Mathematica 在网络爬虫中的应用实验报告

王郅成

19335202

引言

最近的计算机行业，人工智能及大数据风头正盛。而在当今的大数据时代，做任何价值分析的前提是数据，而爬虫则是获得这个前提的一个低成本高收益手段。网络爬虫（又被称为网页蜘蛛，网络机器人），是一种按照一定的规则，自动地抓取万维网信息的程序或者脚本。放大的说，搜索引擎也是网络爬虫的一种。对于个人来说，我们可以运用爬虫，获取大量数据，方便自己生活，比如聚合平台整合信息进行比较、爬取所需要的数据进行统计。

0 法律问题

***以防误解，将申明放于此处

爬虫本身不违法，如果是爬的公开数据的话，与浏览器访问相比没有本质区别。

爬虫目前还处于不明确的蛮荒阶段，“允许哪些行为”这种基本秩序还处于建设中。至少目前来看，如果抓取的数据为个人所用，则不存在问题；如果数据用于转载，那么抓取数据的类型就很重要。

本文中的实验获取的都是公开数据，都仅做个人实验使用。

1 问题描述

名称：Mathematica 在网络爬虫中的应用

目的：使用 Mathematica 进行网络爬虫等应用分析

2 实验所需知识

Mathematica、Python(Selenium、BeautifulSoup)、HTML、Http、CSV

3 实验环境

Windows 系统，安装 Mathematica12.1、Python3.8（包含 Selenium 及 BeautifulSoup 等模块）、Firefox 及 geckodriver

4 实验一

仅用 Mathematica 实现爬虫，爬取豆瓣流浪地球电影海报

4.1 实验实现

4.1.1 网页分析

豆瓣网页：

https://movie.douban.com/subject/26266893/photos?type=R

图片如此排列，分多页展示：

6213x9048
正式海报 北美澳新

1080x1920
正式海报 中国大陆

1080x1920
正式海报 中国大陆 2回应

<前页 1 2 3 4 5 6 7 8 9 10 后页> (共292张)

4.1.2 源文件导入

首先分析网页源文件，需要获得源文件，Mathematica 有两种方法，一个是 Import 导入 html，一个是 URLFetch，但是 Import 导入的 html 会自动排版导致丢失图片的信息，后者 URLFetch 返回的就是原封不动的纯文本源代码，并未结构化，难以进行处理及查找。查看帮助文档，Mathematica 有一种变量 XMLElement，可以结构化源文件。还是通过 Import，但是需要用 XMLObject 转换。

获取源文件代码：

```
st = Import["https://movie.douban.com/subject/26266893/photos?type=R",
     导入
  "XMLObject"]
```

输出：（截取开头部分）

```
Out[48]= XMLObject[Document][{XMLObject[Declaration][Version → 1.0, Standalone → yes]}
       XMLElement[html, {class → , lang → zh-CN, {http://www.w3.org/2000/xmlns/, xml
        {XMLElement[head, {}, {XMLElement[meta, {name → renderer, content → webkit},
           XMLElement[meta, {name → google-site-verification, content → ok0wCgT20tBE
          流浪地球 海报
        }], XMLElement[meta, {name → baidu-site-verification, content → cZdR4xxR7RxmM4
             {}], XMLElement[meta, {http-equiv → Expires, content → Sun, 6 Mar 2005 0
           XMLElement[meta, {name → description, content → 流浪地球电影海报欣赏,高清
```

4.1.3 HTML 分析

使用浏览器工具，检查图片元素：

```
▼<li data-id="2547712073">
  ▼<div class="cover">
    ▼<a href="https://movie.douban.com/photos/photo/2547712073/">
       <img src="https://img2.doubanio.com/view/photo/m/public/p2547712073.webp"> == $0
     </a>
   </div>
   <div class="prop">
                  6213x9048
               </div>
   <div class="name">
                     正式海报 北美澳新
                  </div>
  </li>
▶<li data-id="2548752099">…</li>
▼<li class="last" data-id="2548076368">
  ▼<div class="cover">
    ▼<a href="https://movie.douban.com/photos/photo/2548076368/">
       <img src="https://img1.doubanio.com/view/photo/m/public/p2548076368.webp">
```

