



**COVID-19** 疫情数据科学实战

课程笔记

作者：1210

邮箱：[sx12101184@qq.com](mailto:sx12101184@qq.com) 时间：May 6, 2020

组织：笭

别人都关心你飞的有多高，只有我关心你的翅膀好不好吃！



目录

**1**

**Python** 疫情数据爬取

**1**

1

1

1

1

2

2

4

6

6

6

6

7

8

10

12

12

13

14

14

14

20

22

22

23

28

31

31

1.1

网络爬虫基础 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

1.1.1

1.1.2

1.1.3

1.1.4

1.1.5

爬虫是什么? . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

爬虫可以做什么? . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

爬虫的分类 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

爬虫的基本流程 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

常见的反爬与反反爬 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

1.2

1.3

数据源选择 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

Chrome 浏览器开发者工具简述 . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

1.3.1

1.3.2

1.3.3

1.3.4

1.3.5

1.3.6

什么是浏览器开发者工具 . . . . . . . . . . . . . . . . . . . . . . . . . . . .

如何调出开发者工具 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

开发者工具初步介绍 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

元素(Elements) 面板 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

网络(Network) 面板 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

网络(Network) 面板资源分析 . . . . . . . . . . . . . . . . . . . . . . . . . .

1.4

JSON 数据格式分析 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

1.4.1 Json 函数 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

§标站点分析 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

实时数据爬取 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

1.5

1.6

1.6.1

1.6.2

1.6.3

§标数据抓取 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

全国各省实时数据爬取 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

世界各国实时数据爬取 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

1.7

历史数据爬取 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

1.7.1

1.7.2

1.7.3

全国历史数据爬取 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

全国各省历史数据爬取 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

世界各国历史数据爬取 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

1.8

总结 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

1.8.1 核心命令总结 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**2**

**Pandas** 疫情探索性分析

**32**

32

32

32

33

33

39

45

45

50

2.1

数据及 Pandas 工具介绍 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

2.1.1

2.1.2

数据介绍 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

Pandas 介绍 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

2.2

实时数据探索性分析 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

2.2.1

2.2.2

世界各国实时数据探索性分析 . . . . . . . . . . . . . . . . . . . . . . . . . .

全国各省实时数据探索性分析 . . . . . . . . . . . . . . . . . . . . . . . . . .

2.3

历史数据探索性分析 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

2.3.1

2.3.2

全国历史数据探索性分析 . . . . . . . . . . . . . . . . . . . . . . . . . . . .

世界各国历史数据探索性分析 . . . . . . . . . . . . . . . . . . . . . . . . . .



– ii –

目录

2.4 总结 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

57

**3**

**Python** 疫情数据可视化

**58**

58

58

61

62

66

69

3.1

3.2

3.3

3.4

3.5

3.6

可视化概述 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

世界各国家现存确诊人数地图 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

世界国家累计死亡人数玫瑰图 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

3 月美国单日新增确诊人数与股票指数涨幅折线图 . . . . . . . . . . . . . . . . . .

3 月世界国家累计确诊人数动态条形图 . . . . . . . . . . . . . . . . . . . . . . . . .

总结 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**4**

**Scipy** 疫情数据建模分析

**70**

70

70

71

75

78

78

78

80

85

88

4.1

SIR 模型 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

4.1.1

4.1.2

4.1.3

SIR 模型原理 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

运用湖北省疫情数据建立SIR 模型 . . . . . . . . . . . . . . . . . . . . . . .

基本再生数 *R*0 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

4.2

使用数据拟合参数 *β* 和 *γ* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

4.2.1

4.2.2

4.2.3

定义损失函数 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

读取数据并划分训练集验证集 . . . . . . . . . . . . . . . . . . . . . . . . . .

训练模型 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

4.3

4.4

疫情发展趋势预测 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

总结 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .



第一章 **Python** 疫情数据爬取

内容提要

**1.1** 网络爬虫基础

**1.1.1** 爬虫是什么**?**

爬虫是一个模拟人类请求网站行为, 并批量下载网站资源的一种程序或自动化脚本

* + 1. 爬虫可以做什么**?**
       1. 搜索引擎
       2. 采集金融数据
       3. 采集商品数据
       4. 采集竞争对手的客户数据
       5. 采集行业相关数据，进行数据分析
       6. 刷流量

**1.1.3** 爬虫的分类

通用网络爬虫

又称为全网爬虫，其爬取对象山一批 URL 扩充至整个 Web，主要山搜索引擎或大型 Web 服务商使用。

聚焦网络爬虫

又称为主题网络爬虫，其特点是只选择性的地爬取与预设的主题相关的页面，相比通用网络爬虫，聚焦网络爬虫仅需要爬取与主题相关的页面，极大地节省硬件及网络资源，能更快的更新保存页面，更好的满足特定人群对特定领域的需求。

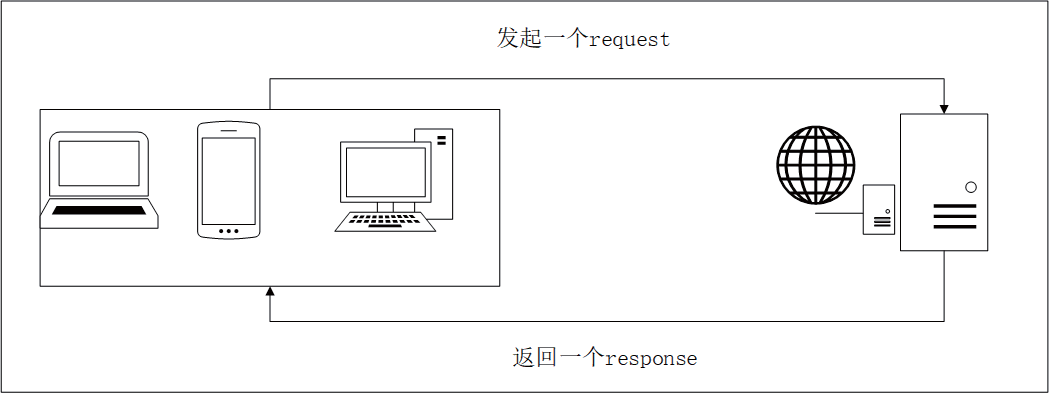
增量网络爬虫

只对已下载的网页采取增量式更新，或只爬取新产生的及已经发生变化的网页，这种机制能够在某种程度上保证所爬取的网页尽可能的新。

深度网络爬虫

Web 页面按照存在的方式可以分为表层页面和深层页面两类。表层页面是只传统搜索引擎可以索引到的页面，以超链接可以达到的静态页面为主。深层页面是指大部分内容无法通过静态链接获取，隐藏在搜索表单之后的，需要用户提交关键词后才能获得的 Web 页面， 如一些登陆后可见的网页。

* 网络爬虫基础 □ 目标站点分析
* 数据源选择 □ 实时数据爬取
* 浏览器开发者工具 □ 历史数据爬取
* JSON 数据格式 □ 总结



1.1 网络爬虫基础

– 2 –

**1.1.4** 爬虫的基本流程

**1.1.4.1** 浏览网页的流程

图 **1.1:** 浏览网页的流程

**1.1.4.2** 爬虫的基本流程

1.

请求网页

通过 HTTP 库向§标站点发起请求，即发送一个 Request，请求可以包含额外的 headers 等信息，等待服务器响应!

获得相应内容

如果服务器能正常响应，会得到一个Response，Response 的内容便是所要获取的页面内容， 类型可能有HTML，Json 字符串，二进制数据（如图片视频）等类型。

解析内容

得到的内容可能是 HTML，可以用正则表达式、网页解析库进行解析。可能是 Json，可以直接转为Json 对象解析，可能是二进制数据，可以做保存或者进一步的处理。

存储解析的数据

保存形式多样，可以存为文本，也可以保存至数据库，或者保存特定格式的文件

2.

3.

4.

**1.1.5** 常见的反爬与反反爬

守: 通过User-Agent 来控制访问：

从用户请求的 Headers 反爬虫是最常见的反爬虫策略。山千正常用户访问网站时是通过浏览器访问的，所以§标网站通常会在收到请求时校验 Headers 中的 User-Agent 字段，如果

令

不是携带正常的 User-Agent 信息的请求便无法通过请求。

笔记User Agent 中文名为用户代理，简称 UA，它是一个特殊字符串头，使得服务器能够识别客户使用的操作系统及版本、CPU 类型、浏览器及版本、浏览器渲朵引擎、浏览器语言、浏览器插件等。

破：应对措施：如果遇到了这类反爬虫机制，可以直接在自己写的爬虫中添加Headers，将 浏览器的User-Agent 复制到爬虫的Headers 中

守: 基千行为检测(限制 IP)

还有一些网站会通过用户的行为来检测网站的访问者是否是爬虫，例如同一IP 短时间内多次访问同一页面，或者同一账户短时间内多次进行相同操作。大多数网站都是前一种情况， 对千这种情况有两种策略：



1.1 网络爬虫基础

– 3 –

破：应对措施：

1.

可以专门写一个在网上抓取可用代理 ip 的脚本，然后将抓取到的代理 ip 维护到代理池中供爬虫使用，当然，实际上抓取的ip 不论是免费的还是付费的，通常的使用效果都极为一般，如果需要抓取高价值数据的话也可以考虑购买宽带adsl 拨号的VPS，如果ip 被§标网站被封掉，重新拨号即可。

降低请求频率。例如每个一个时间段请求一次或者请求若干次之后 sleep 一段时间。山千网站获取到的 ip 是一个区域网的 ip，该 ip 被区域内的所有人共享，因此这个间隔时间并不需要特别长, 对千第二种情况，可以在每次请求后随机间隔几秒再进行下一次请求。对千有逻辑漏洞的网站，可以通过请求几次，退出登录，重新登录，继续请 求来绕过同一账号短时间内不能多次进行相同请求的限制，如果能有多个账户，切换

使用，效果更佳。

2.

守: 通过账号权限反爬(ookie 限制)

部分网站需要登录才能继续操作，这部分网站虽然并不是为了反爬虫才要求登录操作，但确实起到了反爬虫，的作用, 可是网页上有一部分内容如: 新浪微博是需要用户登录才能查看更多内容。限制每个每天下载 300 张.

破：应对措施：

因此可以通过注册账号，访问时带 cookie, 模拟登录的方法进行规避。

守: 验证码限制

这是一个相当古老但却不失有效性的反爬虫策略。更早的时候，这种验证码可以通过 OCR 技术进行简单的图像识别破解，但是现在来说，验证码的干扰线，噪点已经多到肉眼都无法轻易识别的地步。所以§前而言，山千OCR 技术发展不力，验证码技术反而成为了许多网站最有效的手段之一。

破：应对措施：

1. 神经网络训练NLP(图像识别)
2. 人工识别
3. 打码平台
4. 第三方 OCR 库

守: 动态页面的反爬虫(通过变换网页结构反爬)

一些社交网站常常会更换网页结构，而爬虫大部分情况下都需要通过网页结构来解析需要的数据，所以这种做法也能起到反爬虫的作用。在网页结构变换后，爬虫往往无法在原本的网页位置找到原本需要的内容.

破：应对措施：

1. 只爬取一次时，在其网站结构调整之前，将需要的数据全部爬取下来；使用脚本对网 站结构进行监测，结构变化时，发出告警并及时停止爬虫。
2. 逆向分析，抓包
3. selenium 库



1.2 数据源选择

– 4 –

**1.2** 数据源选择

新型冠状病毒感染的肺炎疫情爆发后，对人们的生活产生很大的影响。当前感染人数依然在 不断变化。每天国家卫健委和各大新闻媒体都会公布疫情的数据，包括累计确诊人数、现有确诊 人数等。

首先我们需要找到合适的数据源，卫健委和各媒体每天都会报道新冠肺炎的疫情数据，如下 图所示，因此我们可以考虑将这些网页作为数据源。

以北京市和湖南省为例查看卫健委的数据，结果发现，两地卫健委的数据各式并不统一，北 京市卫健委数据为文本，而湖南省卫健委数据为图片，并且山千各卫健委的网络地址还需要再花 时间去寻找，因此卫健委的数据并不便千采集。

北京市卫健委疫情通报地址：点击跳转

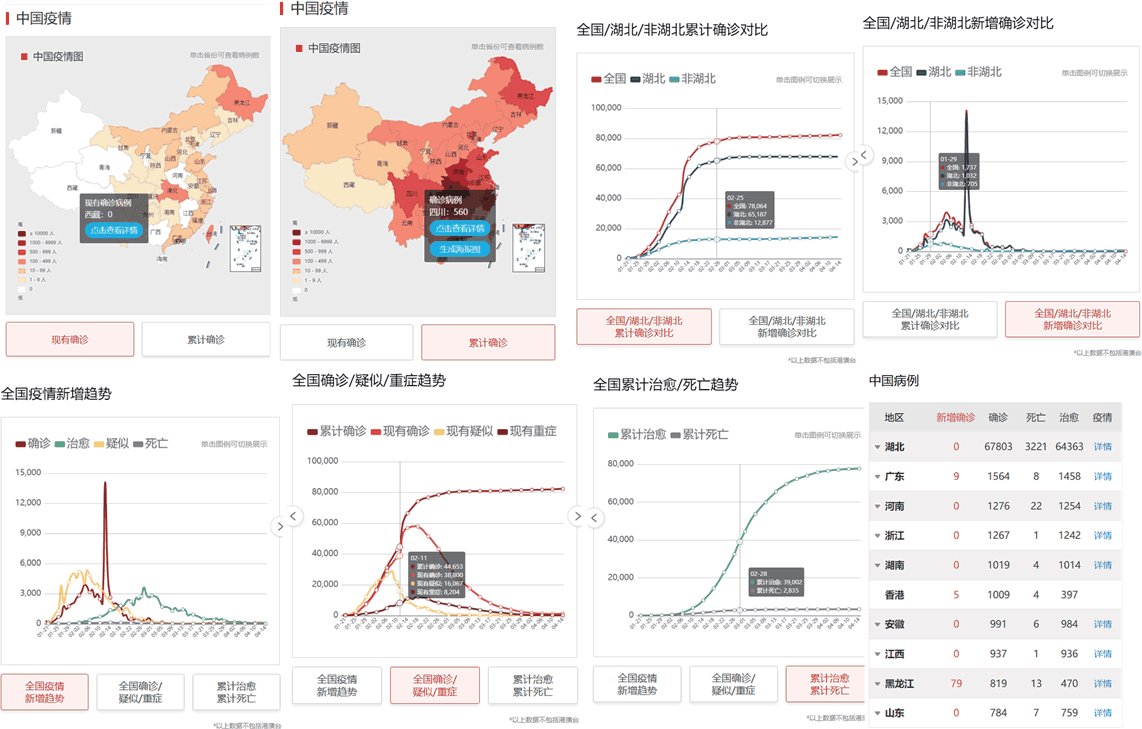
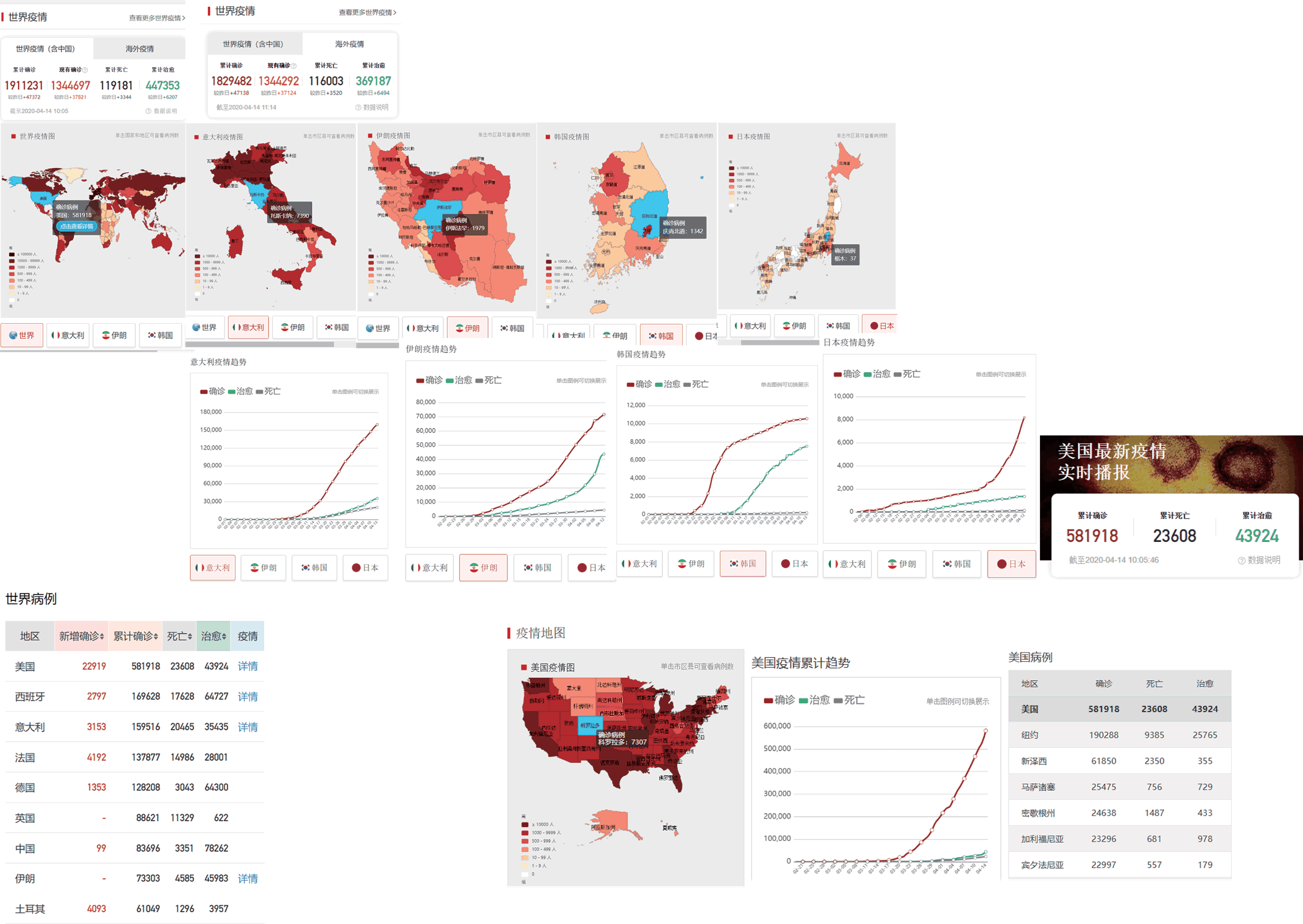
地址 2：点击跳转

湖南省卫健委疫情通报地址：点击跳转

地址 2：点击跳转

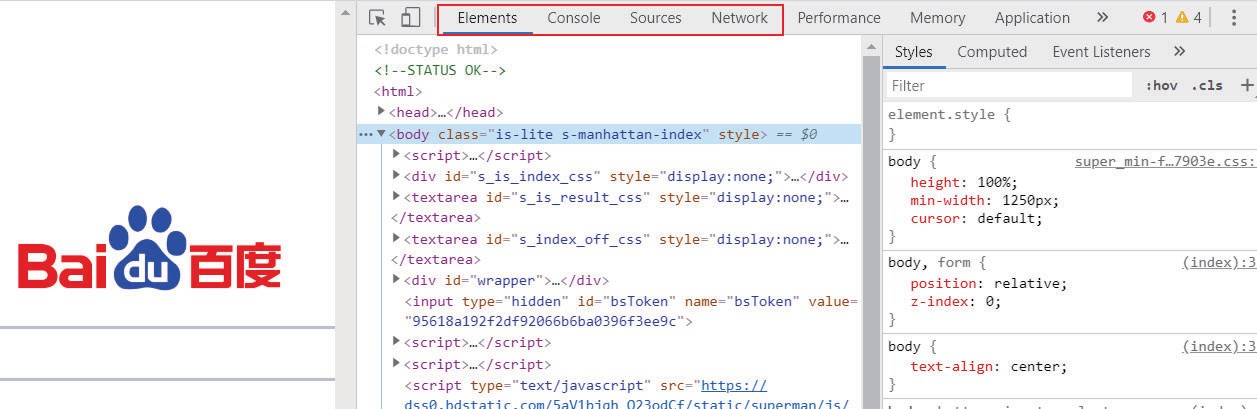
我们考虑将新闻媒体的播报平台作为数据源，以网易的疫情播报平台为例，如下图所示可以

看到它的数据内容非常丰富，不仅包括国内的数据还包括国外的数据，且作为大平台，公信度也 比较高。因此我们选择网易的疫情实时动态播报平台作为数据源，其地址如下：点击跳转



1.2 数据源选择

– 5 –



1.3 Chrome 浏览器开发者工具简述

– 6 –

**1.3 Chrome** 浏览器开发者工具简述

**1.3.1** 什么是浏览器开发者工具

其实简单的说，浏览器开发者工具就是给专业的 web 应用和网站开发人员使用的工具，它的作用在千，帮助开发人员对网页进行布局，比如 HTML+CSS，帮助前端工程师更好的调试脚本

（JavaScript、jQuery）之类的，还可以使用工具查看网页加载过程，获取网页请求（这个过程也叫做抓包），抓包是非常有意思的过程，而每一个浏览器厂商生产出来的浏览器都会有自己的杀手铜，也就是功能上的差别，那么这个时候你就找一个最适合自己的浏览器使用就可以，接下来就

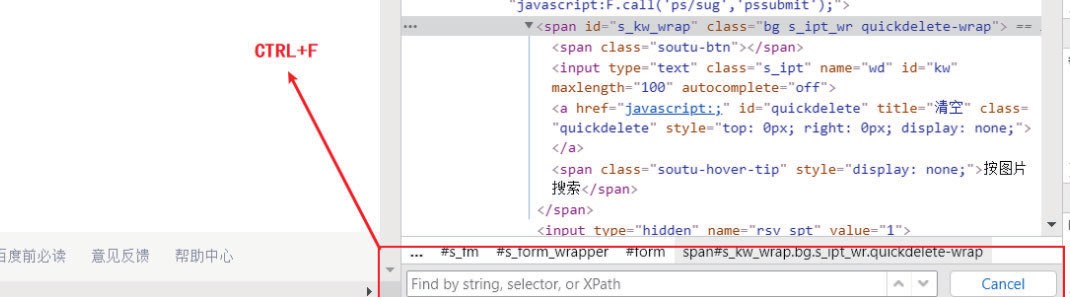
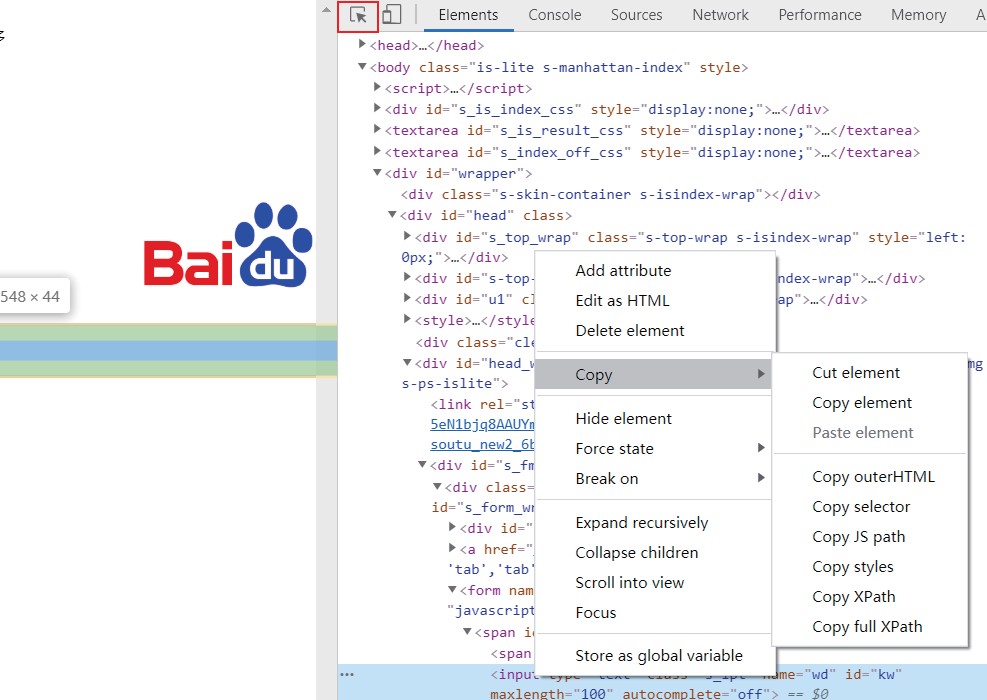
是介绍我常用的三个浏览器。

* + 1. 如何调出开发者工具
       - 按F12 调出
       - 右键检查（或快捷键Ctrl+Shift+i）调出

图 **1.2:** 浏览器开发者工具

**1.3.3** 开发者工具初步介绍

图 **1.3:** 浏览器开发者工具



1.3 Chrome 浏览器开发者工具简述

– 7 –

chrome 开发者工具最常用的四个功能模块：元素（ELements）、控制台（Console）、源代码

（Sources），网络（Network）。爬虫这块用的比较多的是元素 **(ELements),** 网络 **(Network)**.

