**用Python抓取最便宜的机票信息**

陈瑞琪 2018210764 2019.6.27

**摘要**：当我们准备出去旅游时会考虑什么，目的地？时间？价格？本论文将探讨如何在目的地与触发点已知的情况下寻找一定时间内最便宜的机票信息。

web抓取一直是一个很有意思的东西，在不断的学习中会慢慢的了解平时使用互联网的内侧，这种不断揭开秘密的过程如儿时探险般有趣。搜索机票的过程一直不是一个有趣的过程，明明已经准备好了一切，脑内已经形成完美的计划，但就是找不到合适的机票，价钱时间等等信息的收集都需要我们打开数个网页对比价格，看相关规定。懒一直是驱动人们进步的一大动力，而python几乎可以帮我们做到互联网上的任何事，既然有了好的工具，制作一个可以将最便宜的机票信息爬取下来的程序的想法自然就浮现出来。

**关键词**：爬虫，webdriver，邮件，表格

首先选择爬网页的工具，这里用比较灵活的webdriver做为我们爬网页的工具，它可以通过python对浏览器做任何事。然后是选择网站，这里选择国内的携程作为我们爬的网站，它的服务器架在国内，访问时加载快，信息全，且不用考虑验证码的问题，唯一缺点是它没有提供城市与对应缩写，需要我们对城市和其缩写进行爬取才能做到交互，这里就体现出webdriver相比其他模块的优势了。表格用pandas的矩阵进行存取，生成的文件通过smtplib与email模块进行发送。

# 1.**Selenium 与 ChromeDriver**

Selnium是webdriver的前身，它是 ThoughtWorks 提供的一个强大的基于浏览器的开源自动化测试工具，又叫Selenium 1。  
 Selenium 是一个用于 Web 应用程序测试的工具，测试直接自动运行在浏览器中，就像真正的用户在手工操作一样。支持的浏览器包括 IE、Chrome 和 Firefox 等。这个工具的主要功能包括：测试与浏览器的兼容性 - 测试您的应用程序看是否能够很好地工作在不同浏览器和操作系统之上；测试系统功能 - 创建回归测试检验软件功能和用户需求；支持自动录制动作，和自动生成 .NET、Perl、Python、Ruby 和 Java 等不同语言的测试脚本。

Selenium 2，又名 WebDriver，它的主要新功能是集成了 Selenium 1.0 以及 WebDriver， 是两个项目的合并，既兼容 Selenium API 也支持 WebDriver API。

WebDriver 曾经是 Selenium 的竞争对手（最开始是google的一个人弄的，主要用于避免在JavaScript的沙箱环境里存在的各种限制），他主要是通过利用浏览器原生API的方式来操控浏览器执行各种动作（还包括系统级别的调用来模拟用户输入）。

Selenium WebDriver 就是对浏览器提供的原生API进行封装，使其成为一套更加面向对象的Selenium WebDriver API。  
使用这套API可以操控浏览器的开启、关闭，打开网页，操作界面元素，控制Cookie，还可以操作浏览器截屏、安装插件、设置代理、配置证书等。由于使用的原生API，其速度与稳定性都会好很多。

在代码中 new ChromeDriver() 时，selenium会随机挑选一个端口调用chromedriver程序，调用成功后 chromedriver 会在指定的端口启动一个服务（会有一个进程）。

|  |
| --- |
| #填入chromedriver的路径，一般在C:\Program Files (x86)\Google\Chrome\Application\,如果没有需下载  chromedriver\_path = 'C:\Program Files (x86)\Google\Chrome\Application\chromedriver.exe'  driver = webdriver.Chrome(executable\_path=chromedriver\_path) # 打开chrome窗口 |

