内容简介

集 Python、Excel、数据分析为一体式本书的特色。

本书围绕整个数据分析的常规流程：熟悉工具——明确目的——获取数据——熟悉数据——处理数据——分析数据——得出结论——验证结论——展示结论进行 Excel 和 Python 的对比实现，告诉你每一个过程中都会用到什么，过程和过程之间有什么联系。本书既可以作为系统学习数据分析操作流程的说明书，也可以作为一本数据分析师案头必备的实操工具书。

本书通过对比 Excel 功能操作去学习 Python 的代码实现，而不是直接学习 Python 代码，大大降低了学习门槛，消除了读者对代码的恐惧心理。适合刚入行的数据分析师，也适合对 Excel 比较熟练的数据分析师，以及从事其他岗位想提高工作效率的职场人。

本书可配合尚学堂科技推出的《Python400 集大型视频课程》，作为参考教材。

尚学堂·百战程序员

前言

本书特色

学员需求：学习不枯燥、能够快速入门、实战操作、熟悉底层原理；企业要求：程序员既有实战技能可以快速上手，也内功扎实熟悉底层原理后劲十足。因此，针对以上需求我们北京尚学堂科技有限公司设计了本书。

本书的主要有以下几个特点：

1. 循序渐进，由浅入深

为了方便读者学习，本书首先让读者了解了 Python 的历史和特点，通过具体的例子逐渐把读者带入 Python 的世界，掌握 Python 语言的基本要点以及基础类库、常用库和工具的使用。

2. 技术全面，内容充实

本书在保证内容使用的前提下，详细介绍了 Python 语言的各个知识点。同时，本书所涉及的内容非常全面，无论从事什么行业的读者，都可以从本书中找到可应用 Python 与本行业的地方。

3. 代码完整，详解详尽

对于书中的每个知识点都有一段示例代码，并对代码的关键点进行了注释说明。每段代码的后面都有详细的分析，同时给出了代码运行后的结果。读者可以参考运行结果阅读源程序，可以加深对程序的理解。

本书如何学习

本书共分六章，为了方便大家的学习，我们对各章节做简要说明。

第一章：本章讲解了 Numpy 数组创建、索引切片、拼接和一些通用函数。Numpy 是一个非常强大的科学计算库，为学习数据科学、人工智能打下一定基础。

第二章：本章讲解了 Matplotlib 的基本使用方法，通过对本章的学习，可以对 Matplotlib 的基本使用方法有一个全面的了解，进而对大多数数据进行可视化。

第三章：本章讲解了 Anaconda 开发环境搭建、Pandas 数据类型及处理数据的函数和方法。通过本章的学习，可以掌握 Series 对象和 DataFrame 对象的基本操作，从而为数据分析鉴定基础。

第四章：网络爬虫基础。本章讲解了一些网络爬虫概念、分析爬虫的基本流程及 HTTP 协议请求响应内容，重点讲解了 Beautiful Soup 和 XPath 的使用。

第五章：本章首先讲解了 Python 如何操作 Excel，其次使用具体的示例爬取网络数据，

尚学堂·百战程序员

并将数据保存到 TXT 文件或 excel 文件中

第六章：本章以两个数据集为例来讲解如何用 Pandas 来分析数据。

尚学堂·百战程序员

鸣谢

本书由北京尚学堂科技的教研部编制。北京尚学堂科技是一家从事人工智能、大数据、区块链、Java 技术、数据库、前端全栈、UI 设计等互联网技术培训的公司，其编写的教材和发布的教程影响了一代程序员。公司多次获得新浪、腾讯、网易、央视等年度最佳机构的荣誉。

本书在出版过程中，得到了清华大学出版社杨如林老师的大力支持，在此表示衷心感谢。另外，本书所有的编审、发行人员为本书的出版发行付出了辛勤的劳动，在此一并致以诚挚的谢意。

本书主编为高淇，主要参编人员有：高淇、张巧英等。我们以科学、严谨的态度，力求精益求精，但错误之处，在所难免，敬请广大读者批评指正，我们将不胜感激。

教研部出版组邮箱：book@sxt.cn　　高淇老师邮箱：gaoqi@sxt.cn

尚学堂·百战程序员

目录

第五篇【数据分析】

第一章 Numpy.....1

1.1 Numpy 开发环境搭建.....1

1.2 数组的创建.....2

1.2.1 array 创建.....2

1.2.2 arange 创建.....4

1.2.3 随机数创建.....5

1.2.4 ndarray 对象.....8

1.2.5 其他方式创建.....10

1.3 切片和索引.....14

1.4 改变数组的维度.....18

1.5 数组的拼接.....20

1.5.1 水平数组组合.....20

1.5.2 垂直数组组合.....21

1.6 数组的分隔.....24

1.6.1 split 分隔.....24

1.6.2 水平分隔.....26

1.6.3 垂直分隔数组.....27

1.7 算术函数.....29

1.8 数学函数.....31

1.9 聚合函数.....33

习题.....37

第二章 Matplotlib.....39

2.1 matplotlib 开发环境搭建.....39

2.2 绘制基础.....40

2.2.1 绘制直线.....41

2.2.2 绘制折线.....42

2.2.3 设置标签文字和线条粗细.....42

2.2.4 绘制一元二次方程的曲线 y=x^2.....44

2.2.5 绘制正弦曲线和余弦曲线.....45

尚学堂·百战程序员

2.3 绘制散点图.....47
2.4 绘制柱状图.....52
2.5 绘制饼状图.....56
2.6 绘制直方图.....58
2.7 绘制等高线图.....60
2.8 绘制三维图.....61
习题.....63
第三章 Pandas 数据分析库.....65
3.1 Pandas 开发环境搭建.....65
3.1.1 Anaconda 下载和安装.....65
3.1.2 新建 Jupyter Notebook 文件.....71
3.1.3 运行第一段代码.....72
3.1.4 重命名 Jupyter Notebook 文件.....73
3.1.5 保存 Jupyter Notebook 文件.....73
3.2 Pandas 数据类型.....74
3.2.1 Series 对象创建.....74
3.2.2 DataFrame 对象创建.....77
3.2.3 获取 Series 对象的值.....80
3.2.4 获取 DataFrame 的值.....82
3.2.5 Series 的方法.....87
3.2.6 Series 的条件过滤.....89
3.2.7 DataFrame 的条件过滤.....90
3.3 处理缺失值.....91
3.3.1 缺失值查看.....91
3.3.2 缺失值删除.....93
3.3.3 缺失值填充.....95
3.4 拼接.....98
3.5 merge 的使用.....101
习题.....106
第四章 网络爬虫基础.....107
4.1 初识网络爬虫.....107
4.1.1 什么是网络爬虫.....107
4.1.2 能从网络上爬取什么数据.....107
4.1.3 应不应该学爬虫.....108

尚学堂·百战程序员

4.1.4 网络爬虫是否合法 108
4.2 爬虫的基本流程 108
4.3 HTTP 协议 109
4.3.1 HTTP 请求过程 109
4.3.2 HTTP 状态码 110
4.3.3 HTTP 头部信息 111
4.3.4 HTTP 请求方式 112
4.3.5 请求 114
4.3.6 响应 115
4.4Beautiful Soup 库 118
4.4.1 Beautiful Soup 简介 118
4.4.2 Beautiful Soup 安装 118
4.4.3Beautiful Soup 库解析器 119
4.4.4 BeautifulSoup 120
4.5XPath 137
4.5.1 XPath 简介 137
4.5.2 安装 137
4.5.3 XPath 语法 137
4.5.4 XPath 使用 139
习题 143
第五章 Python 操作 Excel 存储数据 144
5.1 openpyxl 模块 144
5.1.1 安装 openpyxl 模块 144
5.1.2 Excel 文件的三个对象 144
5.2 Python 操作 Excel 146
5.2.1 Python 操作 Excel 之读取 146
5.2.2 Python 操作 Excel 之写 152
5.2.3 Python 操作 Excel 之修改样式 154
5.2.4 Python 操作 Excel 之图表 157
5.3 数据存储 163
5.3.1 存储至 TXT 163
5.3.2 存储至 excel 165
习题 170
第六章 Python 进行数据分析 171

尚学堂·百战程序员

6.1 导入外部数据.....171
6.1.1 导入.xlsx 文件.....171
6.1.2 导入.csv 文件.....174
6.1.3 导入.txt 文件.....177
6.2 查看数据.....178
6.2.1 预览前、后几行.....178
6.2.2 调用 shape 获取数据表的大小.....179
6.2.3 调用 info()获取数据类型.....180
6.2.4 调用 describe()获取数值分布情况.....181
6.3 足球运动员数据分析.....182
6.3.1 加载数据.....182
6.3.2 查看数据.....182
6.3.3 缺值处理.....184
6.3.4 运动员身高和体重分布.....186
6.3.5 左脚右脚使用数量.....189
6.3.6 相关性分析.....191
6.4 航班准点分析.....193
6.4.1 数据探索和清洗.....194
6.4.2 统计起飞是否延迟情况.....196
6.4.3 统计到达是否延迟情况.....197
6.4.4 统计机场航班起飞、到达延迟数量.....198
6.4.5 统计机场航班起飞、到达延迟的百分比.....200
习题.....202

尚学堂·百战程序员

案例目录

第一章 Numpy.....1

【示例 1-1】测试 numpy 模块安装是否成功.....1

【示例 1-2】创建一维数组.....2

【示例 1-3】创建二维数组.....3

【示例 1-4】array()函数 ndmin 参数的使用.....3

【示例 1-5】array()函数 dtype 参数的使用.....3

【示例 1-6】arange()函数创建数组.....4

【示例 1-7】arange()函数指定起始值、终止值及步长.....4

【示例 1-8】arange()函数创建二维数组.....5

【示例 1-9】numpy.random.random(size=None)的使用.....5

【示例 1-10】numpy.random.randint()的使用.....6

【示例 1-11】numpy.random. randn ()的使用.....7

【示例 1-12】np.random.normal()的使用.....8

【示例 1-13】ndarray 属性的使用.....9

【示例 1-14】zeros()函数创建数组.....10

【示例 1-15】ones()函数创建数组.....11

【示例 1-16】empty()函数创建数组.....12

【示例 1-17】linspace()函数创建等差数列.....12

【示例 1-18】linspace()函数创建等差数列指定 endpoint 参数为 true.....13

【示例 1-19】linspace()函数创建等差数列 指定 endpoint 参数为 true，retstep 为 True.....13

【示例 1-20】logspace()函数创建等比数列.....13

【示例 1-21】一维数组切片和索引的使用.....14

【示例 1-22】二维数组索引的使用一.....14

【示例 1-23】二维数组切片的使用.....15

【示例 1-24】二维数组索引的使用二.....16

【示例 1-25】负索引的使用.....17

尚学堂·百战程序员

【示例 1-26】copy()函数实现数组的复制.....18
【示例 1-27】改变数组的维度.....19
【示例 1-28】concatenate()函数实现数组的拼接.....22
【示例 1-29】vstack() 与 hstack()实现二维数组的拼接.....22
【示例 1-30】三维数组的拼接.....23
【示例 1-31】split()函数分隔一维数组.....24
【示例 1-32】split()函数分隔二维数组.....25
【示例 1-33】hsplit()的使用.....27
【示例 1-34】vsplit()垂直分隔数组.....28
【示例 1-35】transpose()函数的使用.....29
【示例 1-36】算术函数的使用.....30
【示例 1-37】函数指定输出结果的用法.....31
【示例 1-38】三角函数的使用.....31
【示例 1-39】around()、floor()、ceil()函数的使用.....32
【示例 1-40】power()函数的使用.....34
【示例 1-41】median ()函数的使用一.....34
【示例 1-42】mean ()函数的使用二.....35
第二章 Matplotlib.....39
【示例 2-1】使用 matplotlib 根据两点绘制一条线.....41
【示例 2-2】使用 matplotlib 绘制折线图.....42
【示例 2-3】使用 matplotlib 绘制折线图并设置样式.....43
【示例 2-4】解决标签、标题中的中文问题.....43
【示例 2-5】使用 matplotlib 绘制一元二次方程曲线.....45
【示例 2-6】使用 matplotlib 绘制正弦曲线和余弦曲线.....46
【示例 2-7】subplot()函数将画布分区的使用.....47
【示例 2-8】使用 matplotlib 绘制 sin()函数的散点图.....48
【示例 2-9】绘制 10 种大小 100 种颜色的散点图.....48
【示例 2-10】绘制不同种类不同颜色的线.....50
【示例 2-11】添加图例.....51

尚学堂·百战程序员

【示例 2-12】使用 bar()绘制柱状图，并设置柱的宽度.....52
【示例 2-13】使用 bar()和 barh()函数绘制柱状图.....53
【示例 2-14】对柱状图的部分柱状设置颜色.....54
【示例 2-15】使用 bar()绘制三天中三部电影的票房变化.....55
【示例 2-16】绘制饼状图.....57
【示例 2-17】绘制直方图.....58
【示例 2-18】同一画布绘制三个直方图.....59
【示例 2-19】使用 matplotlib 绘制等高线图.....60
【示例 2-20】使用 Matplotlib 绘制三维图.....61
第三章 Pandas 数据分析库.....65
【示例 3-1】使用列表创建 Series 对象.....74
【示例 3-2】Series 对象中两个重要的属性 values 和 index.....75
【示例 3-3】创建 Series 对象时候，指定 index 属性.....75
【示例 3-4】创建 Series 对象时候，使用 list 列表指定 index 属性.....75
【示例 3-5】使用字典创建 Series 对象，默认将 key 作为 index 属性.....76
【示例 3-6】使用字典创建 Series 对象，又指定了 index 属性值，如果 key 不存在，则值为 NaN.....76
【示例 3-7】标量与 index 属性创建 Series.....77
【示例 3-8】创建 DataFrame 对象.....77
【示例 3-9】查看 DataFrame 对象的 values、index 和 columns 属性.....78
【示例 3-10】使用列表创建 DataFrame 对象.....78
【示例 3-11】创建 DataFrame 对象，指定 index 属性.....79
【示例 3-12】创建 DataFrame 对象，指定列索引 columns.....79
【示例 3-13】使用列表创建 Dataframe 对象.....79
【示例 3-14】使用一个二维数组并指定 columns 和 index 创建 DataFrame 对象.....80
【示例 3-15】Series 对象的切片、索引.....80
【示例 3-16】位置索引与标签索引相同的问题.....81
【示例 3-17】Series 对象中 loc 与 iloc 的使用.....81
【示例 3-18】创建 DataFrame 对象.....82

尚学堂·百战程序员

【示例 3-19】获取 DataFrame 对象中某一列/某几列..... 83
【示例 3-20】通过传入位置获取 DataFrame 对象中某一列/某几列..... 83
【示例 3-21】获取 DataFrame 对象中连续几列..... 84
【示例 3-22】DataFrame 对象中选择某一行/某几行..... 85
【示例 3-23】DataFrame 对象中选择某一行/某几行..... 86
【示例 3-24】同时选择连续的部分行和部分列..... 87
【示例 3-25】同时选择不连续的部分行和部分列..... 87
【示例 3-26】使用 mean、min、max、std 等方法对数据集进行各种运算，最后对数据集进行排序操作..... 88
【示例 3-27】Series 对象指定条件来过滤数据..... 89
【示例 3-28】DataFrame 对象指定条件来过滤数据..... 90
【示例 3-29】缺失值查看..... 92
【示例 3-30】获取所有缺失值..... 92
【示例 3-31】删除缺失值..... 93
【示例 3-32】删除空白行..... 94
【示例 3-33】Series 对象缺失值填充..... 95
【示例 3-34】DataFrame 对象缺失值填充..... 96
【示例 3-35】列的平均值来填充..... 97
【示例 3-36】Series 对象拼接..... 98
【示例 3-37】两个 df 对象拼接，默认找相同的列索引进行合并..... 98
【示例 3-38】两个 df 对象拼接，添加 axis 参数..... 99
【示例 3-39】两个 df 对象拼接，索引重复问题..... 99
【示例 3-40】两个 df 对象拼接，解决索引重复问题加 ignore_index 属性..... 100
【示例 3-41】两个 df 对象拼接，解决索引重复问题，加 keys 属性..... 100
【示例 3-42】两个 df 对象拼接，join 内连接做交集..... 100
【示例 3-43】merge 的使用，默认以 how=’inner’进行合并..... 101
【示例 3-44】merge 的使用，参数 how=’outer’进行合并..... 102
【示例 3-45】merge 的使用，参数 how=’left’进行合并..... 103
【示例 3-46】行索引不同， how=’right’进行合并..... 104

尚学堂·百战程序员

第四章 网络爬虫基础..... 107
【示例 4-1】第一个简单的爬虫..... 115
【示例 4-2】Response 设置编码..... 116
【示例 4-3】查看 Beautiful Soup 安装是否成功..... 118
【示例 4-4】初始化 BeautifulSoup 使用 lxml，把第二个参数改为 lxml..... 119
【示例 4-5】获取 title 节点，查看它的类型..... 120
【示例 4-6】选择元素..... 122
【示例 4-7】调用 name 属性获取节点的名称..... 122
【示例 4-8】调用 attrs 获取所有属性..... 123
【示例 4-9】简单获取属性的方式..... 123
【示例 4-10】调用 string 属性获取节点元素包含的文本内容..... 124
【示例 4-11】嵌套选择..... 124
【示例 4-12】调用 children 属性，获取它的直接子节点..... 125
【示例 4-13】调用 parent 属性，获取某个节点元素的父节点..... 126
【示例 4-14】调用 parents 属性，获取某个节点元素的祖先节点..... 127
【示例 4-15】调用 next_sibling 和 previous_sibling 分别获取节点的下一个和上一个兄弟元素
..... 128
【示例 4-16】find_all 方法传入 name 参数，根据节点名来查询元素..... 129
【示例 4-17】find_all 方法传入 attrs 参数，根据属性来查询..... 130
【示例 4-18】find_all 方法根据属性来查询..... 132
【示例 4-19】find_all 方法根据文本来查询..... 133
【示例 4-20】find 方法查询第一个匹配的元素..... 134
【示例 4-21】CSS 选择器的使用..... 135
【示例 4-22】用以 // 开头的 XPath 规则来选取所有符合要求的节点..... 139
【示例 4-23】指定匹配的节点名称..... 140
【示例 4-24】选择 li 节点的所有直接 a 子节点..... 140
【示例 4-25】用@符号进行属性过滤..... 140
【示例 4-26】调用 text()方法获取文本..... 141
【示例 4-27】@符号跟属性名直接获取节点的属性值..... 141

