jzqx=1.1743086770.1743417476.3.jzqsr=hip%2Elianjia%2Ecom|jzqct=/.jzqsr=bj%2Elianjia%2Ecom|jzqct=/; login\_ucid=2000000475594965; lianjia\_token=2.00143e639042ce0ea905934aa172e74f88; lianjia\_token\_secure=2.00143e639042ce0ea905934aa172e74f88; security\_ticket=kJTW0O78ZiEbLJdC52BMmhOVWkmk+JRY2VbTz+6M2fz1Vedzx57t0Cl+jZLbTTOORFvFDMaoSp6PYHT3+SQQLgTcNO41eMPcXvJsnkTzR9Hqr2n5WPsbYyuvEOcmJosQmPuEdxqV3I/kaNM5vhlwEd9fjJFTlx5FHUZkfBZVb6E=; ftkrc\_=49b169bc-e6bc-497c-8ad5-176098d9e4b4; srcid=; \_gat=1; \_gat\_past=1; \_gat\_global=1; \_gat\_new\_global=1; \_gat\_dianpu\_agent=1; \_ga\_KJTRWRHDL1=GS1.2.1743417489.3.1.1743418198.0.0.0; \_ga\_QJN1VP0CMS=GS1.2.1743417489.3.1.1743418198.0.0.0; Hm\_lpvt\_46bf127ac9b856df503ec2dbf942b67e=1743418199; \_qzja=1.936581872.1743327950261.1743413095555.1743417475988.1743418186630.1743418198786.0.0.0.17.3; \_qzjb=1.1743417475988.4.0.0.0; \_qzjto=10.2.0; \_jzqb=1.4.10.1743417476.1',

            'User-Agent':'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36'

        }

    async def scrape(self, url):

        async with self.semaphore:

            delay = random.uniform(\*self.delay\_range)

            await asyncio.sleep(delay)

            timeout = aiohttp.ClientTimeout(total=60)  *# 设置超时时间为60秒*

            async with aiohttp.ClientSession(headers=self.header, timeout=timeout) as session:

                try:

                    response = await session.get(url)

                    response.raise\_for\_status()  *# 检查请求是否成功*

                    return await response.text()

                except asyncio.TimeoutError:

                    logging.error(f"Timeout occurred for {url}")

                except aiohttp.ClientError as e:

                    logging.error(f"Request failed for {url} with error: {e}")

                return None

    async def scrape\_index(self, page):

        url = f'https://bj.lianjia.com/ershoufang/pg{page}/'

        text = await self.scrape(url)

        await self.parse(text)

        page\_delay = random.uniform(1, 5)

        await asyncio.sleep(page\_delay)

    async def parse(self, text):

        html = etree.HTML(text)

        lis = html.xpath('//\*[@id="content"]/div[1]/ul/li')

        for li in lis:

            try:

                house\_data = li.xpath('.//div[@class="title"]/a/text()')[0]  *# 房源*

                house\_info = li.xpath('.//div[@class="houseInfo"]/text()')[0]  *# 房子信息*

                address = ' '.join(li.xpath('.//div[@class="positionInfo"]/a/text()'))  *# 位置信息*

                price = li.xpath('.//div[@class="priceInfo"]/div[2]/span/text()')[0]  *# 单价 元/平米*

                attention\_num = li.xpath('.//div[@class="followInfo"]/text()')[0]  *# 关注人数和发布时间*

                tag = ' '.join(li.xpath('.//div[@class="tag"]/span/text()'))  *# 标签*

                sheet.append([house\_data, house\_info, address, price, attention\_num, tag])

                logging.info([house\_data, house\_info, address, price, attention\_num, tag])

            except IndexError:

                continue  *# 忽略空白或错误的房源数据*

    def main(self, start\_page, stop\_page):

        scrape\_index\_tasks = [asyncio.ensure\_future(self.scrape\_index(page)) for page in range(start\_page, stop\_page)]

        loop = asyncio.get\_event\_loop()

        tasks = asyncio.gather(\*scrape\_index\_tasks)

        loop.run\_until\_complete(tasks)

if \_\_name\_\_ == '\_\_main\_\_':

    spider = Spider()

    spider.main(1, 501)

    wb.save('house\_more.xlsx')

    delta = (datetime.datetime.now() - start).total\_seconds()

    print("用时：{:.3f}s".format(delta))

爬虫的机制是先爬取网页，再保存数据。如果网页爬取出现错误，数据自然也不会保存。

因为从异步爬虫改为简单爬虫，同时又增加了很多随机时延，所以爬虫速度肯定很慢。若大量爬取数据，只要网页爬取某处出现错误，则出现此错误前的数据将一概被丢弃，这样很浪费效率和资源。所以我想到的解决方法是：边爬取边存档。