在打开窗口后就可以通过模块中的函数对浏览器进行操作了。

**（0）获取城市的字母缩写**

携程没有提供关于城市的字母缩写，而访问携程的机票查询需要城市的字母缩写，如果想要在用户交互过程中输入城市即可得到信息，需要预先写一个爬字母与城市关系的程序。



如图所示，在城市的扩展表中没有关于城市字母缩写的相关信息，反而在上面的输入框，所以要想爬这个信息，就必须点击城市扩展表中的每一个城市，再在输入框中获取信息。如果是单纯的request是无法得到这个信息的，这几个点击操作是javascript模块而不是链接，无法通过重复访问的方式不断使输入框的信息更新，而webdriver完美的解决了这一点。用于控制网页的模块自然可以简单的操作浏览器自动通过点击来不断更新数据，来达到获取城市字母缩写的需求，模拟浏览器的行为让一切都变得简单。

首先第一步是要通过webdriver打开携程的网页，这里城市和时间不做要求，只是爬取城市字母缩写，只要打开的是机票查询的网页就可以做到。

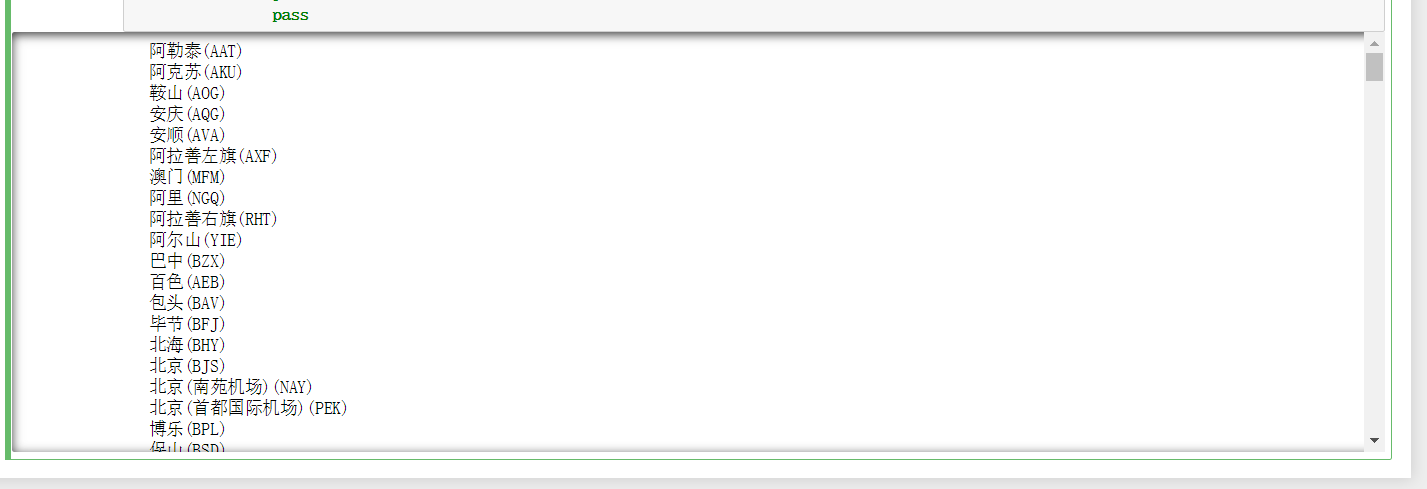
|  |
| --- |
| #随便打开一个机票查询的网页  kayakURL='https://flights.ctrip.com/itinerary/roundtrip/bjs-sha?date=2019-07-03,2019-07-18'  driver.get(kayakURL)  #睡眠防止网速太慢导致无法读取到元素  sleep(randint(4,5)) |

别忘了加上sleep，如果没有的话会因为网速太慢导致无法读取到元素。

|  |
| --- |
| #三次循环分别对应：ABCD的字母大引索框，ABCD的字母小引索框，每个首字母的所有城市  for awww in range(2,7):  for aww in range(1,10):  for aw in range(1,30):  try:  caa='//\*[@id="dcity0"]'#输入框的id  driver.find\_element\_by\_xpath(caa).click()  cheap\_paice='//\*[@id="address\_hot"]/div[2]/ol/li['+str(awww)+']/span'  driver.find\_element\_by\_xpath(cheap\_paice).click()  aaa='//\*[@id="address\_hot"]/div[2]/dl/div['+str(aww)+']/dd/a['+str(aw)+']'  driver.find\_element\_by\_xpath(aaa).click()  awe='//\*[@id="dcity0"]'  awe2=driver.find\_element\_by\_xpath(awe)  print(awe2.get\_attribute("value"))  sleep(1)  except:  pass |