尚学堂·百战程序员

【示例 4-28】多属性匹配.....142
第五章 Python 操作 Excel 存储数据.....144
【示例 5-1】创建 Workbook 对象.....145
【示例 5-2】打开本地工作簿，获取所有工作表名称.....146
【示例 5-3】创建工作表.....147
【示例 5-4】根据工作表名称获取工作表.....147
【示例 5-5】获取单元格对象及单元格的值.....148
【示例 5-6】获取单元格的行、列及值.....148
【示例 5-7】读取整行、整列及部分几行.....149
【示例 5-8】使用 iter_rows()逐行读取.....150
【示例 5-9】部分行部分列切片读取.....151
【示例 5-10】列数字与字母的对应转换.....151
【示例 5-11】创建、删除工作表.....152
【示例 5-12】使用 list 写入.....153
【示例 5-13】使用 range 方式写入.....153
【示例 5-14】使用 cell()方法写入.....154
【示例 5-15】修改字体样式.....154
【示例 5-16】设置单元格公式.....155
【示例 5-17】设置行高和列宽.....155
【示例 5-18】合并单元格.....156
【示例 5-19】拆分单元格.....157
【示例 5-20】饼图.....157
【示例 5-21】条形图.....159
【示例 5-22】气泡图.....160
【示例 5-23】散点图.....162
【示例 5-24】爬取数据存储至 txt.....164
【示例 5-25】爬取数据保存到 excel.....165
【示例 5-26】使用 bs4 爬取数据保存到 excel.....166
【示例 5-27】使用 XPath 爬取数据保存到 excel.....168

尚学堂·百战程序员

第六章 Python 进行数据分析.....171
【示例 6-1】导入.xlsx 文件.....171
【示例 6-2】导入.xlsx 文件时，指定导入哪个 Sheet.....172
【示例 6-3】导入.xlsx 文件时，传入 Sheet 的顺序.....172
【示例 6-4】导入.xlsx 文件时，通过 index_col 指定行索引.....172
【示例 6-5】导入.xlsx 文件时，通过 header 指定行索引.....173
【示例 6-6】导入.xlsx 文件时，通过 header 指定行索引.....173
【示例 6-7】导入.xlsx 文件时，通过 usecols 指定列索引.....174
【示例 6-8】导入.csv 文件，文件编码格式是 gbk.....174
【示例 6-9】导入.csv 文件，指明分隔符.....175
【示例 6-10】导入.csv 文件，nrows 参数的使用.....176
【示例 6-11】导入.csv 文件，engine 参数的使用.....177
【示例 6-12】导入.txt 文件.....177
【示例 6-13】使用 read_table()方法导入.csv 文件.....178
【示例 6-14】预览前、后几行.....178
【示例 6-15】调用 shape 获取数据表的行、列数.....179
【示例 6-16】调用 info()获取数据类型.....180
【示例 6-17】调用 describe()获取数值分布情况.....181
【示例 6-18】加载足球运动员数据.....182
【示例 6-19】查看数据前 5 行.....182
【示例 6-20】查看数据的数据类型.....182
【示例 6-21】删除 Club_Position 的缺失值.....184
【示例 6-22】查看是否有重复数据.....186
【示例 6-23】去掉 Height 和 Weight 数据的单位.....187
【示例 6-24】运动员 Height 数值分布.....187
【示例 6-25】运动员 Weight 数值分布.....188
【示例 6-26】查看足球运动员左脚右脚使用情况.....189
【示例 6-27】使用饼状图来显示左脚右脚选手数量的差别.....190
【示例 6-28】value_counts()方法的使用.....191

尚学堂·百战程序员

【示例 6-29】分析足球运动员的身高和体重是否相关.....191
【示例 6-30】分析足球运动员的身高和评分是否相关.....192
【示例 6-31】通过相关系数查看两个变量是否相关.....193
【示例 6-32】分析对评分影响前 10 的变量.....193
【示例 6-33】加载数据集.....194
【示例 6-34】查看及删除空值 Unnamed 列.....195
【示例 6-35】删除重复数据.....195
【示例 6-36】查询起飞是否延迟.....196
【示例 6-37】统计起飞延迟与不延迟，使用饼状图来显示.....197
【示例 6-38】统计到达延迟与不延迟，使用饼状图显示.....197
【示例 6-39】统计机场航班起飞延迟数量，使用柱状图显示.....198
【示例 6-40】统计机场航班到达延迟数量，使用柱状图显示.....199
【示例 6-41】合并机场航班起飞和到达延迟.....199
【示例 6-42】机场起飞总航班数.....200
【示例 6-43】机场航班起飞延迟的百分比.....200
【示例 6-44】机场航班到达延迟的百分比.....201
【示例 6-45】组合起飞延迟和到达延迟百分比，柱状图描述.....201

尚学堂·百战程序员

第一章 Numpy

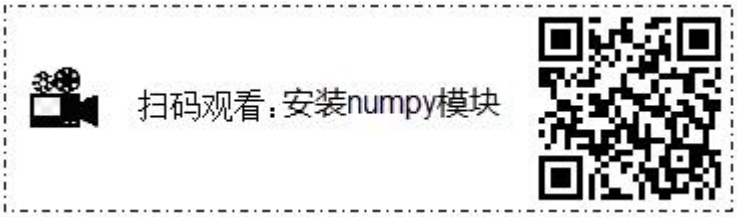
NumPy(Numerical Python) 是科学计算基础库，提供大量科学计算相关功能，比如数据统计，随机数生成等。其提供最核心类型为多维数组类型（ndarray），支持大量的维度数组与矩阵运算，Numpy 支持向量处理 ndarray 对象，提高程序运算速度。

那么 Numpy 到底有什么功能呢？其实 Numpy 的功能非常多，主要用于数组计算。Numpy 可以让用户在 Python 语言中使用向量和数学矩阵。Numpy 是 Python 语言在科学计算领域取得成功的关键之一，如果想通过 Python 语言学习数据科学、人工智能（包括深度学习、语言处理等分支），就必须学习 Numpy。

通过阅读本章，你可以：

- 了解什么是 Numpy
- 掌握如何使 Numpy 操作数组，如创建数组、改变数组的维度、拼接和分隔数组等
- 掌握 Numpy 的常用函数，如数组存取函数、加权平均数函数、最大值和最小值函数等

1.1 Numpy 开发环境搭建



Numpy 是第三方程序库，所以在使用 Numpy 之前必须安装 Numpy。如果使用的是 Anaconda Python 开发环境，那么 Numpy 已经集成到 Anaconda 环境中了，不需要再安装。如果使用的是官方开发环境，可以使用 pip 命令安装 Numpy，语法格式如下：

```
pip install numpy
```

【示例 1-1】测试 numpy 模块安装是否成功

```
import numpy as np
a=np.arange(10)
print(a)
```

执行结果如图 1-1 所示：

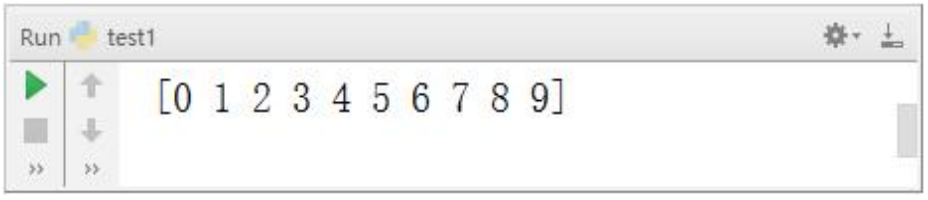


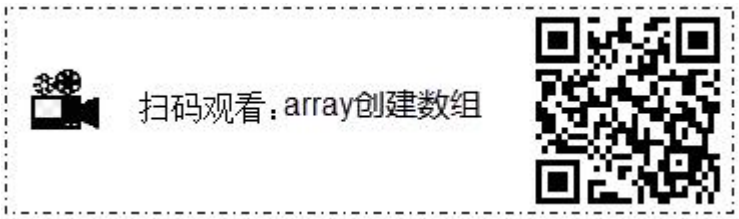
图 1-1 示例 1-1 运行效果图

在上面的程序中只涉及 numpy 模块中的一个 arange 函数，该函数可以传入一个整数类

型的参数 n，函数返回值看着像一个列表，其实返回值类型是 numpy.ndarray。这是 Numpy 中特有的数组类型。如果传入 arange 函数的参数值是 n，那么 arange 函数会返回 0 到 n-1 的 ndarray 类型的数组。

1.2 数组的创建

1.2.1 array 创建



numpy 模块的 array 函数可以生成多维数组。例如，如果要生成一个二维数组，需要向 array 函数传递一个列表类型的参数。每一个列表元素是一维的 ndarray 类型数组，作为二维数组的行。另外，通过 ndarray 类的 shape 属性可以获得数组每一维的元素个数（元组形式），也可以通过 shape[n]形式获得每一维的元素个数，其中 n 是维度，从 0 开始。

语法格式如下：

```
numpy.array(object, dtype = None, copy = True, order = None, subok = False, ndmin = 0)
```

表 1-1 array 参数说明

名称	描述
object	数组或嵌套的数列
dtype	数组元素的数据类型，可选
copy	对象是否需要复制，可选
order	创建数组的样式，C 为行方向，F 为列方向，A 为任意方向（默认）
subok	默认返回一个与基类类型一致的数组
ndmin	指定生成数组的最小维度

【示例 1-2】创建一维数组

```
b=np.array([1,2,3,4,5,6])
print(b)
print('b 数组的维度：',b.shape)
```

执行结果如图 1-2 所示：

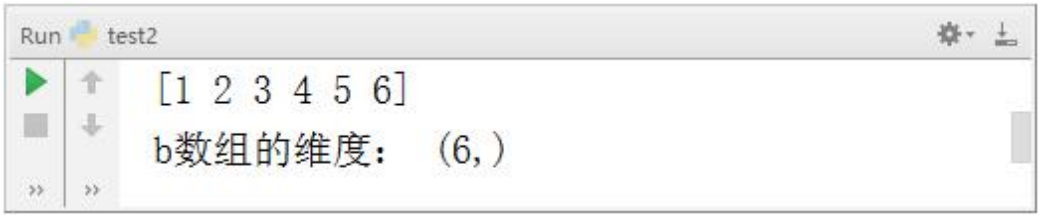


图 1-2 示例 1-2 运行效果图

【示例 1-3】创建二维数组

```
a=np.array([[1,2,3],[4,5,6],[7,8,9]])
print(a)
print('a 数组的维度:',a.shape)
```

执行结果如图 1-3 所示：

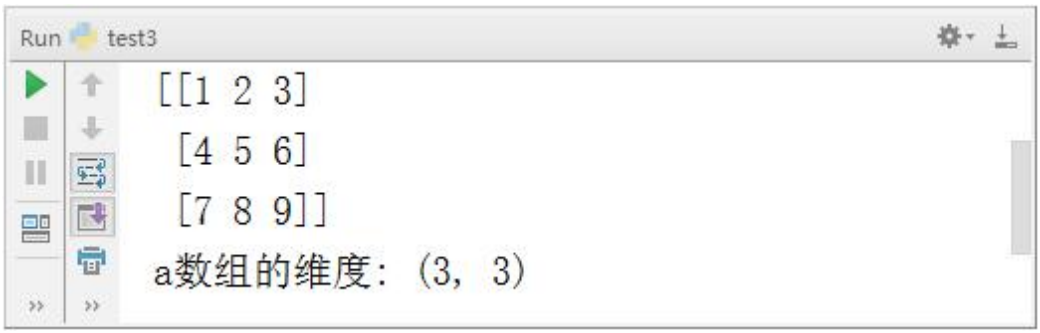


图 1-3 示例 1-3 运行效果图

【示例 1-4】array()函数 ndmin 参数的使用

```
import numpy as np
a=np.array([1,2,3,4,5,6],ndmin=3)
print(a)
```

执行结果如图 1-4 所示：

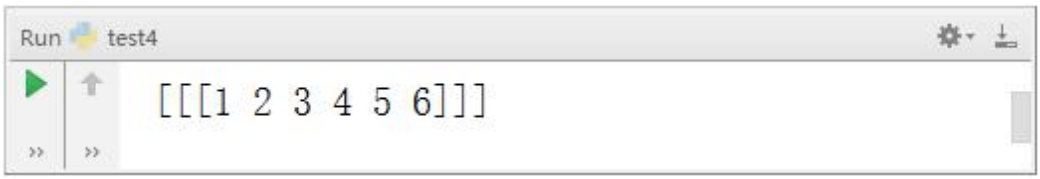


图 1-4 示例 1-4 运行效果图

【示例 1-5】array()函数 dtype 参数的使用

```
a=np.array([1,2,3,4,5,6],dtype=complex)
print(a)
```

执行结果如图 1-5 所示：

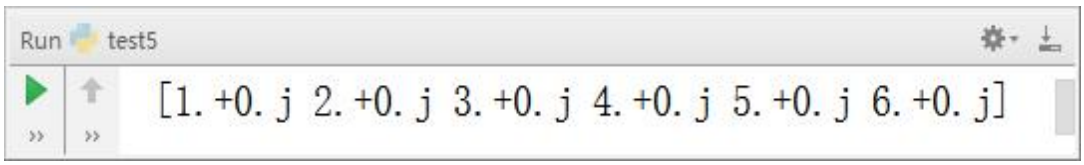


图 1-5 示例 1-5 运行效果图

1.2.2 arange 创建



使用 arange 函数创建数值范围并返回 ndarray 对象，函数语法格式如下：

```
numpy.arange(start, stop, step, dtype)
```

表 1-2 arange 参数说明

参数	描述
start	起始值，默认为 0
stop	终止值（不包含）
step	步长，默认为 1
dtype	返回 ndarray 的数据类型，如果没有提供，则会使用输入数据的类型。

【示例 1-6】arange()函数创建数组

```
import numpy as np
x=np.arange(0,6,dtype=int)
print(x)
```

执行结果如图 1-6 所示：



图 1-6 示例 1-6 运行效果图

【示例 1-7】arange()函数指定起始值、终止值及步长

```
x=np.arange(10,20,2,dtype=float)
print(x)
```

执行结果如图 1-7 所示：

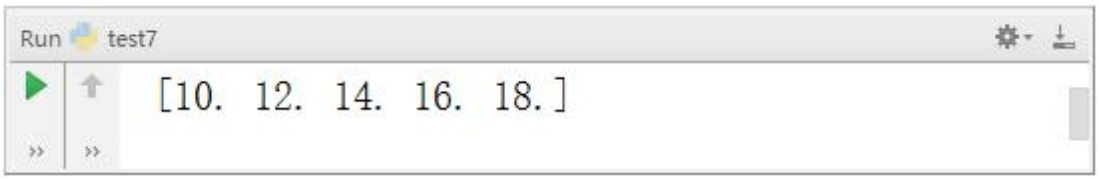


图 1-7 示例 1-7 运行效果图

【示例 1-8】arange()函数创建二维数组

```
b=np.array([np.arange(1,4),np.arange(4,7),np.arange(7,10)])
print(b)
print('b 数组的维度：',b.shape)
```

执行结果如图 1-8 所示：

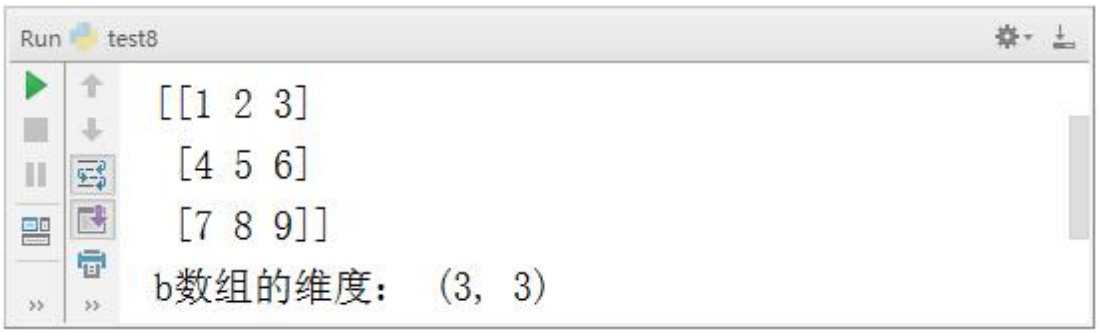
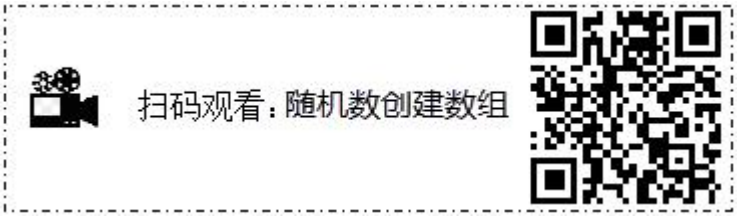


图 1-8 示例 1-8 运行效果图

1.2.3 随机数创建



numpy 中的 random 模块包含了很多方法可以用来产生随机数，对 random 中的一些常用方法如表 1-3 所示：

表 1-3 random 的常用方法

函数	说明
seed	确定随机生成器种子
permutation	返回一个序列的随机排列或返回一个随机排列的范围
shuffle	对一个序列就地随机排列
rand	产生均匀分布的样本值
randint	该方法有三个参数 low、high、size 三个参数。默认 high 是 None,如果只有 low，那范围就是[0,low)。如果有 high，范围就是[low,high)。
random(size=None)	该方法返回[0.0, 1.0)范围的随机数。
randn(d0,d1,…,dn)	randn 函数返回一个或一组样本，具有标准正态分布（期望为 0，方差为 1）。dn 表格每个维度，返回值为指定维度的 array

【示例 1-9】numpy.random.random(size=None)的使用

```
#numpy.random.random(size=None)
#返回[0.0, 1.0)范围的随机数
```

```
import numpy as np
print('生成一维（4，）的随机数组：')
x=np.random.random(size=4)
print(x)
print('生成二维（3,4）的随机数组：')
y=np.random.random(size=(3,4))
print(y)
```

执行结果如图 1-9 所示：



图 1-9 示例 1-9 运行效果图

【示例 1-10】numpy.random.randint()的使用

```
import numpy as np
#numpy.random.randint()的使用
#生成 [0,low)范围的随机整数
x=np.random.randint(5,size=10)
print(x)
#生成[low,high)范围的随机整数
y=np.random.randint(5,10,size=10)
print(y)

#生成[low,high)范围的 2*4 的随机整数
z=np.random.randint(5,10,size=(2,4))
print(z)
```