* 元素(**Elements**)：用千查看或修改 HTML 元素的属性、CSS 属性、监听事件、断点等。css

可以即时修改，即时显示。大大方便了开发者调试页面

* 控制台(**Console**)：控制台一般用千执行一次性代码，查看 JavaScript 对象，查看调试日志

信息或异常信息。还可以当作 Javascript API 查看用。例如我想查看 console 都有哪些方法和属性，我可以直接在Console 中输入"console" 并执行

* 源代码(**Sources**)：该页面用千查看页面的 HTML 文件源代码、JavaScript 源代码、CSS 源代码，此外最重要的是可以调试JavaScript 源代码，可以给JS 代码添加断点等。
* 网络(**Network**)：网络页面主要用千查看 header 等与网络连接相关的信息。

**1.3.4** 元素 **(Elements)** 面板

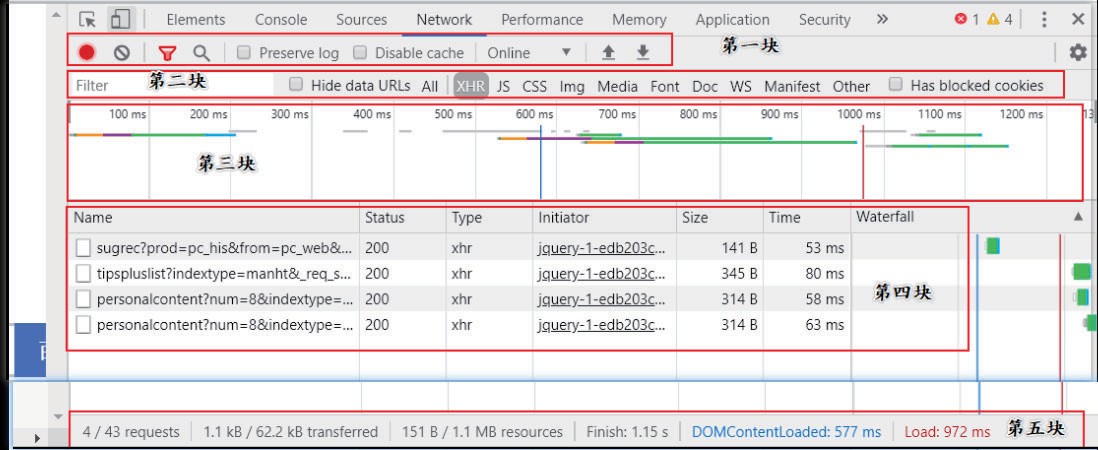
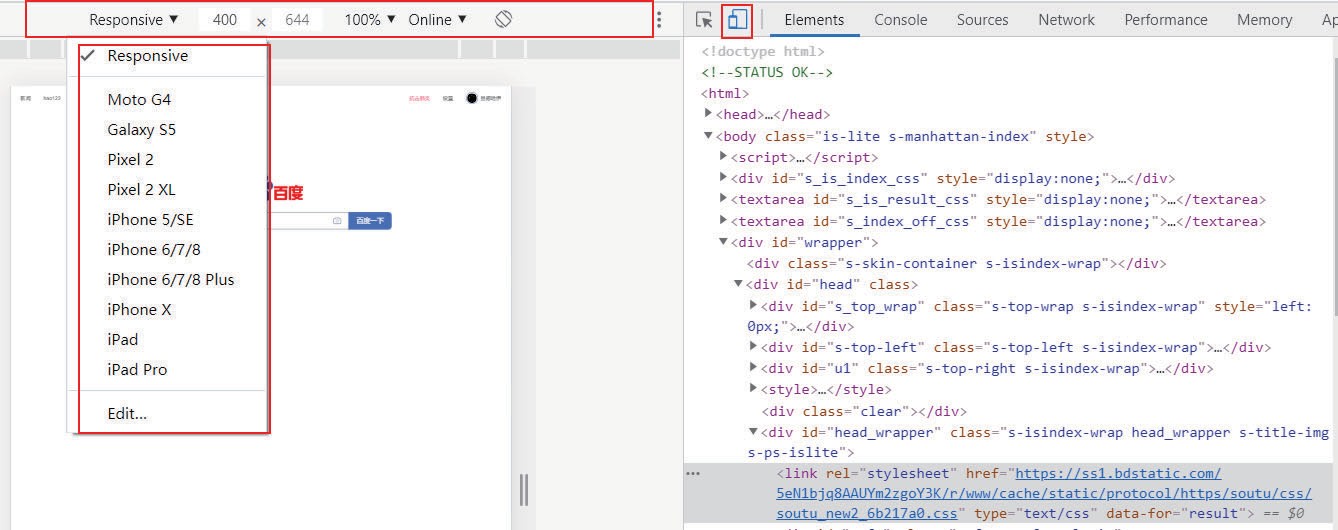
查看元素的代码：点击左上角的箭头图标（或按快捷键 Ctrl+Shift+C）进入选择元素模式，然后从页面中选择需要查看的元素，然后可以在开发者工具元素（Elements）一栏中定位到该元素 源代码的具体位置

图 **1.4:** 浏览器开发者工具

用千定位元素，复制某类元素路径，这个在使用 BS4 库,selenium 库时，选择定位器会用到!

图 **1.5:** 浏览器开发者工具

用千写XPATH 路径语言时使用!



1.3 Chrome 浏览器开发者工具简述

– 8 –

图 **1.6:** 浏览器开发者工具

可以更改浏览器显示方式!, 使用手机或者其他设备! 也可以更改当前使用的设备的网络连接

速度!

**1.3.5** 网络 **(Network)** 面板

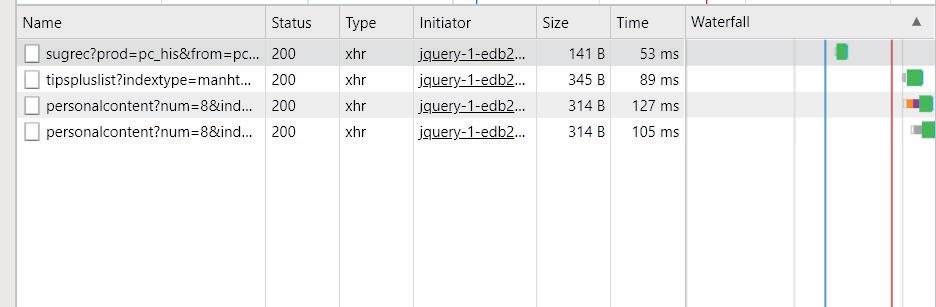
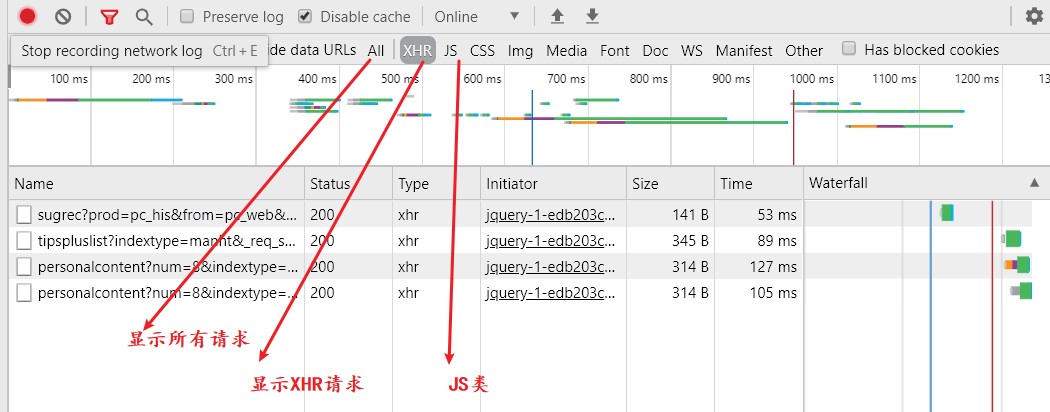
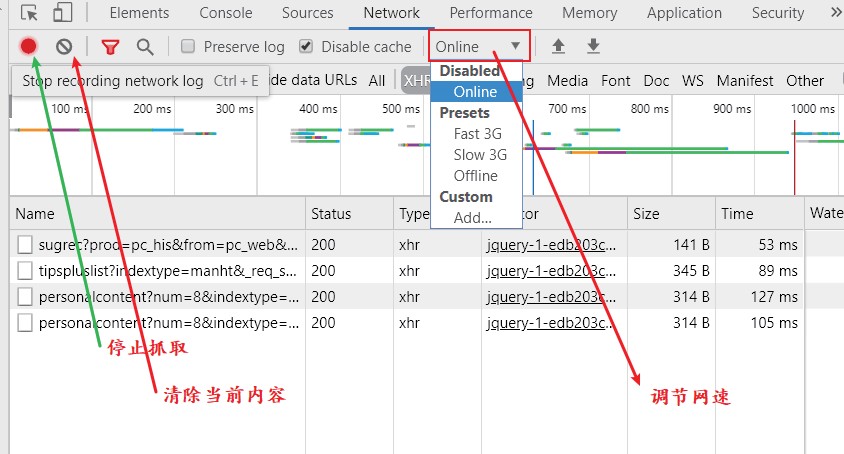
Network 面板可以记录页面上的网络请求的详情信息，从发起网页页面请求 Request 后分析 HTTP 请求后得到的各个请求资源信息（包括状态、资源类型、大小、所用时间、Request 和

Response 等），可以根据这个进行网络性能优化。该面板主要包括 5 大块窗格：

图 **1.7:** 浏览器开发者工具

* **Controls**ă 控制Network 的外观和功能。
* **Filters**ă 控制Requests Table 具体显示哪些内容。
* **Overviewă** 显示获取到资源的时间轴信息。
* **Requests Table**ă 按资源获取的前后顺序显示所有获取到的资源信息，点击资源名可以查看该资源的详细信息。
* **Summary**ă 显示总的请求数、数据传输量、加载时间信息。

其中用的比较多的是:**Controls,Filter,Requests Table**



1.3 Chrome 浏览器开发者工具简述

– 9 –

图 **1.8:** 浏览器开发者工具

使用频率一般，在部分网站上抓取某些请求时使用! 有时需要选择 Disable Cache, 不需要缓

存。

图 **1.9:** 浏览器开发者工具

令

这个主要用来选择一些请求时用的！常用的俩选项XHR 和JS, 其他偶尔用用!

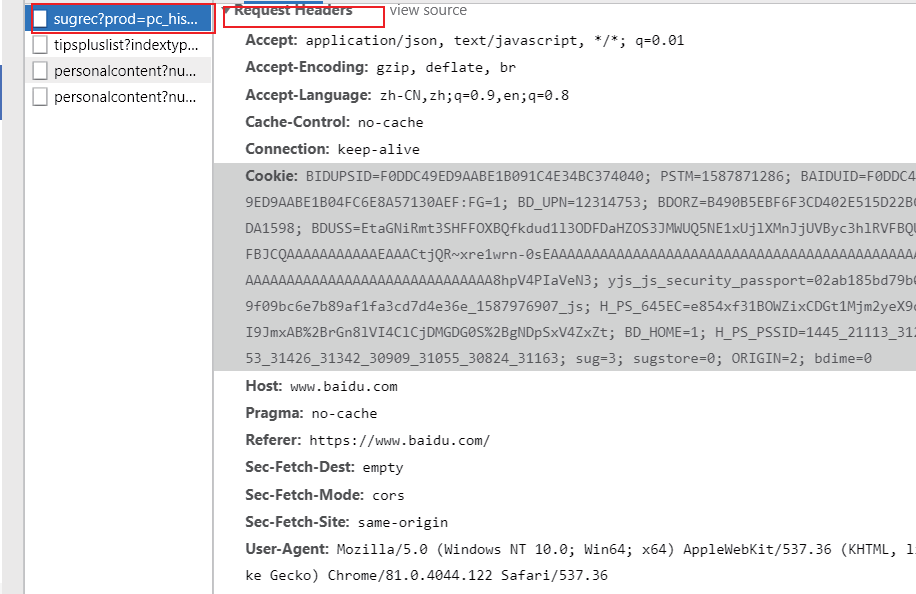
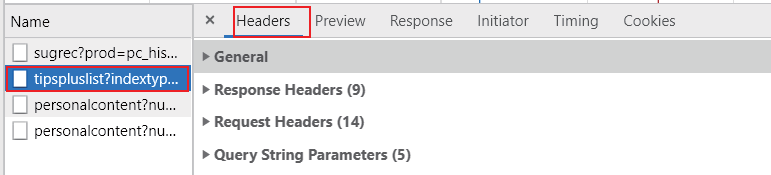
笔记xhr，全称为 XMLHttpRequest，用于与服务器交互数据，是 ajax 功能实现所依赖的对象，jquery

中的ajax 就是对xhr 的封装。

图 **1.10:** 浏览器开发者工具

查看一些请求的名字，状态码，类型，大小和类型! 这个不是重点！重点是这个资源本身的

一些属性!



1.3 Chrome 浏览器开发者工具简述

– 10 –

**1.3.6** 网络 **(Network)** 面板资源分析

点击任意一个资源我们可以得到如下资源!

图 **1.11:** 浏览器开发者工具

这个主要用千查看这个请求的内容! 知道数据是否在这块放着!

图 **1.12:** 浏览器开发者工具

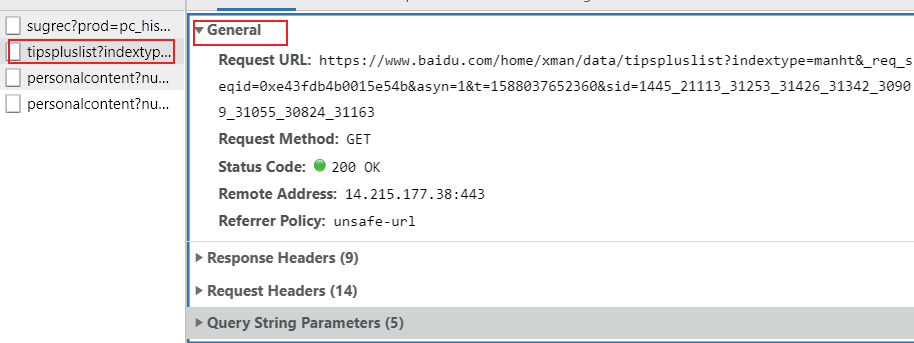
主要用千查看某个资源是如何请求的, 请求头信息，请求体信息，响应体信息等等，下面会

具体介绍! 这部分内容相对来说还是很重要的!

图 **1.13:** 浏览器开发者工具

Request Headers 请求头信息，里面信息很多，常用来做伪装浏览器使用，最常见的伪装方法就是使用User-Agent!

当然还有一些其他的伪装方法!

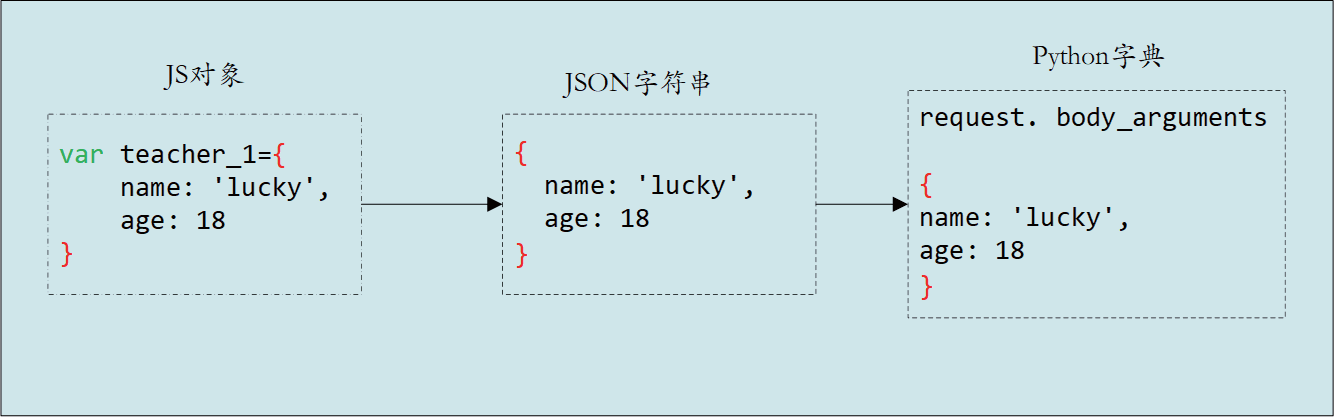


1.3 Chrome 浏览器开发者工具简述

– 11 –

图 **1.14:** 浏览器开发者工具

generel 主要用千查看请求的 url 和请求方式，响应状态码，等信息，常用的就这四个!



1.4 JSON 数据格式分析

– 12 –

**1.4 JSON** 数据格式分析

JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式，易千人阅读和编写。

图 **1.15:** JS 数据格式

SON 注意事项：

1. 名称必须用双引号（即：＂）来包括
2. 值可以是双引号包括的字符串、数字、true、false、null、JavaScript 数组，或子对象。

**1.4.1 Json** 函数

使用JSON 函数需要导入json 库：

1

* **json.dumps()** 将Python 对象编码成JSON 字符串
* **json.loads()** 将已编码的JSON 字符串解码为 Python 对象
* **json.dump()** 将Python 内置类型序列化为json 对象后写入文件
* **json.load()** 读取文件中json 形式的字符串元素转化为Python 类型。其中用最多的大概是 **json.loads()** 了!

1

2

3

4

5

6

7

8

9

10

11

import json

data = {’name’:’nanbei’,’age’:18}

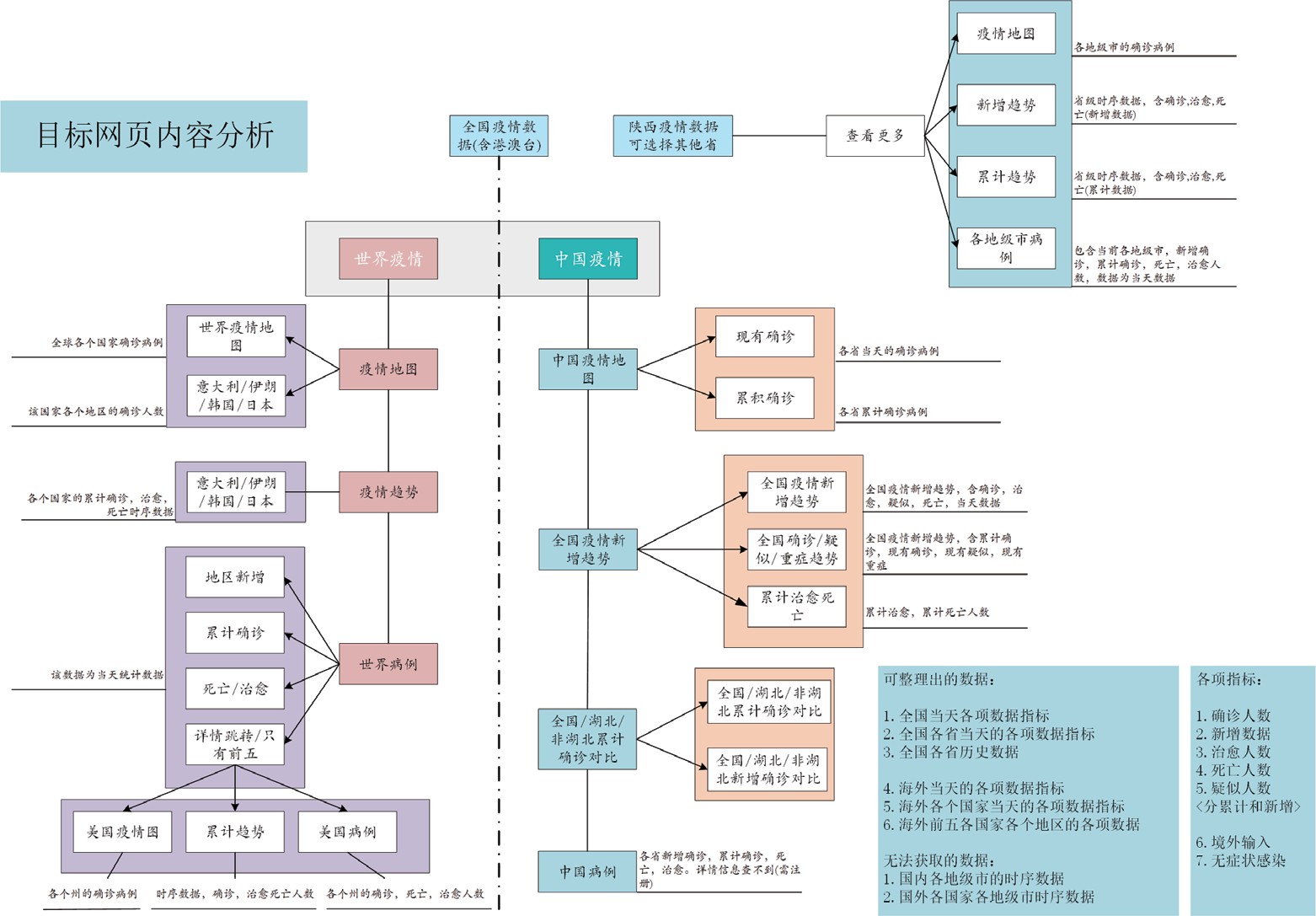
#将Python对象编码成json字符串#print(json.dumps(data))

#将json字符串编码成Python对象 a = json.dumps(data) print(json.loads(a))

-----------------------------------

{’name’: ’nanbei’, ’age’: 18}

import json



1.5 目标站点分析

– 13 –

**1.5** 目标站点分析

结合开发者工具的使用，我们对§标网站可以进行简单分析，看看每个图我们能拿到什么样 的数据! 基本情况如下:



1.6 实时数据爬取

– 14 –

**1.6** 实时数据爬取

**1.6.1** 目标数据抓取

❖

导入相关的库

1

2

3

4

构建目标网址，对爬虫进行伪装

❖

1

2

3

4

5

❖

抓取目标数据，查看相应状态

1

2

3

4

5

6

7

8

9

**1.6.2** 全国各省实时数据爬取

首先我们抓取全国各省的实时数据，在 areaTree 键值对中，存放着世界各地的实时数据，

areaTree 是一个列表，每一个元素都是一个国家的数据，每一个元素的 children 是各国家省份的数据。

* + - 1. 使用 **Json** 模块对响应内容进行初步解析
         * 使用 **Json** 模块对响应内容进行初步解析

1

2

3

4

5

6

7

data\_json =json.loads(req.text) #使用json.loads()解析响应内容type(data\_json) #查看相应的内容

----------------------

输出: dict

---------------------

data\_json.keys() #查看相应内容的键

req = requests.get(url=url,headers=headers) #发起请求

print(req.status\_code) #查看响应状态码print( type(req.text)) #查看响应内容文件类型print( len(req.text)) #查看响应内容长度

----------------------------- 200

<class ’str’> 254972

#设置请求头，伪装为浏览器

headers = {’User-Agent’: ’Mozilla/5.0␣(Windows␣NT␣10.0;␣Win64;␣x64)␣AppleWebKit

/537.36␣(KHTML,␣like␣Gecko)␣Chrome/81.0.4044.122␣Safari/537.36’}

#找到的第一个数据原作为请求目标。

url = "https://c.m.163.com/ug/api/wuhan/app/data/list-total?t=317590785231"

import requests #用于发起请求import pandas as pd #数据分析库import time #导入时同模块import json #解析JSON文件



1.6 实时数据爬取

– 15 –

8

9

10

在这一块，我们第一次解析 JSON 文件得到的上述几个键，其中我们待获取的数据在 data 键中，

因此我们对此可以进行下一步操作! 获取data 键中的数据

1

2

3

4

5

在这一步中我们得到了 data 键中的内容, 并且可以看到他们有上述四个键，通过§标站点分析，我们可以知道，这四个键分别对应的数据为:

1. **chinaTotal** 全国当日数据
2. **chinaDayList** 全国历史数据
3. **lastUpdateTime** 更新时间
4. **areaTree** 世界各地实时数据

在这里我们选择中国的数据,areaTree 中中国的数据! 然后尝试遍历前五个数据，看是否可以正常获得各省数据

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

就结果而言，看来是可以的，接着我们进行进一步操作，是否可以使用DataFrame 以字典的形式

保存数据!