三次循环分别对应：ABCD的字母大引索框，ABCD的字母小引索框，每个首字母的所有城市，然后只需找到输入框的xpath（由于只是简单的读取，且id不会变化，所有可直接在chrome的检查中复制当前元素的xpath），控制浏览器点击，会弹出城市的扩展表，控制浏览器点击城市的首字母，控制浏览器点击城市名，然后通过循环不断重复这个过程。其中通过range生成的列表进行循环，不断变换xpath的路径以到达逐个点击的功能。

但是这样会又一个大问题，就是每个字母大引索框的内部元素数量和字母小引索框的内部元素数量都不同，如果range范围设的太小会有几个城市读不到，太大会使个别找不到xpath使程序报错，这里的解决方案是区最大的，然后跳过错误，这样就可以读到所有城市的缩写字母了



在之后的用户输入会用到它。（**注：请勿在获取信息途中对打开的浏览器进行任何操作，否则会出现无法对信息进行检测的情况，这时程序不会报错且不会关闭，只能重新打开jpnodebook重启程序**）

**（0.5）用户交互与信息处理**

通过对产品的界面和行为进行交互设计，让产品和它的使用者之间建立一种有机关系，从而可以有效达到使用者的目标。在用户输入城市名就能访问网页而不是打开携程查看城市对应的字母之后输入是我们需要的，如果需要打开网页的话为什么还需要python呢？

在上个程序中我们已经得到了城市和其对应的字母，现在需要我们将其中的信息提取出来。先确定信息的储存结构，输入城市名得到对应的字母，自然让人想到用字典储存。

然后确定处理方式，我将城市和其对应的字母保存在一个txt文档中，它的特征信息是：城市（城市对应的字母）\n，且如果将文档读出的话为字符串形式，所以选择字符分割的方式对信息进行处理。通过对')\n'的分割可将每个城市分开，再对每个城市通过')'进行分割，这样就得到了一个二维的列表，然后再将其一一录入字典中就完成了。（注：由于有两个城市中有数个机场，它们以括号区分，与上述处理方式产生了冲突，由于产生异常的数据很少，且数据可以容易改动，所以在文件中将他们的不同机场改为通过中括号[]进行区分，排掉了冲突）

|  |
| --- |
| #搜索城市的id  def cityid\_search(cityname):  global password#赋值操作是局部的，所以对全局变量进行修改时需要声明  data0=open('XC\_data.txt','r')  data1=data0.read()  data0.close()  data2=data1.split(')\n')#特征信息是：城市（城市对应的字母）\n  data3=[]  for dd in data2:  data3.append(dd.split('('))#对每个城市通过')'进行分割  data6={}  for dd in data3:  data6.update({dd[0]:dd[1]})  data6['遵义']='ZYI'#最后的括号没有\n，单独拿出修改  try:  data7=data6[cityname]  except KeyError:  print("\n警告！！！所输入城市错误或当地没有机场\n")  password=1  return 0  return data7 |

当然不要忘了加上异常处理，通过password判读是否出错，如果出错直接driver.quit()将网页关闭。

|  |
| --- |
| global password#赋值操作是局部的，所以对全局变量进行修改时需要声明  print('网页加载中不可对页面进行任何操作，否则会使程序出错')  city\_out=cityid\_search(input("请输入起点城市名称（注：若想同一城市不同机场出发，请用‘[]’进行注释，例：北京[南苑机场]）"))  city\_to=cityid\_search(input("请输入终点城市名称（注：若想同一城市不同机场出发，请用‘[]’进行注释，例：北京[南苑机场]）"))  if(password==1):  return  date\_out=input("请输入可选出发时间的最早日期（例：2019-07-18）")  date\_return=input("请输入可选出发时间的最晚日期（例：2019-07-21）") |