执行结果如图 1-10 所示：

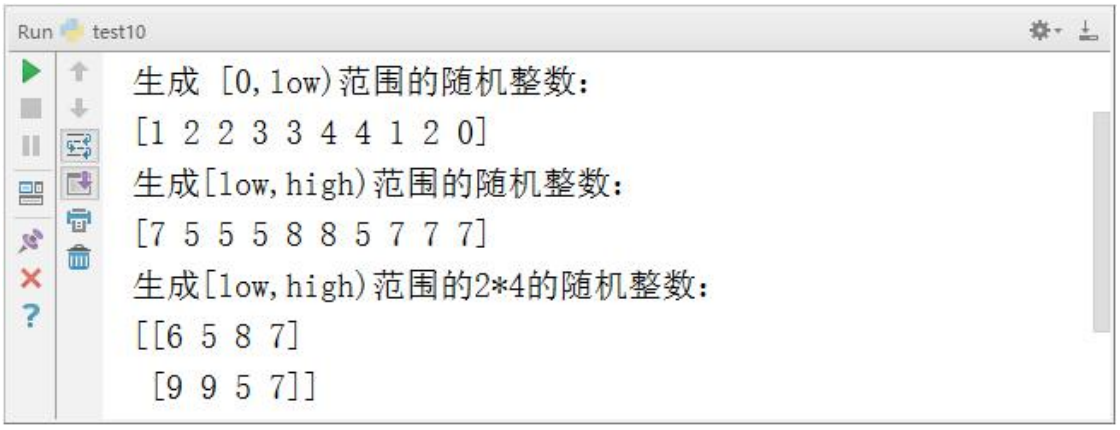


图 1-10 示例 1-10 运行效果图

【示例 1-11】numpy.random. randn ()的使用

```
# randn 函数返回一个或一组样本，具有标准正态分布
import numpy as np
x=np.random.randn()
print(x)
y=np.random.randn(2,4)
print(y)
z=np.random.randn(2,3,4)
print(z)
```

执行结果如图 1-11 所示：

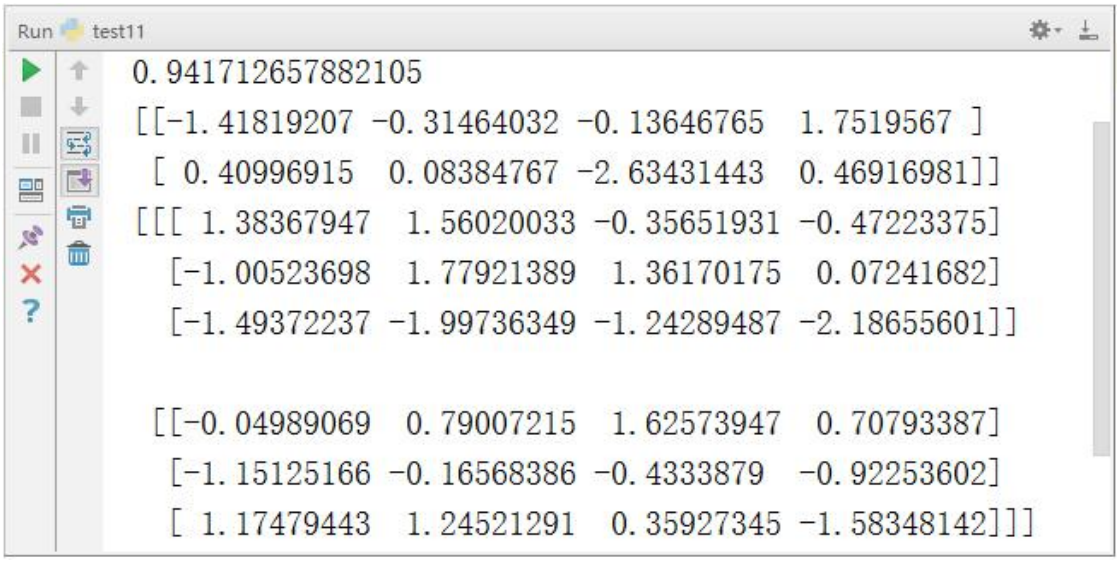


图 1-11 示例 1-11 运行效果图

【示例 1-12】np.random.normal()的使用

```
import numpy as np
#正太分布（高斯分布）loc：期望  scale：方差  size  形状
print(np.random.normal(loc=3,scale=4,size=(2,2,3)))
```

执行结果如图 1-12 所示：

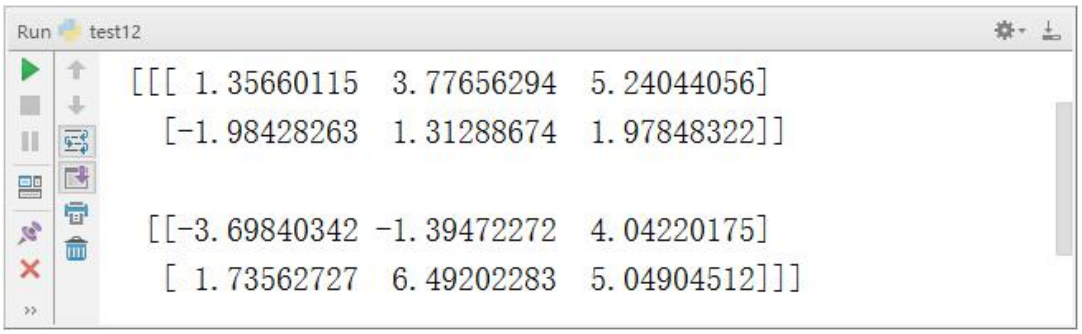
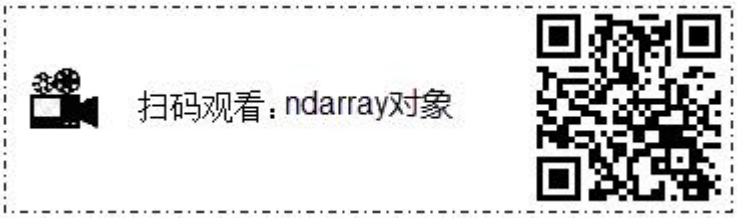


图 1-12 示例 1-12 运行效果图

1.2.4 ndarray 对象

NumPy 最重要的一个特点是其 N 维数组对象 ndarray，它是一系列同类型数据的集合，以 0 下标为开始进行集合中元素的索引。



ndarray 对象是用于存放同类型元素的多维数组。

ndarray 中的每个元素在内存中都有相同存储大小的区域。

ndarray 内部由以下内容组成：

- 一个指向数据（内存或内存映射文件中的一块数据）的指针。
- 数据类型或 dtype，描述在数组中的固定大小值的格子。
- 一个表示数组形状（shape）的元组，表示各维度大小的元组。

NumPy 的数组中比较重要 ndarray 对象属性有：

表 1-4 ndarray 对象属性

属性	说明
ndarray.ndim	秩，即轴的数量或维度的数量
ndarray.shape	数组的维度，对于矩阵，n 行 m 列
ndarray.size	数组元素的总个数，相当于 .shape 中 n*m 的值
ndarray.dtype	ndarray 对象的元素类型
ndarray.itemsize	ndarray 对象中每个元素的大小，以字节为单位
ndarray.flags	ndarray 对象的内存信息

属性	说明
ndarray.real	ndarray 元素的实部
ndarray.imag	ndarray 元素的虚部
ndarray.data	包含实际数组元素的缓冲区，由于一般通过数组的索引获取元素，所以通常不需要使用这个属性。

【示例 1-13】ndarray 属性的使用

```
import numpy as np
x1=np.random.randint(10,size=6)
print(x1)
x2=np.random.randint(10,size=(3,4))
print(x2)
x3=np.random.randn(3,4,5)
print('ndim 属性'.center(20,'*'))
print('ndim:',x1.ndim,x2.ndim,x3.ndim)
print('shape 属性'.center(20,'*'))
print('shape:',x1.shape,x2.shape,x3.shape)
print('dtype 属性'.center(20,'*'))
print('dtype:',x1.dtype,x2.dtype,x3.dtype)
print('size 属性'.center(20,'*'))
print('size:',x1.size,x2.size,x3.size)
print('itemsize 属性'.center(20,'*'))#一个字节是 8 位
print('itemsize:',x1.itemsize,x2.itemsize,x3.itemsize)
```

执行结果如图 1-13 所示：

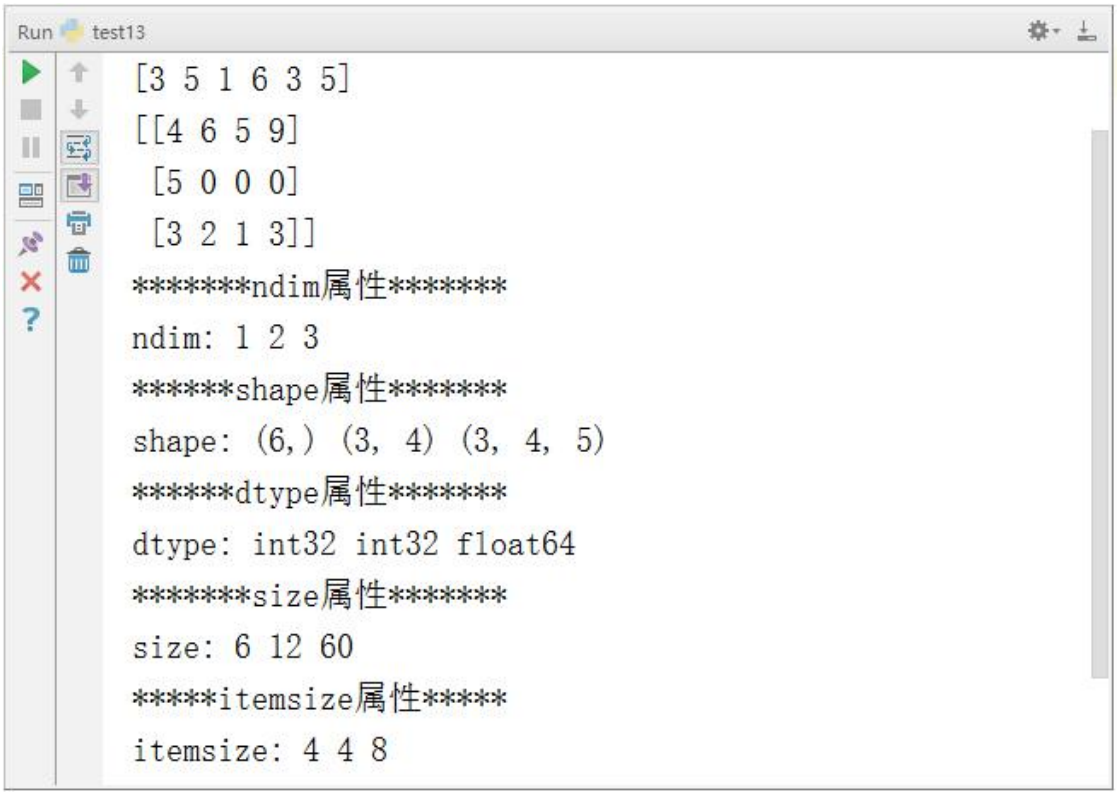
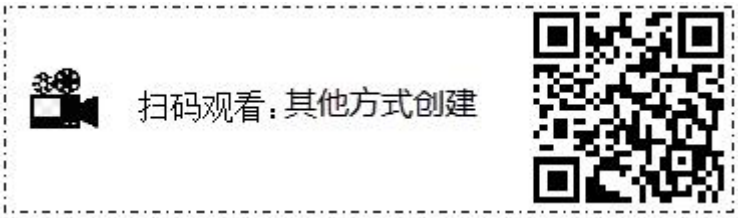


图 1-13 示例 1-13 运行效果图

1.2.5 其他方式创建



ndarray 数组除了可以使用底层 ndarray 构造器来创建外，也可以通过以下几种方式来创建。

zeros 创建指定大小的数组，数组元素以 0 来填充：

```
numpy.zeros(shape, dtype = float, order = 'C')
```

【示例 1-14】zeros()函数创建数组

```
import numpy as np
x=np.zeros(5)
print(x)
#设置类型为整数
y=np.zeros((5,),dtype=int)
print(y)
z=np.zeros((2,2))
print(z)
```

执行结果如图 1-14 所示：



图 1-14 示例 1-14 运行效果图

numpy.ones 创建指定形状的数组，数组元素以 1 来填充：

```
numpy.ones(shape, dtype = None, order = 'C')
```

【示例 1-15】ones()函数创建数组

```
import numpy as np
x=np.ones(5)
print(x)
y=np.ones((3,4),dtype=int)
print(y)
```

执行结果如图 1-15 所示：



图 1-15 示例 1-15 运行效果图

numpy.empty() 方法用来创建一个指定形状（shape）、数据类型（dtype）且未初始化的数组，里面的元素的值是之前内存的值：

```
numpy.empty(shape, dtype = float, order = 'C')
```

表 1-5 empty 参数说明

参数	描述
shape	数组形状
dtype	数据类型，可选
order	有"C"和"F"两个选项,分别代表，行优先和列优先，在计算机内存中的存储元素的顺序。

【示例 1-16】empty()函数创建数组

```
import numpy as np
x=np.empty([3,2],dtype=int)
print(x)
```

执行结果如图 1-16 所示：

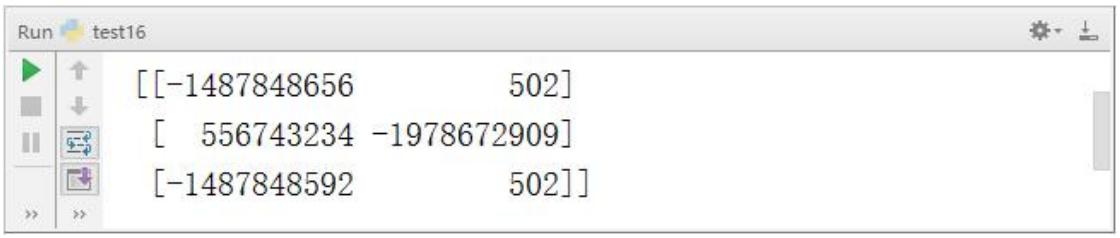


图 1-16 示例 1-16 运行效果图

linspace 函数用于创建一个一维数组，数组是一个等差数列构成的，格式如下：

```
np.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)
```

表 1-6 linspace 参数说明

参数	描述
start	序列的起始值
stop	序列的终止值，如果 endpoint 为 true，该值包含于数列中
num	要生成的等步长的样本数量，默认为 50
endpoint	该值为 ture 时，数列中中包含 stop 值，反之不包含，默认是 True。
retstep	如果为 True 时，生成的数组中会显示间距，反之不显示。
dtype	ndarray 的数据类型

【示例 1-17】linspace()函数创建等差数列

```
import numpy as np
x=np.linspace(1,10,10)
print(x)
```

执行结果如图 1-17 所示：

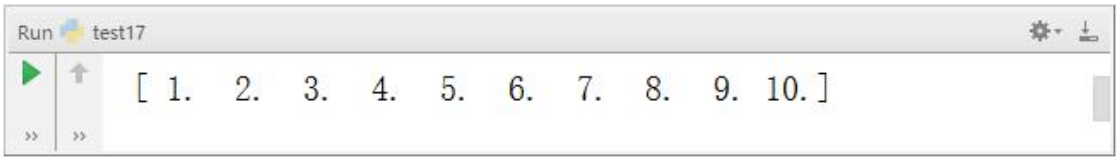


图 1-17 示例 1-17 运行效果图

【示例 1-18】linspace()函数创建等差数列指定 endpoint 参数为 true

```
import numpy as np
x=np.linspace(10,20,5,endpoint=True)
print(x)
```

执行结果如图 1-18 所示：

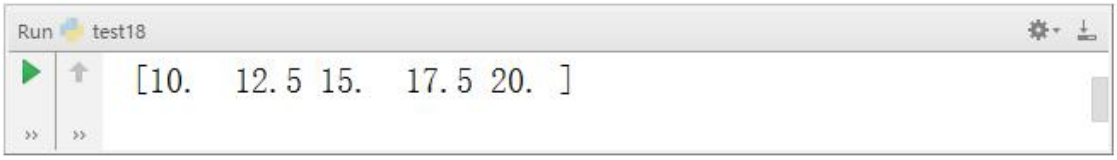


图 1-18 示例 1-18 运行效果图

【示例 1-19】linspace()函数创建等差数列 指定 endpoint 参数为 true，retstep 为 True

```
import numpy as np
x=np.linspace(10,20,5,endpoint=True,retstep=True)
print(x)
```

执行结果如图 1-19 所示：

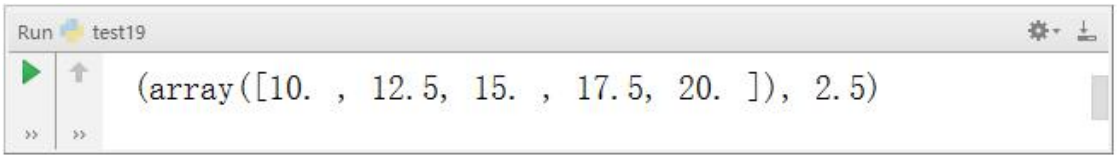


图 1-19 示例 1-19 运行效果图

numpy.logspace 函数用于创建一个于等比数列。格式如下：

```
np.logspace(start, stop, num=50, endpoint=True, base=10.0, dtype=None)
```

表 1-7 logspace 参数说明

参数	描述
start	序列的起始值为：base ** start
stop	序列的终止值为：base ** stop。如果 endpoint 为 true，该值包含于数列中
num	要生成的等步长的样本数量，默认为 50
endpoint	该值为 ture 时，数列中中包含 stop 值，反之不包含，默认是 True。
base	对数 log 的底数。
dtype	ndarray 的数据类型

【示例 1-20】logspace()函数创建等比数列

```
import numpy as np
```

```
x=np.logspace(0,9,10,base=2)
print(x)
```

执行结果如图 1-20 所示：

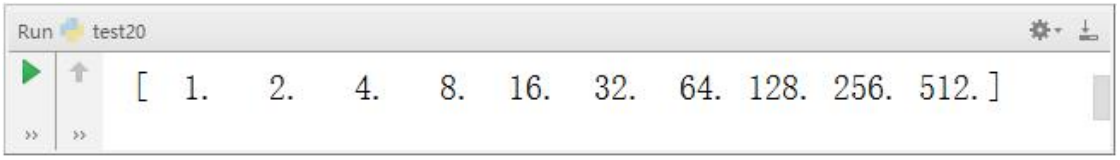
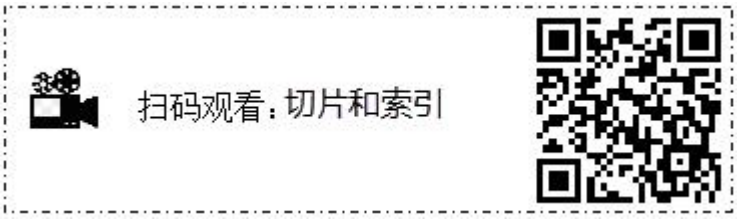