具体的实现是，每次爬500pages，每爬取完10pages写入一次excel文件，注意这里的写excel文件的操作要设置成“在原有信息后新增信息”。

下面是最终代码实现：

import asyncio

import aiohttp

from lxml import etree

import pandas as pd

import logging

import datetime

import openpyxl

import random

wb = openpyxl.Workbook()

sheet = wb.active

sheet.append(['房源', '房子信息', '所在区域', '单价', '关注人数和发布时间', '标签'])

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s: %(message)s')

start = datetime.datetime.now()

class Spider(object):

    def \_\_init\_\_(self):

        self.semaphore = asyncio.Semaphore(1)  *# 信号量，控制协程数，防止爬得过快被反爬*

        self.delay\_range = (0.5, 3)  *# 随机延时的范围*

        self.header = {

            "Host": "bj.lianjia.com",

            "Referer": "https://bj.lianjia.com/ershoufang/",

            'Cookie': 'lianjia\_uuid=0a245c01-0249-4548-a9de-66e6367fdd82; \_ga=GA1.2.1314715887.1743086781; \_ga\_EYZV9X59TQ=GS1.2.1743086782.1.1.1743086797.0.0.0; \_ga\_DX18CJBZRT=GS1.2.1743086782.1.1.1743086797.0.0.0; \_ga\_BKB2RJCQXZ=GS1.2.1743088349.1.0.1743088349.0.0.0; Qs\_lvt\_200116=1743090960; Qs\_pv\_200116=3209232641914898400%2C2610558853758770700%2C2860350900287569000%2C2313803583990493000; \_ga\_E91JCCJY3Z=GS1.2.1743090925.1.1.1743091086.0.0.0; \_ga\_MFYNHLJT0H=GS1.2.1743090925.1.1.1743091086.0.0.0; Hm\_lvt\_46bf127ac9b856df503ec2dbf942b67e=1743086769,1743327950; HMACCOUNT=37989191F901F592; \_jzqc=1; \_qzjc=1; \_gid=GA1.2.1869468630.1743327970; crosSdkDT2019DeviceId=-2gp41z-o3lzyu-63pkn8jey4ausm7-e52x58qto; sensorsdata2015jssdkcross=%7B%22distinct\_id%22%3A%22195d81156b0581-08cc2e949a02e-4c657b58-1821369-195d81156b125a3%22%2C%22%24device\_id%22%3A%22195d81156b0581-08cc2e949a02e-4c657b58-1821369-195d81156b125a3%22%2C%22props%22%3A%7B%22%24latest\_traffic\_source\_type%22%3A%22%E8%87%AA%E7%84%B6%E6%90%9C%E7%B4%A2%E6%B5%81%E9%87%8F%22%2C%22%24latest\_referrer%22%3A%22https%3A%2F%2Fcn.bing.com%2F%22%2C%22%24latest\_referrer\_host%22%3A%22cn.bing.com%22%2C%22%24latest\_search\_keyword%22%3A%22%E6%9C%AA%E5%8F%96%E5%88%B0%E5%80%BC%22%7D%7D; select\_city=110000; \_jzqckmp=1; lfrc\_=7ac40290-a05f-4af6-a343-5cf68223992f; lianjia\_ssid=cef6d138-2a67-4902-825c-acca81cd9b18; hip=ZqJNHesc-q8M-cLKVXS1DczLesgAfOukX5k-TW536DddyCxdjzFacxTuoT6OFIL9zPCvPoLBTCWdqch3wo14TobVAlrneXVkD5LWjb5-nAG\_Okvr3CEe251SMxD8pkD4Vk97E8DHYOWUUedxHu-\_FUOV3EytI2WFqr0wcAcr75y\_KMth0Pay2WVBDA%3D%3D; \_jzqa=1.2936646379323121700.1743086770.1743413095.1743417476.5; \_jzqx=1.1743086770.1743417476.3.jzqsr=hip%2Elianjia%2Ecom|jzqct=/.jzqsr=bj%2Elianjia%2Ecom|jzqct=/; login\_ucid=2000000475594965; lianjia\_token=2.00143e639042ce0ea905934aa172e74f88; lianjia\_token\_secure=2.00143e639042ce0ea905934aa172e74f88; security\_ticket=kJTW0O78ZiEbLJdC52BMmhOVWkmk+JRY2VbTz+6M2fz1Vedzx57t0Cl+jZLbTTOORFvFDMaoSp6PYHT3+SQQLgTcNO41eMPcXvJsnkTzR9Hqr2n5WPsbYyuvEOcmJosQmPuEdxqV3I/kaNM5vhlwEd9fjJFTlx5FHUZkfBZVb6E=; ftkrc\_=49b169b',

            'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36'