输入操作，获取出发的地点等信息

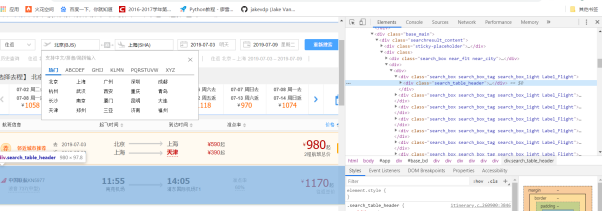
1. **网页的打开与爬取**

漫长准备过程终于结束了，现在我们终于可以通过webdriver爬取信息了，首先还是打开网页

|  |
| --- |
| #打开网页  def web\_open(date\_out,city\_out,city\_to,date\_return):  global password#赋值操作是局部的，所以对全局变量进行修改时需要声明  kayakURL='https://flights.ctrip.com/itinerary/oneway/'+city\_out+'-'+city\_to+'?date='+date\_out  try:  driver.get(kayakURL)  print('正在打开页面，请勿对打开的浏览器进行任何操作')  except:  print('\n警告！！！时间输入错误\n')  password=1  return  sleep(randint(4,8)) |

别忘了sleep和异常处理。

打开网页后可以对信息进行爬取了，这里用driver.find\_elements\_by\_xpath对多个对象进行爬取，在选择xpath时请不要复制，而是直接写class，复制的id会在下一次打开网页变化，class即使网站更新也基本不会变化，然后将每一个对象的text进行读取，再通过'\n'进行分割，基本数据就可以获得了。



**选择class↑**

|  |
| --- |
| #获取当前页面的初始数据  def web\_result():  try:  result\_k='//\*[@class="search\_table\_header"]'#选择xpath时请不要复制  #result\_k='//\*[@class="cabinV2"]'  result\_k\_contain=driver.find\_elements\_by\_xpath(result\_k)  print('正在读取页面数据.........')#信息交互让用户知道程序正在流畅运行  except NoSuchElementException:  print('由于网速问题使访问过慢，程序出错')  pass  result\_k\_list=[]  result\_k\_split0=[]  for result in result\_k\_contain:  result\_k\_list.append(result.text)  for result in result\_k\_list:  result\_k\_split0.append(result.split('\n'))  #print(result\_k\_split0)  return result\_k\_split0#获得基本数据 |

在某些地方可以print一些信息，让用户知道程序正在运行。

只爬取一天的数据当然不能满足我们，我们的目的是获取一段时间内的机票信息，以求得最便宜的机票

|  |
| --- |
| #翻到下一个页面  def next\_click\_botton(number):  cheap\_paice='//\*[@data-ubt="c\_calendar\_banner\_'+str(number)+'"]'  driver.find\_element\_by\_xpath(cheap\_paice).click()  print('页面'+str(number)+'加载中.........')  sleep(4)#防止因为网速问题而产生无法定位元素的错误  #测试是否成功点击  #strtest=driver.find\_element\_by\_xpath('//\*[@class="calendar\_date"]').text  #print(strtest) |

点击传入编号number的界面，number由日期而定，在输入时就以确定。

有了这两步，只需通过输入的日期写一个循环就可以把日期内的数据爬下来

|  |
| --- |
| date\_list=putinit()  if(password==0):  for number in range(int(date\_list[0][2]),int(date\_list[1][2])+1):  flight\_list.append(web\_result())  next\_click\_botton(number)  print('程序读取完毕') |

（**二次提醒：请勿在获取信息途中对打开的浏览器进行任何操作，否则会出现无法对信息进行检测的情况，这时程序不会报错且不会关闭，只能重新打开jpnodebook重启程序**）

