《Python科学计算与数据处理》结课报告

用Python爬虫实现网络小说和桌面壁纸的批量下载

姓 名：承子杰

学 号：202228000243001

研究所： 数学与系统科学研究院

1. **报告简介**

本篇报告主要讲述使用Python爬虫进行网络信息的抓取。作者首先将介绍实现Python爬虫所需要的一些基础模块，例如requests模块、re模块、Etree模块等、正则表达式的初步知识、多线程下载与线程池的使用的一些基础内容。之后作者将通过一个实例---爬取英文网络小说计算语言信息熵---来展示具体实现流程，从而进一步地加深读者对Python爬虫的了解与使用，方便大家在今后的应用过程中编写自己所需要爬虫程序。

1. **基础知识介绍**
2. **Requests模块简介**
3. Requests模块背景

Requests模块是使用Apache2 Licensed许可证的基于Python开发的HTTP库，其在Python内置模块（urllib3）的基础上进行了高度的封装，从而使得用户在进行网络请求时，变得方便了许多。Requests支持HTTP连接保持和连接池，支持使用cookie保持会话，支持文件上传，支持自动确定响应内容的编码，支持国际化的 URL 和 POST 数据自动编码等多种功能，基本上能完全满足当前网络的各种需求。

1. Requests模块的安装

Requests模块是Python的第三方库，我们可以使用pip install命令来进行安装，具体代码如下：

pip install requests

1. Requests模块提供的请求方法。

Requests提供了如下五种基本的请求方法：

* GET： 请求指定的页面信息，并返回实体主体。
* HEAD： 只请求页面的首部。
* POST： 请求服务器接受所指定的文档作为对所标识的URI的新的从属实体。
* PUT： 从客户端向服务器传送的数据取代指定的文档的内容。
* DELETE： 请求服务器删除指定的页面。

对于一般的网络爬虫而言，GET请求和POST请求使用较多，下面我们对这两个方法进行详细的介绍。

1. GET请求

**基本的GET请求**

使用的基本语法如下：

import requests #导入模块

#使用get方法请求url，会得到一个response对象

response = requests.get(url)

**带请求头headers的GET请求**

有些网站在访问时必须带有浏览器等信息，我们可以设置请求头伪装成浏览器浏览。例如，在爬取豆瓣书评网站时，若不设置请求头，则会访问失败。

import requests

url = "https://book.douban.com"

# 不设置请求头

resp = requests.get(url)

# 打印请求结果代码

print(resp.status\_code)

Result：

418 # 请求成功打印数字200

下面我们设置参数headers，headers需要写成一个字典的形式，基本格式如下：

headers = {

'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_11\_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.116 Safari/537.36'

}

其中字典的键是固定的，为 'User-Agent'，其键值可以在浏览的网页中打开页面源码（作者使用的Google Chrome浏览器可以网页右击选择检查打开），选择网络菜单，打开对应网页的标头选项中可以查找到相应信息。

图形用户界面, 文本, 应用程序

描述已自动生成

作者在此处显示的值如下。

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_15\_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/105.0.0.0 Safari/537.36

此处显示了基本的浏览器信息与访问设备信息，现在我们加入请求头，重新测试爬取结果。

import requests

url = "https://book.douban.com"

# 设置请求头

headers = {'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_15\_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/105.0.0.0 Safari/537.36'}

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

# 打印请求结果代码

print(resp.status\_code)

Result:

200 # 请求成功

1. POST请求

post() 方法类似于GET()方法，可以发送 POST 请求到指定 url，其一般的语法格式如下：

requests.post(url, data={key: value}, json={key: value}, args)

其中：

* **url**：请求 url。
* **data**参数：为要发送到指定 url 的字典、元组列表、字节或文件对象。
* **json**参数：为要发送到指定 url 的 JSON 对象。
* **args**为其他参数，比如 cookies、headers、verify等。

1. Response响应属性

POST()方法与GET()的返回值都是都是一个response对象，比较常用的属性如下：

|  |  |
| --- | --- |
| 属性或方法 | 说明 |
| close() | 关闭与服务器的连接 |
| encoding | 解码 r.text 的编码方式 |
| headers | 返回响应头，字典格式 |
| ok | 检查 "status\_code" 的值，如果小于400，则返回 True，如果不小于 400，则返回 False |
| reason | 响应状态的描述，比如 "Not Found" 或 "OK" |
| request | 返回请求此响应的请求对象 |
| status\_code | 返回 http 的状态码，比如 404 和 200（200 是 OK，404 是 Not Found） |
| text | 返回响应的内容，unicode 类型数据 |
| url | 返回响应的 URL |