1

pd.DataFrame(data\_province).head() #直接生成数据效果并不理想

data\_province = data[’areaTree’][2][’children’] type(data\_province)

---------------------------------------------

输出: list #输出的数据类型是一个列表

---------------------------------------------

for i in range( len(data\_province)): print(data\_province[i][’name’],data\_province[i][’lastUpdateTime’]) if i ==5:

break

---------------------------------------------

输出:湖北 2020-04-27 08:54:31

广东 2020-04-27 08:27:52

河南 2020-04-27 09:43:17

沿工 2020-04-27 09:01:02

香港 2020-04-27 08:39:10

湖南 2020-04-27 08:55:53

---------------------------------------------

data=data\_json[’data’] #取出json中的数据data.keys() #查看相应内容的键

---------------------------- -----------------

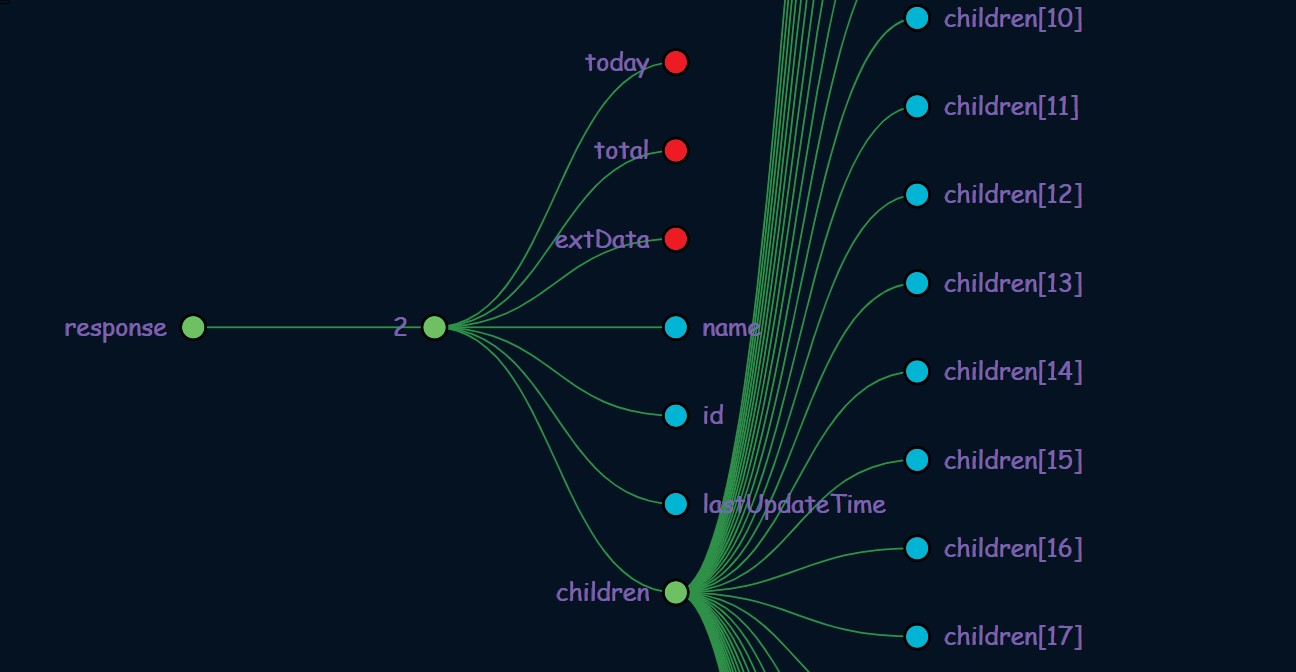
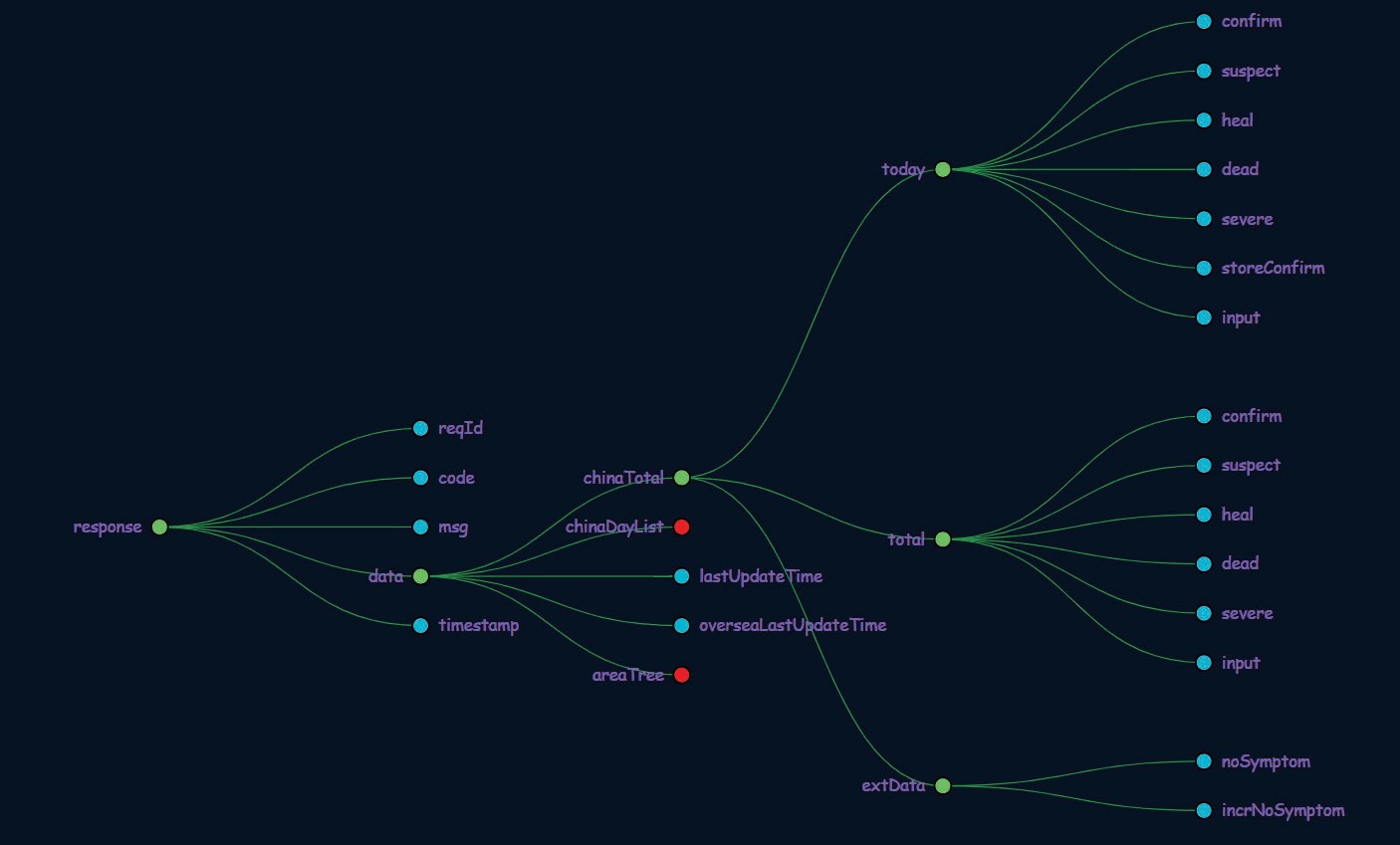
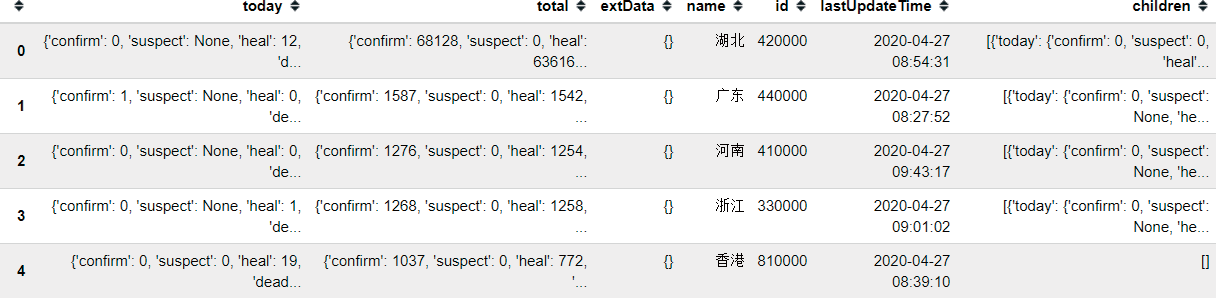
输出:dict\_keys([’chinaTotal’, ’chinaDayList’, ’lastUpdateTime’, ’overseaLastUpdateTime’, ’areaTree’])

---------------------------------------------

----------------------

输出:dict\_keys([’reqId’, ’code’, ’msg’, ’data’, ’timestamp’])

---------------------



1.6 实时数据爬取

– 16 –

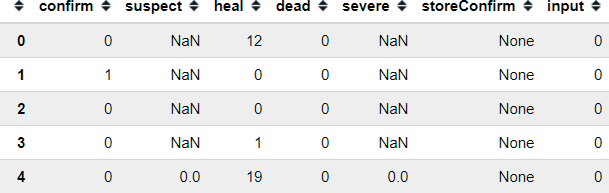
不能直接生成DataFrame 是因为数据中嵌套着字典，例如湖北省数据如下：标红线表示带有嵌套字典，篮筐内没有嵌套字典。数据结构层级看起来有点难受! 我们可以使用 **JSON Viewer Awesome**

这个插件看下当前数据的结构层级，便千分析!

这是整个的数据分级结构，其中areaTree 中的第二个为中国，具体如下：

可以看到在 areaTree 2 中有的为列表，有的为字典! 因此直接套用会出现问题, 所以我们需要

再进行改正!



1.6 实时数据爬取

– 17 –

* + - 1. 获取目标 **ID,** 最后更新日期**,** 以及当前名字
         * 获取目标 **ID,** 最后更新日期**,** 以及当前名字

1

2

3

然后我们可以获取today 中的内容。

* + - 1. **today** 内容获取
         * **today** 内容获取

1

2

3

输出如下：

注意此处获取的 today 内容是 areaTree 中, 中国数据下 children 中每个 children 的 today 数据，

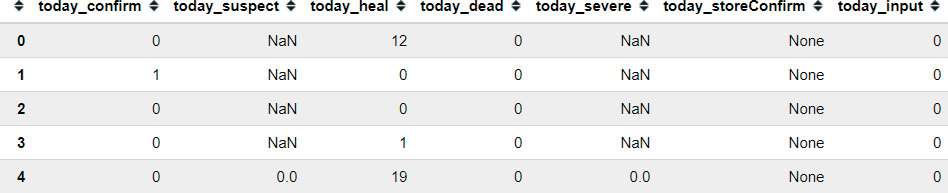
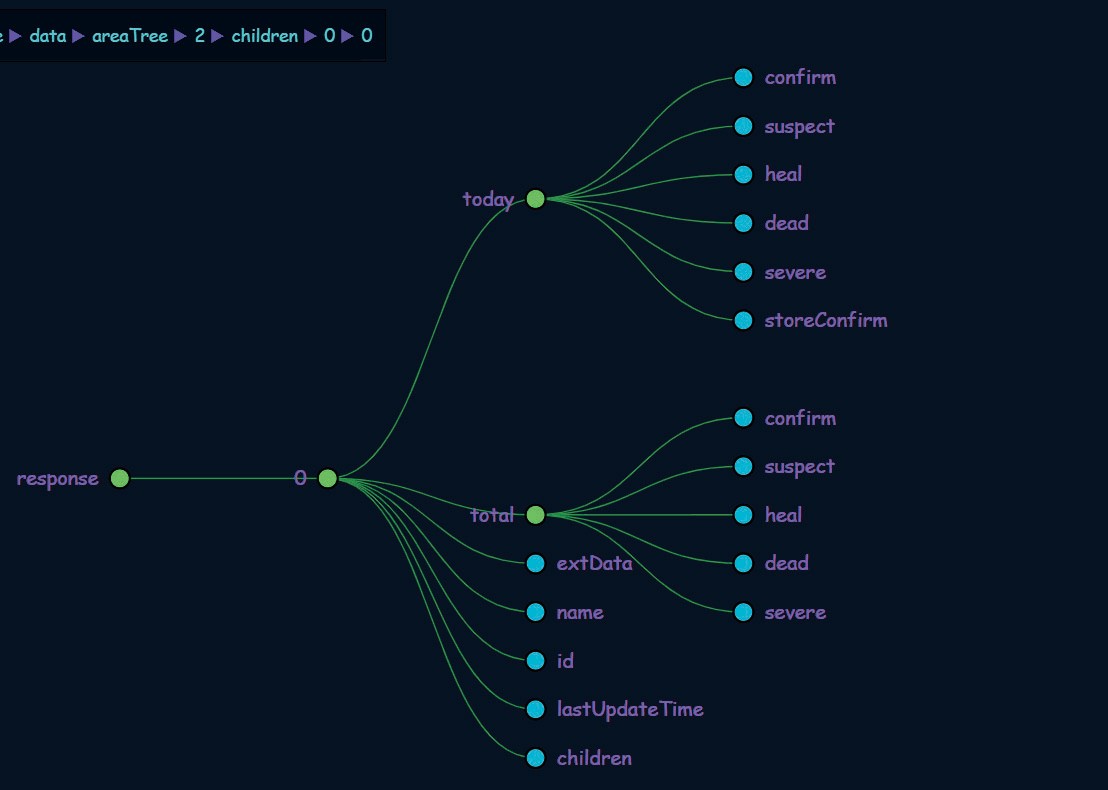
如下图所示：

# 获取today中的数据

today\_data = pd.DataFrame([province[’today’] for province in data\_province ]) today\_data.head()

# 获取id、lastUpdateTime、name

info = pd.DataFrame(data\_province)[[’id’,’lastUpdateTime’,’name’]] info.head()



1.6 实时数据爬取

– 18 –

这里我们可以查看解析到的数据结构!

1

2

3

4

5

此外还有一个问题, 山千today 中键名和total 键名相同，因此需要修改列名称!

1

2

3

4

可以看到解析到内容是没问题的! 接着我们使用同样的方法获取total 中的数据

* + - 1. **total** 数据获取
         * **total** 数据获取

1

2

# 获取total中的数据

total\_data = pd.DataFrame([province[’total’] for province in data\_province ])

# 白于today中键名和total键名相同，因此需要修改列名称

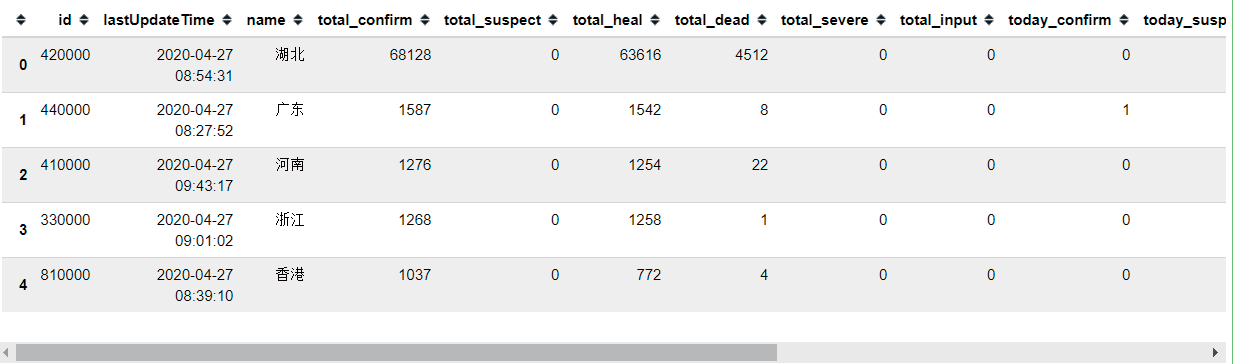
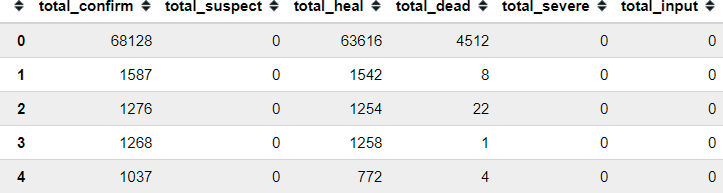
today\_data.columns = [’today\_’+i for i in today\_data.columns]

today\_data.head() #查看前五行数据

today\_data.shape

----------------- (34, 7)

-----------------



1.6 实时数据爬取

– 19 –

3

4

1

2

3

4

5

* + - 1. 三者数据合并
         * 三者数据合并

1

**1.6.2.6** 代码封装

1

2

3

4

5

6

7

8

9

10

11

# 将提取数据的方法封装为函数

def get\_data(data,info\_list):

info = pd.DataFrame(data)[info\_list] # 主要信息

today\_data = pd.DataFrame([i[’today’] for i in data ]) # 提取today的数据

today\_data.columns = [’today\_’+i for i in today\_data.columns] # 修改列名 columns

total\_data = pd.DataFrame([i[’total’] for i in data ]) # 提取total的数据

total\_data.columns= [’total\_’+i for i in total\_data.columns] # 修改列名 columns

return pd.concat([info,total\_data,today\_data],axis=1) # info、today和total横向合并最终得到汇总的数据

pd.concat([info,total\_data,today\_data],axis=1).head() # 将三个数据合并

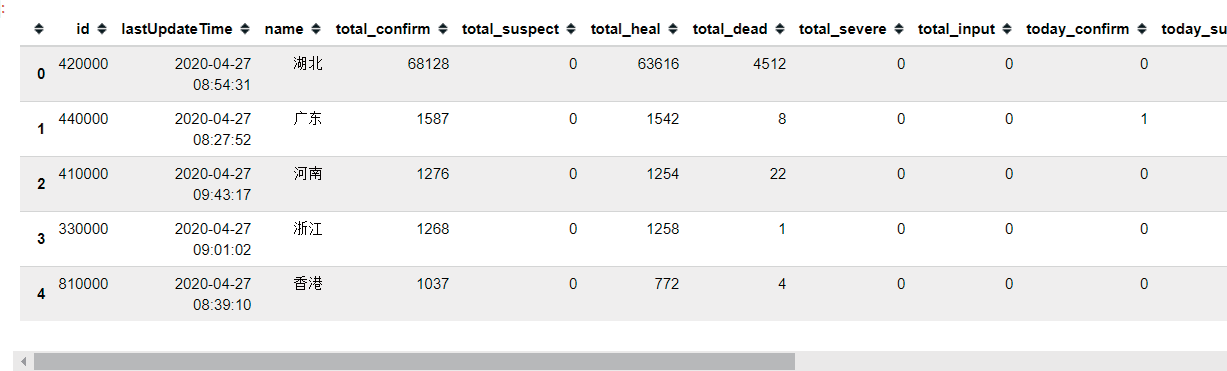
total\_data.shape #查看数据维度

------------------

输出：(34, 6)

------------------

total\_data.columns = [’total\_’+i for i in total\_data.columns] total\_data.head()



1.6 实时数据爬取

– 20 –

调用函数，获取数据

1

2

数据保存

1

2

3

4

调用函数

1

2

3

4

5

**1.6.3** 世界各国实时数据爬取

有了前面定义好的函数，那么现在获取数据就相对容易了很多！

* + - 1. 获取 **areaTree** 数据
         * 获取 **areaTree** 数据

1

2

3

4

5

6

7

8

9

10

11

12

areaTree = data[’areaTree’] # 取出areaTree

areaTree[0] # 查看第一个国家的数据

------------------------

输出：{’today’: {’confirm’: 10,

’suspect’: 0,

’heal’: 9,

’dead’: 0,

’severe’: 0, ’storeConfirm’: None},

’total’: {’confirm’: 949, ’suspect’: 0, ’heal’: 216, ’dead’: 38, ’severe’: 0}, ’extData’: {},

save\_data(today\_province,’today\_province’)