**（2）整理数据**

读取的数据并不规整，肯定无法将其写成xlsx文件，所以要求我们对其进行整理。虽然大量的数据不规整，但是还是有规律可循的，我们随便拿出一条数据

|  |
| --- |
| ['中国联航KN5737', '波音 737-800(中型)', '07:50', '南苑机场', '09:50', '虹桥国际机场T2', '准点率', '70%', '¥532起', '经济舱3.6折', '订票'] |

可以看到价格，准点率等等数据都在固定的位置，索取用for循环即可将数据取出。但在实际做的过程中有部分数据和其他数据不同，会多出一两条优惠信息，所以需要分多种情况讨论。

|  |
| --- |
| for a in flight\_list:  for aa in a:  if aa[8]!='订立减¥3' and aa[8]!='准点率' and aa[8]!='订立减¥5':  f\_g.append(aa[0])  f\_x.append(aa[1])  f\_outt.append(aa[2])  f\_outf.append(aa[3])  f\_tot.append(aa[4])  f\_tof.append(aa[5])  f\_z.append(aa[7])  f\_q.append(int(aa[8][1:4]))  #在统计数据时随便找最便宜的机票  if(int(f\_payi1[8][1:4])>int(aa[8][1:4])):  f\_payi1=aa  f\_k.append(aa[9])  elif aa[8]=='订立减¥3':  f\_g.append(aa[0])  f\_x.append(aa[1])  f\_outt.append(aa[2])  f\_outf.append(aa[3])  f\_tot.append(aa[4])  f\_tof.append(aa[5])  f\_z.append(aa[7])  f\_q.append(int(aa[9][1:4]))  #在统计数据时随便找最便宜的机票  if(int(f\_payi2[9][1:4])>int(aa[9][1:4])):  f\_payi2=aa  f\_k.append(aa[10])  elif aa[8]=='订立减¥5':  f\_g.append(aa[0])  f\_x.append(aa[1])  f\_outt.append(aa[2])  f\_outf.append(aa[3])  f\_tot.append(aa[4])  f\_tof.append(aa[5])  f\_z.append(aa[7])  f\_q.append(int(aa[9][1:4]))  #在统计数据时随便找最便宜的机票  if(int(f\_payi2[9][1:4])>int(aa[9][1:4])):  f\_payi2=aa  f\_k.append(aa[10])  else:  f\_g.append(aa[0])  f\_x.append(aa[1])  f\_outt.append(aa[2])  f\_outf.append(aa[3])  f\_tot.append(aa[6])  f\_tof.append(aa[7])  f\_z.append(aa[9])  try:  f\_q.append(int(aa[10][1:4]))  except ValueError:  print(aa)  driver.quit()  #在统计数据时随便找最便宜的机票  if(int(f\_payi3[10][1:4])>int(aa[10][1:4])):  f\_payi3=aa  f\_k.append(aa[11]) |

这样基本上每条机票信息的数据就保存在列表中了，还顺便统计了价格最小的机票，将其单独作为正文直接发到邮箱，在统计最小的过程依然要分情况。

|  |
| --- |
| if(int(f\_payi1[8][1:4])<int(f\_payi2[9][1:4])):  f\_payi=f\_payi1  if(int(f\_payi1[8][1:4])>int(f\_payi3[10][1:4])):  f\_payi=\_f\_payi3  elif(int(f\_payi2[9][1:4])<int(f\_payi3[10][1:4])):  f\_payi=f\_payi2  else:  f\_payi=f\_payi3 |

下面为了将其发到邮箱，将其存为矩阵形式，并记录当前查询的时间，以区别不同时间的数据。

|  |
| --- |
| flights\_df = pd.DataFrame({'Airline company|': f\_g,  'Aircraft model|':f\_x,  'Out Early Day|': f\_outt,  'Out Duration|': f\_outf,  'Out Later Day|': f\_tot,  'To Duration|': f\_tof,  'Aircraft model|':f\_z,  'Price|':f\_q,  'Preferentail':f\_k,  })  flights\_df['timestamp'] = strftime("%Y%m%d-%H%M")#记录当前查询的时间 |