图片的链接在/li　/div　/a　/img 的 src 中

像这个图片的链接就为：

https://img2.doubanio.com/view/photo/m/public/p2547712073.webp

4.1.4 XMLElement 匹配获取图片链接

```
In[59]:= PicAddress = Cases[st,
                      模式匹配
          XMLElement["li", {"data-id" → ___},
          XML元素
            {___, XMLElement["div", {"class" → "cover"},
                  XML元素
              {___, XMLElement["a", {"shape" → "rect", "href" → ___},
                    XML元素
                {___, XMLElement["img", {"src" → src_}, {}], ___}], ___}], ___}] :→ src,
                      XML元素
          {0, Infinity}]
              无穷大
```

代码里面大量的“___“通配号，是因为 Mathematica 会把换行符及空格等东西全部保存下来，若没考虑到这些，匹配一般都会失败。

这里因为豆瓣获取原图需要登录账号，mathematica 无法操作，所以只能获取一个压缩过后的图片。

匹配结果：

```
Out[59]= {https://img2.doubanio.com/view/photo/m/public/p2545472803.jpg,
          https://img9.doubanio.com/view/photo/m/public/p2545553064.jpg,
          https://img9.doubanio.com/view/photo/m/public/p2547575075.jpg,
          https://img9.doubanio.com/view/photo/m/public/p2542316954.jpg,
          https://img3.doubanio.com/view/photo/m/public/p2626714671.jpg,
          https://img9.doubanio.com/view/photo/m/public/p2547115834.jpg,
          https://img9.doubanio.com/view/photo/m/public/p2547115835.jpg,
          https://img3.doubanio.com/view/photo/m/public/p2542316941.jpg,
          https://img1.doubanio.com/view/photo/m/public/p2542316947.jpg,
          https://img1.doubanio.com/view/photo/m/public/p2547115829.jpg,
          https://img9.doubanio.com/view/photo/m/public/p2542316944.jpg,
          https://img9.doubanio.com/view/photo/m/public/p2547761236.jpg,
          https://img9.doubanio.com/view/photo/m/public/p2546876215.jpg,
          https://img9.doubanio.com/view/photo/m/public/p2547761224.jpg,
          https://img2.doubanio.com/view/photo/m/public/p2549796462.jpg,
          https://img1.doubanio.com/view/photo/m/public/p2547761239.jpg,
          https://img9.doubanio.com/view/photo/m/public/p2548447525.jpg,
          https://img2.doubanio.com/view/photo/m/public/p2547761222.jpg,
          https://img2.doubanio.com/view/photo/m/public/p2547712073.jpg,
          https://img1.doubanio.com/view/photo/m/public/p2548752099.jpg}
```

获取了同一页面的链接

4.1.5 下一页读取

因为图片分成多页面展示，所以需要获取下一页的链接

豆瓣的下一页直接显示于 HTML 中：

```
<span class="thispage" data-total-page="10">1</span> == $0
<a href="https://movie.douban.com/subject/26266893/photos?type=R&start=30&sortby=like&size=a&subtype=a">2</a>
<a href="https://movie.douban.com/subject/26266893/photos?type=R&start=60&sortby=like&size=a&subtype=a">3</a>
<a href="https://movie.douban.com/subject/26266893/photos?type=R&start=90&sortby=like&size=a&subtype=a">4</a>
<a href="https://movie.douban.com/subject/26266893/photos?type=R&start=120&sortby=like&size=a&subtype=a">5</a>
<a href="https://movie.douban.com/subject/26266893/photos?type=R&start=150&sortby=like&size=a&subtype=a">6</a>
<a href="https://movie.douban.com/subject/26266893/photos?type=R&start=180&sortby=like&size=a&subtype=a">7</a>
<a href="https://movie.douban.com/subject/26266893/photos?type=R&start=210&sortby=like&size=a&subtype=a">8</a>
<a href="https://movie.douban.com/subject/26266893/photos?type=R&start=240&sortby=like&size=a&subtype=a">9</a>
<a href="https://movie.douban.com/subject/26266893/photos?type=R&start=270&sortby=like&size=a&subtype=a">10</a>
```