图 1-20 示例 1-20 运行效果图

1.3 切片和索引



ndarray 对象的内容可以通过索引或切片来访问和修改，与 Python 中 list 的切片操作一样。ndarray 数组可以基于 0 - n 的下标进行索引，并设置 start, stop 及 step 参数进行，从原数组中切割出一个新数组。

【示例 1-21】一维数组切片和索引的使用

```
import numpy as np
x=np.arange(10)
print('原数组：',x)
y = x[2:7:2]
z = x[2:]
print('对数组进行[2:7:2]切片：',y)
print('对数组进行[2:]切片：',z)
```

执行结果如图 1-21 所示：



图 1-21 示例 1-21 运行效果图

【示例 1-22】二维数组索引的使用一

```
import numpy as np
x=np.arange(1,13)
```

```
a=x.reshape(4,3)
print('数组元素')
print(a)
print('获取第二行')
print(a[1])
print('获取第三行第二列')
print(a[2][1])
```

执行结果如图 1-22 所示：

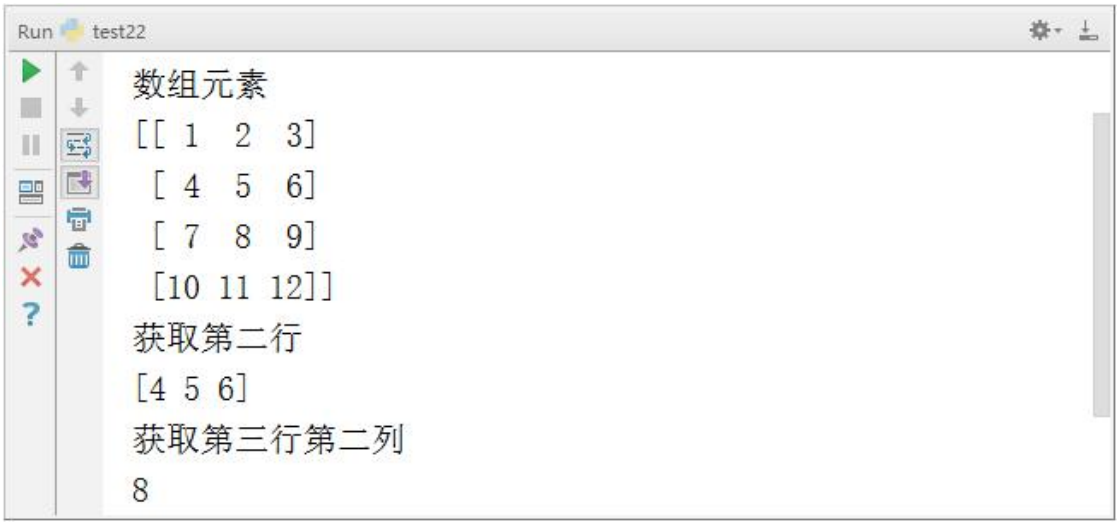


图 1-22 示例 1-22 运行效果图

【示例 1-23】二维数组切片的使用

```
import numpy as np
x=np.arange(1,13)
a=x.reshape(4,3)
print('数组元素')
print(a)
#使用索引获取
print('所有行的第二列')
print(a[:,1])
print('奇数行的第一列')
print(a[::2,0])
```

执行结果如图 1-23 所示：

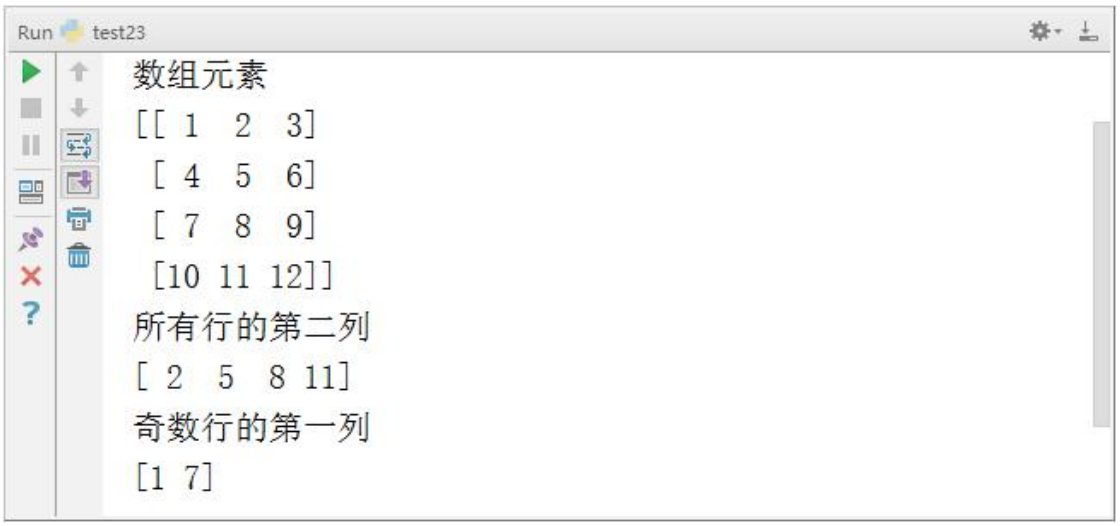


图 1-23 示例 1-23 运行效果图

【示例 1-24】二维数组索引的使用二

```
import numpy as np
a = np.arange(1,13).reshape(4,3)
print('数组元素')
print(a)
print('获取第三行第二列的结果：',a[2,1])
print('同时获取第三行第二列，第四行第一列')
print('分别获取：',np.array((a[2,1],a[3,0])))
print('第一个元组是行索引，第二个元组是列索引获取：',a[(2,3),(1,0)])
```

执行结果如图 1-24 所示：

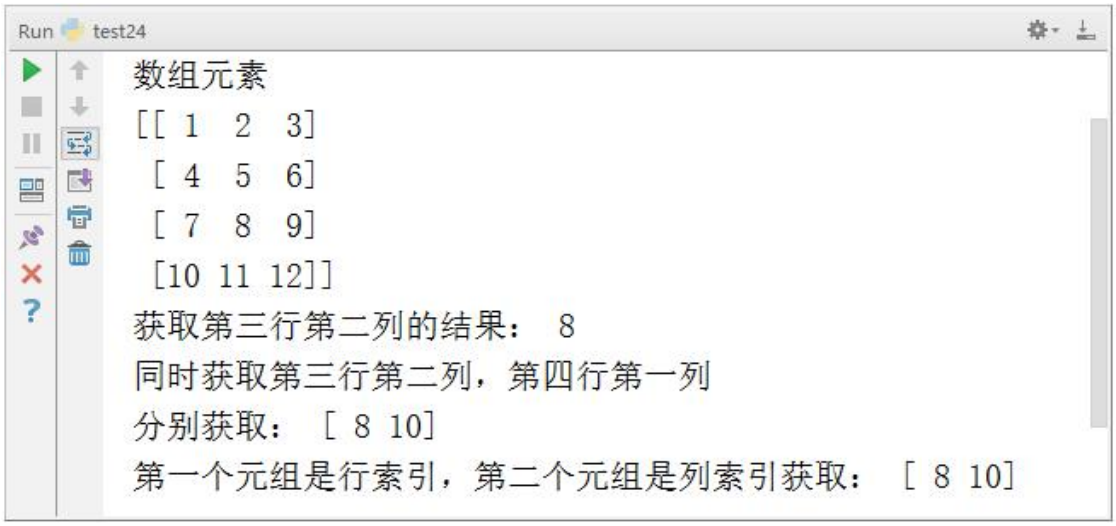


图 1-24 示例 1-24 运行效果图

【示例 1-25】负索引的使用

```
import numpy as np
x=np.arange(1,13).reshape(4,3)
print('数组元素')
print(x)
print('获取最后一行')
print(a[-1])
print('行进行倒序')
print(a[::-1])
print('行列都倒序')
print(a[::-1,::-1])
```

执行结果如图 1-25 所示：

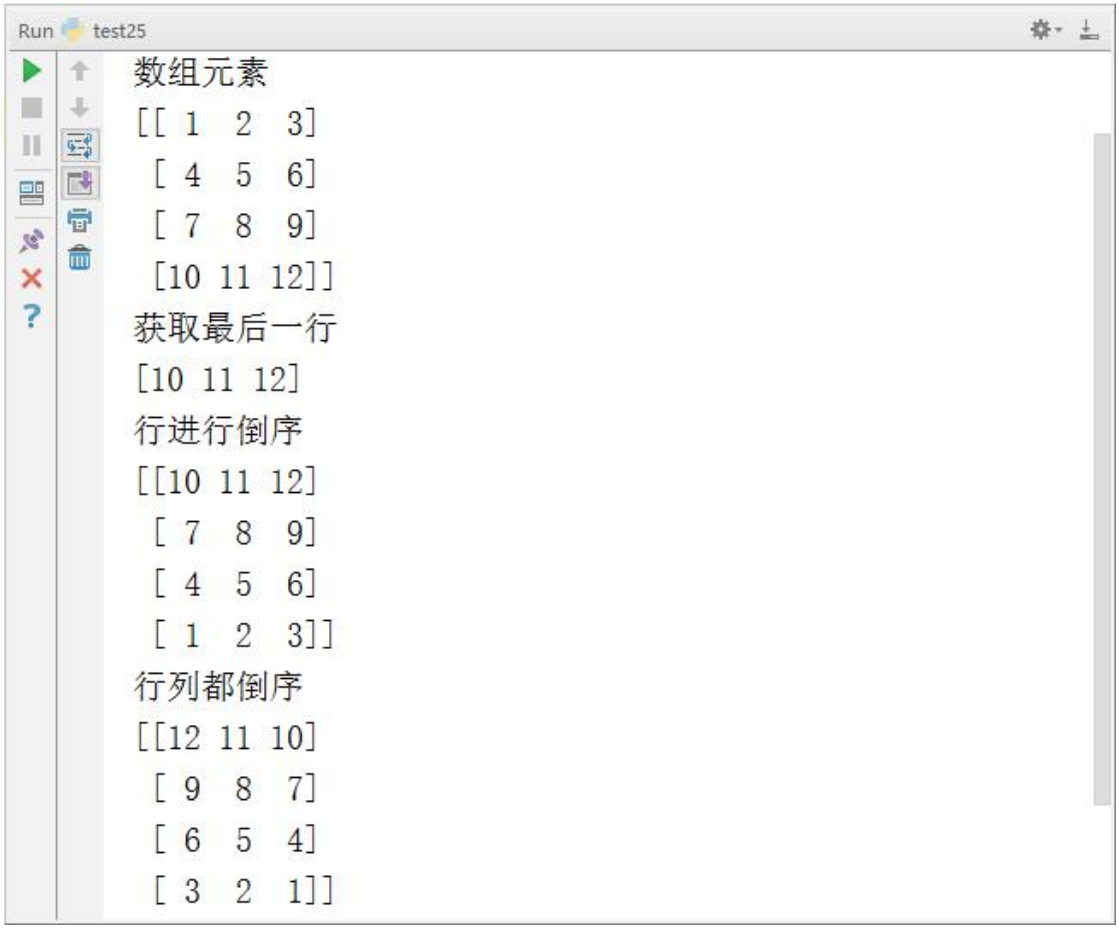


图 1-25 示例 1-25 运行效果图

所有切片取出来的数组，即使你把它赋值给了新的变量，它仍是原来数组的视图。

【示例 1-26】copy()函数实现数组的复制

```
import numpy as np
a=np.arange(1,13).reshape(3,4)
sub_array=a[:2,:2]
sub_array[0][0]=1000
print(a)
print(sub_array)
print('copy'*20)
sub_array=np.copy(a[:2,:2])
sub_array[0][0]=2000
print(a)
print(sub_array)
```

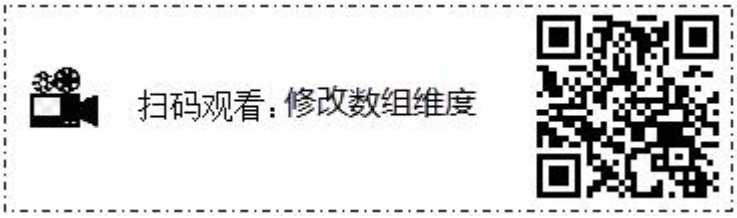
执行结果如图 1-26 所示：



图 1-26 示例 1-26 运行效果图

1.4 改变数组的维度

处理数组的一项重要工作就是改变数组的维度，包含提高数组的维度和降低数



组的维度，还包括数组的转置。Numpy 提供的大量 API 可以很轻松地完成这些数组的操作。例如，通过 reshape 方法可以将一维数组变成二维、三维或者多维数组。通过 ravel 方法或 flatten 方法可以将多维数组变成一维数组。改变数组的维度还可以直接设置 Numpy 数组的 shape 属性（元组类型），通过 resize 方法也可以改变数组的维度。

【示例 1-27】改变数组的维度

```
import numpy as np
#创建一维的数组
a=np.arange(24)
print(a)
print('数组 a 的维度：',a.shape)
print('-'*30)

#使用 reshape 将一维数组变成三维数组
b=a.reshape(2,3,4)
print(b)
print('数组 b 的维度：',b.shape)
print('-'*30)

#将 a 变成二维数组
c=a.reshape(3,8)
print(c)
print('数组 c 的维度：',c.shape)
print('-'*30)

#使用 ravel 函数将三维的 b 变成一维的数组
a1=b.ravel()
print(a1)
print('-'*30)

#使用 flatten 函数将二维的 c 变成一维的数组
a2=c.flatten()
print(a2)
print('-'*30)
```

执行结果如图 1-27 所示：

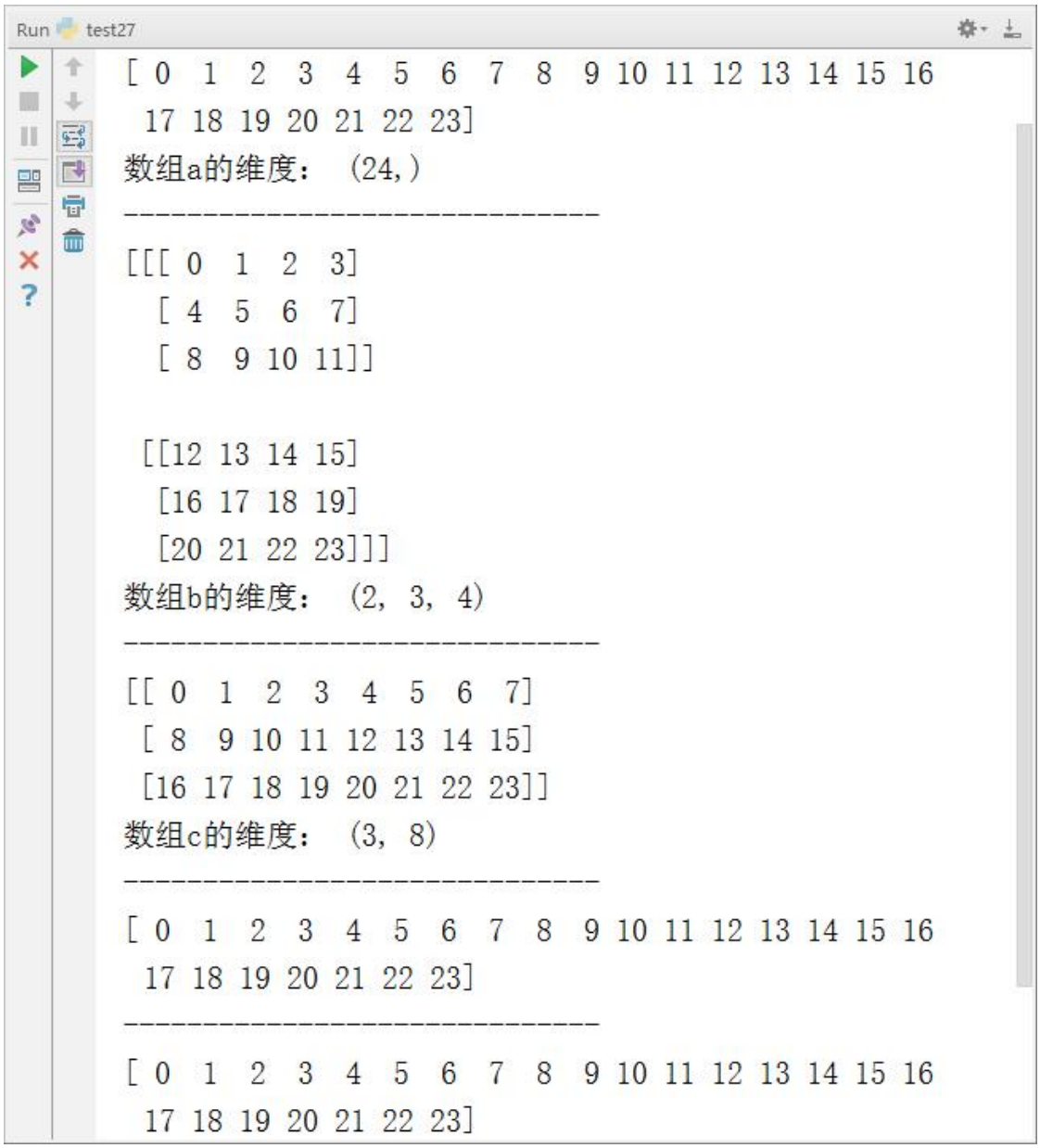
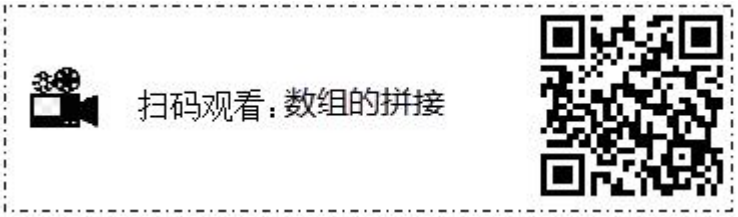


图 1-27 示例 1-27 运行效果图

1.5 数组的拼接

1.5.1 水平数组组合



通过 hstack 函数可以将两个或多个数组水平组合起来形成一个数组，那么什么叫做数组的水平组合。现在有两个 2*3 的数组 A 和 B。

数组 A

0 1 2

3　4　5

数组 B

6　7　8

9　10　11

使用 hstack 函数将两个数组水平组合的代码：hstack(A,B)。返回的结果：

0　1　2　6　7　8

3　4　5　9　10　11

可以看到，数组 A 和数组 B 在水平方向首尾连接了起来，形成了一个新的数组。这就是数组的水平组合。多个数组进行水平组合的效果类似。但数组水平组合必须要满足一个条件，就是所有参与水平组合的数组的行数必须相同，否则进行水平组合会抛出异常。