这样数据就存储好了，基础已经搭好，下面可以开始邮件发送和绘图的工作了。

# **邮件发送与绘图**

# SMTP是发送邮件的协议，Python内置对SMTP的支持，可以发送纯文本邮件、HTML邮件以及带附件的邮件。Python对SMTP支持有smtplib和email两个模块，email负责构造邮件，smtplib负责发送邮件。

# （实际上smtplib和email一直有在更新，所以在找教程的过程中除了官方文档外一定要找新的，老旧的教程方法无法使用）

**（0）保存起始数据**

首先将我们的矩阵保存为xlsx，并将最便宜的机票的机票信息转换为字符串。

|  |
| --- |
| final\_df.to\_excel(r'C:\Users\user\Music//{}\_flights\_{}-{}\_\_{}\_to\_{}.xlsx'.format(strftime("%Y%m%d-%H%M"),city\_out\_, city\_to\_, date\_out\_, date\_to\_), index=False)  print('保存表格.....')  f\_payi0=[str(i) for i in f\_payi]#将列表变为字符串，方面发送  f\_payi4=' '.join(f\_payi0) |

**（0.5）获取邮件的等等信息**

之后通过输入获取发送人的邮箱和接受的邮箱，还需要填写邮箱的服务器，QQ邮箱的SMTP服务器为smtp.qq.com。用qq邮件服务器发送邮件需要先到邮箱里设开启SMTP/POP3服务。然后获取授权码，这样才能创建SSL安全连接，然后再使用SMTP协议发送邮件。

|  |
| --- |
| from\_addr=input('发信邮箱（最好为QQ邮箱，例：1719913794@qq.com')  to\_addr=input('收信邮箱（最好为QQ邮箱，例：1719913794@qq.com')  smtp\_server=input('邮箱服务器，若为QQ邮箱则为smtp.qq.com')  pwd=input('密钥，需要查询例：kqjapjuxnoxeefbb')  mail\_content='测试'  mail\_title='测试邮件' |

**（1）构建邮件对象**

在必要的信息得到后，我们可以开始通过email构建我们的信息了。

|  |
| --- |
| # 邮件对象:  msg = MIMEMultipart()  msg['From'] = from\_addr  msg['To'] =Header("接受测试",'utf-8')  msg['Subject'] = Header(mail\_title, 'utf-8')  # 邮件正文是MIMEText:  msg.attach(MIMEText('send with file...'+'\n'+f\_payi4, 'html', 'utf-8'))  # 构造附件，传送当前目录下的 文件  att1 = MIMEText(open(r'C:\Users\user\Music//{}\_flights\_{}-{}\_\_{}\_to\_{}.xlsx'.format(strftime("%Y%m%d-%H%M"),city\_out\_, city\_to\_, date\_out\_, date\_to\_), 'rb').read(), 'base64', 'utf-8')  att1["Content-Type"] = 'application/octet-stream'  # 这里的filename可以任意写，写什么名字，邮件中显示什么名字  att1["Content-Disposition"] = 'attachment; filename="www.xlsx"'  msg.attach(att1) |

通过MIMEMultipart()构建邮件对象，然后设置它的发信人和收信人，标题。再将正文和附件通过attach构入邮件对象中。这样，一个完整的邮件对象就构建完成，下面改开始发送。

**（2）发送邮件**

早期SMTP 服务器都采用普通的网络连接，因此默认端口是 25。但现在绝大部分 SMTP 都是基于 SSL（Secure Socket Layer）的，这样保证网络上传输的信息都是加密过的，从而使得信息更加安全。这种基于 SSL 的 SMTP 服务器的默认端口是 465。上面程序中连接的是 QQ 邮箱的基于 SSL 的 SMTP 服务器，QQ 邮箱服务器不支持普通的 SMTP。国内有些公司的免费邮箱（比如 QQ 邮箱）默认是关闭了 SMTP 的，因此需要登录邮箱进行设置，并获取授权码。

|  |
| --- |
| #ssl登录  smtp = SMTP\_SSL(smtp\_server)  #set\_debuglevel()是用来调试的。参数值为1表示开启调试模式，参数值为0关闭调试模式  smtp.set\_debuglevel(1)  smtp.ehlo(smtp\_server)  smtp.login(from\_addr, pwd)  print('正在发送邮件')  smtp.sendmail(from\_addr, to\_addr, msg.as\_string())  smtp.quit() |