运用 Cases 进行操作

Cases 中进行递归，所以需要将主程序写成函数，这个之后会介绍。

```
NextPage =
 Cases[st, XMLElement["a", {___, "href" → src_, ___},
 |模式匹配    |XML元素
    {ToString[CurPage + 1]}] :→ src, {0, Infinity}];
     |转换为字符串                    |无穷大
If[Length[NextPage] ≠ 0, DownLoadDouBan[CurPage + 1, NextPage[[1]]],
|···  |长度
 Return]]
 |返回
```

4.1.6 图片试读取

Import 图片

```
In[52]:= Import[#] & /@ PicAddress[[1 ;; 3]]
         |导入
```

Out[52]=

尝试导入三张，测试成功

4.1.7 函数书写

```
In[57]:= DownLoadDouBan[CurPage_, URL_] :=
 Block[{st, PicAddress, rawpic, NextPage},
  (*读取网页XML*)
  st = Import[URL, "XMLObject"];
  (*获取图片URL*)
  PicAddress =
   Cases[st, XMLElement["li", {___, "data-id" → ___, ___},
     {___, XMLElement["div", {"class" → "cover"},
       {___, XMLElement["a", {"shape" → "rect", "href" → ___},
         {___, XMLElement["img", {"src" → src_}, {}], ___}], ___}],
     ___}] :> src, {0, Infinity}];
  Print[PicAddress];
  (*输出*)
  Export["C:/Users/predator/Downloads/pic/" <>
    StringSplit[#, "/"][[-1]], Import[#]] & /@ PicAddress;
  NextPage =
   Cases[st, XMLElement["a", {___, "href" → src_, ___},
     {ToString[CurPage + 1]}] :> src, {0, Infinity}];
  If[Length[NextPage] ≠ 0, DownLoadDouBan[CurPage + 1, NextPage[[1]]],
   Return[]]
```