        }

    async def scrape(self, url):

        async with self.semaphore:

            delay = random.uniform(\*self.delay\_range)

            await asyncio.sleep(delay)  *# 延时*

            timeout = aiohttp.ClientTimeout(total=60)  *# 设置超时时间为60秒*

            async with aiohttp.ClientSession(headers=self.header, timeout=timeout) as session:

                try:

                    response = await session.get(url)

                    response.raise\_for\_status()  *# 检查请求是否成功*

                    return await response.text()

                except asyncio.TimeoutError:

                    logging.error(f"Timeout occurred for {url}")

                except aiohttp.ClientError as e:

                    logging.error(f"Request failed for {url} with error: {e}")

                return None

    async def scrape\_index(self, page):

        url = f'https://bj.lianjia.com/ershoufang/pg{page}/'

        text = await self.scrape(url)

        if text:

            await self.parse(text)

    async def parse(self, text):

        html = etree.HTML(text)

        lis = html.xpath('//\*[@id="content"]/div[1]/ul/li')

        for li in lis:

            try:

                house\_data = li.xpath('.//div[@class="title"]/a/text()')[0]  *# 房源*

                house\_info = li.xpath('.//div[@class="houseInfo"]/text()')[0]  *# 房子信息*

                address = ' '.join(li.xpath('.//div[@class="positionInfo"]/a/text()'))  *# 位置信息*

                price = li.xpath('.//div[@class="priceInfo"]/div[2]/span/text()')[0]  *# 单价 元/平米*

                attention\_num = li.xpath('.//div[@class="followInfo"]/text()')[0]  *# 关注人数和发布时间*

                tag = ' '.join(li.xpath('.//div[@class="tag"]/span/text()'))  *# 标签*

                sheet.append([house\_data, house\_info, address, price, attention\_num, tag])

                logging.info([house\_data, house\_info, address, price, attention\_num, tag])

*# 每次爬取完一页后保存文件*

                file\_name = 'house\_data\_8.xlsx'

                wb.save(file\_name)

                logging.info('Data saved to ' + file\_name)

            except IndexError:

                continue  *# 忽略空白或错误的房源数据*

    async def main(self, start\_page, stop\_page):

        scrape\_index\_tasks = []

        for page in range(start\_page, stop\_page + 1):

            scrape\_index\_tasks.append(asyncio.ensure\_future(self.scrape\_index(page)))

*# 执行所有页面爬取任务*

        loop = asyncio.get\_event\_loop()

        tasks = asyncio.gather(\*scrape\_index\_tasks)

        await tasks

if \_\_name\_\_ == '\_\_main\_\_':

    spider = Spider()

    asyncio.run(spider.main(start\_page=0, stop\_page=50))  *# 设置爬取页数为1到500*

    end = datetime.datetime.now()

    logging.info(f"爬取结束, 总耗时: {end - start}")

至此爬虫数据爬取部分基本完成。测试爬取速度为1.5w data/2~3h。

数据爬取完成后涉及去重，因此我调用了pandas库对excel文件进行操作，去掉完全重复的行，并返回给我去重后新生成的文件。Python代码如下：

*# \_\*\_ coding : utf-8 \_\*\_*

*# @Time : 2025/3/31 21:47*

*# @Author : 张振哲*

*# @File : data\_clean*

*# @Project : spider\_project*

import pandas as pd

def remove\_duplicate\_rows(input\_file, output\_file):

    """

    读取Excel文件，删除完全重复的行，保存结果到新文件

    :param input\_file: 输入文件名（xlsx格式）

    :param output\_file: 输出文件名（xlsx格式）

    """

*# 读取Excel文件*

    df = pd.read\_excel(input\_file)

*# 记录原始行数*

    original\_rows = df.shape[0]

*# 删除完全重复的行（保留第一次出现的行）*

    df.drop\_duplicates(inplace=True)

*# 保存处理后的数据到新文件*

    df.to\_excel(output\_file, index=False)

*# 统计信息*

    removed\_rows = original\_rows - df.shape[0]

    print(f"处理完成！共删除 {removed\_rows} 个重复行")

    print(f"原始行数: {original\_rows}，处理后行数: {df.shape[0]}")

    print(f"结果已保存到: {output\_file}")

*# 使用示例*

if \_\_name\_\_ == "\_\_main\_\_":

    input\_filename = "house\_data.xlsx"  *# 请修改为你的输入文件名*

    output\_filename = "house\_cleared.xlsx"  *# 请修改为你想要的输出文件名*

    remove\_duplicate\_rows(input\_filename, output\_filename)