信息发送结果如图所示



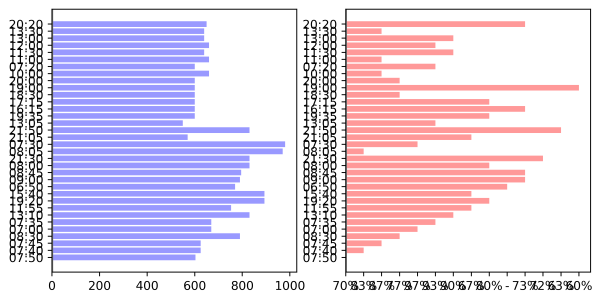
**（3）绘图**

单纯的表格数据是无法看出其中的某些特性和数据之间的关联的，所以可以通过绘图的方式具体观察数据的某些特性和关联。

因为所得数据中带有数字的只有时间，准点率和价格，而其他信息多为字符串且重复度不高，所以就这三个信息作为我们讨论的对象。

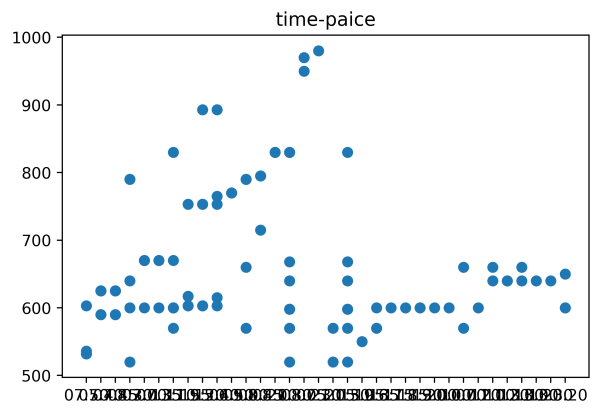
|  |
| --- |
| plt.title("time-paice")  plt.scatter(f\_outt,f\_q)  fig, axes = plt.subplots(ncols=2, figsize=plt.figaspect(1./2))  vert\_bars = axes[0].barh(f\_outt, f\_q, color='#9999ff', align='center')  horiz\_bars = axes[1].barh(f\_outt, f\_z, color='#ff9999', align='center')  #在水平或者垂直方向上画线  axes[0].axvline(0, color='gray', linewidth=2)  axes[1].axvline(0, color='gray', linewidth=2)  plt.rcParams['savefig.dpi'] = 500 #图片像素  plt.rcParams['figure.dpi'] = 500 #分辨率  plt.show() |

分别对时间和价格，时间和准点率做了柱状图，对时间和价格的分布做了散点图，以观察之间的关系。



柱状图↑

由这量张图可以看出一般离启航日越远的飞机票，就会越便宜，而离启航日越近的飞机，准点率越高。周五晚上和周六早上的票买的最贵，周末的票价比较便宜，这一点在散点图中更能体现。



散点图中可以明显的看到突起，那个就是周五晚上和周六早上的票价。

1. **结束语**

以上就是用python抓取最便宜的机票的程序的全部内容，当我们想旅游时，不用再繁琐的查询，打开电脑，运行一下此程序，就可以等待邮箱的信息了。当然，此程序还有可以改进的地方，每次还要打开电脑太麻烦？将程序放置在服务器上，写一个sleep很长时间的死循环，就可以不断获取机票信息了；想查看最便宜的飞机票的详细信息？可以通过webdriver点击购票，再进行爬取查看......总之，查取信息的功能基本实现，更方便更准确的功能在之后时间充足的情况改进。