1. **正则表达式基础知识与Re模块简介**
2. 正则表达式的定义

正则表达式是对字符串（包括普通字符（例如，a到z之间的字母）和特殊字符（称为“元字符”））操作的一种逻辑公式，就是用事先定义好的一些特定字符、及这些特定字符的组合，组成一个“规则字符串”，这个“规则字符串”用来表达对字符串的一种过滤逻辑。正则表达式是一种文本模式，该模式描述在搜索文本时要匹配的一个或多个字符串。

1. 正则表达式的基本语法

|  |  |
| --- | --- |
| 表达式 | 描述 |
| X,a,9,< | 普通字符的完全匹配 |
| . | 匹配任何单个字符，除了换行符’\n’ |
| \w | 匹配“单词”字符：字母或数字或下划线[a-zA-Z0-9\_] |
| \b | 字词与非字词之间的界限 |
| \s | 匹配单个空格字符-空格、换行符，返回，制表符 |
| \S | 匹配任何非空格字符 |
| \t,\n,\r | 匹配制表符、换行符、退格符 |
| \d | 匹配十进制数[0-9] |
| ^ | 匹配字符串的开头 |
| $ | 匹配字符串的末尾 |
| \ | 抑制字符的特殊性，即转义字符。 |

1. 正则表达式的特殊语法

|  |  |
| --- | --- |
| 表达式 | 描述 |
| <.\*> | 贪婪重复 |
| <.\*?> | 非贪婪重复 |
| [a-z] | 匹配任何小写字母 |
| [A-Z] | 匹配任何大写字母 |
| [0-9] | 匹配任何数字 |
| [\u4e00-\u9fa5] | 匹配所有汉字 |

1. Re模块背景

Re模块是Python的标准库，它使 Python 语言拥有全部的正则表达式功能。该模块提供了一些关于正则表达式操作的函数，这些函数使用一个模式字符串做为它们的第一个参数，下面我们将对常用的几个函数进行详细的介绍。

1. **match函数**

**match函数**从字符串的最开始与pattern进行匹配，下面是该函数的基本语法 ：

**re.match(pattern, string, flags = 0)**

**其中：**

* pattern - 这是要匹配的正则表达式。
* string - 这是字符串，它将被搜索用于匹配字符串开头的模式。
* flags - 可以使用按位OR(|)指定不同的标志。这些是修饰符，如下表所列。

当re.match函数匹配成功时，会返回匹配对象，失败时则返回None。使用match(num)或groups()函数可以匹配对象来获取匹配的表达式。

下面是一个使用match函数的例子。

#未从初始位置匹配，会返回None

import re

line = 'i can speak good english'

matchObj = re.match(r'(i)\s(\w\*)\s(\w\*).\*',line)

if matchObj:

print('matchObj.group() :',matchObj.group())

print('matchObj.group(1) :',matchObj.group(1))

print('matchObj.group(2) :',matchObj.group(2))

print('matchObj.group(3) :',matchObj.group(3))

else:

print('no match!')

Result:

matchObj.group() : i can speak good english

matchObj.group(1) : i

matchObj.group(2) : can

matchObj.group(3) : speak

1. **search 函数**

search函数与match()工作的方式类似，但是search()不是从最开始匹配的，而是从任意位置查找第一次匹配的内容。下面是这个函数的语法 ：

**re.search(pattern, string, flags = 0)**

**其中：**

* pattern - 这是要匹配的正则表达式。
* string - 这是字符串，它将被搜索用于匹配字符串开头的模式。
* flags - 可以使用按位OR(|)指定不同的标志。这些是修饰符，如下表所列。

当re.research函数匹配成功时，会返回匹配对象，失败时则返回None。使用match(num)或groups()函数可以匹配对象来获取匹配的表达式。

下面是一个使用search函数的例子。

**i**mport re

line = 'i can speak good english'

searchObj = re.search(r'\s(\w\*)\s(\w\*).\*',line)

if searchObj:

print('searchObj.group() :',searchObj.group())

print('searchObj.group(1) :',searchObj.group(1))

print('searchObj.group(2) :',searchObj.group(2))

else:

print('no match!')