------------------------

输出：today\_province\_2020\_04\_27.csv 保存成功！

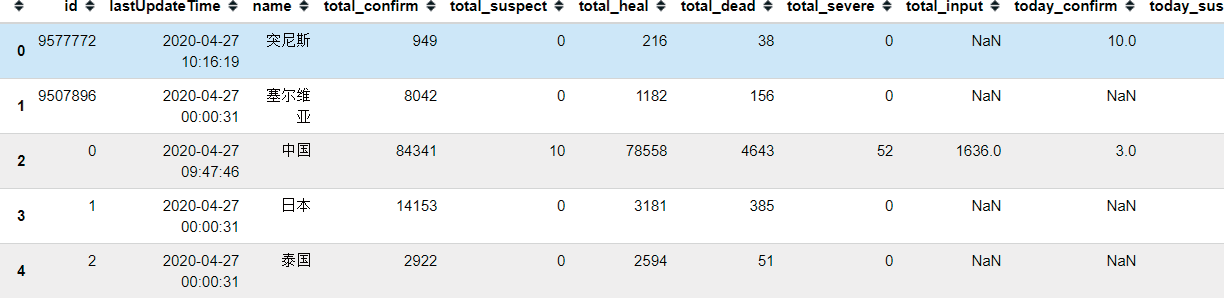
------------------------

def save\_data(data,name): # 定义保存数据方法

file\_name = name+’\_’+time.strftime(’%Y\_%m\_%d’,time.localtime(time.time()))+’.csv’ data.to\_csv(file\_name,index=None,encoding=’utf\_8\_sig’)

print(file\_name+’␣保存成功！’)

today\_province = get\_data(data\_province,[’id’,’lastUpdateTime’,’name’]) today\_province.head()



1.6 实时数据爬取

– 21 –

13

14

15

16

17

* + - 1. 遍历前五个国家数据
         * 遍历前五个国家数据

1

2

3

4

5

6

7

8

9

10

11

12

13

**1.6.3.3**

调用封装的函数获取数据

1

2

**1.6.3.4**

调用封装的函数存储数据

1

2

3

4

5

save\_data(today\_world,’today\_world’)

------------------------------------

输出:today\_world\_2020\_04\_27.csv 保存成功！

------------------------------------

today\_world = get\_data(areaTree,[’id’,’lastUpdateTime’,’name’]) # areaTree today\_world.head()

for i in range( len(areaTree)): # 查看各国家名称和更新时同print(areaTree[i][’name’],areaTree[i][’lastUpdateTime’]) if i == 5:

break

----------------------

输出:突尼斯 2020-04-27 10:16:19

塞尔维亚 2020-04-27 00:00:31

中国 2020-04-27 09:47:46

日本 2020-04-27 00:00:31

泰国 2020-04-27 00:00:31

新加坡 2020-04-27 00:00:31

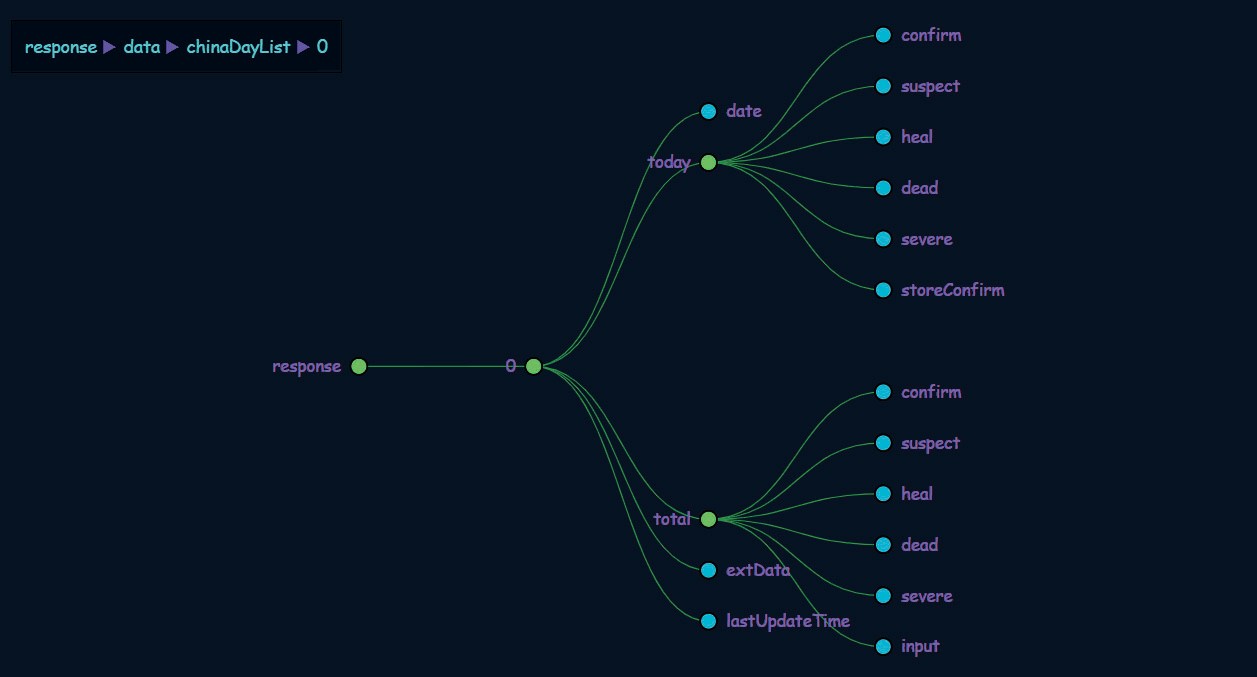
-----------------------

’name’: ’突尼斯’,

’id’: ’9577772’,

’lastUpdateTime’: ’2020-04-27␣10:16:19’, ’children’: []}

------------------------



1.7 历史数据爬取

– 22 –

**1.7** 历史数据爬取

**1.7.1** 全国历史数据爬取

我们打开data 数据下面的chinadaylist 的第一个数据得到如下的数据层级! 同上数据层级关系变化不是太大？因此我们可以直接进行抓取!

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

data.keys() #查看data的键名

------------------------------------

输出:dict\_keys([’chinaTotal’, ’chinaDayList’, ’lastUpdateTime’, ’overseaLastUpdateTime’, ’areaTree’])

------------------------------------

chinaDayList = data[’chinaDayList’] # 取出chinaDayList type(chinaDayList) # 查看chinaDayList的格武

------------------------------------

输出： list

------------------------------------

chinaDayList[0] #查看第一个数据

------------------------------------

输出：{’date’: ’2020-01-20’,

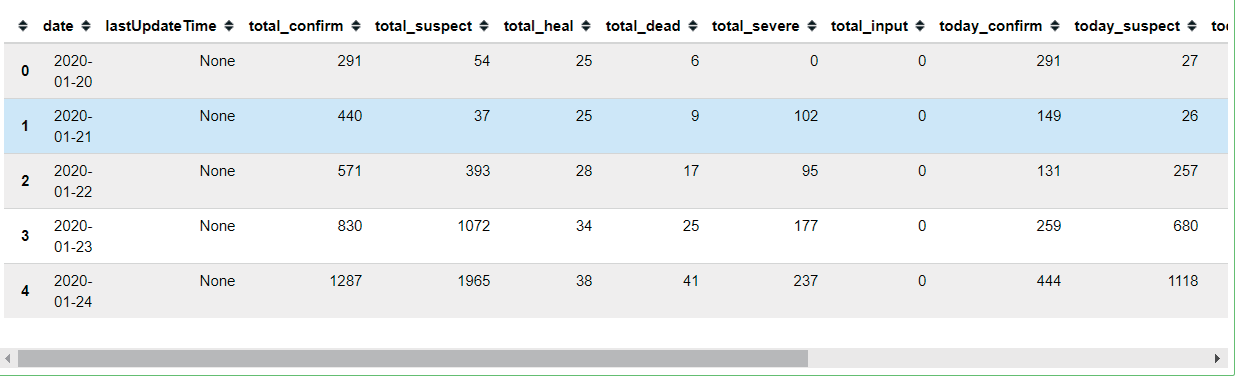
’today’: {’confirm’: 291,

’suspect’: 27,

’heal’: 25,

’dead’: 6,

’severe’: 0, ’storeConfirm’: None}, ’total’: {’confirm’: 291,



1.7 历史数据爬取

– 23 –

25

26

27

28

29

30

31

32

**1.7.1.1**

调用函数抓取数据

1

2

**1.7.1.2**

调用函数存储数据

1

2

3

4

5

**1.7.2** 全国各省历史数据爬取

首先找到§标地址!

save\_data(alltime\_China,’alltime\_China’)

------------------------------------

输出:alltime\_China\_2020\_04\_27.csv 保存成功！

------------------------------------

alltime\_China = get\_data(chinaDayList,[’date’,’lastUpdateTime’]) #调用getdata抓取alltime\_China.head()

’suspect’: 54,

’heal’: 25,

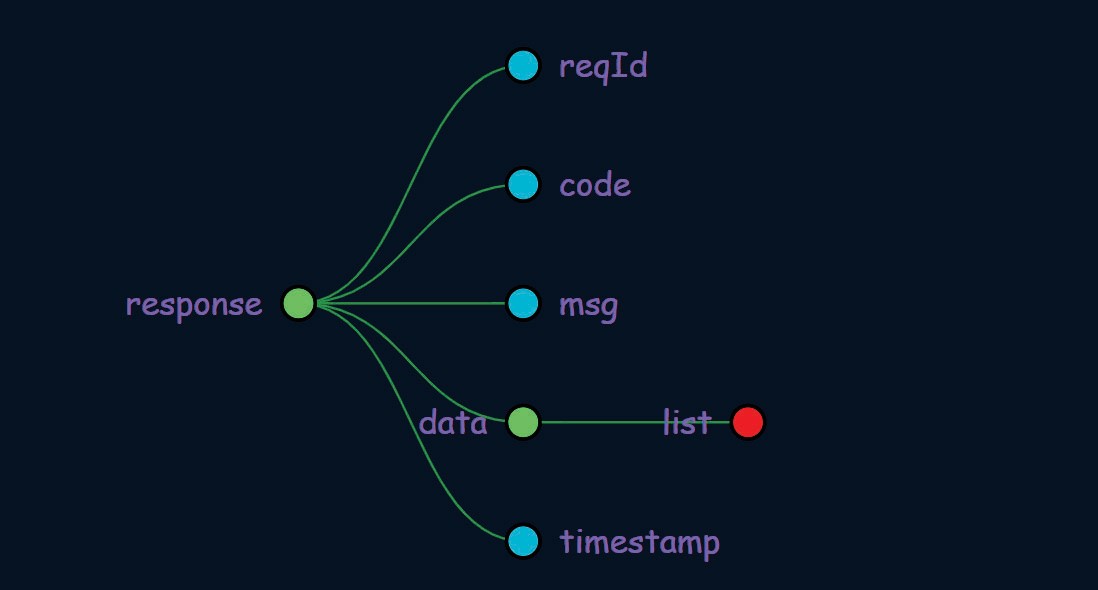
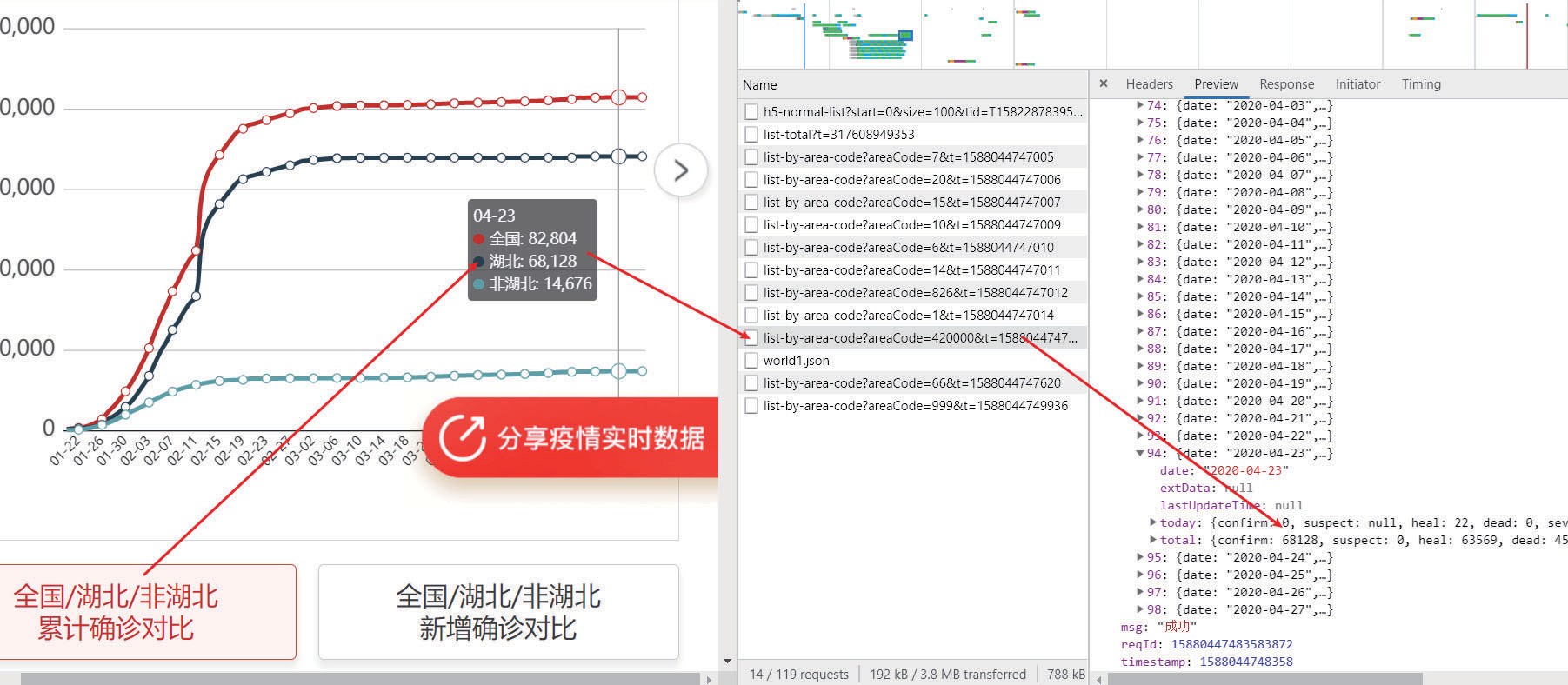
’dead’: 6,

’severe’: 0,

’input’: 0}, ’extData’: None,

’lastUpdateTime’: None}

------------------------------------



1.7 历史数据爬取

– 24 –

从而得到湖北省的历史数据网址:湖北省历史数据网址:

先以其中湖北省为例，先尝试获取其历史数据，其他的省可以使用同样的方法。

1

2

3

4

5

6

7

8

9

10

11

这是对应的数据关系!, 再此我们尝试查看data 下list 中第一个数据层

1

2

3

data\_json[’data’][’list’][0]

------------------------------------

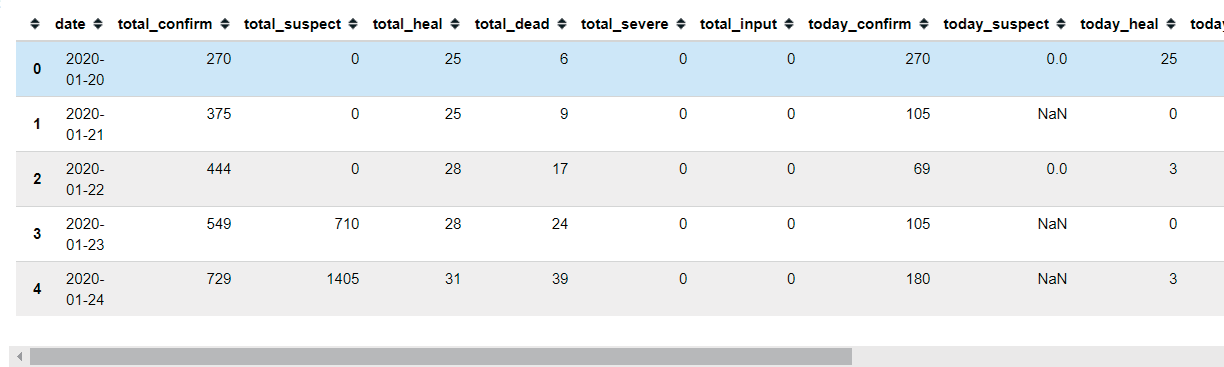
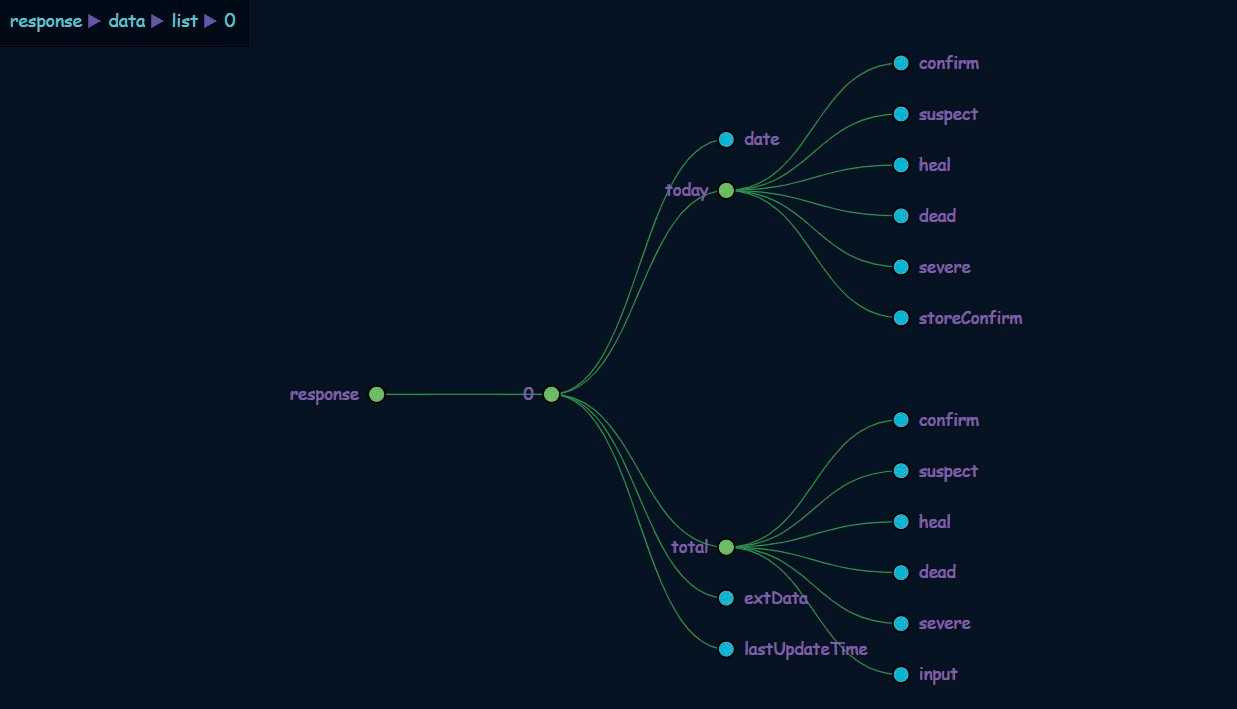
url = ’https://c.m.163.com/ug/api/wuhan/app/data/list-by-area-code?areaCode=420000’ # 定义数据地址

r = requests.get(url, headers=headers) # 进行请求data\_json = json.loads(r.text) # 获取json数据data\_json.keys() #查看键名

------------------------------------

输出:dict\_keys([’reqId’, ’code’, ’msg’, ’data’, ’timestamp’])

------------------------------------



1.7 历史数据爬取

– 25 –

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

可以看出一个省一天的的数据格式如上，这和之前的数据结构一样，因此可以使用之前的方法得

到数据。

1

2

3

data\_test = get\_data(data\_json[’data’][’list’],[’date’]) data\_test[’name’] = ’湖北省’

data\_test.head()

输出:{’date’: ’2020-01-20’,

’today’: {’confirm’: 270,

’suspect’: 0,

’heal’: 25,

’dead’: 6, ’severe’: None,

’storeConfirm’: None}, ’total’: {’confirm’: 270,

’suspect’: 0,

’heal’: 25,

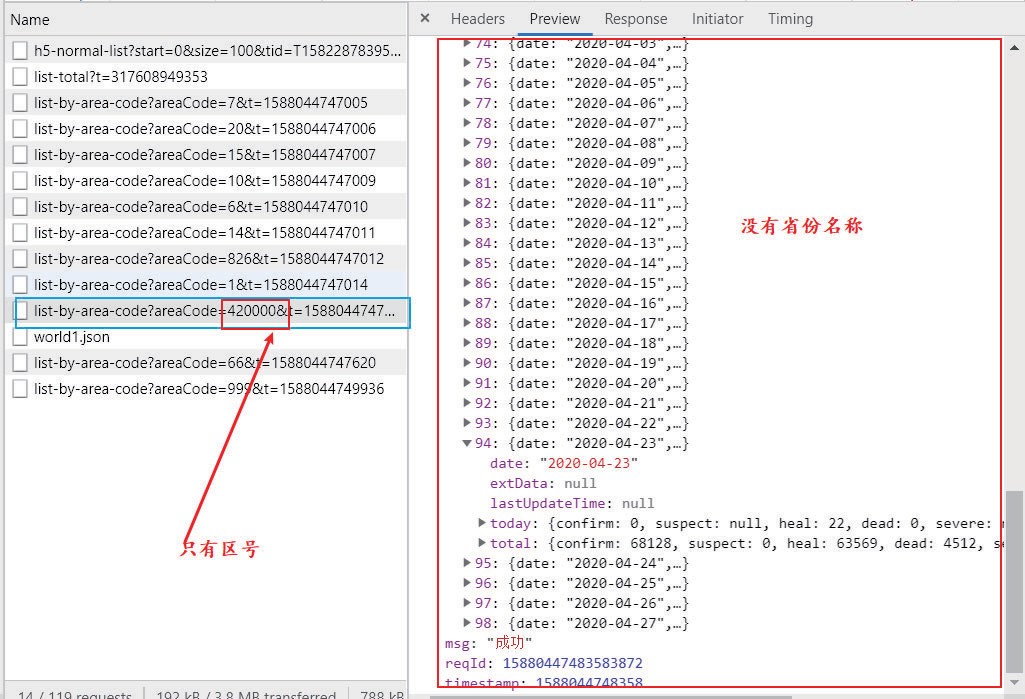
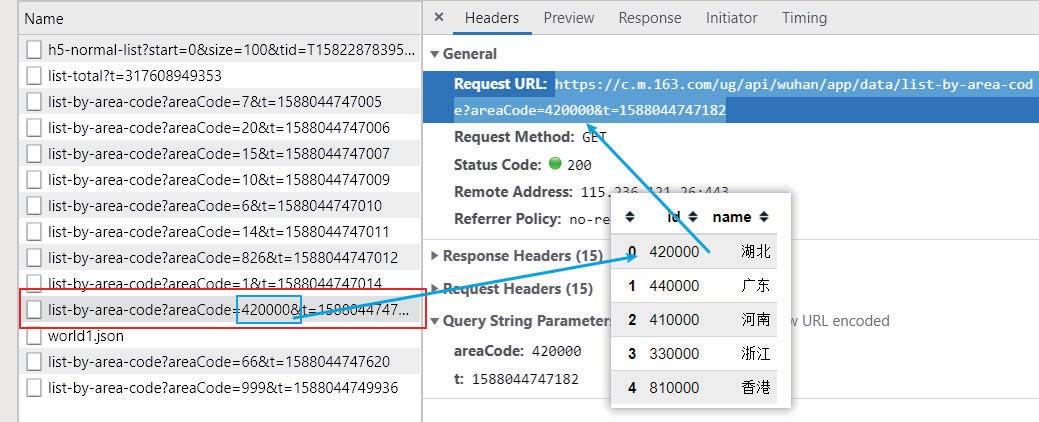
’dead’: 6,

’severe’: 0,

’input’: 0}, ’extData’: None,

’lastUpdateTime’: None}

------------------------------------



1.7 历史数据爬取

– 26 –

通过上述方法得到了湖北省的历史数据，想要得到每个省的历史数据怎么做呢？

在湖北省历史数据的地址中，我们发现参数aeraCode=420000 ，而这刚好和全国各省实时数据 today\_province 中的id 对应.