基本分成三部分，分别为读取 XML，获取图片 URL，输出并导出，截图中为本机的下载目录下的 pic 文件夹。

函数设计方面，预留两个接口，分别是 Current Pages 以及 URL，表示页码以及网址。

使用 Block 赋初始值

4.2 实验结果

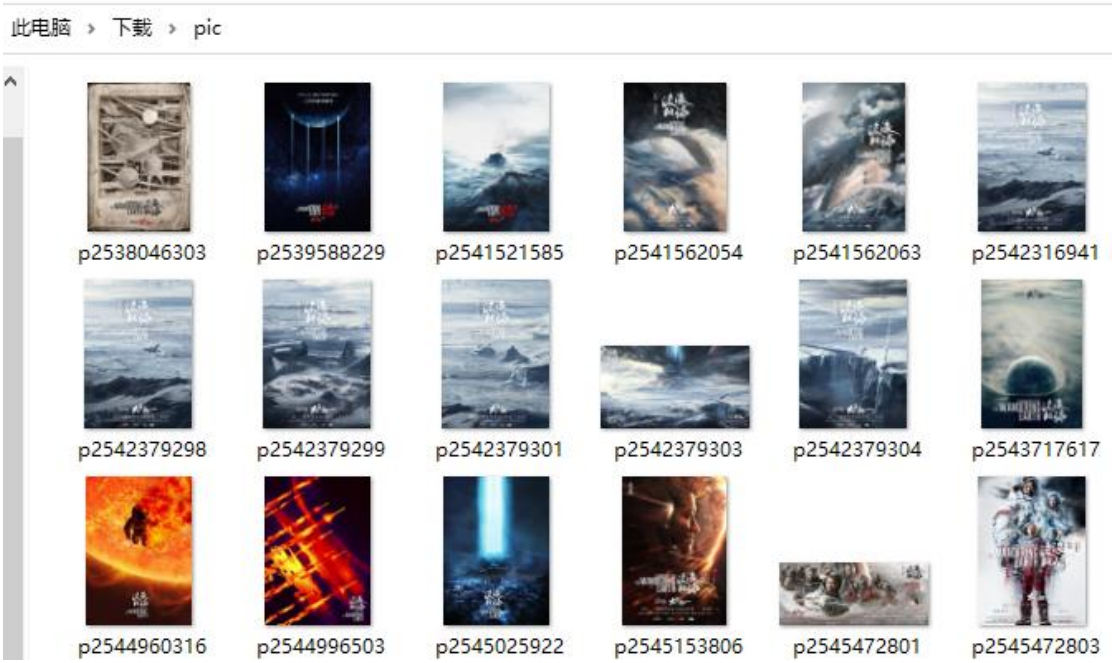
Mathematica 输出结果（局部截图）

```
In[58]:= DownLoadDouBan[1,
          "https://movie.douban.com/subject/26266893/photos?type=R"]

{https://img2.doubanio.com/view/photo/m/public/p2545472803.jpg,
 https://img9.doubanio.com/view/photo/m/public/p2545553064.jpg,
 https://img2.doubanio.com/view/photo/m/public/p2546876262.jpg,
 https://img9.doubanio.com/view/photo/m/public/p2547575075.jpg,
 https://img9.doubanio.com/view/photo/m/public/p2542316954.jpg,
 https://img2.doubanio.com/view/photo/m/public/p2542316943.jpg,
 https://img3.doubanio.com/view/photo/m/public/p2626714671.jpg,
 https://img9.doubanio.com/view/photo/m/public/p2547115834.jpg,
 https://img9.doubanio.com/view/photo/m/public/p2542316946.jpg,
 https://img9.doubanio.com/view/photo/m/public/p2547115835.jpg,
 https://img3.doubanio.com/view/photo/m/public/p2542316941.jpg,
 https://img3.doubanio.com/view/photo/m/public/p2547115830.jpg,
 https://img1.doubanio.com/view/photo/m/public/p2542316947.jpg,
 https://img1.doubanio.com/view/photo/m/public/p2547115829.jpg,
 https://img3.doubanio.com/view/photo/m/public/p2547761230.jpg,
 https://img9.doubanio.com/view/photo/m/public/p2542316944.jpg,
 https://img9.doubanio.com/view/photo/m/public/p2547761236.jpg,
 https://img2.doubanio.com/view/photo/m/public/p2542316952.jpg,
 https://img9.doubanio.com/view/photo/m/public/p2546876215.jpg,
 https://img9.doubanio.com/view/photo/m/public/p2547761224.jpg,
```

图片下载至文件夹中：（局部截图）

与之前一致



5 实验二

因为 mathematica 毕竟不是专业的爬虫工具，在爬虫方面有不少局限，比如点击按钮才能翻页的网页以及下翻网页才能加载的网页，这些网页不能经由 mathematica 处理，我在上学期学过 python 语言，就想通过 python 丰富的仓库来抓取数据，交由 mathematica 进行数据处理，毕竟 mma 的强项就在数据处理。

我尝试爬取的数据为艺龙网的广州番禺区的旅馆价格，从 2020-12-29 起 30 天的旅馆数据，为运行考量，仅取前五十页，如此也有近一万五千条有效数据。

5.1 Python 代码解释

报告主要是 mathematica 部分，这边的 python 部分我会挑重

点部分进行解释。因为初学爬虫，代码可能仍有缺陷，可读性不是特别强，请谅解。

代码主要调用 selenium，一个网页自动化工具。BeautifulSoup，HTML 文本处理工具。pandas，一个方便的数据结构库，其中的 Dataframe 适合存储表格类数据。

基础网页加载：

```
option = webdriver.FirefoxOptions()
#无头模式
option.add_argument("-headless")
#禁止加载css样式表
option.set_preference('permissions.default.stylesheet',2)
#timeou后停止网页加载
desired_capabilities = DesiredCapabilities.FIREFOX
desired_capabilities["pageLoadStrategy"] = "none"
#webdriver
driver = webdriver.Firefox(firefox_options=option)
#driver.set_page_load_timeout(8)
```