## 04-01（周二）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 生成文件 | 开始爬取 | 终止爬取 | Time | Rawdata |
| house\_data\_0.xlsx | 03-31- 19:26:27 | ‏‎03-31-‏‎ 21:41:13 | 02:14:46 | 14941 |
| house\_data\_1.xlsx | 04-01- 0‏‎0:00:12 | ‏‎04-01- 0‏‎2:19:08 | 02:18:56 | 15031 |
| house\_data\_2.xlsx | 04-01- ‏‎6:18:41 | 04-01- 08:21:23 | 02:02:42 | 15031 |
| house\_data\_3.xlsx | 04-01-‏‎ 08:59:17 | ‏‎04-01- 10:52:38 | 01:53:21 | 15031 |
| house\_data\_4.xlsx | 04-01- 12:16:23 | 04-01- 14:10:25 | 01:54:02 | 15031 |
| house\_data\_5.xlsx | 04-01- 14:41:57 | 04-01- 16:29:38 | 01:47:41 | 14371 |
| house\_data\_6.xlsx | 04-01- 16:37:37 | ‏‎04-01- 20:07:36 | 03:29:59 | 18031 |
| house\_data\_7.xlsx | 04-02- 00:36:02 | ‏‎04-02- 8:28:15 | 07:52:13 | 30031 |

因为爬取得到的文件数量过多，因此03-31提供的仅支持单文件数据去重服务的python文件将不再使用；因此，支持多文件终端输入选定的多文件数据去重服务的python代码产出。代码如下：

import pandas as pd

import os

import shlex

def merge\_and\_deduplicate\_excel():

*# 获取用户输入的文件列表*

    file\_input = input("请输入要处理的Excel文件（支持多个文件，用空格分隔）：\n").strip()

    if not file\_input:

        print("未输入任何文件名！")

        return

    try:

        raw\_files = shlex.split(file\_input)

    except ValueError as e:

        print(f"输入解析失败：{e}")

        return

*# 验证文件有效性*

    valid\_files = []

    for f in raw\_files:

        if not f.endswith('.xlsx'):

            print(f"跳过非xlsx文件：{f}")

            continue

        if os.path.isfile(f):

            valid\_files.append(f)

        else:

            print(f"文件不存在：{f}")

    if not valid\_files:

        print("没有有效的xlsx文件可处理！")

        return

*# 读取所有数据（无表头模式）*

    dfs = []

    for file in valid\_files:

        try:

*# 关键修改：添加 header=None 参数*

            df = pd.read\_excel(file, engine='openpyxl', header=None)

            dfs.append(df)

            print(f"成功读取：{file}（{len(df)}行）")

        except Exception as e:

            print(f"读取失败【{file}】：{str(e)}")

    if not dfs:

        print("所有文件读取失败")

        return

    merged\_df = pd.concat(dfs, ignore\_index=True)

*# 关键修改：去重时保持所有列相同*

    original\_count = len(merged\_df)

    final\_df = merged\_df.drop\_duplicates(keep='first')

*# 保存结果（不保留列名）*

    output\_file = "house\_merged\_result.xlsx"

    final\_df.to\_excel(output\_file, index=False, header=False, engine='openpyxl')

    print("\n处理结果：")

    print(f"输入行总计：{original\_count}")

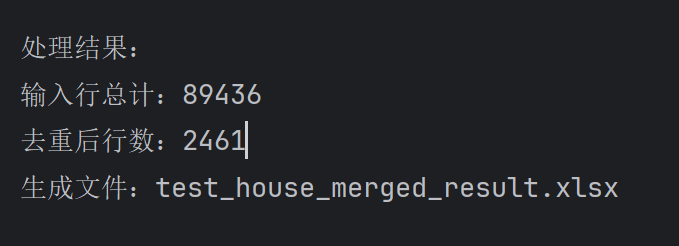
    print(f"去重后行数：{len(final\_df)}")

    print(f"生成文件：{output\_file}")