1.5.2 垂直数组组合

通过 vstack 函数可以将两个或多个数组垂直组合起来形成一个数组，那么什么叫数组的垂直组合呢？现在以两个 2*3 的数组 A 和 B 为例。

数组 A

0　1　2

3　4　5

数组 B

6　7　8

9　10　11

使用 vstack 函数将两个数组垂直组合代码：vstack(A,B)。运行的结果是：

0　1　2

3　4　5

6　7　8

9　10　11

表 1-8 数组的拼接

函数	描述
concatenate	连接沿现有轴的数组序列
hstack	水平堆叠序列中的数组（列方向）
vstack	竖直堆叠序列中的数组（行方向）

numpy.concatenate 函数用于沿指定轴连接相同形状的两个或多个数组，格式如下：

```
numpy.concatenate((a1, a2, ...), axis)
```

其中参数 a1, a2, ...指相同类型的数组；axis 指沿着它连接数组的轴，默认为 0。

【示例 1-28】concatenate()函数实现数组的拼接

```
a=np.array([[1,2,3],[4,5,6]])
print(a)
b=np.array([['a','b','c'],['d','e','f']])
print(b)
print(np.concatenate([a,b]))
print('垂直方向拼接 相当于 vstack')
print(np.concatenate([a,b],axis=0))
print('水平方向拼接 相当于 hstack')
print(np.concatenate([a,b],axis=1))
```

执行结果如图 1-28 所示：

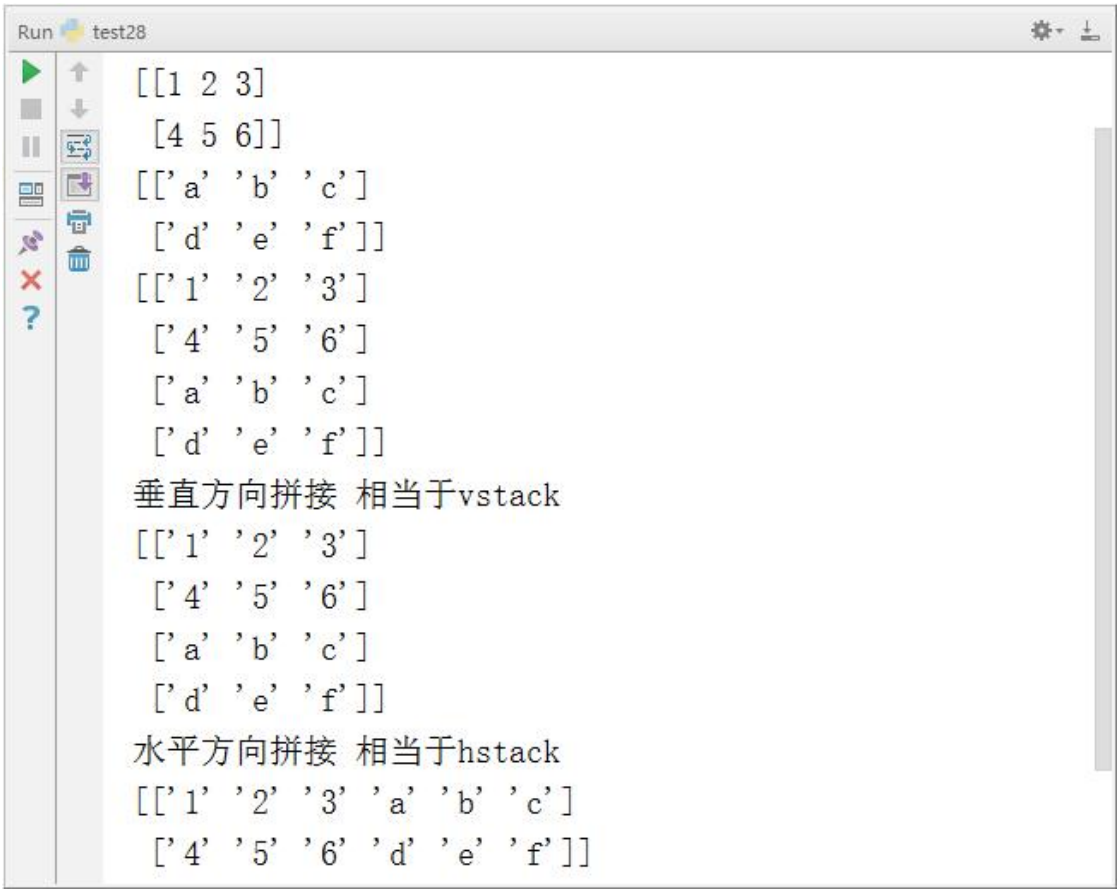


图 1-28 示例 1-28 运行效果图

numpy.hstack 它通过水平堆叠来生成数组。

numpy.vstack 它通过垂直堆叠来生成数组。

【示例 1-29】vstack() 与 hstack()实现二维数组的拼接

```
import numpy as np
```

```
a=np.array([[1,2,3],[4,5,6]])
print(a)
b=np.array([['a','b','c'],['d','e','f']])
print(b)
print('x 轴方向及垂直堆叠')
print(np.vstack([a,b]))
print('y 轴方向及水平堆叠')
print(np.hstack([a,b]))
```

执行结果如图 1-29 所示：

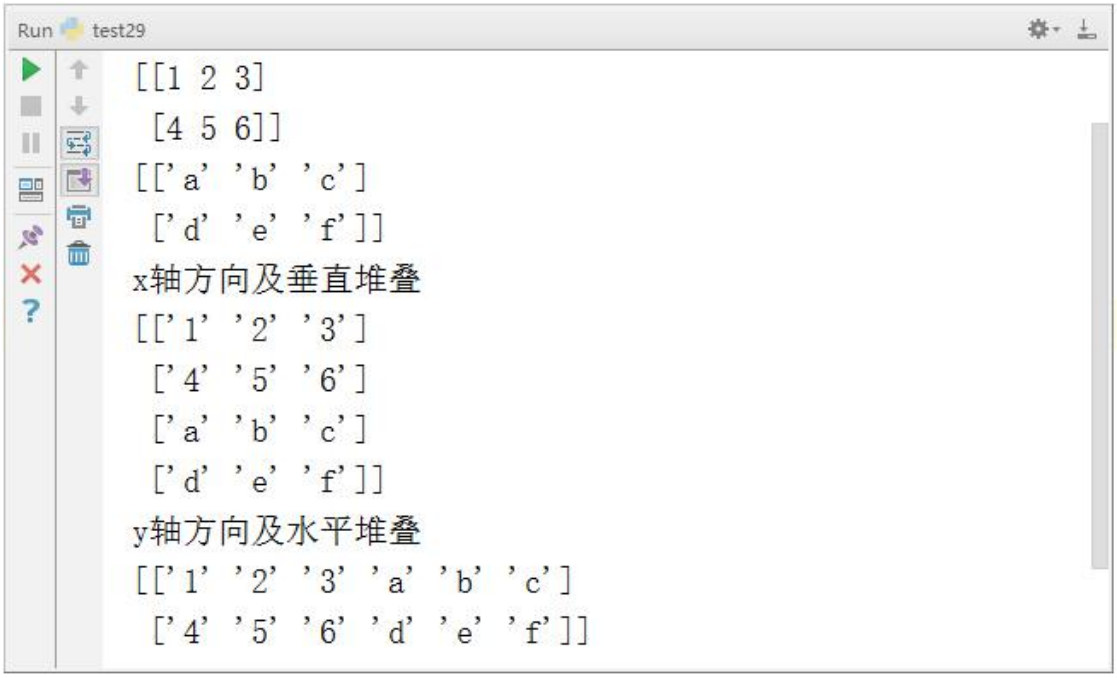


图 1-29 示例 1-29 运行效果图

注意：

- 如果拼接的行和列数目不同，则会报错。

【示例 1-30】三维数组的拼接

```
import numpy as np
aa=np.arange(1,37).reshape(3,4,3)
print('数组 aa.shape：',aa.shape)
bb=np.arange(101,137).reshape(3,4,3)
print('数组 bb.shape：',bb.shape)
print('concatenate((aa,bb),axis=0)的结果：')
```

```
print(np.concatenate((aa,bb),axis=0).shape)    #6 4 3
print('concatenate((aa,bb),axis=1)的结果：')
print(np.concatenate((aa,bb),axis=1).shape)    #3,8,3
print('concatenate((aa,bb),axis=2)的结果：')
print(np.concatenate((aa,bb),axis=2).shape)    #3,4,6
```

执行结果如图 1-30 所示：

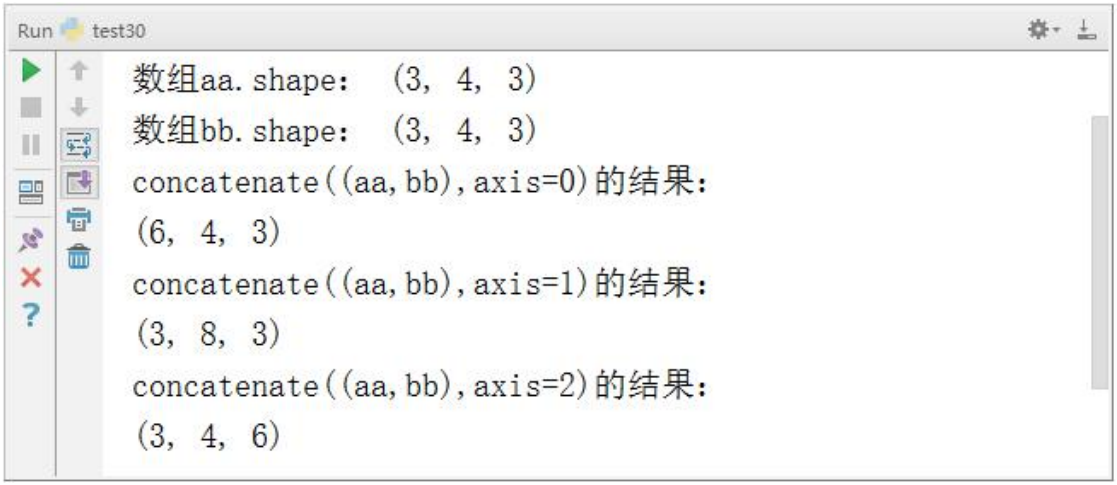
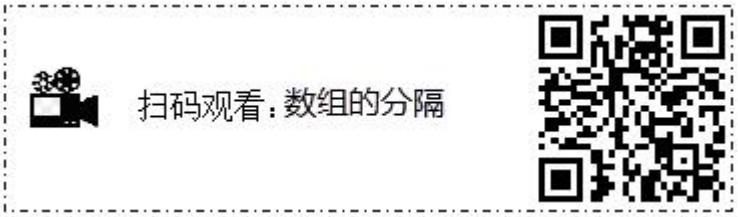


图 1-30 示例 1-30 运行效果图

1.6 数组的分隔



1.6.1 split 分隔

numpy.split 函数沿特定的轴将数组分割为子数组，格式如下：

```
numpy.split(ary, indices_or_sections, axis)
```

参数说明：

arry：被分割的数组。

indices_or_sections：如果是一个整数，就用该数平均切分，如果是一个数组，为沿轴切分的位置。

axis：沿着哪个维度进行切分，默认为 0，横向切分。为 1 时，纵向切分。

【示例 1-31】split()函数分隔一维数组

```
import numpy as np
x=np.arange(1,9)
a=np.split(x,4)
print(a)
print(a[0])
```

```
print(a[1])
print(a[2])
print(a[3])
#传递数组进行分隔
b=np.split(x,[3,5])
print(b)
```

执行结果如图 1-31 所示：

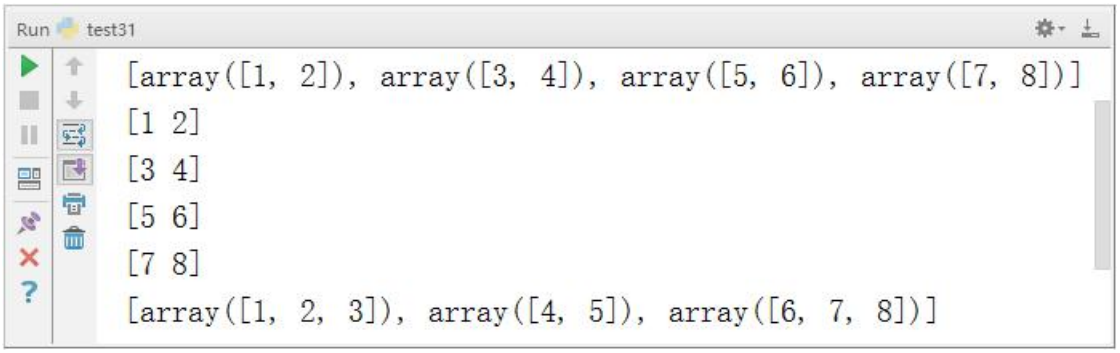


图 1-31 示例 1-31 运行效果图

【示例 1-32】split()函数分隔二维数组

```
#导入 numpy
import numpy as np
#创建两个数组
a=np.array([[1,2,3],[4,5,6],[11,12,13],[14,15,16]])
print('axis=0 垂直方向 平均分隔')
r=np.split(a,2,axis=0)
print(r[0])
print(r[1])
print('axis=1 水平方向 按位置分隔')
r=np.split(a,[2],axis=1)
print(r)
```

执行结果如图 1-32 所示：

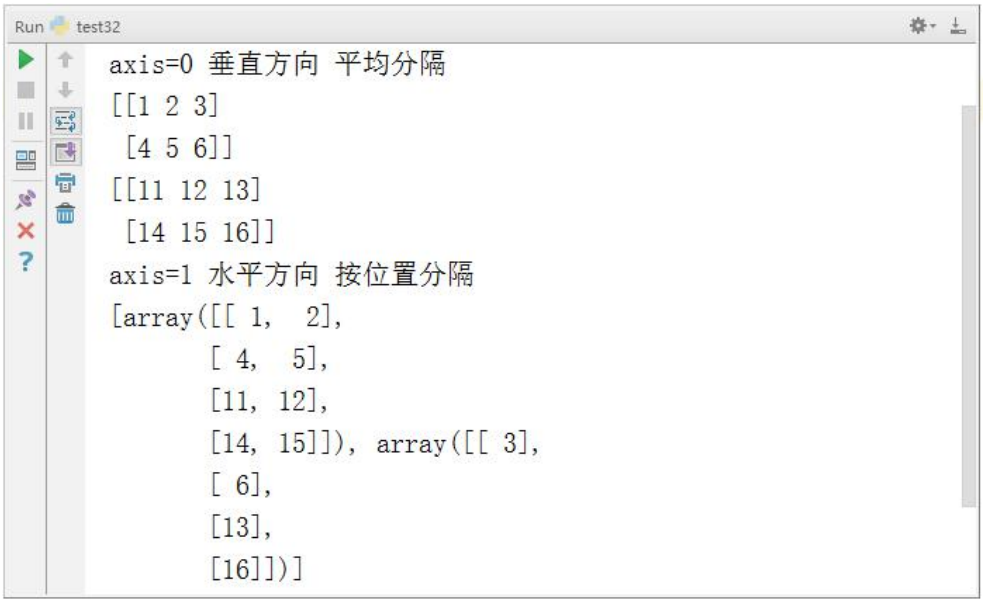


图 1-32 示例 1-32 运行效果图

1.6.2 水平分隔

分隔数组是组合数组的逆过程，与组合数组一样，分隔数组也分为水平分隔数组和垂直分隔数组。水平分隔数组与水平组合数组对应。水平组合数组是将两个或多个数组水平进行收尾相接，而水平分隔数组是将已经水平组合到一起的数组再分开。

使用 hsplit 函数可以水平分隔数组，该函数有两个参数，第 1 个参数表示待分隔的数组，第 2 个参数表示要将数组水平分隔成几个小数组，现在先来看一个例子。下面是一个 2*6 的二维数组 X。

数组 X

1 2 3 4 5 6

7 8 9 10 11 12

使用如下代码对 X 数组进行分隔

```
np.hsplit(X,2)
```

分隔的结果如下：

1 2 3 4 5 6

7 8 9 10 11 12

很明显，将数组 X 分隔成了列数相同的两个数组。现在使用下面的代码重新对数组 X 进行分隔。

```
np.hsplit(X,3)
```

分隔的结果如下：

1 2 3 4 5 6

7 8 9 10 11 12

现在讲数组 X 分隔成了 3 个列数都为 2 的数组，但要是使用 hsplit(X,4)分隔数组 X 就

会抛出异常，这是因为数组 X 是没有办法被分隔成列数相同的 4 个数组的，所以使用 hsplit 函数分隔数组的一个规则就是第 2 个参数值必须可以整除待分隔数组的列数。

【示例 1-33】hsplit()的使用

```
import numpy as np
grid=np.arange(16).reshape(4,4)
a,b=np.hsplit(grid,2)
print(a)
print(b)
```

执行结果如图 1-33 所示：



图 1-33 示例 1-33 运行效果图

1.6.3 垂直分隔数组

垂直分隔数组是垂直组合数组的逆过程。垂直组合数组是将两个或多个数组垂直进行首尾相接，而垂直分隔数组是将已经垂直组合到一起的数组再分开。

使用 vsplit 函数可以垂直分隔数组，该函数有两个参数，第 1 个参数表示待分隔的数组，第 2 个参数表示将数组垂直分隔成几个小数组。示例如下，一个 4*3 的二维数组 X。

数组 X

1 2 3

4 5 6

7 8 9

10 11 12

现在使用如下的代码对数组 X 进行分隔：

vsplit(X,2)

分隔后的结果如下：

1 2 3

4　5　6

7　8　9

10　11　12

很明显，将数组 X 分隔成了行数相同的两个数组。

【示例 1-34】vsplit()垂直分隔数组

```
import numpy as np
arr=np.arange(16).reshape(4,4)
a,b=np.vsplit(arr,[3])
print('vsplit(arr,[3])的结果：')
print(a)
print(b)
print('vsplit(arr,[1,3])的结果：')
a,b,c=np.vsplit(arr,[1,3])
print(a)
print(b)
print(c)
```