使用 headless、设置错误时间以及禁用 CSS 以加快加载速度，浏览器选用 firefox

页面打开：

```
driver.get("http://hotel.elong.com/")
time.sleep(3)

driver.find_element_by_class_name("input_text").clear()
driver.find_element_by_class_name("input_text").send_keys("番禺")
time.sleep(1)
driver.find_element_by_class_name("input_text.w170").clear()
driver.find_element_by_class_name("input_text.w170").send_keys(day1)
time.sleep(1)
driver.find_element_by_class_name("search_input").clear()
driver.find_element_by_class_name("search_input").send_keys(day2)
time.sleep(1)
try:
    driver.find_element_by_class_name("licon").click()
except:
    print("licon not found")
    one_day(day1,day2,lenth)
time.sleep(3)
```

通过搜索页面进入各个日期页面

页面向下拉并读取数据：

```
j = 0
while j < 2:
    js = "q=document.documentElement.scrollTop=40000"  # 将页面滚动条拖到底部
    driver.execute_script(js)
    time.sleep(0.5)
    print("scroll down")
    j = j + 1

soup = BeautifulSoup(driver.page_source, features="html.parser")
for name in soup.select('.info_cn'):
    temp_name.append(name.next.string)
for score in soup.select('.t20.c37e'):
    temp_score.append(score.next.string.replace(u'\xa0', u''))
for price in soup.select('.h_pri_num '):
    temp_price.append(price.next.string)
```

根据标签读取数据

模拟点击下一页按钮：

```
try:
    driver.find_element_by_class_name("page_next").click()
    time.sleep(2)
    i = i + 1
except:
    print("No button found")
```

失败返回错误信息重新尝试

根据日期调用爬虫函数：

```
d = {}

length = 1
in_date = '2020-12-29'
for date_num in range(30):
    print("day")
    print(length)
    dt = datetime.datetime.strptime(in_date, "%Y-%m-%d")
    out_date = (dt + datetime.timedelta(days=1)).strftime("%Y-%m-%d")
    one_day(in_date, out_date, length)
    in_date = out_date
    length = length + 1
```

输出成 csv 文件：

```
df.to_csv("test.csv",index=True,sep=',',encoding='utf_8_sig')
```

5.2 Mathematica 代码解释

导入：

```
In[1]:= data = Import["C:/Users/predator/Documents/test2.csv"]
          导入
```

结果（截取部分）：

In[60]:= `Grid[data]`

格子

Out[ ]=

		name	广州米兰小筑公寓	美朵精品公寓(广州南站店)
		date	1	2
1	2020/12/29	0	36	249
2	2020/12/30	1	56	263
3	2020/12/31	2	68	498
4	2021/1/1	3	68	466
5	2021/1/2	4	68	406
6	2021/1/3	5	68	406
7	2021/1/4	6	100	298
8	2021/1/5	7	100	298
9	2021/1/6	8	100	298
10	2021/1/7	9	100	298
11	2021/1/8	10	100	298
12	2021/1/9	11	100	298
13	2021/1/10	12	100	298
14	2021/1/11	13	100	298
15	2021/1/12	14	100	298
16	2021/1/13	15	100	298