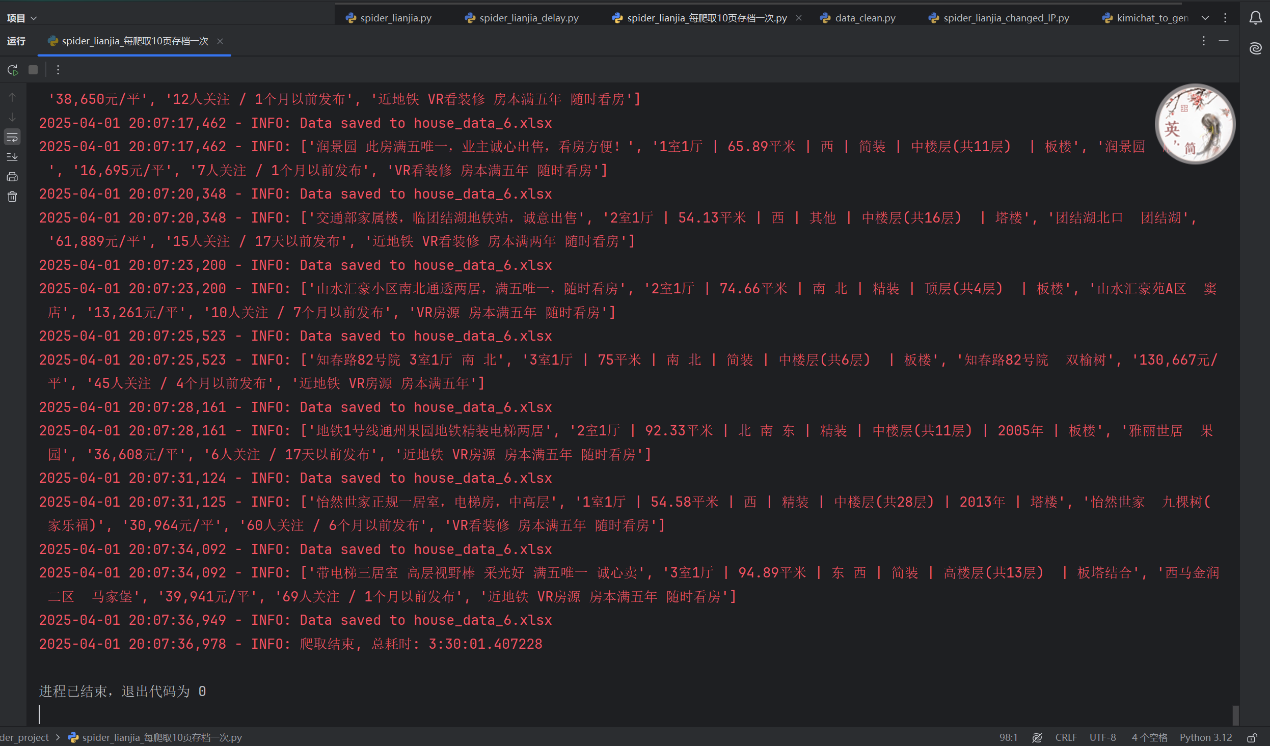
if \_\_name\_\_ == '\_\_main\_\_':

    merge\_and\_deduplicate\_excel()