1

这个可以进行验证中华人民共和国行政区划

因此为了得到每个省的历史数据，我们只需要将各省的行政代码作为参数传入这个地址即 可，如下所示：

**https://c.m.163.com/ug/api/wuhan/app/data/list-by-area-code?areaCode=** 各省行政代码广 东 省 历 史 数 据 广 东 省 历 史 数 据 的 地 址 为 ： **https://c.m.163.com/ug/api/wuhan/app/data/list-by-area-code?areaCode=440000**

湖南省历史数据的地址为：

**https://c.m.163.com/ug/api/wuhan/app/data/list-by-area-code?areaCode=430000**

但是数据中并没有显示省的名称，因此需要写入每个省的名称。

为了便千写入各省的名称，我们需要生成一个各省行政代码和省名称对应的字典。

today\_province[[’id’,’name’]].head()



1.7 历史数据爬取

– 27 –

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

每一个省的列名是相同的，因此多个省的数据合并起来就可以存入一个数据中！

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

start = time.time()

for province\_id in province\_dict: # 遍历各省编号

try:

# 按照省编号访问每个省的数据地址，并获取json数据

url =’https://c.m.163.com/ug/api/wuhan/app/data/list-by-area-code?areaCode=’+ province\_id

req2 = requests.get(url, headers=headers) data\_json = json.loads(req2.text)

# 提取各省数据，然后写入各省名称

province\_data = get\_data(data\_json[’data’][’list’],[’date’]) province\_data[’name’] = province\_dict[province\_id]

# 合并数据

if province\_id == ’420000’:

alltime\_province = province\_data else:

alltime\_province = pd.concat([alltime\_province,province\_data]) print(’-’\*20,province\_dict[province\_id],’成功’,province\_data.shape,

alltime\_province.shape,’,累计耗时:’, round(time.time()-start),’-’\*20)

# 设置廷迟等待

time.sleep(20) except:

print(’-’\*20,province\_dict[province\_id],’wrong’,’-’\*20)

province\_dict = {num:name for num,name

in zip(today\_province[’id’],today\_province[’name’])}

# 不要刻意换行，这是一行代码！

# 查看前五个内容

count = 0

for i in province\_dict: print(i,province\_dict[i]) count += 1

if count == 5:

break

------------------------------------

输出:420000 湖北

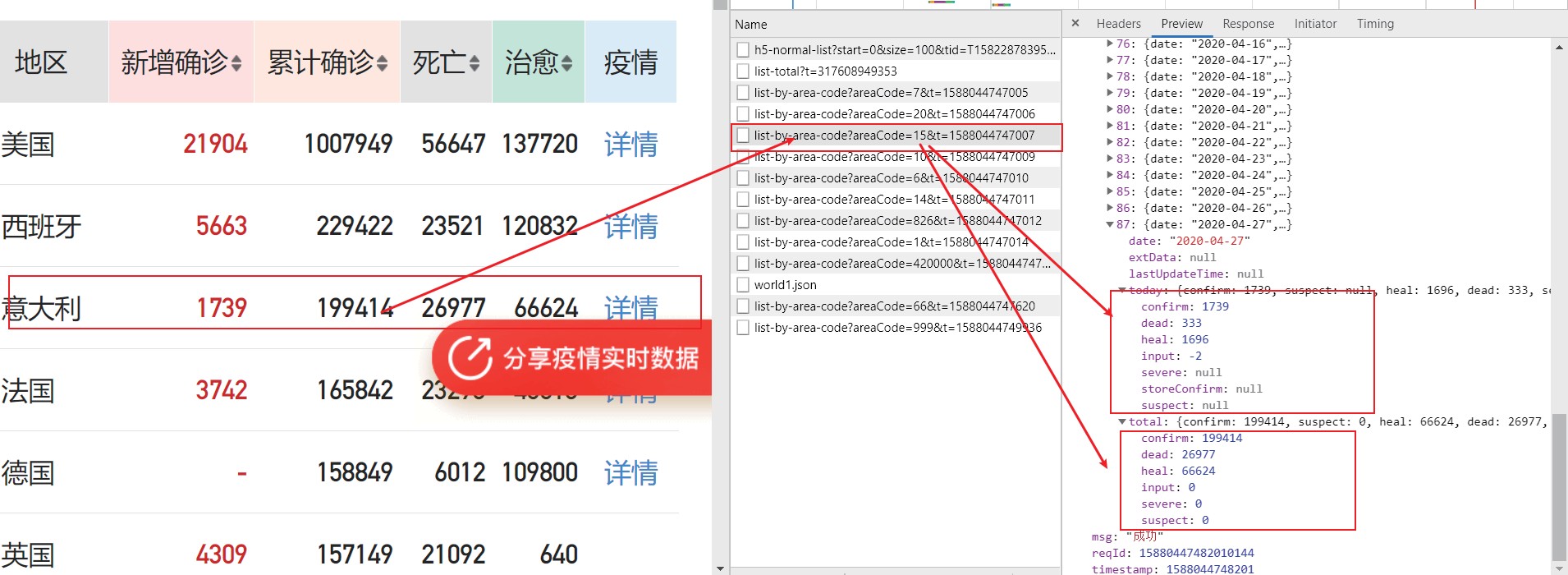
440000 广东

410000 河南

330000 沿工

430000 湖南

------------------------------------



1.7 历史数据爬取

– 28 –

26

27

28

29

30

31

32

33

保存数据

1

2

3

4

5

**1.7.3** 世界各国历史数据爬取

接着找到下一个数据地址，对比后发现是意大利数据，首先尝试爬取意大利的历史数据，其 他国家的数据可以用同样的方法爬取。

1

2

3

4

5

6

7

8

9

10

11

12

13

14

url\_italy = ’https://c.m.163.com/ug/api/wuhan/app/data/list-by-area-code?areaCode=15’ #

意大利的数据地址

r = requests.get(url\_italy, headers=headers) # 进行访问

italy\_json = json.loads(r.text) # 导出json数据

italy\_json.keys()

------------------------------------

输出:dict\_keys([’reqId’, ’code’, ’msg’, ’data’, ’timestamp’])

------------------------------------

italy\_json[’data’].keys()

------------------------------------

输出:dict\_keys([’list’])

------------------------------------

save\_data(alltime\_province,’alltime\_province’)

------------------------------------

输出:alltime\_province\_2020\_04\_27.csv 保存成功！

------------------------------------

------------------------------------

输出:

-------------------- 湖北 成功 (98, 15) (98, 15) ,累计耗时: 0 --------------------

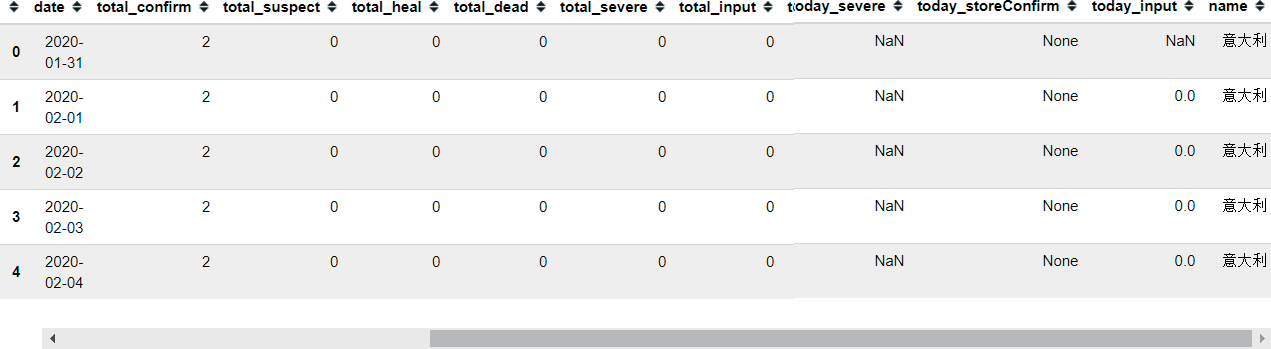
-------------------- 广东 成功 (95, 15) (193, 15) ,累计耗时: 20 --------------------

-------------------- 河南 成功 (97, 15) (290, 15) ,累计耗时: 41 --------------------

-------------------- 沿工 成功 (94, 15) (384, 15) ,累计耗时: 61 --------------------

-------------------- 香港 成功 (97, 15) (481, 15) ,累计耗时: 81 --------------------

------------------------------------



1.7 历史数据爬取

– 29 –

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

1

2

3

因为原始数据中没有国家名称，为了得到每个国家的名称，需要生成国家编号和国家名称的 键值对，这样就可以存储国家名称，在之前的世界各国实时数据 today\_world 中有国家的编号和

名称，可以用它来生成键值对。

1

today\_world[[’id’,’name’]].head()

data\_italy = get\_data(italy\_json[’data’][’list’],[’date’]) # 生成数据

data\_italy[’name’] = ’意大利’ # 写入意大利国家名称

data\_italy.head()

italy\_json[’data’][’list’][0] # 查看数据内容

------------------------------------

输出:{’date’: ’2020-01-31’,

’today’: {’confirm’: 2,

’suspect’: 0,

’heal’: 0,

’dead’: 0, ’severe’: None,

’storeConfirm’: None}, ’total’: {’confirm’: 2,

’suspect’: 0,

’heal’: 0,

’dead’: 0,

’severe’: 0,

’input’: 0}, ’extData’: None,

’lastUpdateTime’: None}

------------------------------------



1.7 历史数据爬取

– 30 –

生成数据字典!

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

通过每个国家的编号访问每个国家历史数据的地址，然后获取每一个国家的历史数据

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

round(time.time()-start),’-’\*20)

20

21

22

23

time.sleep(20) except:

print(’-’\*20,country\_dict[country\_id],’wrong’,’-’\*20)

start = time.time()

for country\_id in country\_dict: # 遍历每个国家的编号

try:

# 按照编号访问每个国家的数据地址，并获取json数据

url = ’https://c.m.163.com/ug/api/wuhan/app/data/list-by-area-code?areaCode=’+ country\_id

r = requests.get(url, headers=headers)

json\_data = json.loads(r.text)

# 生成每个国家的数据

country\_data = get\_data(json\_data[’data’][’list’],[’date’]) country\_data[’name’] = country\_dict[country\_id]

# 数据叠加

if country\_id == ’9577772’:

alltime\_world = country\_data else:

alltime\_world = pd.concat([alltime\_world,country\_data])

print(’-’\*20,country\_dict[country\_id],’成功’,country\_data.shape,alltime\_world.shape,

’,累计耗时:’,

country\_dict = {key:value for key,value in zip(today\_world[’id’], today\_world[’name’])}

# 查看前五个内容

count = 0

for i in country\_dict: print(i,country\_dict[i]) count += 1

if count == 5:

break

------------------------------------

输出:

9577772 突尼斯

9507896 塞尔维亚

1. 中国
2. 日本
3. 泰国

------------------------------------



1.8 总结

– 31 –

24

25

26

27

28

29

30

保存数据

1

2

3

4

5

**1.8** 总结

本篇案例的主要内容是新冠肺炎疫情的数据采集，首先我们寻找合适的数据源，最终选择网 易的实时疫情播报平台作为数据源，然后逐步地解析找到我们需要的数据，最后通过 python 爬虫获得我们得到了疫情的数据

**1.8.1** 核心命令总结

save\_data(alltime\_world,’alltime\_world’)

------------------------------------

输出:alltime\_world\_2020\_04\_27.csv 保存成功！

------------------------------------

------------------------------------

输出:

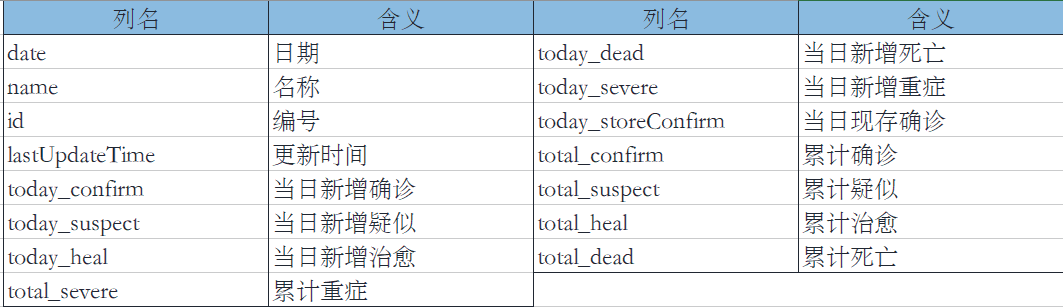
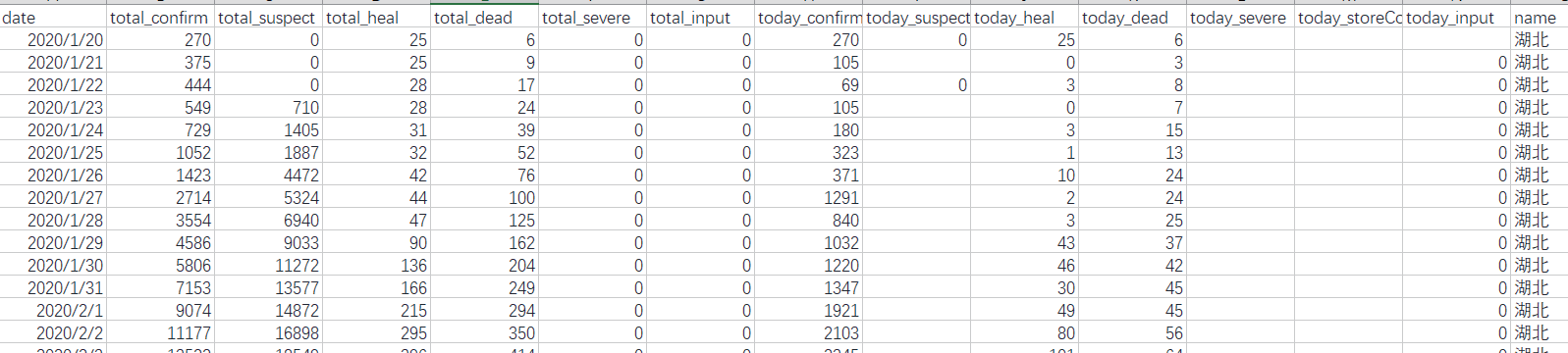
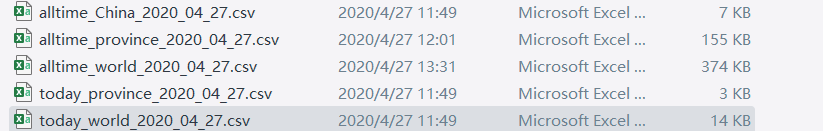
-------------------- 突尼斯 成功 (47, 15) (47, 15) ,累计耗时: 0 --------------------

-------------------- 塞尔维亚 成功 (46, 15) (93, 15) ,累计耗时: 20 --------------------

-------------------- 中国 成功 (83, 15) (176, 15) ,累计耗时: 41 --------------------

-------------------- 日本 成功 (85, 15) (261, 15) ,累计耗时: 61 --------------------

------------------------------------



第二章 **Pandas** 疫情探索性分析

**2.1** 数据及 **Pandas** 工具介绍

**2.1.1** 数据介绍

在上一章节案例中我们基千网易实时疫情播报平台，使用 Python 对疫情数据进行了爬取。本篇案例的主要内容是新冠肺炎疫情数据的探索性分析，包括中国各省和世界各国的实时数据，及 中国和世界各国的历史数据。

图 **2.1:** 数据示例如上

数据集中出现的特征和含义如下表所示：

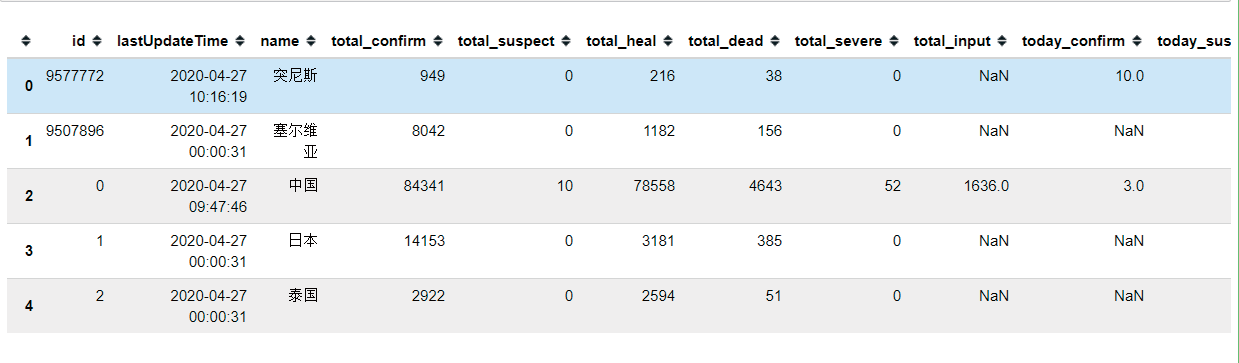
图 **2.2:** 数据示例如上

在上述字段列表，存在很多空值，无效值，如何处理这些数据，将是这章内容的重点! 下面

我们看关千Pandas 工具的介绍!

**2.1.2 Pandas** 介绍

Pandas 是一个开源的、专注千数据分析的 Python 库。最初山 AQR Capital Management 千 2008 年 4 月开发，并千 2009 年底开源出来，§前山专注千 Python 数据包开发的PyData 开发小组继续开发和维护，属千 PyData 项§的一部分。Pandas 最初被作为金融数据分析工具而开发出来，因此，Pandas 为时间序列分析提供了很好的支持。



2.2 实时数据探索性分析

– 33 –

Pandas 是基千NumPy 数组构建的，能够灵活处理关系型数据，可便捷的完成索引、切片、组

合以及选取数据子集等操作。接下来就让我们一起使用Pandas 对疫情数据进行探索性分析。

**2.2** 实时数据探索性分析

**2.2.1** 世界各国实时数据探索性分析

我们首先读入数据，将列名英文改为中文。接着，查看数据的基本信息并进行缺失值处理。此外，我们还将新增病死率一列，并将国家设置为索引。数据预处理之后我们将查看世界当前累计确诊人数前十名的国家，并绘制累计确诊、累计死亡和病死率的水平条形图来分析各国疫情状况。

1

2

3

4

5

6

7

数据表各列名称为英文不便千观察，我们将列名修改为中文。首先，创建中英文对照的列名

字典，使用rename() 函数修改列名:

1

2

3

4

5

6

7

8

9

10

11

name\_dict = {’date’:’日期’,’name’:’名称’,’id’:’编号’,’lastUpdateTime’:’更新时同’,

’today\_confirm’:’当日新增确诊’,’today\_suspect’:’当日新增疑似’,

’today\_heal’:’当日新增冶愈’,’today\_dead’:’当日新增死亡’,

’today\_severe’:’当日新增重症’,’today\_storeConfirm’:’当日现存确诊’,

’total\_confirm’:’累计确诊’,’total\_suspect’:’累计疑似’,

’total\_heal’:’累计冶愈’,’total\_dead’:’累计死亡’,’total\_severe’:’累计重症’}

# 更改列名

today\_world.rename(columns=name\_dict,inplace=True) # inplace参数判断是否在原数据上进行修改

today\_world.head(5)

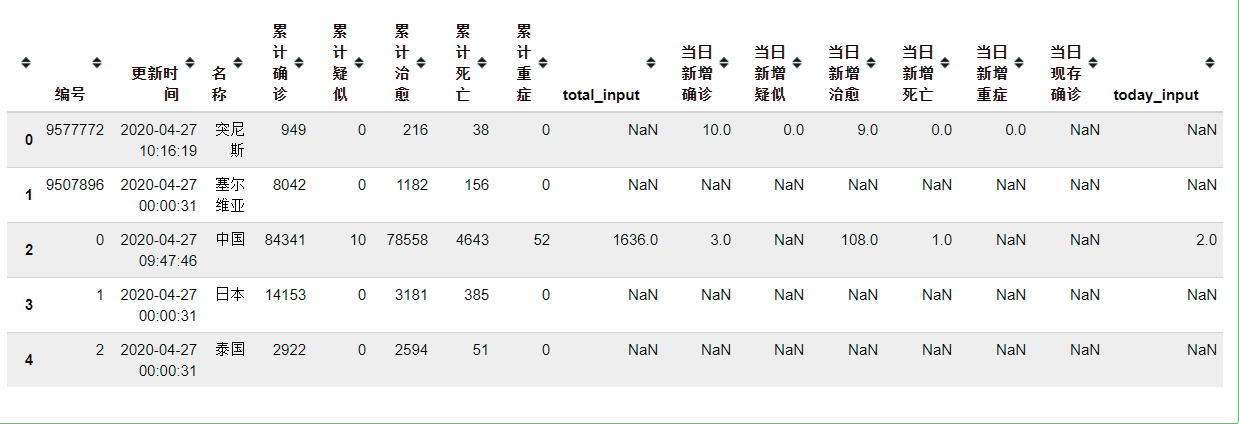
import pandas as pd

# 读取数据

today\_world = pd.read\_csv("today\_world\_2020\_04\_27.csv")

# 查看世界各国实时数据

today\_world.head()



2.2 实时数据探索性分析

– 34 –

当我们拿到一份数据的时候，首先需要观察一下数据的基本信息和特征的统计信息。我们可

以使用 info() 查看数据的基本信息：

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

查看数据的统计信息可以使用 describe() 函数：

1

2

# 默认只计算数值型特征的统计信息

today\_world.describe()