Result:

searchObj.group() : can speak good english

searchObj.group(1) : can

searchObj.group(2) : speak

1. sub函数

re模块中使用频率最多的函数应该是sub函数。此方法使用repl替换所有出现在模式里的字符串，并且替换所有次的出现，除非提供了参数max。此方法返回修改的字符串，一般的语法格式如下：

re.sub(pattern, repl, string, max=0)

1. compile函数

**compile 函数用于编译正则表达式，生成一个 Pattern 对象，**它的一般使用形式如下：

re.compile(pattern[, flag])

参数含义与上面函数类似，这里不再做过多解释。

1. findall函数

findall函数返回包含所有匹配项的列表。返回string中所有与pattern相匹配的全部字串，返回形式为数组。它的一般使用形式如下：

re.findall(pattern, string, flags=0)

findall函数一般与compile函数连用，下面我们举一个例子。

**import re**

**line = 'a0bb3c45djs8'**

**obj = re.compile(r"[0-9]")**

**result = obj.findall(line)**

**print(result)**

**Result:**

**['0', '3', '4', '5', '8']**

1. **Etree模块简介**
2. LXML模块

Python 的lxml模块是XML和HTML的解析器，其主要功能是解析和提取XML和HTML中的数据。lxml和正则一样，也是用C语言实现的，是一款高性能的解析器。该模块也可以利用XPath语法，来定位特定的元素及节点信息。

* HTML是超文本标记语言，主要用于显示数据，他的焦点是数据的外观。
* XML是可扩展标记语言，主要用于传输和存储数据，他的焦点是数据的内容。

1. Etree 模块

Etree模块是LXML库下的一个子模块，本报告中主要使用了该模块下的HTML函数和xpath函数，下面作者将对这两个函数做一个简单的介绍。

1. LXML模块的安装

LXML模块是Python的第三方库，我们可以使用pip install命令来进行安装，具体代码如下：

pip install xlml

1. Etree.HTML函数

etree.HTML()函数可以用来解析字符串格式的HTML文档对象，将传进去的字符串转变成\_Element对象。作为\_Element对象，可以方便的使xpath()等方法进行字符串匹配等操作。

1. Etree.xpath函数

etree.xpath函数可以根据提供的xpath表达式，匹配出所需要的html内容，其基本语法格式如下：

result = et.xpath('xpath表达式')

下面我们对xpath表达式进行一些简单介绍。

1. Xpath表达式介绍

|  |  |
| --- | --- |
| 表达式 | 描述 |
| '/' | 表示从根节点开始定位，表示一个层级 |
| '//' | 表示多个层级，开头时表示从任意位置开始定位 |
| './ ' | 表示从当前标签开始定位 |
| '标签名[@属性名="属性值"]' | 属性定位 |
| '标签名[索引] ' | 索引定位，索引从1开始 |
| 'xpath表达式1 | xpath表达式2' | 多重定位 |
| 标签名/text() | 获取直系文本 |
| 标签名/@属性名 | 获取非直系文本 |
| 标签名/@属性名 | 获取属性值 |

1. **多线程下载与线程池**
2. 线程与进程

下面我们对线程和进程做一些基本的介绍。

* 进程：计算机程序只是存储在磁盘中的可执行二进制(或其他类型)的文件。只有把他们加载到内存中并被操作系统调用，才具有其生命周期。进程则是一个执行中的程序。每个进程都拥有自己的地址空间，内存，数据栈以及其他用于跟踪执行的辅助数据。进程也可以通过派生新的进程来执行其他任务。由于每个进程有自己的数据，所以只能采用进程间通信(IPC)的方式来共享信息。
* 线程：又称轻量级进程。一个进程开始便会创建一个线程，称为主线程。一个进程可以创建多个线程，多线程即是同一进程下的不同执行路径，同一进程下的线程共享该进程的数据区。线程以并发的方式执行，线程执行时可以被中断和挂起。(在多核cpu中，多线程才可能并行执行)