04-01 17:43将house\_data\_0.clsx到hose\_data\_5.xlsx统一进行清洗去重，产量如下。



04-01晚18点至20点爬完第七组数据，保存至house\_data\_6中，如下图演示。



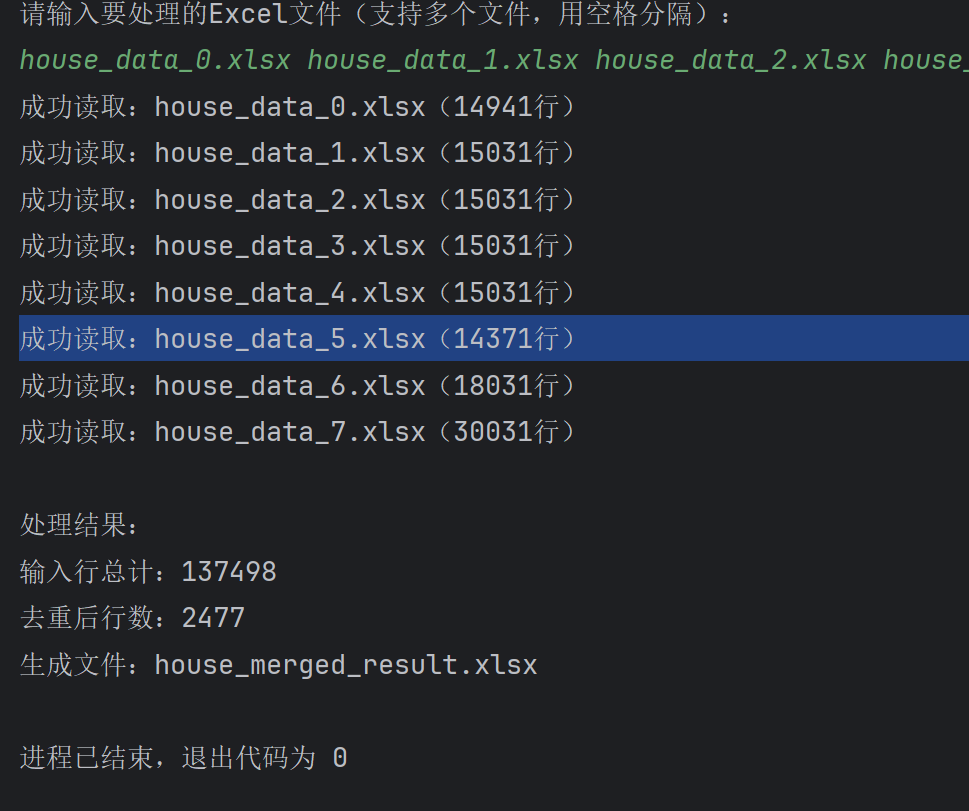
## 04-02（周三）

04-02凌晨00:40至04-02早8：28爬完第八组数据，保存至house\_data\_7.xlsx中。

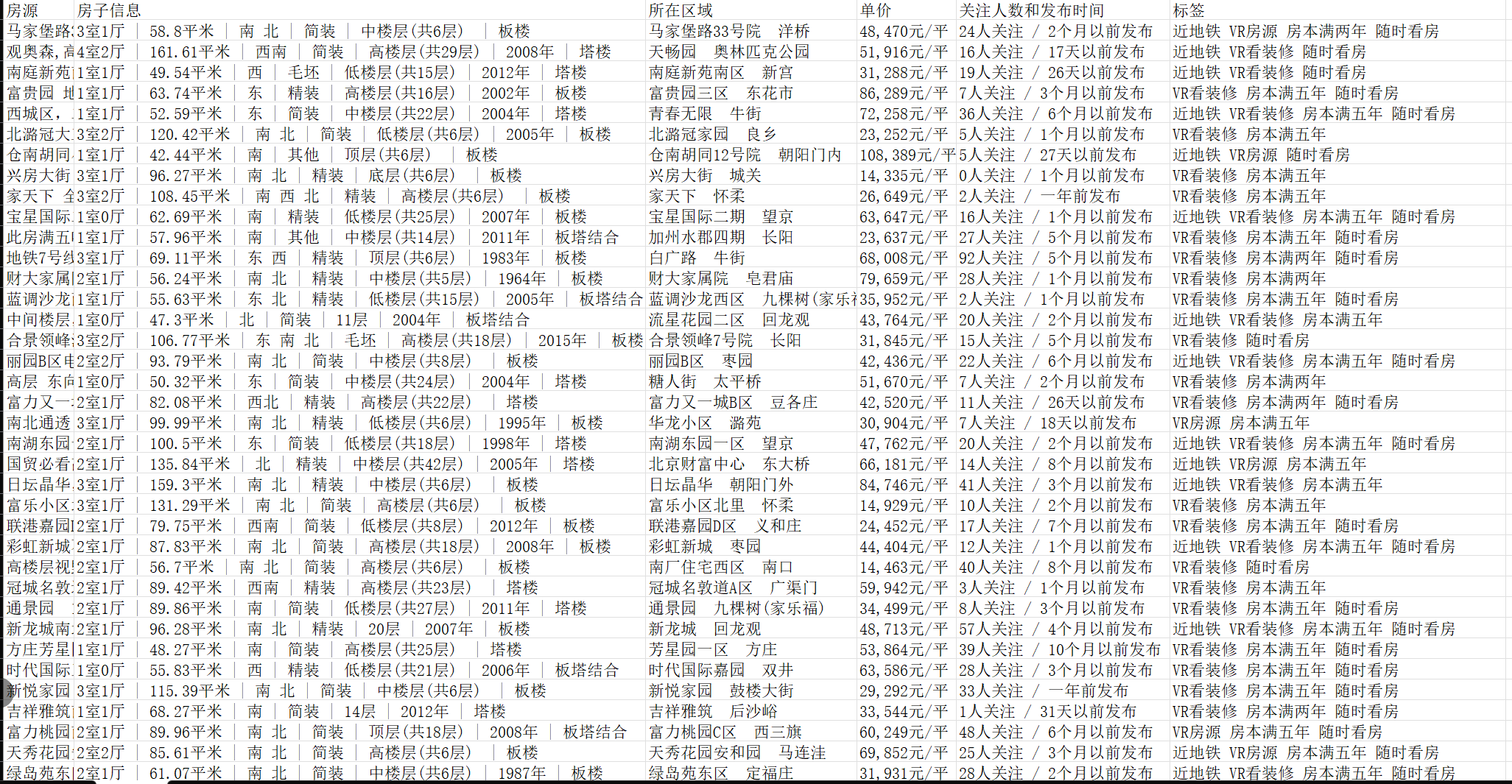
|  |  |  |  |
| --- | --- | --- | --- |
| 生成文件 | Rawdata | Per\_file\_Cooked\_data | Per\_file\_Yield |
| house\_data\_0.xlsx | 14940 | 2412 | 16.145% |
| house\_data\_1.xlsx | 15030 | 45 | 0.299% |
| house\_data\_2.xlsx | 15030 | 45 | 0.299% |
| house\_data\_3.xlsx | 15030 | 50 | 0.333% |
| house\_data\_4.xlsx | 15030 | 60 | 0.399% |
| house\_data\_5.xlsx | 14370 | 57 | 0.397% |
| house\_data\_6.xlsx | 18030 | 57 | 0.316% |
| house\_data\_7.xlsx | 30030 | 44 | 0.147% |