# 查看数据基本信息

today\_world.info()

--------------------------

输出：<class ’pandas.core.frame.DataFrame’> RangeIndex: 203 entries, 0 to 202

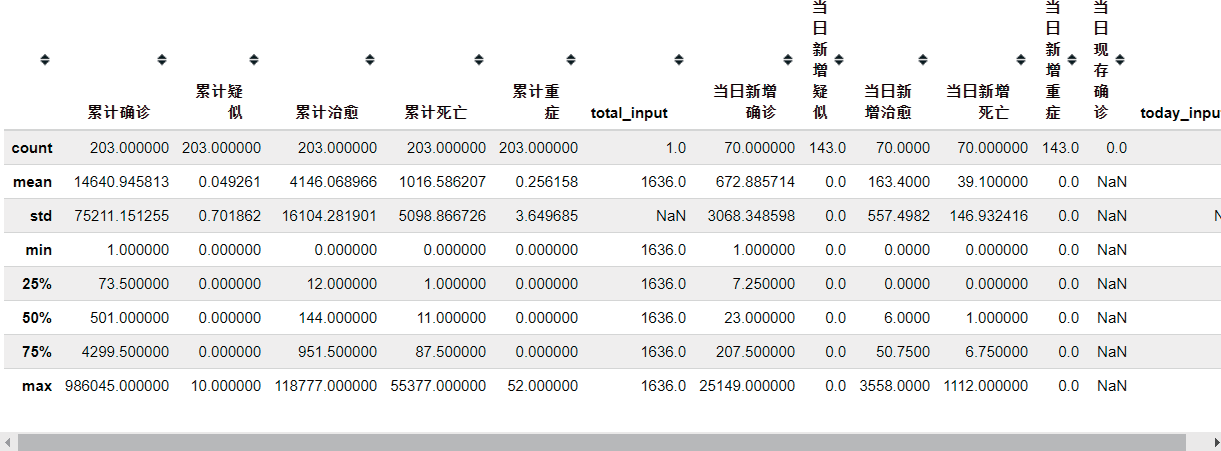
Data columns (total 16 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

1. 编号 203 non-null object
2. 更新时同 203 non-null object
3. 名称 203 non-null object
4. 累计确诊 203 non-null int64
5. 累计疑似 203 non-null int64
6. 累计冶愈 203 non-null int64
7. 累计死亡 203 non-null int64
8. 累计重症 203 non-null int64
9. total\_input 1 non-null float64
10. 当日新增确诊 70 non-null float64
11. 当日新增疑似 143 non-null float64
12. 当日新增冶愈 70 non-null float64
13. 当日新增死亡 70 non-null float64
14. 当日新增重症 143 non-null float64
15. 当日现存确诊 0 non-null float64
16. today\_input 1 non-null float64 dtypes: float64(8), int64(5), object(3) memory usage: 25.5+ KB

-------------------------------



2.2 实时数据探索性分析

– 35 –

数据中当日新增确诊、疑似、治愈、死亡、重症和当日现存确诊中存在大量缺失值。为了便

千观察，我们使用isnull() 函数查看缺失值，并结合sum() 函数计算缺失值比例。

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

我们发现当日新增相关数据缺失值较多，这主要山千采集数据的当天一些国家没有更新数 据，因此我们将不再对其进行分析。当日现存确诊一列虽然全部为空，但该缺失值可以通过已有

数据直接计算，公式为：

当日现存确诊 = 累计确诊累计治愈累计死亡

1

# 缺失值处理

# 计算缺失值比例

today\_world\_nan = today\_world.isnull(). sum()/ len(today\_world)

# 转变为百分数

today\_world\_nan. apply(lambda x: format(x, ’.1%’))

--------------------------

输出：编号 0.0%

更新时同 0.0%

名称 0.0%

累计确诊 0.0%

累计疑似 0.0%

累计冶愈 0.0%

累计死亡 0.0%

累计重症 0.0%

total\_input 99.5%

当日新增确诊 65.5%

当日新增疑似 29.6%

当日新增冶愈 65.5%

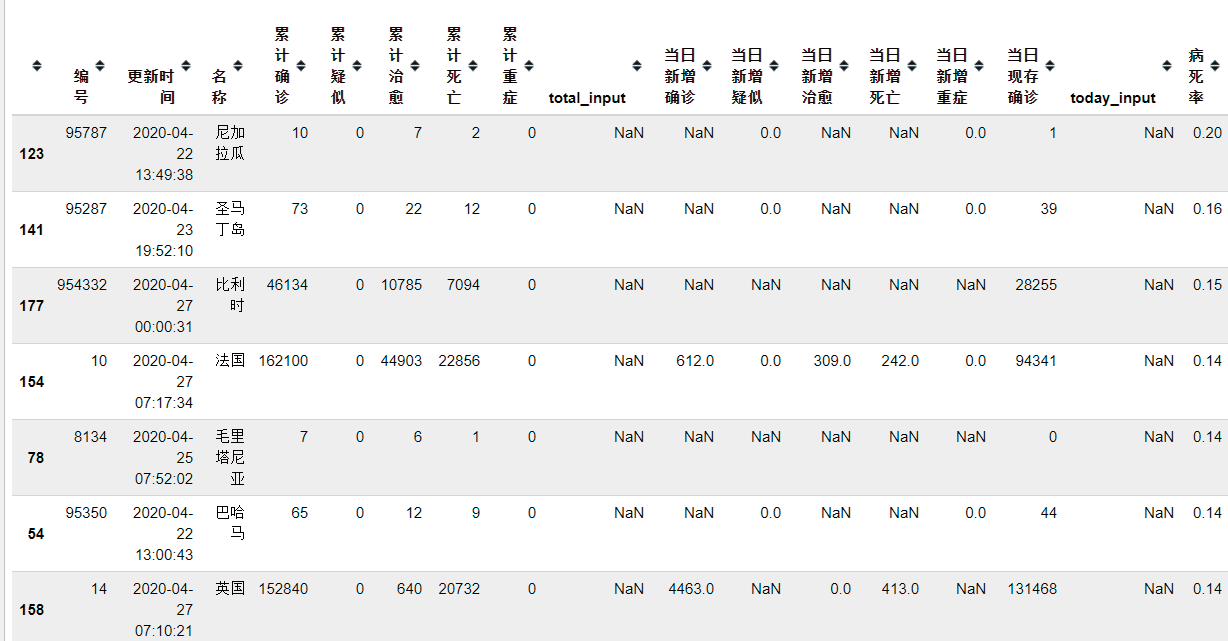
当日新增死亡 65.5%

当日新增重症 29.6%

当日现存确诊 100.0%

today\_input 99.5% dtype: object

-----------------------



2.2 实时数据探索性分析

– 36 –

2

除了我们数据中提供的这些特征，病死率也是一个非常重要的特征，能够反映疾病的严重程度以 及一个地区的臣疗水平。接下来，我们来看看截止数据采集当天，各国的病死率情况。病死率的

计算公式为：

病死率 = 累计死亡累计确诊

1

2

apply(lambda x:

format(x, ’.2f’))

3

4

5

6

7

8

9

10

11

排名第一的国家尼加拉瓜病死率高达 0.2，但从表中可知，该国累计确诊人数只有 10 例。可见，病死率应结合累计确诊人数一起查看。

为了方便查询特定国家的数据，我们使用 set\_index() 函数将国家设置为索引：

1

2

3

4

# 将国家名称设为索引

today\_world.set\_index(’名称’,inplace=True)

today\_world.head(3)

# 将病死率数据类型转换为float

today\_world[’病死率’] = today\_world[’病死率’].astype(’float’)

# 根据病死率降序排序

today\_world.sort\_values(’病死率’,ascending=False,inplace=True)

# 显示病死率前十国家

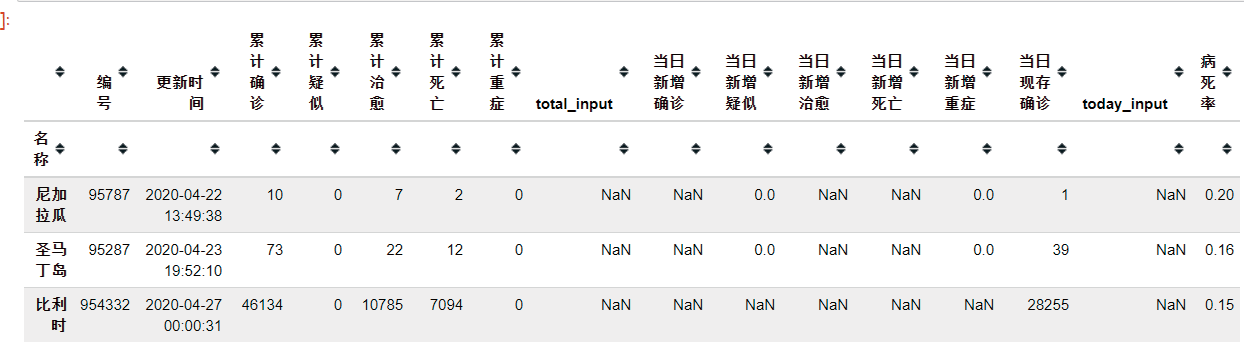
today\_world.head(10)

# 计算病死率,且保留两位小数

today\_world[’病死率’] = (today\_world[’累计死亡’]/today\_world[’累计确诊’]).

today\_world[’当日现存确诊’] = today\_world[’累计确诊’]-today\_world[’累计冶愈’]-

today\_world[’累计死亡’]



2.2 实时数据探索性分析

– 37 –

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

当前累计确诊人数 **top10** 国家接下来，让我们使用sort\_values() 函数根据累计确诊人数进行排序，

找到确诊人数前十国家的累计确诊、累计死亡、病死率三列数据：

1

2

3

4

5

6

# 查看当前累计确诊人数前十国家

world\_top10 = today\_world.sort\_values([’累计确诊’],ascending=False)[:10] world\_top10 = world\_top10[[’累计确诊’,’累计死亡’,’病死率’]]

world\_top10

today\_world.loc[’中国’] #可以通过传入列表获取多个国家的数据

---------------------------

输出:编号 0

更新时同 2020-03-31 18:11:03

累计确诊 82599

累计疑似 183

累计冶愈 76237

累计死亡 3314

累计重症 528

当日新增确诊 151

当日新增疑似 44

当日新增冶愈 300

当日新增死亡 3

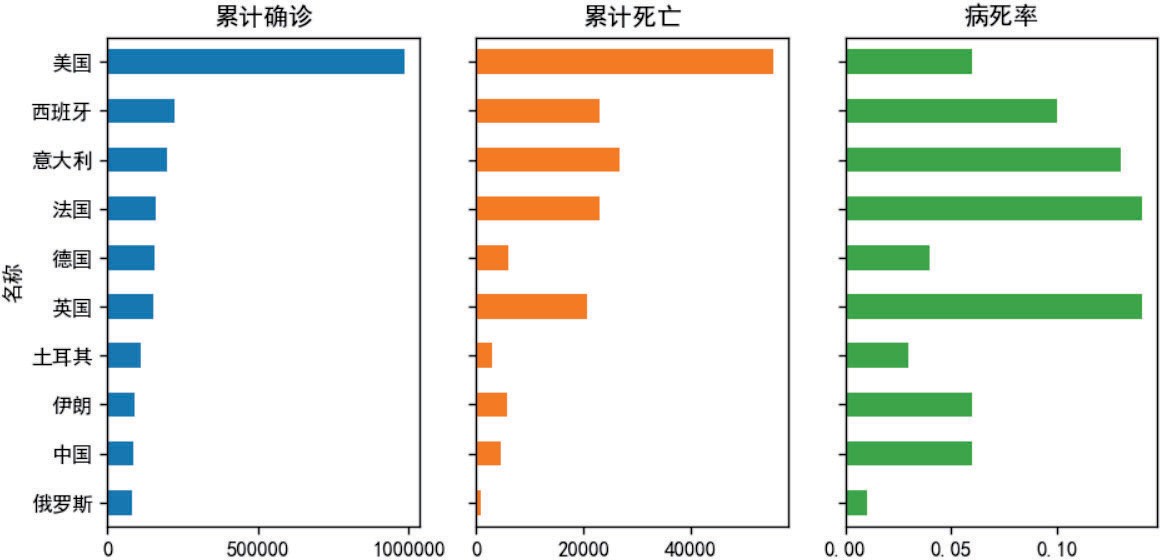
当日新增重症 -105

当日现存确诊 3048

病死率 0.04

Name: 中国, dtype: object

------------------------------



2.2 实时数据探索性分析

– 38 –

直接观察数据并不直观，我们需要借助可视化方法来进行分析。

1

2

3

4

5

6

7

8

9

10

11

山图可知，§前美国累计确诊病例已远远高千其他国家，意大利、西班牙人数也相对较高。而在病死率的图表上意大利则位列第一，可见疫情的严重性，西班牙、法国、伊朗、英国、比利时病死率紧跟其后。然而，累计确诊人数最高的美国病死率却相对较低，其中一个原因就是美国

的臣疗资源丰富，下图是一张每十万人 ICU 病床数量的国家排名条形图：

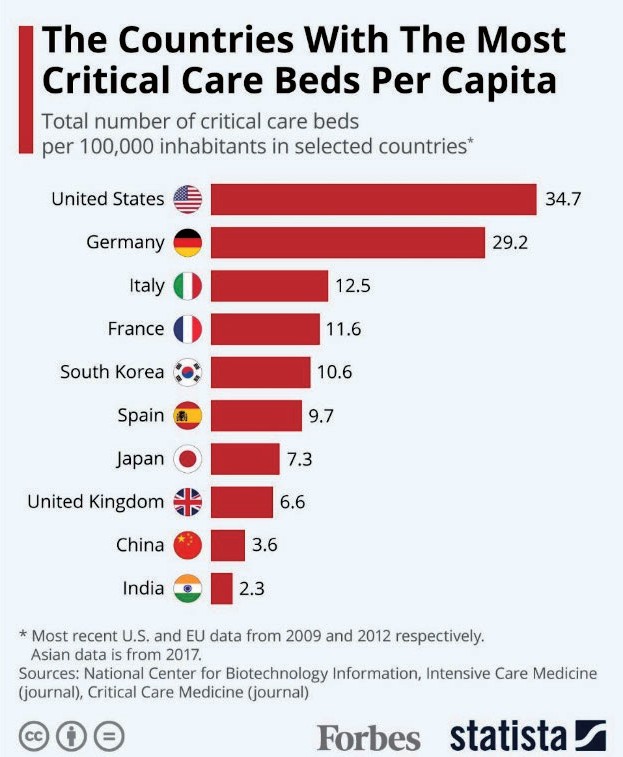
import matplotlib.pyplot as plt

plt.rcParams[’font.sans-serif’]=[’SimHei’] #正常显示中文plt.rcParams[’figure.dpi’] = 120 #设置所有图片的清晰度

# 绘制条形图

world\_top10.sort\_values(’累计确诊’).plot.barh(subplots=True,layout=(1,3),sharex=False, figsize=(7,4),legend=False,sharey=True)

plt.tight\_layout() #自动调整子图同距plt.show()



2.2 实时数据探索性分析

– 39 –

我们发现，美国每十万居民拥有将近 35 个ICU 床位，排在世界第一，德国紧跟其后有近 30

个 ICU 床位，两国的臣疗资源远远高千其他国家，这也是新冠肺炎疫病死率相对较低的原因之一。

**2.2.2** 全国各省实时数据探索性分析

下面让我们一起来分析国内的新冠肺炎疫情情况。全国各省实时数据的预处理工作与前面 的流程基本一致，数据清洗之后我们将分别探索全国新增确诊 top10 地区，及全国现存确诊人数

top10 的地区。

1

2

3

4

5

6

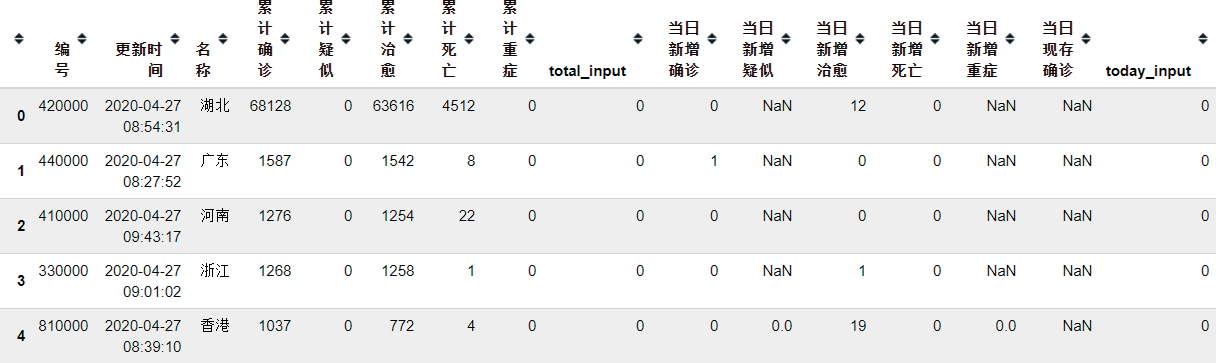
# 读取数据

today\_province = pd.read\_csv("today\_province\_2020\_04\_27.csv")

# 创建中文列名字典

name\_dict = {’date’:’日期’,’name’:’名称’,’id’:’编号’,’lastUpdateTime’:’更新时同’,

’today\_confirm’:’当日新增确诊’,’today\_suspect’:’当日新增疑似’,



2.2 实时数据探索性分析

– 40 –

7

8

9

10

11

12

13

14

15

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

# 查看数据基本信息

today\_province.info()

---------------------------

输出:<class ’pandas.core.frame.DataFrame’> RangeIndex: 34 entries, 0 to 33

Data columns (total 16 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

1. 编号 34 non-null int64
2. 更新时同 34 non-null object
3. 名称 34 non-null object
4. 累计确诊 34 non-null int64
5. 累计疑似 34 non-null int64
6. 累计冶愈 34 non-null int64
7. 累计死亡 34 non-null int64
8. 累计重症 34 non-null int64
9. total\_input 34 non-null int64
10. 当日新增确诊 34 non-null int64
11. 当日新增疑似 3 non-null float64
12. 当日新增冶愈 34 non-null int64
13. 当日新增死亡 34 non-null int64
14. 当日新增重症 3 non-null float64
15. 当日现存确诊 0 non-null float64
16. today\_input 34 non-null int64

’today\_heal’:’当日新增冶愈’,’today\_dead’:’当日新增死亡’,

’today\_severe’:’当日新增重症’,’today\_storeConfirm’:’当日现存确诊’,

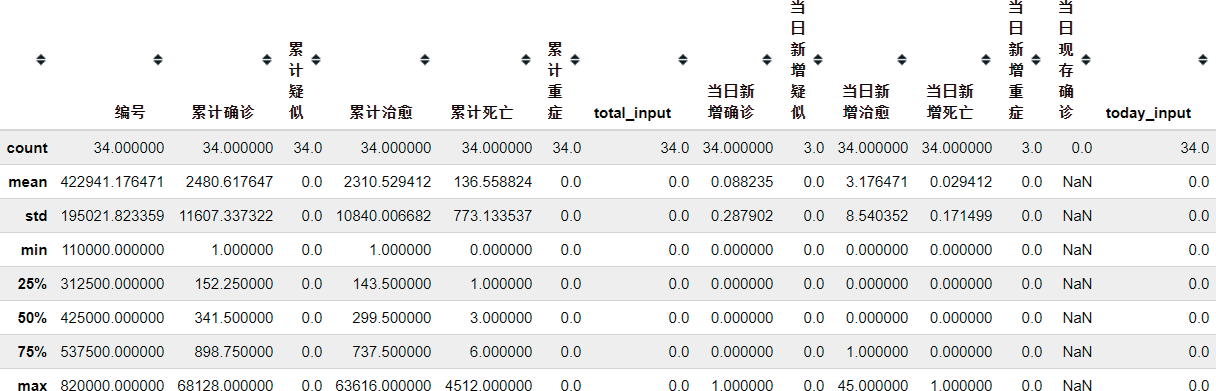
’total\_confirm’:’累计确诊’,’total\_suspect’:’累计疑似’,

’total\_heal’:’累计冶愈’,’total\_dead’:’累计死亡’,’total\_severe’:’累计重症’}

# 更改列名

today\_province.rename(columns=name\_dict,inplace=True) # inplace参数是否在原对象基础上进行修改

today\_province.head()



2.2 实时数据探索性分析

– 41 –

26

27

28

使用describe() 查看数据的统计信息：

1

2

我们发现，累计疑似、累计重症、当日新增疑似和当日新增重症的数据全部为 0，因此我们

便不予考虑。

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

# 计算各省当日现存确诊人数

today\_province[’当日现存确诊’] = today\_province[’累计确诊’]-today\_province[’累计冶愈’]-

today\_province[’累计死亡’]

# 将各省名称设置为索引

today\_province.set\_index(’名称’,inplace=True) today\_province.info()

--------------------------------

输出:<class ’pandas.core.frame.DataFrame’> Index: 34 entries, 湖 北 to 西 藏

Data columns (total 15 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

1. 编号 34 non-null int64
2. 更新时同 34 non-null object
3. 累计确诊 34 non-null int64
4. 累计疑似 34 non-null int64
5. 累计冶愈 34 non-null int64
6. 累计死亡 34 non-null int64
7. 累计重症 34 non-null int64
8. total\_input 34 non-null int64
9. 当日新增确诊 34 non-null int64
10. 当日新增疑似 3 non-null float64

# 查看数值型特征的统计量

today\_province.describe()

dtypes: float64(3), int64(11), object(2) memory usage: 4.4+ KB

------------------------------



2.2 实时数据探索性分析

– 42 –

25

26

27

28

29

30

31

32

全国新增确诊 **top10** 地区

§前，我国疫情已得到很好的控制，我们现在更为关注的是新增确诊病例的地区。

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

1

2

3

4

5

6

7

8

9

# 绘制条形图和饼图

fig,ax = plt.subplots(1,2,figsize=(10,5))

new\_top6.sort\_values(ascending=True).plot.barh(fontsize=10,ax=ax[0]) new\_top6.plot.pie(autopct=’%.1f%%’,fontsize=10,ax=ax[1])

plt.ylabel(’’)

plt.title(’全国新增确诊top10地区’,size=15) plt.show()

# 查看全国新增确诊top10的地区

new\_top6 = today\_province[’当日新增确诊’].sort\_values(ascending=False)[:10] new\_top6