执行结果如图 1-34 所示：

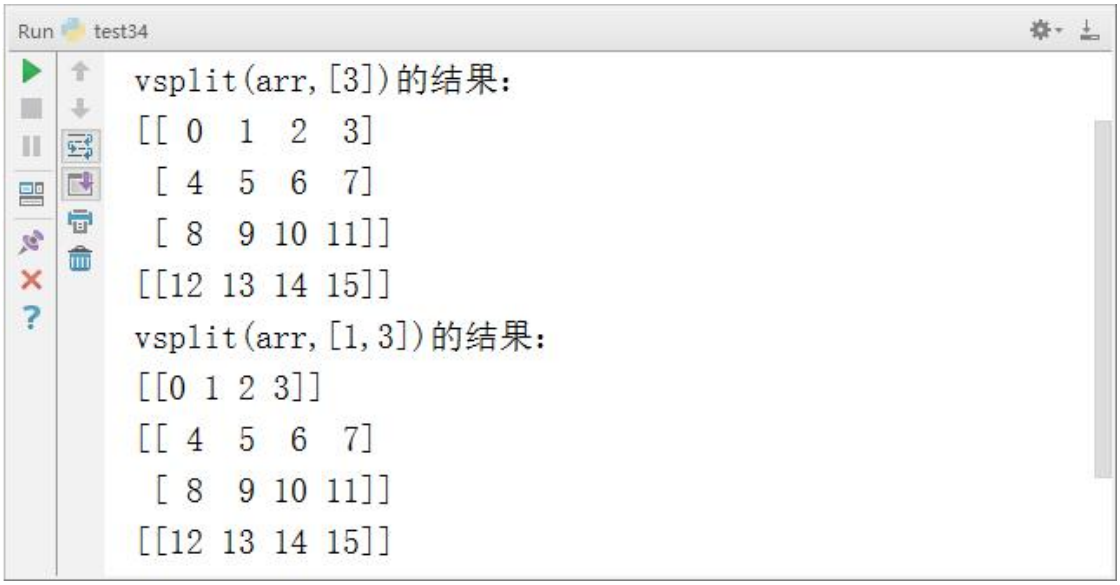


图 1-34 示例 1-34 运行效果图

【示例 1-35】transpose()函数的使用

```
#transpose 进行转置
#二维转置
import numpy as np
a=np.arange(1,13).reshape(2,6)
print('原数组 a')
print(a)
print('转置后的数组')
print(a.transpose())

#多维数组转置
aaa=np.arange(1,37).reshape(1,3,3,4)
#将 1,3,3,4 转换为 3,3,4,1
print(np.transpose(aaa,[1,2,3,0]).shape)
```

执行结果如图 1-35 所示：



图 1-35 示例 1-35 运行效果图

1.7 算术函数



如果参与运算的两个对象 都是 ndarray，并且形状相同，那么会对位彼此之间进

行（+ - * /）运算。NumPy 算术函数包含简单的加减乘除: add()，subtract()，multiply() 和 divide()。

【示例 1-36】算术函数的使用

```
import numpy as np
a=np.arange(9,dtype=np.float).reshape(3,3)
b=np.array([10,10,10])
print('两数组进行加法运算 add 或+的结果：')
print(np.add(a,b))
print('两数组进行减法运算 substract 或-的结果：')
print(a-b)
print('两数组进行乘法运算 multiply 或*的结果：')
print(np.multiply(a,b))
print('两数组进行除法运算 divide 或/的结果：')
print(np.divide(a,b))
```

执行结果如图 1-36 所示：

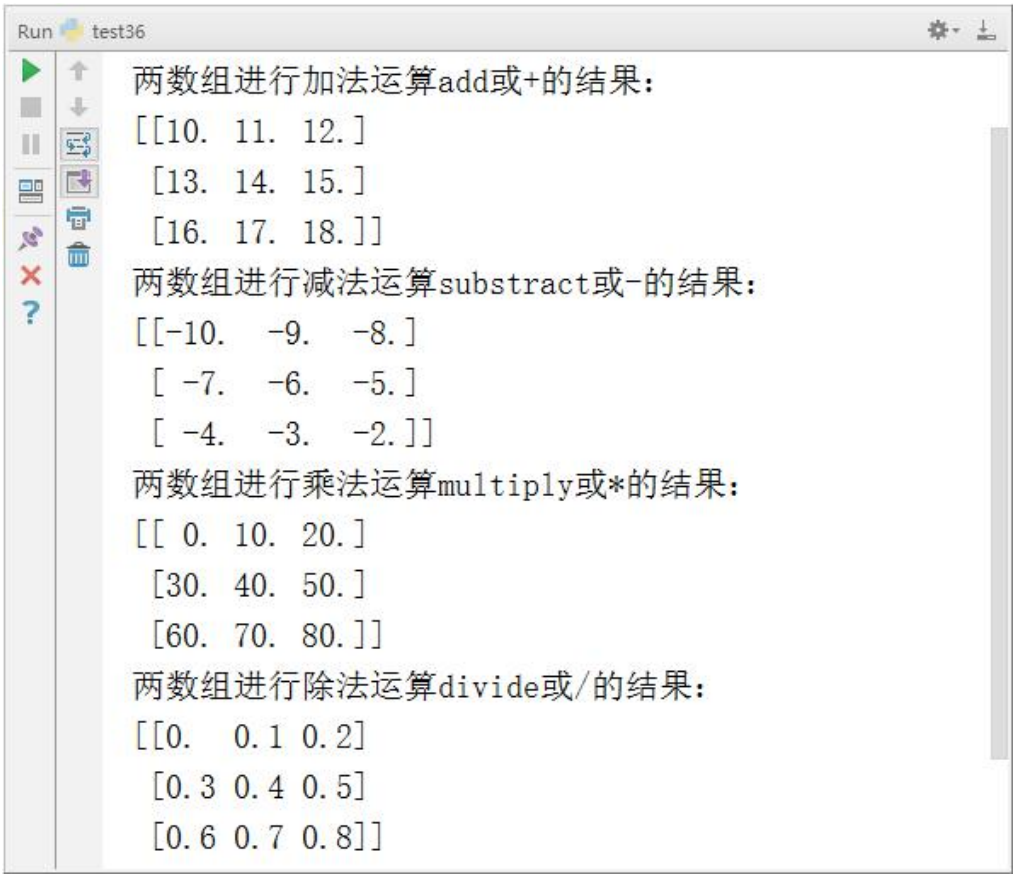


图 1-36 示例 1-36 运行效果图

【示例 1-37】函数指定输出结果的用法

```
# 通用函数指定输出结果的用法
import numpy as np
x = np.arange(5)
y = np.empty(5)
np.multiply(x, 10, out=y)
print(y)
```

执行结果如图 1-37 所示：

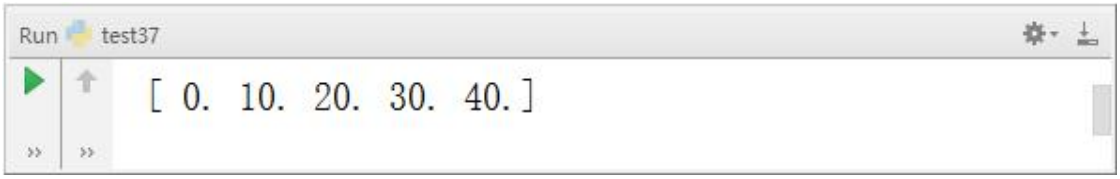


图 1-37 示例 1-37 运行效果图

1.8 数学函数

NumPy 提供了标准的三角函数：sin()、cos()、tan()。

【示例 1-38】三角函数的使用

```
a = np.array([0,30,45,60,90])
print ('不同角度的正弦值：')
# 通过乘 pi/180 转化为弧度
print (np.sin(a*np.pi/180))
print ('数组中角度的余弦值：')
print (np.cos(a*np.pi/180))
print ('数组中角度的正切值：')
print (np.tan(a*np.pi/180))
```

执行结果如图 1-38 所示：



图 1-38 示例 1-38 运行效果图

numpy.around() 函数返回指定数字的四舍五入值。

```
numpy.around(a,decimals)
```

参数说明：

a: 数组。

decimals: 舍入的小数位数。 默认值为 0。 如果为负，整数将四舍五入到小数点左侧的位置。

numpy.floor() 返回数字的下舍整数。

numpy.ceil() 返回数字的上入整数。

【示例 1-39】around()、floor()、ceil()函数的使用

```
import numpy as np
a = np.array([1.0,4.55,  123,  0.567,  25.532])
print  ('原数组：')
print (a)
print ('round 舍入后：')
print (np.around(a))
print (np.around(a, decimals =  1))
print (np.around(a, decimals =  -1))

print('floor 向下取整：')
print(np.floor(a))
print('ceil 向上取整：')
print(np.ceil(a))
```

执行结果如图 1-39 所示：

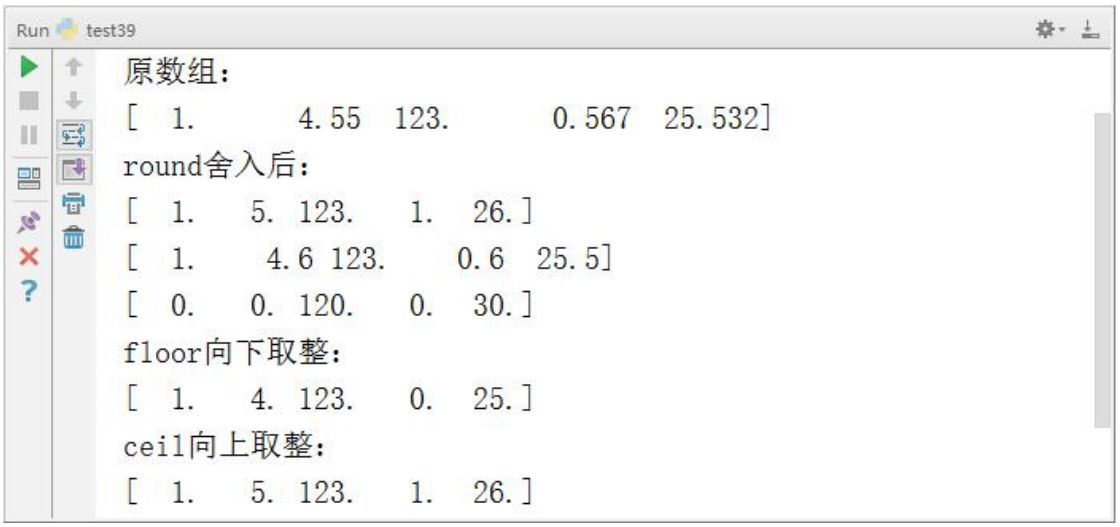


图 1-39 示例 1-39 运行效果图

1.9 聚合函数

NumPy 提供了很多统计函数，用于从数组中查找最小元素，最大元素，百分位标准差和方差等。 具体如表 1-9 所示：

表 1-9 聚合函数

函数名	说明
np.sum()	求和
np.prod()	所有元素相乘
np.mean()	平均值
np.std()	标准差
np.var()	方差
np.median()	中数
np.power()	幂运算
np.sqrt()	开方
np.min()	最小值
np.max()	最大值
np.argmin()	最小值的下标
np.argmax()	最大值的下标
np.inf	无穷大
np.exp(10)	以 e 为底的指数
np.log(10)	对数

numpy.power() 函数将第一个输入数组中的元素作为底数，计算它与第二个输入数组中相应元素的幂。

【示例 1-40】power()函数的使用

```
import numpy as np
a=np.arange(12).reshape(3,4)
print('原来的数组')
print(a)
print('power(a,2)的结果：')
print(np.power(a,2))
```

执行结果如图 1-40 所示：

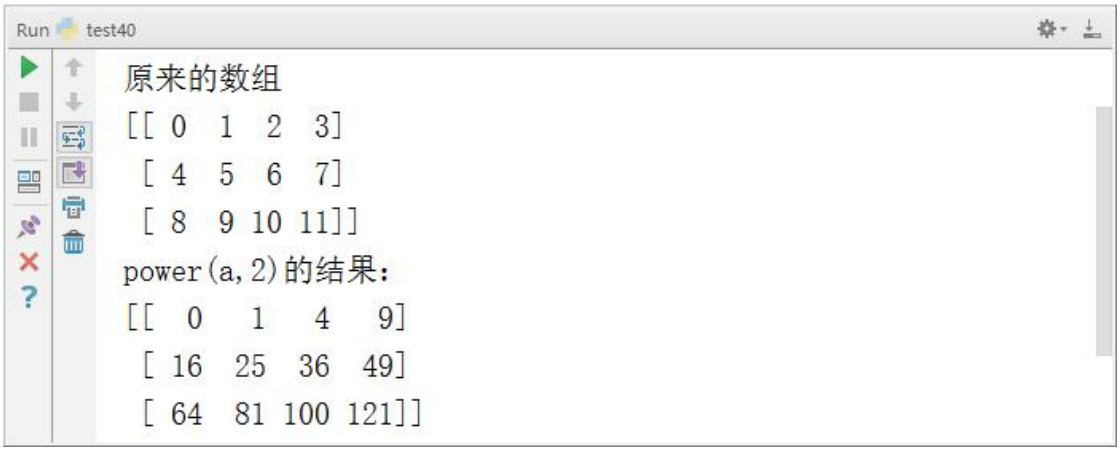


图 1-40 示例 1-40 运行效果图

【示例 1-41】median ()函数的使用一

```
a=np.array([4,2,1,5])
#计算偶数的中位数
print('偶数的中位数：',np.median(a))
a=np.array([4,2,1])
print('奇数个的中位数：',np.median(a))

a=np.arange(1,16).reshape(3,5)
print('原来的数组')
print(a)
print('调用 median 函数')
print(np.median(a))
print('调用 median 函数，axis=1 行的中值')
print(np.median(a,axis=1))
print('调用 median 函数，axis=0 列的中值')
print(np.median(a,axis=0))
```

执行结果如图 1-41 所示：

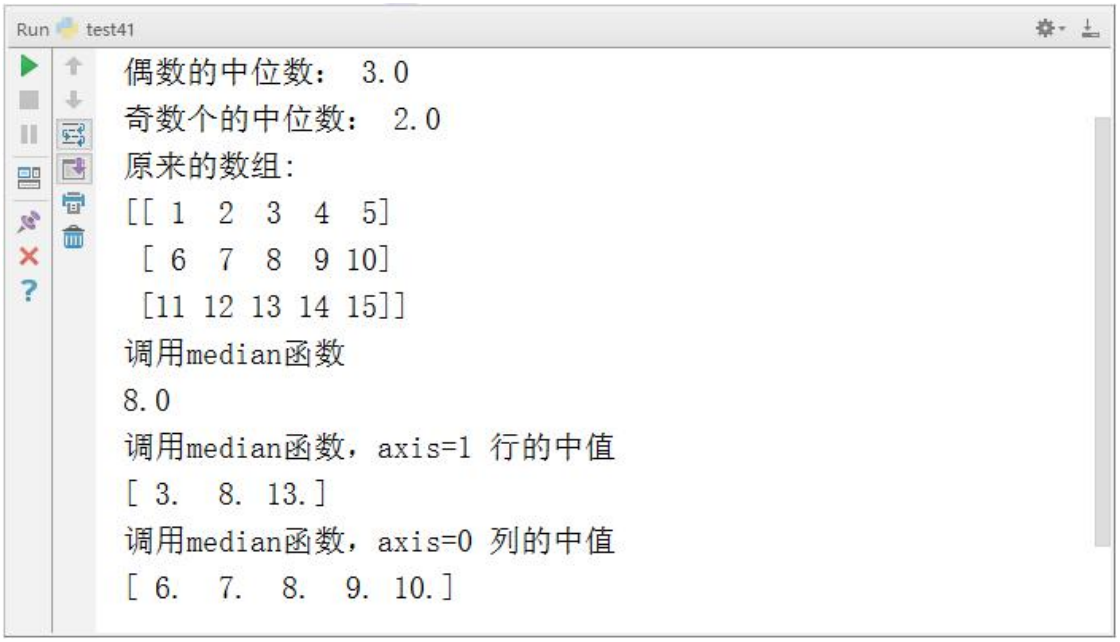


图 1-41 示例 1-41 运行效果图

numpy.mean() 函数返回数组中元素的算术平均值。 如果提供了轴，则沿其计算。算术平均值是沿轴的元素的总和除以元素的数量。

【示例 1-42】mean ()函数的使用二

```
a=np.arange(1,11).reshape(2,5)
print('原来的数组')
print(a)
print('调用 mean 函数')
print(np.mean(a))
print('调用 mean 函数  axis=0  列')
print(np.mean(a,axis=0))
print('调用 mean 函数  axis=1  行')
print(np.mean(a,axis=1))
```

执行结果如图 1-42 所示：

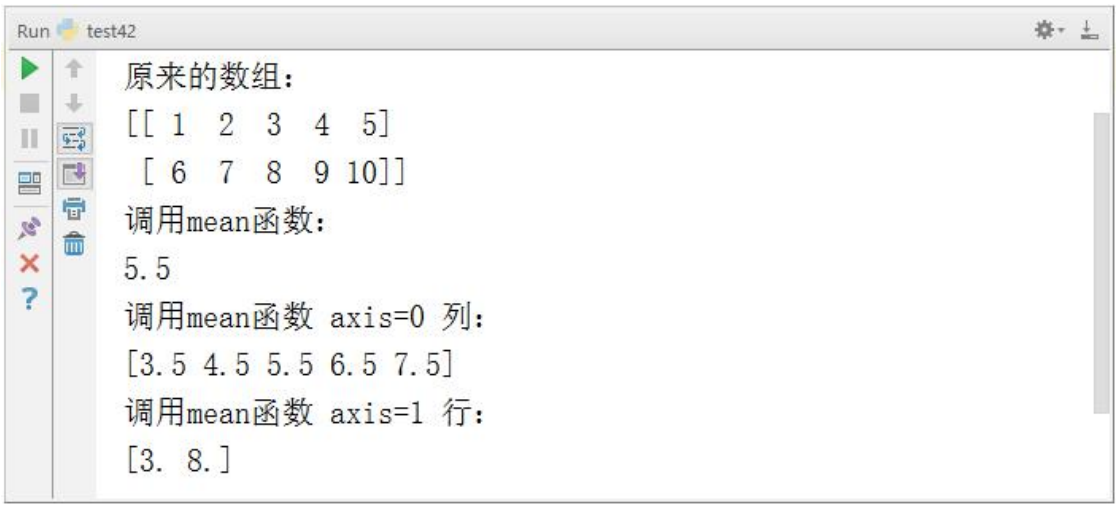


图 1-42 示例 1-42 运行效果图

习题

一、选择题

1. 计算 numpy 中元素个数的方法（）。

A. np.sqrt()

B. np.size()

C. np.identity()

D. np.len()

2. 有数组 n = np.arange(24).reshape(2,-1,2,2)，n.shape 的返回结果是什么？

A. (2,3,2,2)

B. (2,2,2,2)

C. (2,4,2,2)

D. (2,6,2,2)

3. numpy 中创建全为 0 的矩阵使用（）

A. zeros

B. ones

C. empty

D. arange

4. numpy 中向量转成矩阵使用（）

A. reshape

B. reval

C. arange

D. random

5. 已知 c=np.arange(24).reshape(3,4,2)那么 c.sum(axis=0)所的结果为（）

A. array([[12,16],[44,48],[76,80]])

B. array([[1,5,9,13],[17,21,25,29],[33,37,41,45]])

C. array([[24 27],[30 33],[36 39],[42 45]])

二、解答题

1. 简述安装 numpy 的方式。
2. 数组的创建方式。
3. numpy 模块中常用的数学函数。
4. ndarray 对象常用属性。
5. 如何实现数组的分隔，举例说明。