1. Thread与Threading模块

Python提供多线程模块thread及threading，以及队列Queue，其中thread相对比较基础，不容易控制，官方建议使用threading模块，thread模块在python3版本中被重命名为\_thread。由于thread与threading模块需要创建线程，使用完成后需要手动回收线程，相对较为麻烦。于此同时，对于不同大小的问题，我们有时并不知道应该安排多少线程较为合适，而且手动创建多线程需要提供相关锁的程序作为线程守护，否则容易出现问题。因此综上所示，我们选择使用线程池来代替手动创建多线程。

1. 线程池的优点

* 创建合理的线程数量，重用存在的线程，减少线程创建销毁带来的开销。
* 可有效的控制最大并发线程数，提高系统资源的使用率，同时避免过多资源竞争，避免堵塞。提供定时执行、定期执行、单线程、并发数控制等功能。

1. ThreadPoolExecutor模块

ThreadPoolExecutor位于Python3标准库的并发包（concurrent.futures）下，我们可以使用该模块非常简单的创建和使用线程池。该模块具有如下几个特点；

* 主线程可以获取某一个线程的状态，以及返回值。
* 线程同步。
* 让多线程和多进程的编码接口一致。
* 简单粗暴。

1. 多线程的基本使用方法

在这里，我们仅提供一个例子，简单介绍如何使用多线程，对于多线程的详细介绍可以参考相关文档。

**# 创建一个包含2条线程的线程池**

**with ThreadPoolExecutor(max\_workers=2) as pool:**

**# 另一种方式**

**# pool = ThreadPoolExecutor(max\_workers=2)**

**# 向线程池提交一个task, 20会作为action()函数的参数**

**task = pool.submit(action, 20)**

**# 向线程池再提交一个task, 30会作为action()函数的参数**

**task2 = pool.submit(action, 30)**

**# 判断task代表的任务是否结束**

**print(task.done())**

**time.sleep(3)**

**# 判断task2代表的任务是否结束**

**print(task2.done())**

**# 查看task代表的任务返回的结果**

**print(task.result())**

**# 查看task2代表的任务返回的结果**

**print(task2.result())**

**# 关闭线程池**

**pool.shutdown()**

1. **实际案例分析**

**爬取英文网络小说并计算英语语言的信息熵**

1. 问题分析

作者打算爬取英文小说网站：<https://engnovel.com>上当期最流行的一本英语小说。该问题主要分为如下几个步骤：

* 爬取本网站最流行小说排行榜上第一位小说的详情页url。
* 在获得url后，向该url发送请求，获得该小说详情页中目录下第一个章节的url和最后一个章节的url。因为该小说详情页的第一页只提供了部分的章节url，其余章节需要翻页获得url。
* 向第一章的url发送请求，注意到返回到HTML文件中除了小说该章节的内容之外，还有下一章的url。
* 我们再次向下一章的章节发送请求，重复上述操作，模拟翻页的过程，直到最后一章。我们在之前已经获得了最后一章的url，故可以判断是否结束循环。
* 对于每一章的HTML文件，做一定的字符串匹配，获得小说内容，并写入文件
* 小说内容的清洗与熵的计算。

1. 实际操作

下面我们就按上述内容一步一步操作。首先是获得最流行小说详情页的url。我们需要编写一个名为get\_detail\_href的函数，输入值为最流行小说排行榜的url，输出是最流行小说详情页的url，具体代码如下：