---------------------------------

输出：名称黑龙工 1

广东 1

吉林 1

西藏 0

重庆 0

北京 0

上海 0

工苏 0

山东 0

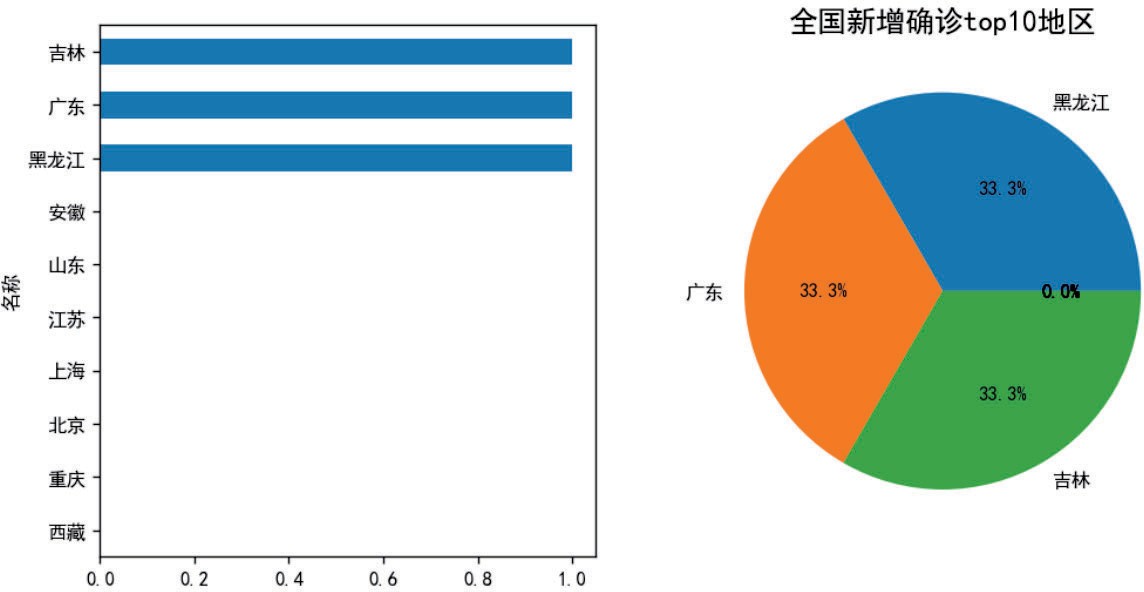
安徽 0

Name: 当日新增确诊, dtype: int64

------------------------------------

1. 当日新增冶愈 34 non-null int64
2. 当日新增死亡 34 non-null int64
3. 当日新增重症 3 non-null float64
4. 当日现存确诊 34 non-null int64
5. today\_input 34 non-null int64 dtypes: float64(2), int64(12), object(1) memory usage: 4.2+ KB

----------------------------



2.2 实时数据探索性分析

– 43 –

从图中可知，吉林广东黑龙江三省有新增案例，但是好在人数不多，(注意别写错了)

全国现存确诊人数 **top10** 的地区

接下来我们查看一下全国现存确诊病例前十名的地区有哪些。

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

1

2

3

4

5

# 绘制条形图

store\_top10.sort\_values(ascending=True).plot.barh(fontsize=10)

plt.title(’全国现存确诊top10地区’,size=15) plt.show()

# 查看全国现存确诊人数top10的省市

store\_top10 = today\_province[’当日现存确诊’].sort\_values(ascending=False)[:10] store\_top10

---------------------------------

输出：名称

黑龙工 367

香港 261

台湾 142

内蒙古 68

上海 67

北京 59

山西 40

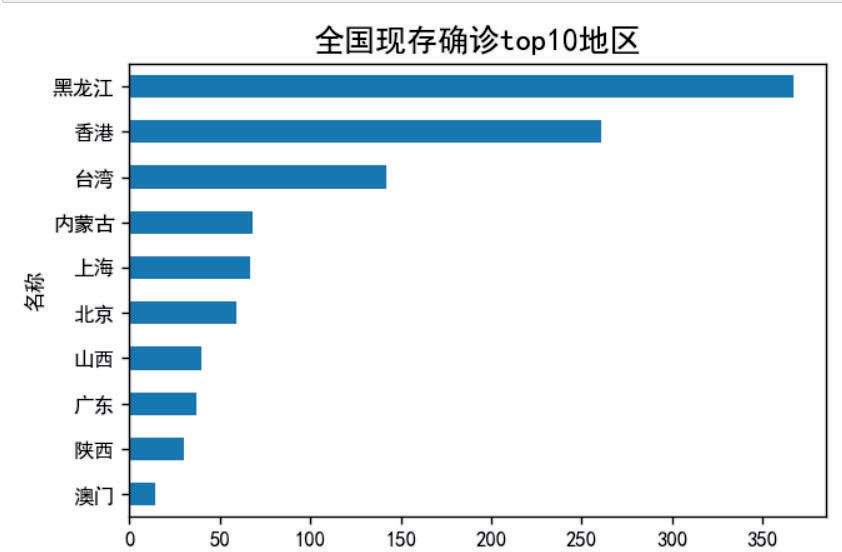
广东 37

陕西 30

澳门 14

Name: 当日现存确诊, dtype: int64

------------------------------------



2.2 实时数据探索性分析

– 44 –



2.3 历史数据探索性分析

– 45 –

**2.3** 历史数据探索性分析

**2.3.1** 全国历史数据探索性分析

全国历史数据是时间序列的数据类型，在数据清洗的时候需要对时间进行处理。本部分的最 后我们将绘制全国历史数据的折线图，并着重分析全国新增确诊人数的变化趋势。

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

alltime\_china.info()

---------------------------------

输出：<class ’pandas.core.frame.DataFrame’> RangeIndex: 98 entries, 0 to 97

Data columns (total 15 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

1. 日期 98 non-null object
2. 更新时同 0 non-null float64
3. 累计确诊 98 non-null int64
4. 累计疑似 98 non-null int64
5. 累计冶愈 98 non-null int64
6. 累计死亡 98 non-null int64
7. 累计重症 98 non-null int64

# 读取数据

alltime\_china = pd.read\_csv("alltime\_China\_2020\_04\_27.csv")

# 创建中文列名字典

name\_dict = {’date’:’日期’,’name’:’名称’,’id’:’编号’,’lastUpdateTime’:’更新时同’,

’today\_confirm’:’当日新增确诊’,’today\_suspect’:’当日新增疑似’,

’today\_heal’:’当日新增冶愈’,’today\_dead’:’当日新增死亡’,

’today\_severe’:’当日新增重症’,’today\_storeConfirm’:’当日现存确诊’,

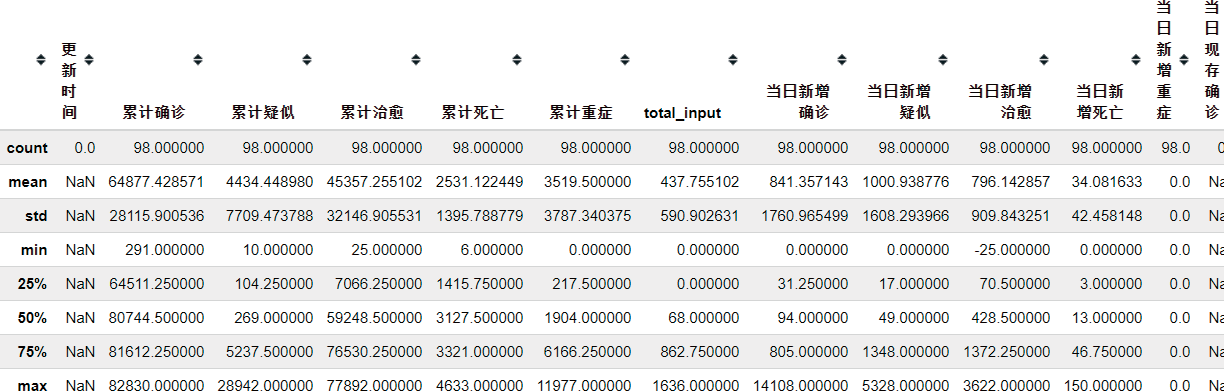
’total\_confirm’:’累计确诊’,’total\_suspect’:’累计疑似’,

’total\_heal’:’累计冶愈’,’total\_dead’:’累计死亡’,’total\_severe’:’累计重症’}

# 更改列名

alltime\_china.rename(columns=name\_dict,inplace=True)

alltime\_china.head()



2.3 历史数据探索性分析

– 46 –

16

17

18

19

20

21

22

23

24

25

26

1

2

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

# 缺失值处理

# 计算当日现存确诊人数

alltime\_china[’当日现存确诊’] = alltime\_china[’累计确诊’]-alltime\_china[’累计冶愈’]-

alltime\_china[’累计死亡’]

# 删除更新时同一列

alltime\_china.drop([’更新时同’,’当日新增重症’],axis=1,inplace=True) alltime\_china.info()

--------------------------------

输出:<class ’pandas.core.frame.DataFrame’> RangeIndex: 98 entries, 0 to 97

Data columns (total 13 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

1. 日期 98 non-null object
2. 累计确诊 98 non-null int64
3. 累计疑似 98 non-null int64

# 查看数据的统计信息

alltime\_china.describe()

1. total\_input 98 non-null int64
2. 当日新增确诊 98 non-null int64
3. 当日新增疑似 98 non-null int64
4. 当日新增冶愈 98 non-null int64
5. 当日新增死亡 98 non-null int64
6. 当日新增重症 98 non-null int64
7. 当日现存确诊 0 non-null float64
8. today\_input 54 non-null float64 dtypes: float64(3), int64(11), object(1) memory usage: 11.6+ KB

------------------------------------



2.3 历史数据探索性分析

– 47 –

20

21

22

23

24

25

26

27

28

29

30

31

32

与实时数据相比，历史数据的日期一列是非常重要的。我们使用 pd.to\_datetime() 将日期的数据类

型设为 datetime，并将其设置为行索引。

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

# 将日期改成datetime格武

alltime\_china[’日期’] = pd.to\_datetime(alltime\_china[’日期’])

# 设置日期为索引

alltime\_china.set\_index(’日期’,inplace=True) # 也可使用pd.read\_csv("./input/ alltime\_China\_2020\_03\_27.csv",parse\_dates=[’date’],index\_col=’date’)

alltime\_china.index

--------------------------------

输出:DatetimeIndex([’2020-01-20’, ’2020-01-21’, ’2020-01-22’, ’2020-01-23’,

’2020-01-24’, ’2020-01-25’, ’2020-01-26’, ’2020-01-27’,

’2020-01-28’, ’2020-01-29’, ’2020-01-30’, ’2020-01-31’,

’2020-02-01’, ’2020-02-02’, ’2020-02-03’, ’2020-02-04’,

’2020-02-05’, ’2020-02-06’, ’2020-02-07’, ’2020-02-08’,

’2020-02-09’, ’2020-02-10’, ’2020-02-11’, ’2020-02-12’,

’2020-02-13’, ’2020-02-14’, ’2020-02-15’, ’2020-02-16’,

’2020-02-17’, ’2020-02-18’, ’2020-02-19’, ’2020-02-20’,

’2020-02-21’, ’2020-02-22’, ’2020-02-23’, ’2020-02-24’,

’2020-02-25’, ’2020-02-26’, ’2020-02-27’, ’2020-02-28’,

’2020-02-29’, ’2020-03-01’, ’2020-03-02’, ’2020-03-03’,

’2020-03-04’, ’2020-03-05’, ’2020-03-06’, ’2020-03-07’,

’2020-03-08’, ’2020-03-09’, ’2020-03-10’, ’2020-03-11’,

’2020-03-12’, ’2020-03-13’, ’2020-03-14’, ’2020-03-15’,

’2020-03-16’, ’2020-03-17’, ’2020-03-18’, ’2020-03-19’,

’2020-03-20’, ’2020-03-21’, ’2020-03-22’, ’2020-03-23’,

’2020-03-24’, ’2020-03-25’, ’2020-03-26’, ’2020-03-27’,

’2020-03-28’, ’2020-03-29’, ’2020-03-30’, ’2020-03-31’,

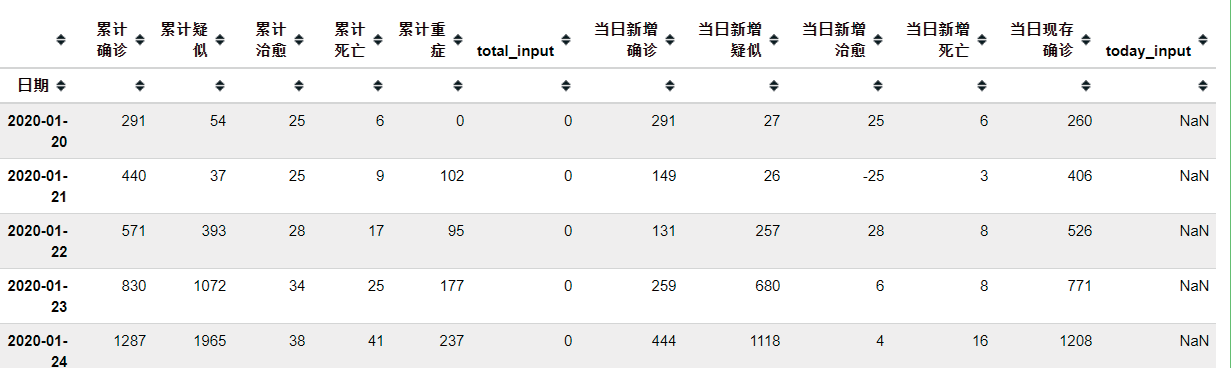
’2020-04-01’, ’2020-04-02’, ’2020-04-03’, ’2020-04-04’,

’2020-04-05’, ’2020-04-06’, ’2020-04-07’, ’2020-04-08’,

’2020-04-09’, ’2020-04-10’, ’2020-04-11’, ’2020-04-12’,

1. 累计冶愈 98 non-null int64
2. 累计死亡 98 non-null int64
3. 累计重症 98 non-null int64
4. total\_input 98 non-null int64
5. 当日新增确诊 98 non-null int64
6. 当日新增疑似 98 non-null int64
7. 当日新增冶愈 98 non-null int64
8. 当日新增死亡 98 non-null int64
9. 当日现存确诊 98 non-null int64
10. today\_input 54 non-null float64 dtypes: float64(1), int64(11), object(1) memory usage: 10.1+ KB

--------------------------------



2.3 历史数据探索性分析

– 48 –

31

32

33

34

35

36

1

2

数据清洗之后，我们将绘制折线图查看新冠肺炎数据的变化趋势：

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

# 时同序列数据绘制折线图

import matplotlib.pyplot as plt import matplotlib.dates as dates import matplotlib.ticker as ticker import datetime

fig, ax = plt.subplots(figsize=(16,9)) alltime\_china.plot(marker=’o’,ms=2,lw=1,ax=ax,use\_index=True)

ax.xaxis.set\_major\_locator(dates.MonthLocator()) #设置同距ax.xaxis.set\_major\_formatter(dates.DateFormatter(’%m/%d’)) #设置日期格武

fig.autofmt\_xdate() #自动调整日期倾斜

#图例位置调整plt.legend(bbox\_to\_anchor = [1,1])

plt.title(’全国新冠肺炎数据折线图’,size=15)

plt.ylabel(’人数’) plt.grid(axis=’y’) #plt.box(False) plt.show()

# 举例

alltime\_china.loc[’2020-01’]

’2020-04-13’, ’2020-04-14’, ’2020-04-15’, ’2020-04-16’,

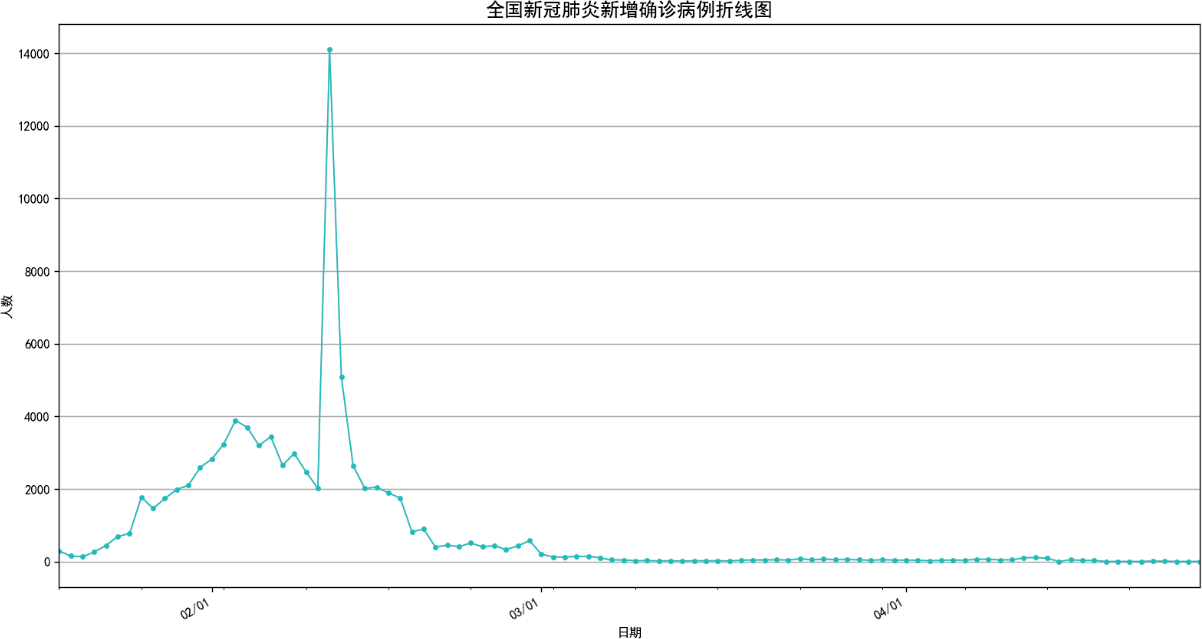
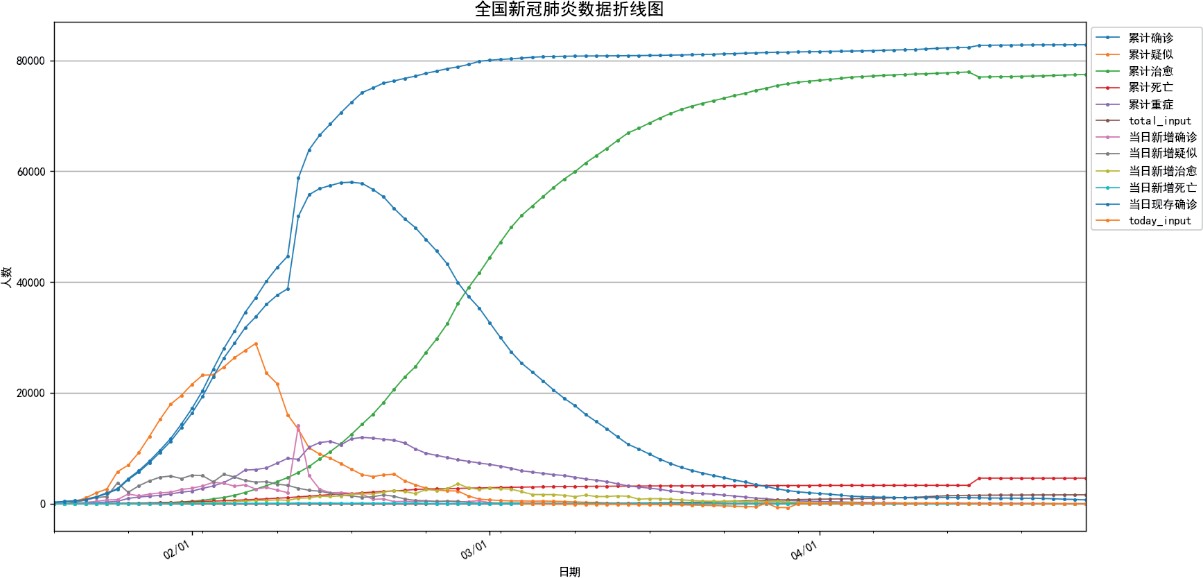
’2020-04-17’, ’2020-04-18’, ’2020-04-19’, ’2020-04-20’,

’2020-04-21’, ’2020-04-22’, ’2020-04-23’, ’2020-04-24’,

’2020-04-25’, ’2020-04-26’],

dtype=’datetime64[ns]’, name=’日期’, freq=None)

--------------------------------



2.3 历史数据探索性分析

– 49 –

1

2

3

4

5

6

7

8

9

10

11

12

13

可以看到 2 月 12 日新增病例大幅上升，这是什么原因呢？如果你非常关注疫情，应该知道这是

因为在当天国家卫健委调整了确诊的标准。此前，患者是否确诊，主要参考指标是核酸检测结果，

fig, ax = plt.subplots(figsize=(16,9))

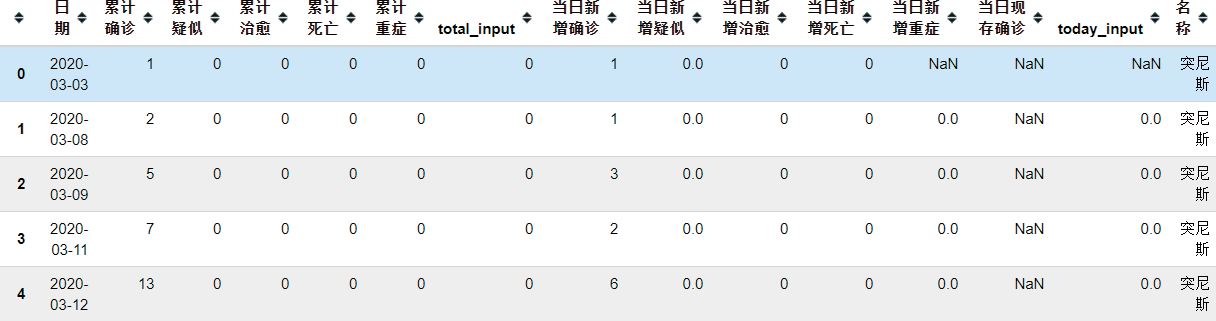
alltime\_china[’当日新增确诊’].plot(ax=ax, style=’-’,lw=1,color=’c’,marker=’o’,ms=3)

ax.xaxis.set\_major\_locator(dates.MonthLocator()) #设置同距ax.xaxis.set\_major\_formatter(dates.DateFormatter(’%m/%d’)) #设置日期格武

fig.autofmt\_xdate() #自动调整日期倾斜

plt.title(’全国新冠肺炎新增确诊病例折线图’,size=15)

plt.ylabel(’人数’) plt.grid(axis=’y’) plt.show()



2.3 历史数据探索性分析

– 50 –

还需要结合 CT 影像，咳嗽等症状来综合判断。随着疫情防控深入，临床数据的不断累积，有个新情况逐渐凸显出来——山千核酸检测的时间较慢，一些患者无法确诊收治，但是病症的临床表 现又高度疑似新冠肺炎。如果不改变认定标准，这部分患者就难以得到有效救助，对整个疫情防

控也造成负面影响。这次主要变化，就是将临床诊断纳入确诊范围。

**2.3.2** 世界各国历史数据探索性分析

最后让我们一起来分析一下世界各国历史数据，山千数据表中每个国家含有多条数据，我们 需要借助 GroupBy 技术对数据进行分组，并通过层次化索引操作选取多个国家的累计确诊和新增确诊数据，来查看各国疫情变化趋势。