三、编码题

1. 创建一个长度为 10 的一维全为 0 的 ndarray 对象，然后让第 5 个元素等于 1。
2. 创建一个元素为从 10 到 49 的 ndarray 对象

3. 创建一个 3×3 的二维数组，值域为 0 到 8。
4. 从数组 np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])中提取所有的奇数。
5. 将一维数组 np.arange(10)转换为 2 行的二维数组。
6. 反转二维数组 np.arange(9).reshape(3,3)的行。

第二章 Matplotlib

为了将数据变成所有人都喜欢的图形，就需要使用本章要介绍的数据可视化库Matplotlib。当然，还有很多类似的程序库。但 Matplotlib 的功能更强大，而且可以很容易与Numpy、Pandas 等程序库结合在一起使用。

Matplotlib 是一个 Python 的 2D 绘图库。通过 Matplotlib，开发者可以仅需要几行代码，便可以生成绘图，直方图，功率谱，条形图，错误图，散点图等。学习 Matplotlib，可让数据可视化，更直观的真实给用户。使数据更加客观、更具有说服力。 Matplotlib 是 Python 的库，又是开发中常用的库。

通过阅读本章，你可以：

- 了解什么是 Matplotlib
- 掌握如何用 Matplotlib 绘制各种图形（柱状图、饼状图、直方图等）
- 掌握如何定制图形的颜色和样式
- 掌握如何用 Matplotlib 绘制三维图

2.1 matplotlib 开发环境搭建



如果使用的是 Anaconda Python 开发环境，那么 Matplotlib 已经被集成进 Anaconda，并不需要单独安装。如果使用的是标准的 Python 开发环境，可以使用下面的命令安装 Matplotlib，语法格式如下：

```
pip install matplotlib
```

如果要了解 Matplotlib 更详细的情况，请访问官方网站。网址如下：https://matplotlib.org。

安装完 Matplotlib 后，可以测试一下 Matplotlib 是否安装成功。进入 Python 的环境使用下面的语句导入 matplotlib.pyplot 模块。如果不出错，就说明 Matplotlib 已经安装成功了。

```
import matplotlib.pyplot as plt
```

虽然上述的安装方式比较简单，但是有时候不能确保安装成功或者并不能保证安装的 Matplotlib 版本适合当今 Python 环境。在这个时候，建议读者登录 Python 官方网站 https://www.python.org/，点击菜单 PyPI 输入 Matplotlib 到下载页如图 2-1 所示，在这个页面中查找与你使用的 Python 版本匹配的 wheel 文件（扩展名为“.whl”的文件）。例如使用的是 64 位的 Python3.6，则需要下载 matplotlib-3.1.0-cp36-cp36m-win_amd64.whl。

Filename, size & hash	File type	Python version	Upload date
matplotlib-3.1.0-cp36-cp36m-macosx_10_6_intel.macosx_10_9_intel.macosx_10_9_x86_64.macosx_10_10_intel.macosx_10_10_x86_64.whl (14.4 MB) SHA256	Wheel	cp36	May 19, 2019
matplotlib-3.1.0-cp36-cp36m-manylinux1_x86_64.whl (13.1 MB) SHA256	Wheel	cp36	May 19, 2019
matplotlib-3.1.0-cp36-cp36m-win32.whl (8.9 MB) SHA256	Wheel	cp36	May 19, 2019
matplotlib-3.1.0-cp36-cp36m-win_amd64.whl (9.1 MB) SHA256	Wheel	cp36	May 19, 2019
matplotlib-3.1.0-cp37-cp37m-macosx_10_6_intel.macosx_10_9_intel.macosx_10_9_x86_64.macosx_10_10_intel.macosx_10_10_x86_64.whl (14.4 MB) SHA256	Wheel	cp37	May 19, 2019
matplotlib-3.1.0-cp37-cp37m-manylinux1_x86_64.whl (13.1 MB) SHA256	Wheel	cp37	May 19, 2019
matplotlib-3.1.0-cp37-cp37m-win32.whl (8.9 MB) SHA256	Wheel	cp37	May 19, 2019
matplotlib-3.1.0-cp37-cp37m-win_amd64.whl (9.1 MB) SHA256	Wheel	cp37	May 19, 2019
matplotlib-3.1.0.tar.gz (37.2 MB) SHA256	Source	None	May 19, 2019

图 2-1 matplotlib 的 wheel 文件下载

当读者下载得到的文件是 matplotlib-3.1.0-cp36-cp36m-win_amd64.whl，将该文件保存到某在目录下如：”E:/matp”。然后打开命令窗口，并切换到“e:/matp”目录下。执行如下命令安装 Matplotlib。

```
pip install matplotlib-3.1.0-cp36-cp36m-win_amd64.whl
```

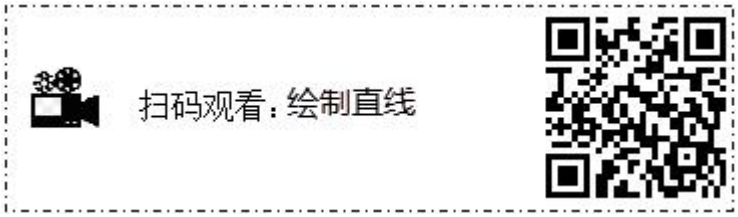
2.2 绘制基础

在使用 Matplotlib 绘制图形时，其中有两个最为常用的场景。一个是画点，一个是画线。pyplot 基本方法的使用如表 2-1。

表 2-1 pyplot 基本方法

方法名	说明
title()	设置图表的名称
xlabel()	设置 x 轴名称
ylabel()	设置 y 轴名称
xticks(x, ticks, rotation)	设置 x 轴的刻度, rotation 旋转角度
yticks()	设置 y 轴的刻度
plot()	绘制线性图表
show()	显示图表
legend()	显示图例
text(x, y, text)	显示每条数据的值　x, y 值的位置
figure(name, figsize=(w, h), dpi=n)	设置图片大小

2.2.1 绘制直线



在使用 Matplotlib 绘制线性图时，其中最简单的是绘制线图。在下面的实例代码中，使用 Matplotlib 绘制了一个简单的直线。具体实现过程如下：

（1） 导入模块 pyplot，并给它指定别名 plt，以免反复输入 pyplot。在模块 pyplot 中包含很多用于生产图表的函数。

（2） 将绘制的直线坐标传递给函数 plot()。

（3） 通过函数 plt.show()打开 Matplotlib 查看器，显示绘制的图形。

【示例 2-1】使用 matplotlib 根据两点绘制一条线

```
import matplotlib.pyplot as plt
#将(0,1)点和(2,4)连起来
plt.plot([0,2],[1,4])
plt.show()
```

运行结果如图 2-2 所示：

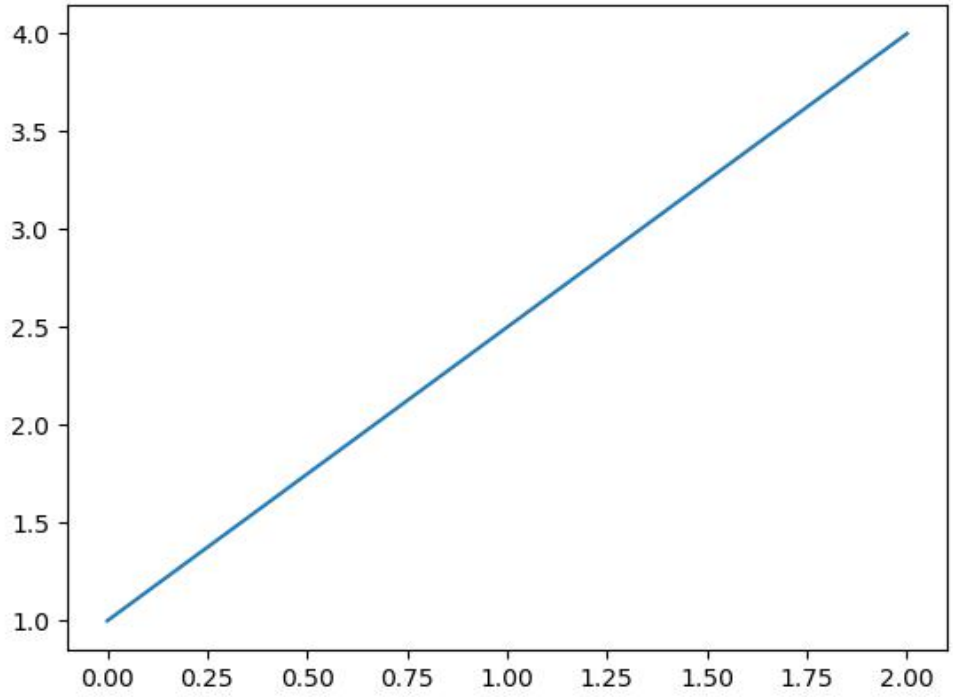


图 2-2 示例 2-1 运行效果图

在上述的实例代码中，使用两个坐标绘制一条直线，接下来使用平方数序列 1、4、9、16 和 25 来绘制一个折线图。

2.2.2 绘制折线

【示例 2-2】使用 matplotlib 绘制折线图

```
import matplotlib.pyplot as plt
x=[1,2,3,4,5]
squares=[1,4,9,16,25]
plt.plot(x,squares)
plt.show()
```

执行结果如图 2-3 所示：

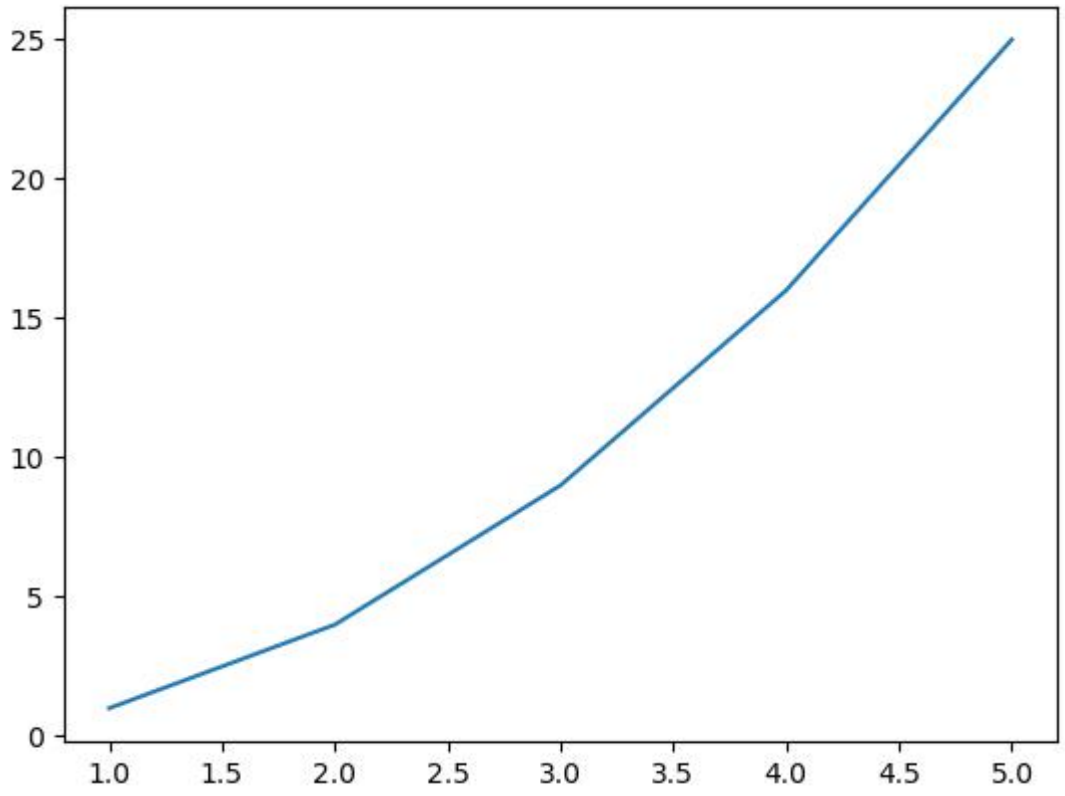


图 2-3 示例 2-2 运行效果图

2.2.3 设置标签文字和线条粗细

在上面的实例直线结果不够完美，开发者可以绘制的线条样式进行灵活设置。例如：可以设置线条的粗细、设置文字等。



【示例 2-3】使用 matplotlib 绘制折线图并设置样式

```
import matplotlib.pyplot as plt
datas=[1,2,3,4,5]
squares=[1,4,9,16,25]
plt.plot(datas,squares,linewidth=5) #设置线条宽度
#设置图标标题，并在坐标轴上添加标签
plt.title('Numbers',fontsize=24)
plt.xlabel('datas',fontsize=14)
plt.ylabel('squares',fontsize=14)
plt.show()
```

执行结果如图 2-4 所示：

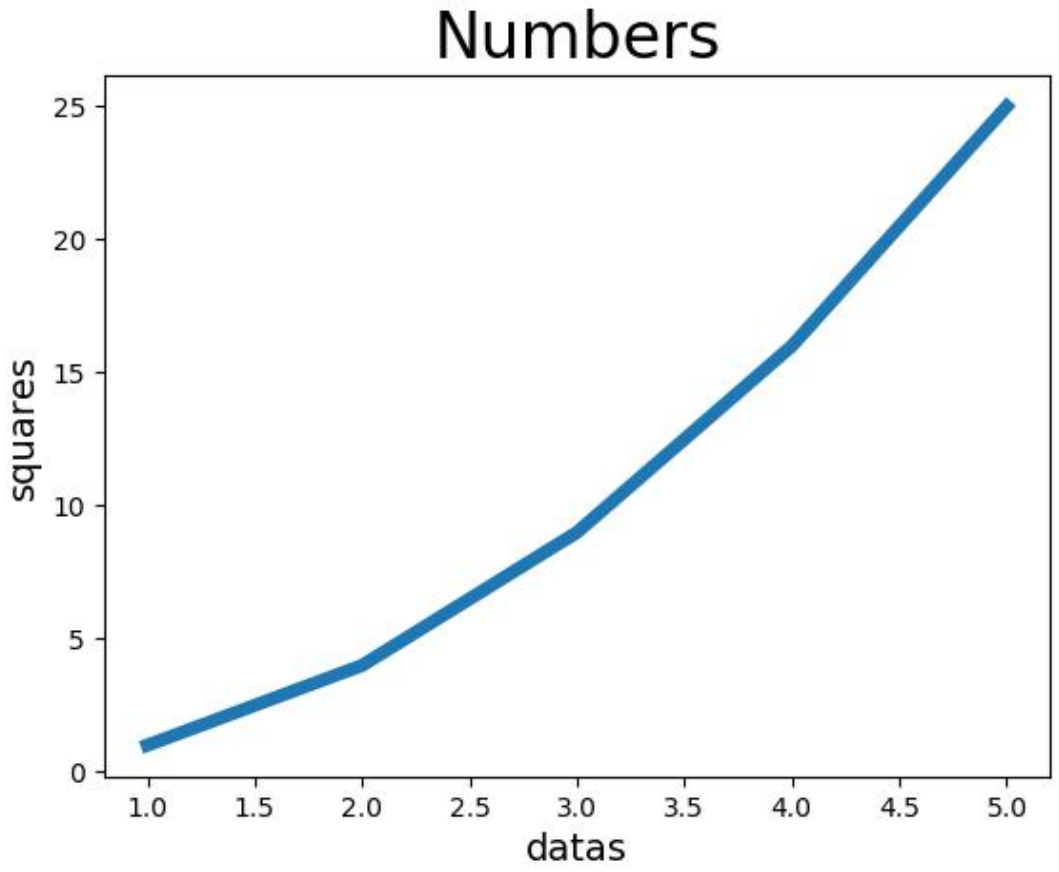


图 2-4 示例 2-3 运行效果图

Matplotlib 默认情况不支持中文，可以使用以下简单的方法来解决：

```
plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签
```

【示例 2-4】解决标签、标题中的中文问题

```
import matplotlib.pyplot as plt
```

```
datas=[1,2,3,4,5]
squares=[1,4,9,16,25]
plt.plot(datas,squares,linewidth=5) #设置线条宽度
#设置中文乱码问题
plt.rcParams['font.sans-serif'] = ['SimHei']
#设置图标标题，并在坐标轴上添加标签
plt.title('标题设置',fontsize=24)
plt.xlabel('x 轴',fontsize=14)
plt.ylabel('y 轴',fontsize=14)
plt.show()
```

执行结果如图 2-5 所示：

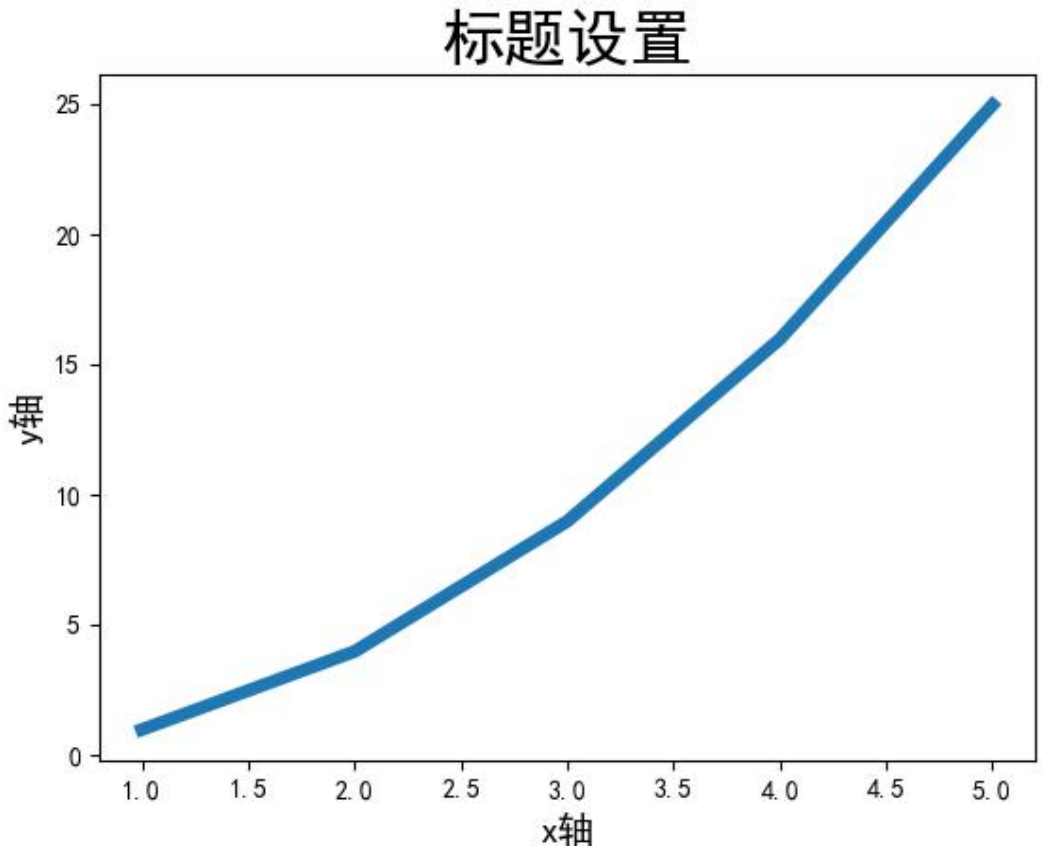


图 2-5 示例 2-4 运行效果图

2.2.4 绘制一元二次方程的曲线 y=x^2

Matplotlib 有很多函数用于绘制各种图形，其中 plot 函数用于曲线，需要将 200 个点的 x 坐标和 Y 坐标分别以序列的形式传入 plot 函数，然后调用 show 函数显示绘制的图形。一元二次方程的曲线。