发现无论再怎么爬取，产率都会很低，这是由于链家网的爬取限制机制。因为爬取到的数据已经够用，所以决定不再爬取。

运行多文件处理的python代码，如下图。



去重后的数据如下图所示：



可以发现，同一列标签中还可以细分出很多不同的特征。

这是因为爬虫使用的是xpath，只能识别html的关键词，无法识别文本信息，所以无法在爬取到数据的同时就处理好数据，只能在爬取完成后对数据进行清洗。

由于机器学习进行数据分析时，需要细化特征，因此需要将去重后的excel的部分列按照观察到的特征进行提取和细化。

可以看出，负责数据处理的python代码应完成如下功能：

（1）读取读取excel文件，找到“房子信息”所在列，对该列进行处理：该列的每一行都对应某房源的很多信息。这些信息有一定的排布规律，按照顺序从前到后依次是布局、面积、朝向、装潢、楼层、年份、材质七个要素，他们中间用“|”分隔。其中，布局信息的关键字是“x室y厅”，其中x，y是数字，朝向信息的关键字是“东西南北”中的任一个或几个的组合，装潢信息的关键字是“毛坯”、“简装”、“精装”中的任一个，“楼层”信息只要包含“层”这个字，就可以判定为“楼层”信息，年份信息的关键字是“x年”，材质信息的关键字是“板楼”、“塔楼”、“板塔结合”中的任一个。 您要帮我实现的功能是，将“房子信息”所在列的每一行的要素分类，将“房子信息”所在列分隔分割成七列，依次的名称如上所述。不是“房子信息”所在列的每一行都有这七个要素，七个要素中的某一个或某几个可能存在缺省的情况，您需要根据我为您提供的关键字来识别该要素究竟是哪一类，并将这类信息填在对应列下面，如果发现某要素缺省，则该行该要素的格子为空。

细化前：

|  |
| --- |
| 房子信息 |
| 3室1厅 | 58.8平米 | 南 北 | 简装 | 中楼层(共6层) | 板楼 |
| 4室2厅 | 161.61平米 | 西南 | 简装 | 高楼层(共29层) | 2008年 | 塔楼 |

细化后：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 布局 | 面积 | 朝向 | 装潢 | 楼层 | 年份 | 材质 |
| 3室1厅 | 58.8平米 | 南北 | 简装 | 中楼层(共6层) |  | 板楼 |
| 4室2厅 | 161.61平米 | 西南 | 简装 | 高楼层(共29层) | 2008年 | 塔楼 |

特征：

|  |  |
| --- | --- |
| 细化特征 | 特征提取 |
| 布局 | X室y厅 |
| 面积 | 平米 |
| 朝向 | 东南西北中的任一个或几个的组合 |
| 装潢 | 毛坯/简装/精装 |
| 楼层 | 包含“层”字 |
| 年份 | 包含“年”字 |
| 材质 | 板楼/塔楼/板塔结合 |

（2）找到“关注人数/发布时间”列，并将本列分隔成两列，分别为“关注人数”和“发布时间”。

细化前：

|  |
| --- |
| 关注人数和发布时间 |
| 24人关注 / 2个月以前发布 |
| 16人关注 / 17天以前发布 |

细化后：

|  |  |
| --- | --- |
| 关注人数 | 发布时间 |
| 24人关注 | 2个月以前发布 |
| 16人关注 | 17天以前发布 |

（3）找到“标签”所在列，将该列重新分成五列，分别为“是否近地铁”，“VR房源”，“VR看装修”，“房本年限”，和“是否随时看房”；检测“标签”所在列的每一行，若出现“近地铁”字样，则在分隔后的“是否近地铁”列下对应行填入“近地铁”，否则填入空；同理，若出现“VR房源”字样，则在分割后的“VR房源”列下对应行填入“VR房源”，否则填入空；若出现“VR看装修”字样，则在分割后的“VR看装修”列下对应行填入“VR看装修”，否则填入空；如果出现“房本”字样，就可判断为“房本年限”，则若出现“房本满x年”（其中x是汉字或数字）字样，则在分割后的“房本年限”列下对应行填入“房本满x年”，否则填入空；若出现“随时看房”字样，则在分割后“是否随时看房”列下对应行填入“随时看房”，否则填入空。

细化前：

|  |
| --- |
| 标签 |
| 近地铁 VR房源 房本满两年 随时看房 |
| 近地铁 VR看装修 随时看房 |
| VR看装修 房本满五年 |

细化后：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 是否近地铁 | VR房源 | VR看装修 | 房本年限 | 是否随时看房 |
| 近地铁 | VR房源 |  | 房本满两年 | 随时看房 |
| 近地铁 |  | VR看装修 |  | 随时看房 |
|  |  | VR看装修 | 房本满五年 |  |