**def get\_detail\_href(url):**

**"""**

**该函数负责获取到详情页的值**

**"""**

**print("开始分析主界面")**

**# 向指定url发送请求**

**resp = requests.get(url)**

**# 定义编码形式**

**resp.encoding = "utf-8"**

**# 解析获得的response请求结果，获得html**

**et = etree.HTML(resp.text)**

**# 使用xpath表达式匹配到指定信息**

**hrefs = et.xpath("//div[@class = 'each\_truyen']/a/@href")**

**print("分析完成")**

**# 返回url链接**

**return hrefs[0]**

**现在我们已经获得了当期最流行小说排行榜第一的小说(PEERLESS MARTIAL GOD)详情页的URL。我们现在需要编写一个函数get\_page\_srcs来获得详情页目录中小说第一章的URL和最后一章的url，具体的代码如下**

**def get\_page\_srcs(url):**

**print("开始抓取子页面")**

**# 请求小说详情页**

**resp = requests.get(url)**

**# 定义编码形式**

**resp.encoding = "utf-8"**

**# 解析获得的response请求结果，获得html**

**et = etree.HTML(resp.text)**

**# 使用xpath表达式匹配到指定信息**

**hrefs = et.xpath("//ul[@class='list-chapter']/li/a/@href")**

**# 从多个目录列表中筛选出第一章和最后一章的url**

**hrefs = [hrefs[0], hrefs[6]]**

**print("抓取成功")**

**# 返回URL列表**

**return hrefs**

现在我们已经获得了第一章和最后一章（第2500章）的url，下面我们编写一个函数download\_text，可以打开对应章节的url，爬取章节页的内容，并且可以模拟点击下一章的操作，从而可以爬取整本小说（建议在连接有线网或网络稳定的情况下进行操作，不然容易失败，或者爬取时间较长。此处由于存在顺序结构，无法使用线程池加快速度，可以考虑在小说目录页模拟翻页操作，将所有的章节url获得，再使用线程池完成任务）。函数的输入项src1是起始页的url，src2是最后页的url，该函数没有输出项，爬去内容直接写入了text文件。函数的具体代码如下：