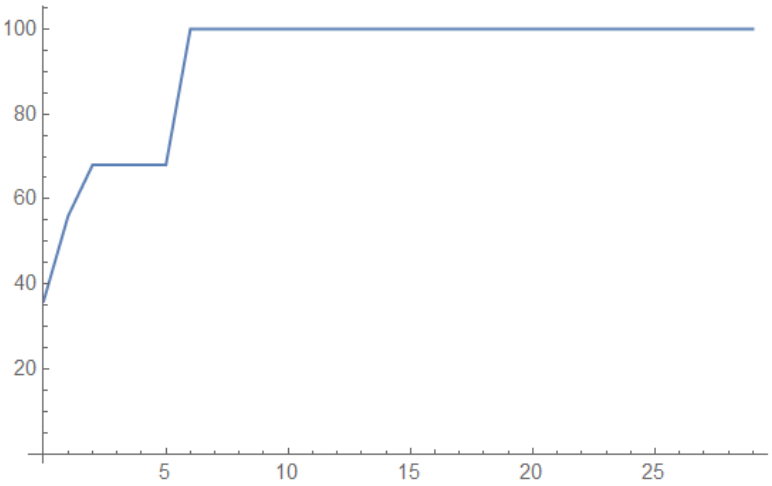
将数据切片，截取第一个酒店的数据并绘制图：

```
In[7]:= data2 = data[[3 ;; 32, 3 ;; 4]]

Out[7]= {{0, 36}, {1, 56}, {2, 68}, {3, 68}, {4, 68}, {5, 68},
  {6, 100}, {7, 100}, {8, 100}, {9, 100}, {10, 100}, {11, 100},
  {12, 100}, {13, 100}, {14, 100}, {15, 100}, {16, 100}, {17, 100},
  {18, 100}, {19, 100}, {20, 100}, {21, 100}, {22, 100}, {23, 100},
  {24, 100}, {25, 100}, {26, 100}, {27, 100}, {28, 100}, {29, 100}}

In[8]:= ListLinePlot[data2, AxesLabel → Automatic]
        绘制点集的线条        坐标轴标签    自动
```

Out[8]=



数据处理，求每天酒店价格的平均值：

```
In[23]:= ave = {};
         For[i = 3, i ≤ 32, i++,
         For循环
               temp1 = Flatten[data[[i ;; i, 4 ;;]]];
                       压平
               temp2 = DeleteCases[temp1, -1];
                       删除匹配元素
               temp3 = DeleteCases[temp2, 100000];
                       删除匹配元素
               tempmean = Mean[temp3];
                          平均值
               AppendTo[ave, tempmean]
               附加
         ]
```

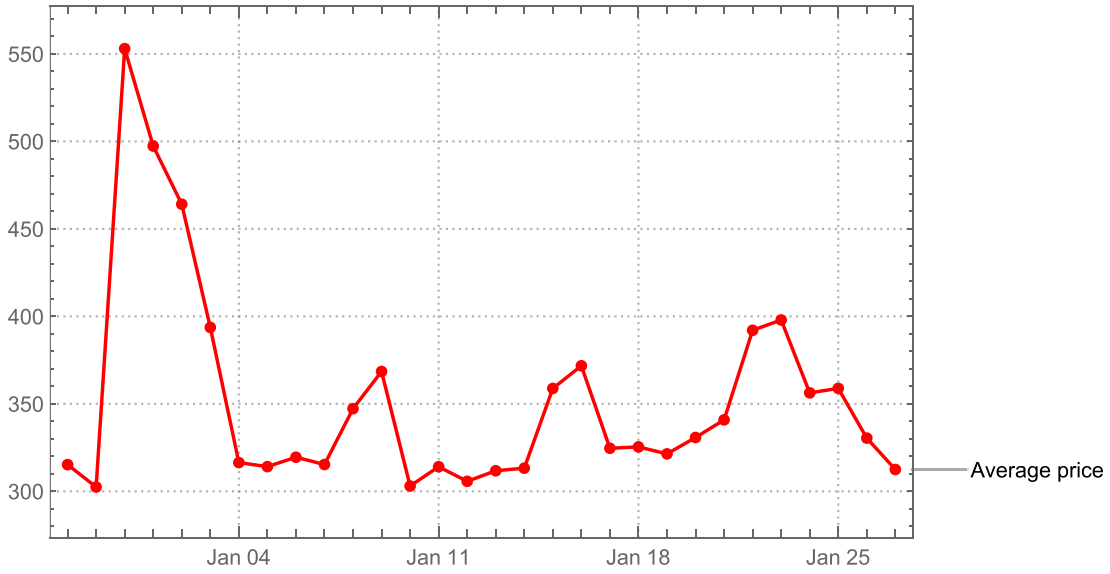
原数据中，-1 代表数据不存在，100000 表示已满房，求平均时需要删除这些数据，处理完的结果如下：

```
In[25]:= ave
Out[25]= {258823/821, 248923/823, 381032/689, 370019/744, 369890/797, 153144/389, 129422/409,
 255049/812, 263265/824, 85452/271, 69793/201, 290006/787, 60304/199, 246528/785, 242727/794,
 248475/797, 124039/396, 284171/792, 297028/799, 247993/764, 259011/796, 83243/259,
 256667/776, 131569/386, 293237/748, 309203/777, 68047/191, 280254/781, 83925/254, 239042/765}
```