特征：

|  |  |
| --- | --- |
| 细化特征 | 特征提取 |
| 是否近地铁 | 有“近地铁”字样 |
| VR房源 | 有“VR房源”字样 |
| VR看装修 | 有“VR看装修”字样 |
| 房本年限 | 包含“房本”字样 |
| 是否随时看房 | 有“随时看房”字样 |

根据上述需求，可以使用pandas库和re库，前者用来操作excel，后者用来对字符串进行操作。

该过程的难点并不是字符串分割，而是分割之后的字段如何精准地匹配到正确的细化特征栏下。Re库中的方法很好地解决了这个问题，通过split方法识别和分割“|”和“/”，通过re.search方法解决“包含字样”类细化，通过part—in方法解决“单选或多选”类细化。

下面是完整的实现代码。

import pandas as pd

import re

*# 定义一个函数来处理“房子信息”列*

def process\_house\_info(row):

*# 定义关键字和对应的列名*

    keys = ["布局", "面积", "朝向", "装潢", "楼层", "年份", "材质"]

*# 初始化一个字典来存储结果*

    result = {key: "" for key in keys}

*# 如果“房子信息”列为空，直接返回空字典*

    if pd.isna(row["房子信息"]):

        return result

*# 按“|”分割“房子信息”列*

    parts = row["房子信息"].split("|")

    for part in parts:

        part = part.strip()

*# 检查每个部分属于哪个类别*

        if re.search(r"\d室\d厅", part):

            result["布局"] = part

        elif re.search(r"\d+\.\d+平米", part):

            result["面积"] = part

        elif re.search(r"[东西南北]", part):

            result["朝向"] = part

        elif part in ["毛坯", "简装", "精装"]:

            result["装潢"] = part

        elif re.search(r"层", part):

            result["楼层"] = part

        elif re.search(r"\d+年", part):

            result["年份"] = part

        elif part in ["板楼", "塔楼", "板塔结合"]:

            result["材质"] = part

    return result

*# 定义一个函数来处理“关注人数/发布时间”列*

def process\_attention\_time(row):

    if pd.isna(row["关注人数和发布时间"]):

        return {"关注人数": "", "发布时间": ""}

    parts = row["关注人数和发布时间"].split(" / ")

    if len(parts) == 2:

        return {"关注人数": parts[0], "发布时间": parts[1]}

    else:

        return {"关注人数": "", "发布时间": ""}

*# 定义一个函数来处理“标签”列*

def process\_tags(row):

*# 初始化结果字典*

    result = {

        "是否近地铁": "",

        "VR房源": "",

        "VR看装修": "",

        "房本年限": "",

        "是否随时看房": ""

    }

    if pd.isna(row["标签"]):

        return result

*# 检查每个标签并填入对应列*

    if "近地铁" in row["标签"]:

        result["是否近地铁"] = "近地铁"

    if "VR房源" in row["标签"]:

        result["VR房源"] = "VR房源"

    if "VR看装修" in row["标签"]:

        result["VR看装修"] = "VR看装修"

*# 检查是否包含“房本满x年”（其中x是汉字或数字）*

    pattern = r"房本满[\d\u4e00-\u9fa5]+年"

    match = re.search(pattern, row["标签"])

    if match:

        result["房本年限"] = match.group()

    if "随时看房" in row["标签"]:

        result["是否随时看房"] = "随时看房"

    return result

*# 读取Excel文件*

def process\_excel(file\_path, output\_path):

    df = pd.read\_excel(file\_path)

*# 处理“房子信息”列*

    house\_info\_df = df.apply(process\_house\_info, axis=1, result\_type="expand")

    house\_info\_df.columns = ["布局", "面积", "朝向", "装潢", "楼层", "年份", "材质"]

*# 处理“关注人数/发布时间”列*

    attention\_time\_df = df.apply(process\_attention\_time, axis=1, result\_type="expand")

    attention\_time\_df.columns = ["关注人数", "发布时间"]

*# 处理“标签”列*

    tags\_df = df.apply(process\_tags, axis=1, result\_type="expand")

    tags\_df.columns = ["是否近地铁", "VR房源", "VR看装修", "房本年限", "是否随时看房"]

*# 合并处理后的数据*

    result\_df = pd.concat([df, house\_info\_df, attention\_time\_df, tags\_df], axis=1)

    result\_df.drop(columns=["房子信息", "关注人数和发布时间", "标签"], inplace=True)

*# 保存到新的Excel文件*

    result\_df.to\_excel(output\_path, index=False)

*# 调用函数处理文件*

process\_excel("house\_merged\_result.xlsx", "data\_divided\_result.xlsx")

该代码将初加工（去重后）的数据进行了二次清洗，最终保存在了data\_divided\_result.xlsx中。

至此，数据清洗完成。

## 04-03及后

尝试解决爬虫产率低的问题，尝试使用无头浏览器参与爬取，对代码进行进一步优化。