**# src1：下载小说起始页url, src2：终止页url**

**def download\_text(src1, src2):**

**# 下载章节计数器**

**t = 0**

**print("开始下载章节")**

**while True:**

**# 请求对于章节的url**

**resp = requests.get(src1)**

**# 定义返回response对象编码形式**

**resp.encoding = "utf-8"**

**# 定义正则表达式**

**obj = re.compile(r'<p>(.\*?)</p>', re.S)**

**# 使用正则表达式匹配返回的html页面，得到章节小说内容**

**content1 = obj.findall(resp.text)**

**# 将返回的列表转化为字符串**

**content = "".join(content1)**

**# 做初步的内容清洗，将&#8230转译为"..."**

**content = content.replace("&#8230", "...")**

**# 编写写入函数，写入程序文件下的book.text文件，写入方式是追加**

**with open(f"book", mode="a") as f:**

**f.write(content)**

**# 判断是否是最后一页，或者章节下载次数超过5000，提高程序的鲁棒性**

**if src1 == src2 or t > 5000:**

**break**

**# 用xpath的方法匹配出下一章节的url**

**et = etree.HTML(resp.text)**

**src1 = et.xpath('//a[@id = "next\_chap" ]/@href')[0]**

**# 计数器加一**

**t = t + 1**

**print(t)**

**print("下载本书完毕")**

必要的功能性函数写完之后，我们需要写一个main函数来统筹一下各个流程，具体的代码如下：

**def main():**

**url = "https://engnovel.com/most-popular-novels"**

**# 1. 抓取到首页中详情页到href**

**hrefs = get\_detail\_href(url)**

**# 2. 获得第一章和最后一章的url**

**page\_src\_list = get\_page\_srcs(hrefs)**

**# 3. 下载小说**

**download\_text(page\_src\_list[1],page\_src\_list[0])**

在我们获取到这本小说的主要内容后，我们来计算一下这本书中文字的信息熵和各个字母的信息熵。要完成这个任务，我们需要进行如下几个步骤的操作：

1. 读取小说文本
2. 清洗文本，将获得的小说内容除英文字母外全部清洗掉。
3. 统计各个单词的词频
4. 计算各个字母的信息熵和英文整体的熵。

对于如上四个问题，我们分别编写了如下四个函数来完成相应的任务。

1. **读取文件内容函数read\_text**

**# 读取文件内容**

**def read\_text():**

**# 打开小说文件，模式是read**

**with open(f"book", mode="r") as f:**

**# 将读取的文件赋值给text**

**text = f.read()**

**# 关闭文件**

**f.close()**

**# 返回读取的内容**

**return text**

1. **文本清洗函数text\_clean**

**# 文本清洗**

**def text\_clean(text):**

**# 使用正则表达式去除除字母以外的所有符号**

**obj = re.compile("[^A-Za-z]")**

**text = obj.sub("", text)**

**# 将所有字母全部转化为小写，方便统计词频**

**text = text.lower()**

**# 返回处理后的文本**

**return text**

1. **统计词频函数count\_alpha**

**# 统计词频**

**def count\_alpha(text, num):**

**# 统计[0,num-1]区间段的文字**

**text = text[0:num - 1]**

**# 使用Counter函数统计**

**count = Counter(text)**

**# 将结果赋值给result**

**result = count.most\_common()**

**return result**

1. **计算熵的函数entropy\_cal**

**# 计算熵的函数**

**def entropy\_cal(alpha\_num):**

**# 读取字母列表**

**alpha\_num\_list = np.array(alpha\_num)**

**# 读取各个字母的词频**

**num = np.array(alpha\_num\_list[:, 1], dtype="int64")**

**alpha = np.array(alpha\_num\_list[:, 0])**

**# 计算总词频**

**total\_num = np.sum(num)**

**# 计算各个词的频率**

**num\_prob = np.divide(num, total\_num)**

**# 计算频率对数**

**log2\_num\_prob = np.log2(num\_prob)**

**# 各个字母的熵**

**en\_alpha = -np.multiply(num\_prob, log2\_num\_prob)**

**# 计算英语语言熵**

**en\_total = -np.dot(num\_prob, log2\_num\_prob)**

**# 拼接字典，返回结果**

**result = dict(np.c\_[alpha, en\_alpha])**

**result['total'] = en\_total**

**return result**

最后我们编写一个main函数来统筹各个函数，完成5000个单词逐步加入样本，计算样本下的熵，观测其变化趋势，main函数的具体代码如下：

**def main():**

**# 读取文本**

**text = read\_text()**

**# 清洗文本**

**text = text\_clean(text)**

**# 以5000个字母为统计单位，依次加入计算熵，观察趋势**

**length = len(text)**

**# 计算加入次数**

**num = length // 5000**

**alpha\_num = count\_alpha(text, length)**

**entropy\_cal(alpha\_num)**

**# 绘制各个英语字母的熵的趋势图**

**for alpha in [chr(i) for i in range(97, 123)]:**

**total = []**

**y = []**

**for i in range(num):**

**if i == num - 1:**

**alpha\_num = count\_alpha(text, length)**

**total.append(round(float(entropy\_cal(alpha\_num)[f"{alpha}"]), 3))**

**y.append(length)**

**else:**

**alpha\_num = count\_alpha(text, (i + 1) \* 5000)**

**total.append(round(float(entropy\_cal(alpha\_num)[f'{alpha}']), 3))**

**y.append((i + 1) \* 5000)**

**plt.plot(y, total, 'r-')**

**plt.xlabel("Number of Words")**

**plt.ylabel("Entropy")**

**plt.title(f'Word "{alpha}" Entropy Tendency')**

**plt.savefig(f"Entropy\_EN/{alpha}.jpg")**

**# 绘制英语语言熵的趋势图**

**total = []**

**y = []**

**for i in range(num):**

**if i == num - 1:**

**alpha\_num = count\_alpha(text, length)**

**total.append(round(float(entropy\_cal(alpha\_num)["total"]), 3))**

**y.append(length)**

**else:**

**alpha\_num = count\_alpha(text, (i + 1) \* 5000)**

**total.append(round(float(entropy\_cal(alpha\_num)["total"]), 3))**

**y.append((i + 1) \* 5000)**

**plt.plot(y, total, 'r-')**

**plt.xlabel("Number of Words")**

**plt.ylabel("Entropy")**

**plt.title('Total Entropy Tendency')**

**plt.savefig("Entropy\_EN/total.jpg")**

**plt.close()**

最后的计算结果如下：

英语语言的总熵趋势图：

图表

描述已自动生成

各个字母的信息熵趋势图：

图表, 直方图

描述已自动生成

图表

描述已自动生成

图表, 箱线图

描述已自动生成

图表

描述已自动生成

图表

描述已自动生成

图表

描述已自动生成

图表

描述已自动生成

图表, 折线图

描述已自动生成

图表, 直方图

描述已自动生成

图表, 箱线图

描述已自动生成

图表

描述已自动生成

图表, 箱线图

描述已自动生成

图表, 直方图

描述已自动生成

图表

描述已自动生成

图表

描述已自动生成

图表, 直方图

描述已自动生成

图表

描述已自动生成

图表

描述已自动生成

图表

描述已自动生成

图表, 箱线图

描述已自动生成

图表

描述已自动生成

图表, 形状

描述已自动生成

图表, 箱线图

描述已自动生成

图表

描述已自动生成

图表

描述已自动生成

图表

描述已自动生成