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

1

2

3

4

5

6

7

8

9

10

11

# 查看数据基本信息

alltime\_world.info()

-----------------------------

输出：

<class ’pandas.core.frame.DataFrame’> RangeIndex: 6519 entries, 0 to 6518 Data columns (total 15 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 日期 6519 non-null object

# 读取数据

alltime\_world = pd.read\_csv("alltime\_world\_2020\_04\_27.csv")

# 创建中文列名字典

name\_dict = {’date’:’日期’,’name’:’名称’,’id’:’编号’,’lastUpdateTime’:’更新时同’,

’today\_confirm’:’当日新增确诊’,’today\_suspect’:’当日新增疑似’,

’today\_heal’:’当日新增冶愈’,’today\_dead’:’当日新增死亡’,

’today\_severe’:’当日新增重症’,’today\_storeConfirm’:’当日现存确诊’,

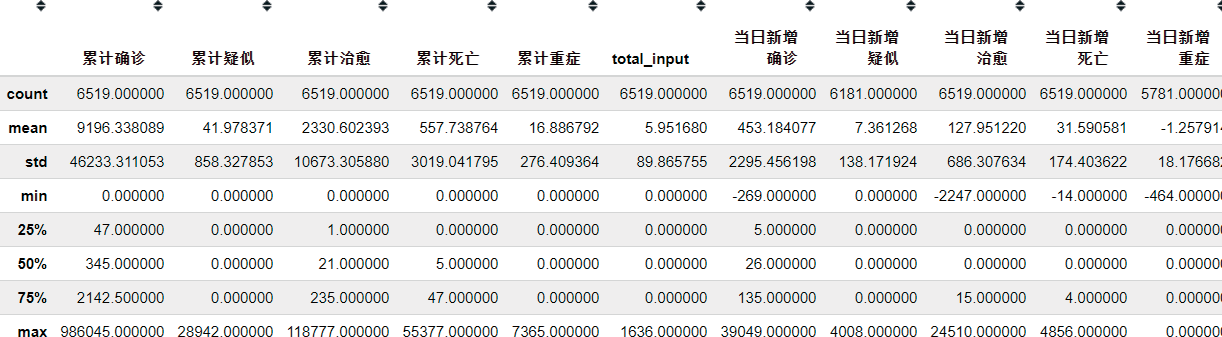
’total\_confirm’:’累计确诊’,’total\_suspect’:’累计疑似’,

’total\_heal’:’累计冶愈’,’total\_dead’:’累计死亡’,’total\_severe’:’累计重症’}

# 更改列名

alltime\_world.rename(columns=name\_dict,inplace=True)

alltime\_world.head()



2.3 历史数据探索性分析

– 51 –

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

1

数据预处理操作与前面的方法基本一致：

1

2

3

4

5

6

7

8

9

10

11

12

13

# 将日期一列数据类型变为datetime

alltime\_world[’日期’] = pd.to\_datetime(alltime\_world[’日期’])

# 计算当日现存确诊

alltime\_world[’当日现存确诊’] = alltime\_world[’累计确诊’]-alltime\_world[’累计冶愈’]-

alltime\_world[’累计死亡’] alltime\_world.info()

------------------------

输出:<class ’pandas.core.frame.DataFrame’> RangeIndex: 6519 entries, 0 to 6518

Data columns (total 15 columns):

# Column Non-Null Count Dtype

alltime\_world.describe()

1. 累计确诊 6519 non-null int64
2. 累计疑似 6519 non-null int64
3. 累计冶愈 6519 non-null int64
4. 累计死亡 6519 non-null int64
5. 累计重症 6519 non-null int64
6. total\_input 6519 non-null int64
7. 当日新增确诊 6519 non-null int64
8. 当日新增疑似 6181 non-null float64
9. 当日新增冶愈 6519 non-null int64
10. 当日新增死亡 6519 non-null int64
11. 当日新增重症 5781 non-null float64
12. 当日现存确诊 0 non-null float64
13. today\_input 6322 non-null float64
14. 名称 6519 non-null object dtypes: float64(4), int64(9), object(2) memory usage: 764.1+ KB

--------------------------------



2.3 历史数据探索性分析

– 52 –

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

数据表中总共有哪些国家呢？我们可以使用 unique() 查看数据中的唯一值：

1

2

3

4

5

6

7

8

9

10

11

同时我们还想了解随着时间的变化，每天有多少国家出现新冠肺炎疫情，value\_counts() 函数可帮

助我们查看每一天记录了多少数据。

1

2

3

4

5

6

7

8

9

10

11

# 统计每天有多少国家出现疫情

alltime\_world[’日期’].value\_counts().head(20)

------------------------ 2020-04-21 203

2020-04-07 162

2020-03-26 157

2020-04-04 153

2020-03-31 148

2020-03-30 147

2020-04-10 146

# 查看唯一值,可使用len()查看个数

alltime\_world[’名称’].unique()

------------------------

输出:array([’突尼斯’, ’塞尔维亚’, ’中国’, ’日本’, ’泰国’, ’新加坡’, ’韩国’, ’澳大利亚’,

’德国’, ’美国’,

...........................

’圣马力诺’, ’哈萨克斯坦’, ’安道尔’, ’牙买加’,

’格恩西岛’, ’罗马尼亚’, ’阿曼’, ’列支敦士登’, ’马达加斯加’], dtype= object)

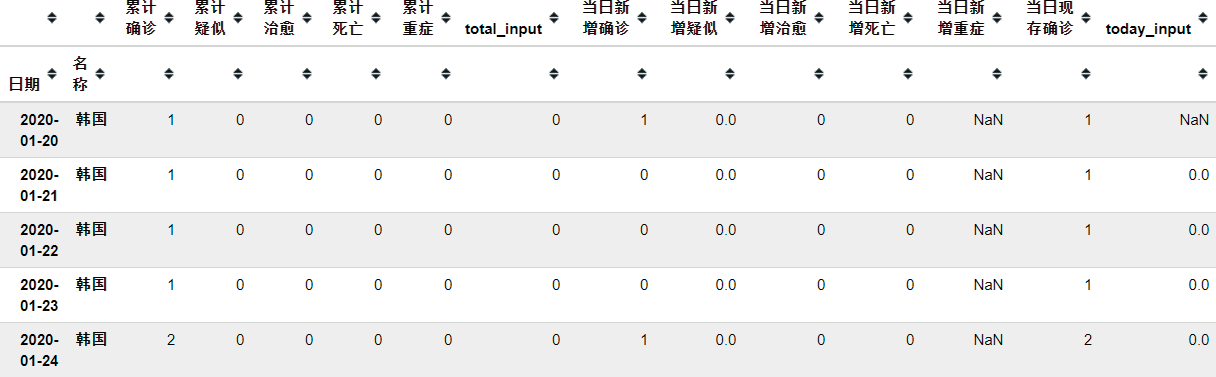
-----------------------------

--- ------ -------------- -----

1. 日期 6519 non-null datetime64[ns]
2. 累计确诊 6519 non-null int64
3. 累计疑似 6519 non-null int64
4. 累计冶愈 6519 non-null int64
5. 累计死亡 6519 non-null int64
6. 累计重症 6519 non-null int64
7. total\_input 6519 non-null int64
8. 当日新增确诊 6519 non-null int64
9. 当日新增疑似 6181 non-null float64
10. 当日新增冶愈 6519 non-null int64
11. 当日新增死亡 6519 non-null int64
12. 当日新增重症 5781 non-null float64
13. 当日现存确诊 6519 non-null int64
14. today\_input 6322 non-null float64
15. 名称 6519 non-null object

dtypes: datetime64[ns](1), float64(3), int64(10), object(1) memory usage: 764.1+ KB

-----------------------------



2.3 历史数据探索性分析

– 53 –

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

数据中显示，3 月 26 日这天出现疫情的国家数量已多达 157 个国家

1

2

3

4

5

选取多国数据

接下来，我们想提取中国、日本、韩国、美国、意大利、英国、西班牙和德国的数据，探索这八个国家的累计确诊和新增确诊病例的变化趋势。我们将使用 GroupBy 技术和层次化索引操作。

GroupBy 技术是对数据进行分组计算并将各组计算结果合并的一项技术，包括如下三个过程： 想要提取多个国家的数据，就需要把国家一列也设置为索引，我们可以使用 groupby() 函数

根据日期和名称两列进行分组，将数据转为层次化索引。

1

2

3

4

想要提取部分数据，同样可以使用.loc 方法，需先通过.loc(axis= ) 指定对行索引还是对列索

引进行操作。比如，我们想提取中国、韩国、美国、意大利、英国、西班牙、德国的数据：

# groupby创建层次化索引

data = alltime\_world.groupby([’日期’,’名称’]).mean()

data.head()

# 设置日期索引

alltime\_world.set\_index(’日期’,inplace=True)

# 3月31日数据统计不完全，我们将其删除alltime\_world = alltime\_world.loc[:’2020-03-31’]

2020-04-22 145

2020-04-16 144

2020-04-11 140

2020-04-09 136

2020-04-15 136

2020-04-06 133

2020-04-14 132

2020-03-27 130

2020-04-08 130

2020-03-22 129

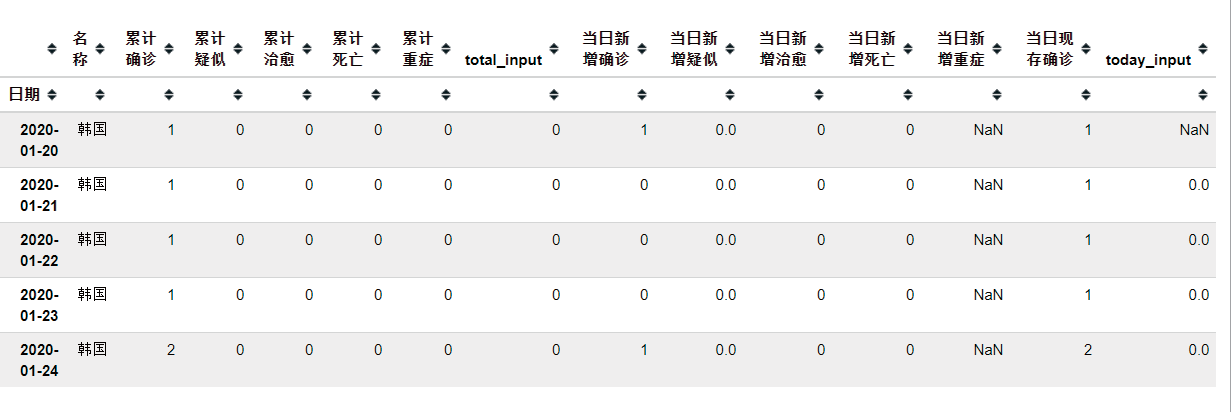
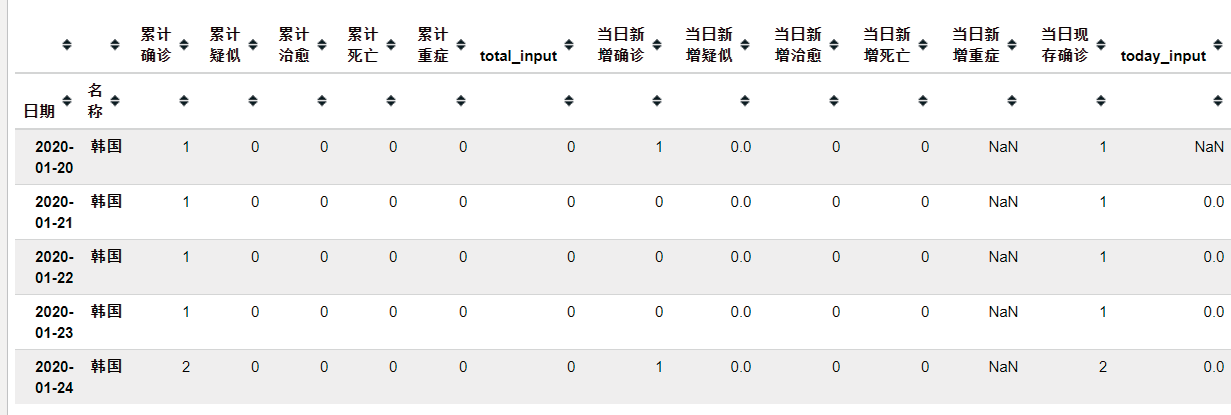
2020-04-24 125

2020-03-25 125

2020-04-23 124

Name: 日期, dtype: int64

-----------------------------



2.3 历史数据探索性分析

– 54 –

1

2

3

4

此时，多级索引已设置成功。如果我们想将其还原则可使用 reset\_index() 函数。

1

2

3

4

绘制多个国家的累计确诊人数折线图

1

2

3

4

5

6

7

8

9

10

11

# 绘制多个国家的累计确诊人数折线图

fig, ax = plt.subplots(figsize=(8,4))

data\_part[’2020-02’:].groupby(’名称’)[’累计确诊’].plot(legend=True,marker=’o’,ms=3,lw=1)

ax.xaxis.set\_major\_locator(dates.MonthLocator()) #设置同距ax.xaxis.set\_major\_formatter(dates.DateFormatter(’%b’)) #设置日期格武

fig.autofmt\_xdate() #自动调整日期倾斜

plt.title(’各国新冠肺炎累计确诊病例折线图’,size=15)

# 将层级索引还原

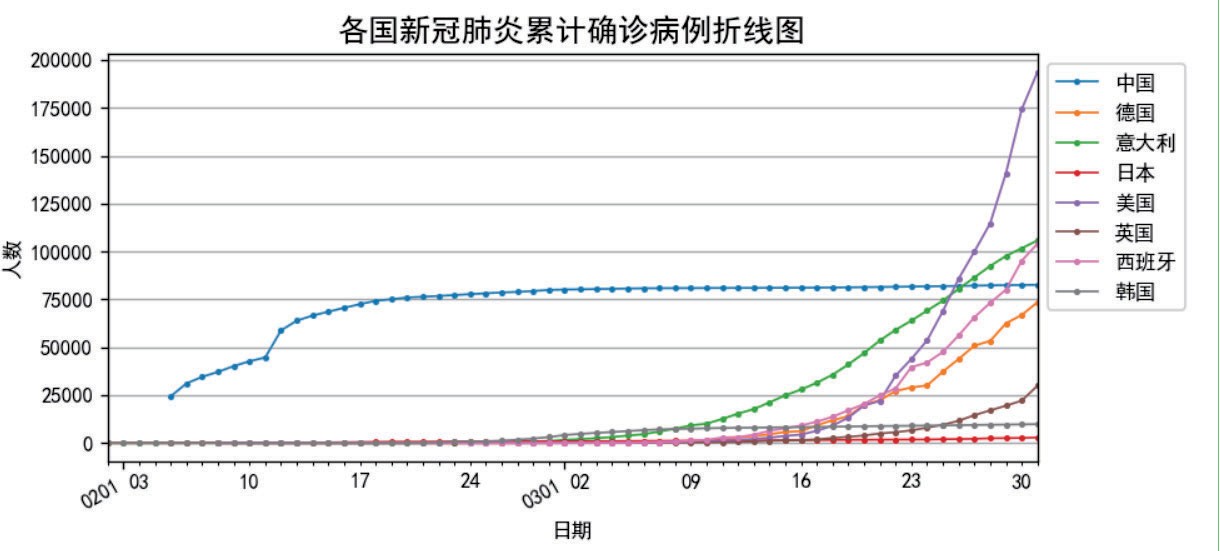
data\_part.reset\_index(’名称’,inplace=True)

data\_part.head()

# 提取部分数据

data\_part = data.loc(axis=0)[:,[’中国’,’日本’,’韩国’,’美国’,’意大利’,’英国’,’西班牙’,’德国’]]

data\_part.head()



2.3 历史数据探索性分析

– 55 –

12

13

14

15

16

山图可知，中韩两国§前累计确诊人数已出现拐点，疫情趋千平稳，而美国在最近几天累计确诊 人数呈现爆发性的增长，已位居世界第一。此外，意大利、西班牙两国增长速度也紧跟其后。

最后，让我们一起来观察一下各国新增确诊人数的变化情况，这里我们只查看 3 月份的最新

数据

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

# 绘制各国新增确诊人数折线图

fig, ax = plt.subplots(figsize=(8,4))

data\_part[’2020-03’:’2020-03-29’].groupby(’名称’)[’当日新增确诊’].plot(legend=True, marker=’o’,ms=3,lw=1)

ax.xaxis.set\_major\_locator(dates.MonthLocator()) #设置同距ax.xaxis.set\_major\_formatter(dates.DateFormatter(’%b’)) #设置日期格武

fig.autofmt\_xdate() #自动调整日期倾斜

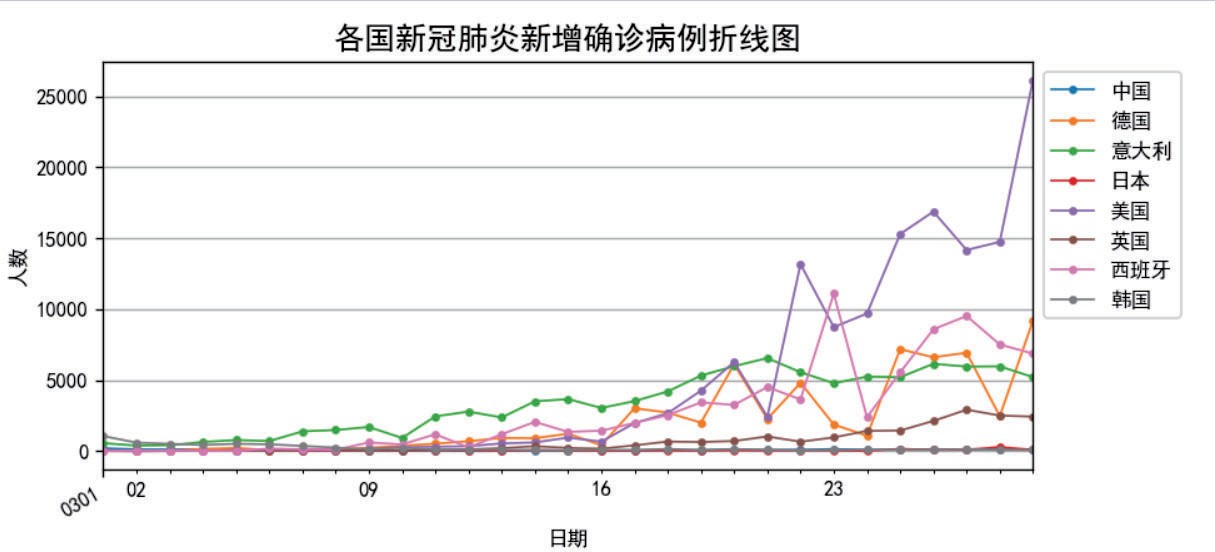
plt.title(’各国新冠肺炎新增确诊病例折线图’,size=15)

plt.ylabel(’人数’) plt.grid(axis=’y’) plt.box(False)

plt.legend(bbox\_to\_anchor = [1,1]) plt.show()

plt.ylabel(’人数’) plt.grid(axis=’y’) plt.box(False)

plt.legend(bbox\_to\_anchor = [1,1]) plt.show()



2.3 历史数据探索性分析

– 56 –

各国新增确诊人数波动较大，但总体趋势呈上升状态。在 3 月下旬，美国和西班牙首次单日新增确诊人数破万，而最新数据显示美国单日新增已突破 25000 例。

在前面两张图里，我们发现日本山千数据较小，很难观察疫情的变化趋势。这里我们单独选

取了日本新冠肺炎的累计确诊和当日新增确诊两列数据进行绘制：

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

japan = alltime\_world[alltime\_world[’名称’]==’日本’] fig, ax = plt.subplots(figsize=(8,4))

japan[’累计确诊’].plot(ax=ax, fontsize=10, style=’-’,lw=1,color=’c’,marker=’o’,ms=3, legend=True)

ax.set\_ylabel(’人数’, fontsize=10)

ax1 = ax.twinx()

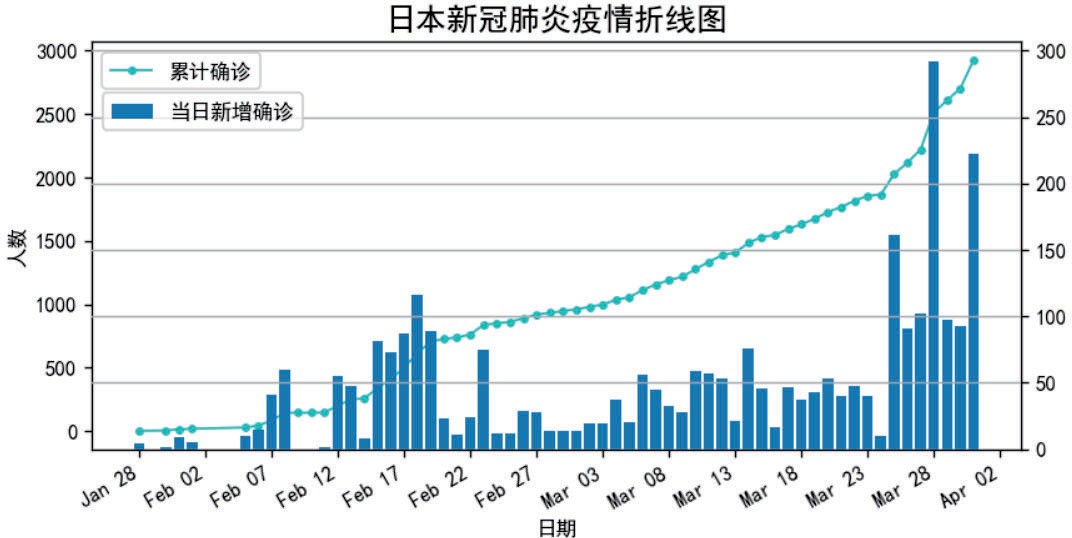
ax1.bar(japan.index, japan[’ 当 日 新 增 确 诊 ’]) ax1.xaxis.set\_major\_locator(dates.DayLocator(interval = 5)) ax1.xaxis.set\_major\_formatter(dates.DateFormatter(’%b␣%d’))

ax1.legend([’当日新增确诊’],loc=’upper␣left’,bbox\_to\_anchor=(0.001, 0.9)) plt.grid(axis=’y’)

plt.box(False)

plt.title(’日本新冠肺炎疫情折线图’,size=15)

plt.show()



2.4 总结

– 57 –

我们发现日本前期新增确诊人数的变化几乎没有太大增幅，但在 3 月 25 日起，日本的新增

确诊人数明显增大，同时累计确诊折线斜率也随之增加。

**2.4** 总结

本案例使用基千网易实时疫情播报平台爬取的数据，进行新冠肺炎疫情数据的探索性分析。其中数据预处理主要包括特征列重命名、缺失值处理、查看重复值、数据类型转换等操作。此外， 我们还使用了Pandas 进行数据可视化，通过图表的绘制探索数据的内涵。同时，我们介绍了时间序列数据的处理方法、如何使用 Groupby 技术进行数据分组，以及层次化索引的操作方法。