绘图并加上 x 轴标签：

```
In[45]:= DateListPlot[ave, {2020, 12, 29}, PlotStyle → RGBColor[1.`, 0, 0],
          日期列表图                          绘图样式      RGB颜色
  PlotTheme -> "Detailed", PlotLabels → "Average price",
  绘图主题                 数据绘制标签
  PlotMarkers → Automatic, Axes -> {True, True},
  绘制点的标记   自动       坐标轴   真    真
  AxesLabel → {"date", "price"}]
  坐标轴标签
```

效果如下：



数据分析：

12 月 31 日至 1 月 3 日的元旦期间，价格都远超平日，而周末（周

五周六)的价格都较平时高,随着春节临近,每周价格也在缓慢上涨,这样的数据符合我们的直觉,可以说明实验数据是正确的。同时,这样的数据也为我们的出行提供参考,在有空闲的前提下,出行应当尽量避开周末及节假日,并且考虑春节的因素,应当尽早出行。

在数据处理方面,我曾试过用 excel 处理此次数据,其公式填写以及异常分析、图像生成均远远比 mma 复杂,而 mma 的代码行,对于程序员来说更为直观,可读性远高于 excel 等文件,图表等生成也拥有更多的自定义空间,可以说,mma 是处理爬虫数据的一个极好的工具。

6 实验总结及感想

本次实验完成了实验目的,给出了两个案例,一个全部使用 mma,一个将 Python 与 mma 结合使用,各自有其使用场景。

其实当知道需要做一个贴合专业的实验项目的时候我是很茫然的,作为大二学生,专业课并未上过很多,更多内容偏向基础,不知道具体可以做什么。最终选定这个题目,主要是我之前学过 Python,想着将 Python 与 mathematica 相结合,而 mathematica 适合处理数据,而 Python 的重要应用之一爬虫可以获得大量数据,于是就想做一个这样的项目。我之前并未接触过爬虫,基本是从零开始学习,经过几周的学习,总算能写出一个像样的代码,还是很有成就感的。我先做了实验二,然而当我去查了官方的帮助文档和 Github 上的某些项目时,我惊喜的发现可以只靠 Mathematica 完成一些简单网页的数据抓取,

于是又写了实验一。在这实验过程中，最让我惊喜的是 mma 的帮助文档，这个文档基本是我见过最为细致的，最为易于使用的帮助文档，我的整个实验都在不停的查询文档，其中很多列出来的元素都能给人不小的惊喜。虽然网上 mma 的教程等的中文资源并不丰富，但靠着自带帮助你也能完成几乎所有工作。除此之外，MMA 中海量的函数也给我留下深刻的印象，通过这些函数，我们可以用极快的速度绘制出一张图片。

在本文的实验中，虽然数据条数已经在一万五左右，但因为设备所限，不给网页服务器造成过大压力，数据仍不全面，仅收集了番禺区的部分酒店，时间也只有 30 天，如果能做个长期的收集，记录不同日期的同一天同一酒店价格，以及不同房型的价格，还可以收集广州各个区的数据进行横向对比，甚至比较不同城市的价格，这样的话数据将更有意义。

7 参考资料

[1] https://reference.wolfram.com/language/

Wolfram 语言与系统参考资料中心

[2] https://selenium-python.readthedocs.io/

Python Selenium 帮助文档

[3]https://www.crummy.com/software/BeautifulSoup/bs4/doc.zh/

Python BeautifulSoup 帮助文档