【示例 2-5】使用 matplotlib 绘制一元二次方程曲线

```
import matplotlib.pyplot as plt
#200 个点的 x 坐标
x=range(-100,100)
#生成 y 点的坐标
y=[i**2 for i in x ]
#绘制一元二次曲线
plt.plot(x,y)
#调用 savefig 将一元二次曲线保存为 result.jpg
plt.savefig('result.jpg') #如果直接写成  plt.savefig('cos')  会生成 cos.png
plt.show()
```

执行结果如图 2-5 所示：

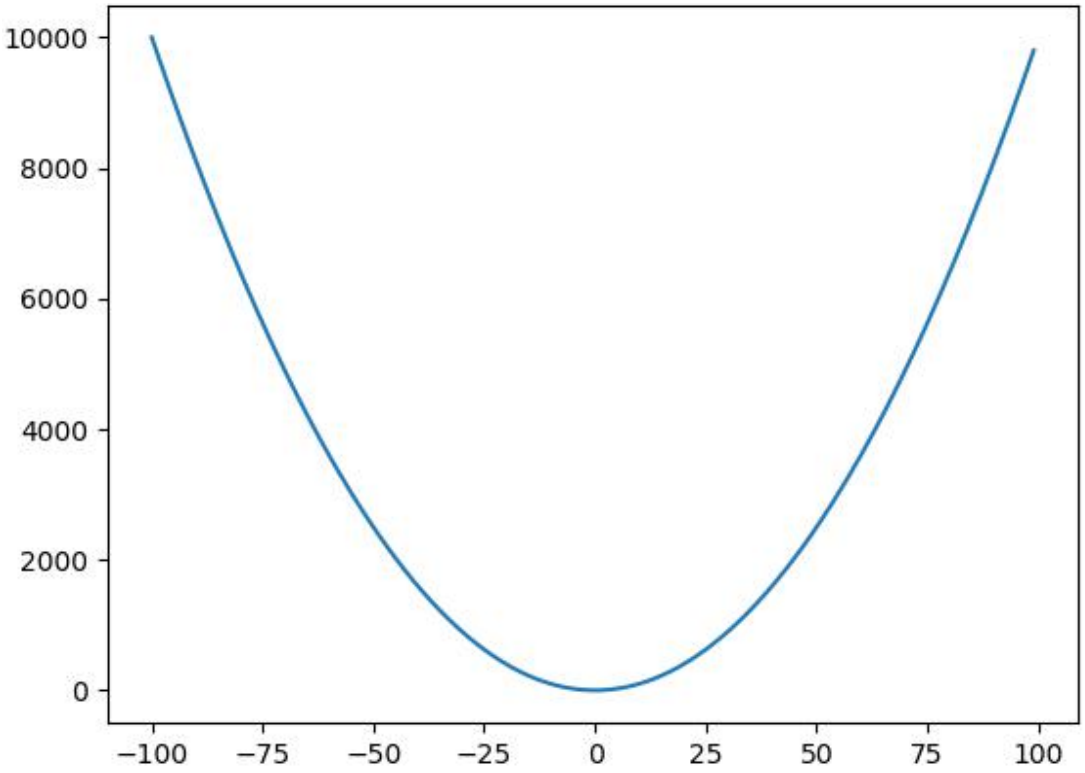


图 2-6 示例 2-5 运行效果图

2.2.5 绘制正弦曲线和余弦曲线

使用 plt 函数绘制任何曲线的第一步都是生成若干个坐标点（x,y），理论上坐标点是越多越好。本例取 0 到 10 之间 100 个等差数作为 x 的坐标，然后将这 100 个 x 坐标值一起传入 Numpy 的 sin 和 cos 函数，就会得到 100 个 y 坐标值，最后就可以使用 plot 函数绘制正弦曲线和余弦曲线。

【示例 2-6】使用 matplotlib 绘制正弦曲线和余弦曲线

```
import matplotlib.pyplot as plt
import numpy as np
#生成 x 的坐标（0-10 的 100 个等差数列）
x=np.linspace(0,10,100)
sin_y=np.sin(x)
#绘制正弦曲线
plt.plot(x,sin_y)
#绘制余弦曲线
cos_y=np.cos(x)
plt.plot(x,cos_y)
plt.show()
```

运行结果如图 2-7 所示：

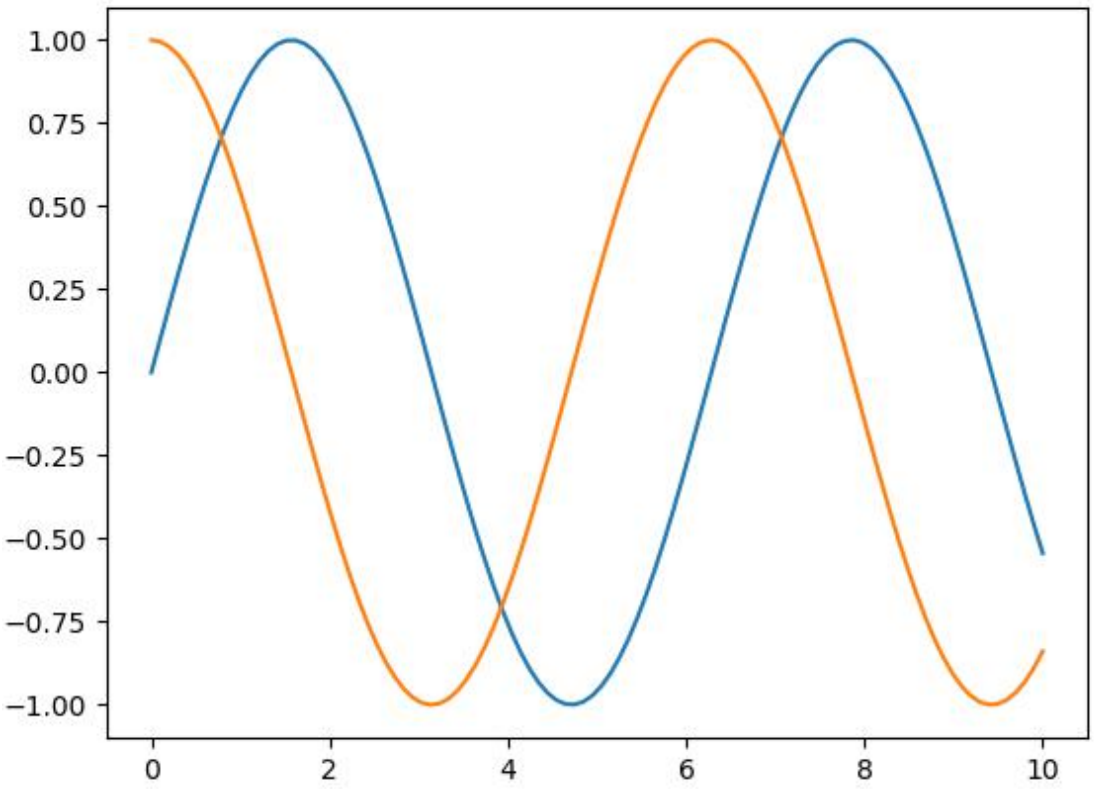


图 2-7 示例 2-6 运行效果图

上面的示例可以看到，调用两次 plot 函数，会将 sin 和 cos 曲线绘制到同一个二维坐标系中，如果想绘制到两张画布中，可以调用 subplot()函数将画布分区。

【示例 2-7】subplot()函数将画布分区的使用

```
import matplotlib.pyplot as plt
import numpy as np
#将画布分为区域，将图画到画布的指定区域
x=np.linspace(1,10,100)
#将画布分为 2 行 2 列，将图画到画布的 1 区域
plt.subplot(2,2,1)
plt.plot(x,np.sin(x))
plt.subplot(2,2,3)
plt.plot(x,np.cos(x))
plt.show()
```

运行结果如图 2-8 所示：

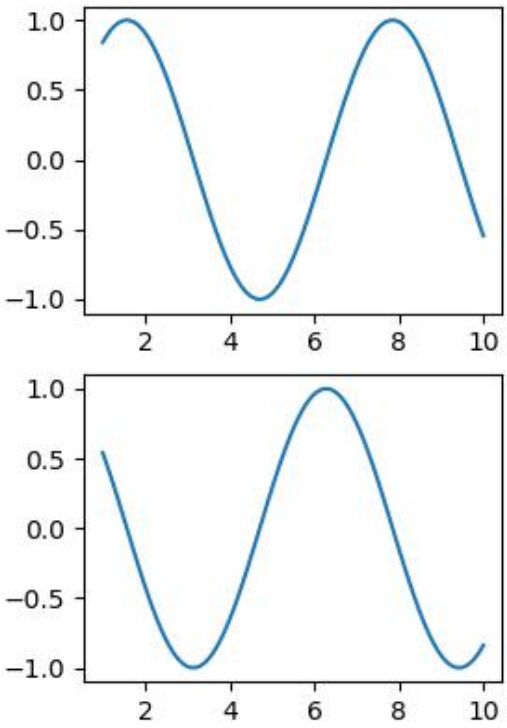
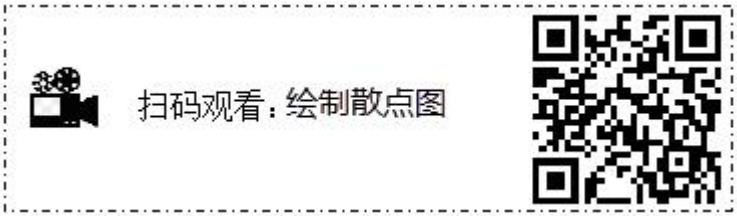


图 2-8 示例 2-7 运行效果图

2.3 绘制散点图



使用 scatter 函数可以绘制随机点，该函数需要接收 x 坐标和 y 坐标的序列。

【示例 2-8】使用 matplotlib 绘制 sin()函数的散点图

```
import matplotlib.pyplot as plt
import numpy as np
#画散点图
x=np.linspace(0,10,100)#生成 0 到 10 中 100 个等差数
plt.scatter(x,np.sin(x))
plt.show()
```

运行结果如图 2-9 所示：

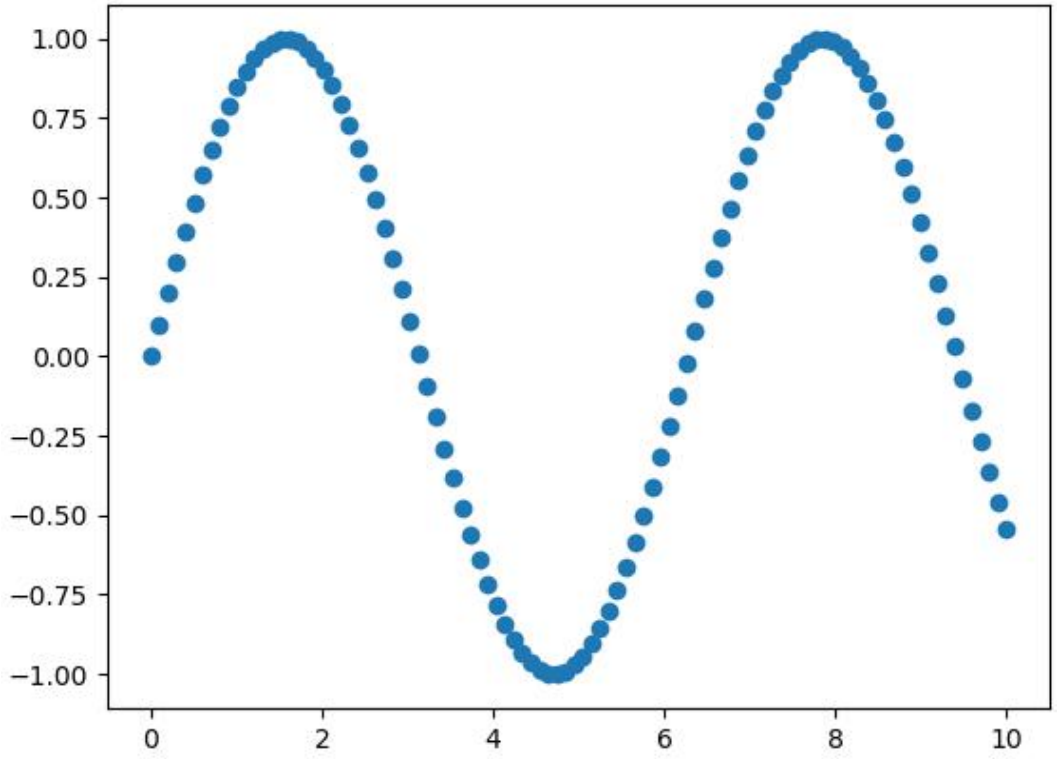


图 2-9 示例 2-8 运行效果图

【示例 2-9】绘制 10 种大小 100 种颜色的散点图

```
import matplotlib.pyplot as plt
import numpy as np
# 画 10 种大小， 100 种颜色的散点图
np.random.seed(0)
x=np.random.rand(100)
y=np.random.rand(100)
colors=np.random.rand(100)
size=np.random.rand(10)*1000
```

```
plt.scatter(x,y,c=colors,s=size,alpha=0.7)
plt.show()
```

运行结果如图 2-10 所示：

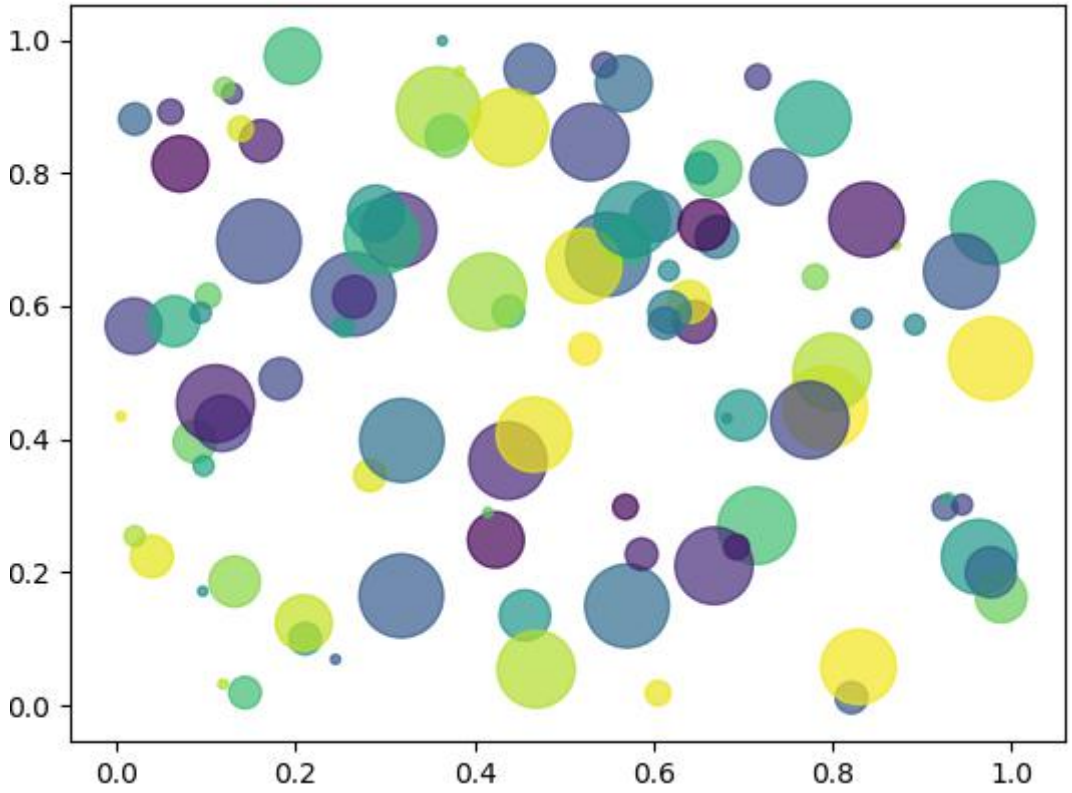


图 2-10 示例 2-9 运行效果图

作为线性图的替代，可以通过向 plot() 函数添加格式字符串来显示离散值。 可以使用如表 2-2 格式化字符。

表 2-2 格式化字符

字符	描述
'-'	实线样式
'--'	短横线样式
'-.'	点划线样式
':'	虚线样式
'.'	点标记
','	像素标记
'o'	圆标记
'v'	倒三角标记
'^'	正三角标记
'1'	下箭头标记
'2'	上箭头标记

字符	描述
'3'	左箭头标记
'4'	右箭头标记
's'	正方形标记
'p'	五边形标记
'*'	星形标记
'h'	六边形标记 1
'H'	六边形标记 2
'+'	加号标记
'x'	X 标记
'D'	菱形标记
'd'	窄菱形标记
'_'	水平线标记

颜色的缩写如表 2-3 所示：

图 2-3 颜色的缩写

字符	颜色
'b'	蓝色
'g'	绿色
'r'	红色
'c'	青色
'm'	品红色
'y'	黄色
'k'	黑色
'w'	白色

【示例 2-10】绘制不同种类不同颜色的线

```
import matplotlib.pyplot as plt
import numpy as np
#不同种类不同颜色的线
x=np.linspace(0,10,100)
plt.plot(x,x+0,'-g')       #实线    绿色
plt.plot(x,x+1,'--c')     #虚线  浅蓝色
plt.plot(x,x+2,'-.k')     #点划线  黑色
plt.plot(x,x+3,'-r')       #实线    红色
```

```
plt.plot(x,x+4,'o')        #点     默认是蓝色
plt.plot(x,x+5,'x')        #叉叉   默认是蓝色
plt.plot(x,x+6,'d')       #砖石   红色
plt.show()
```

【示例 2-11】添加图例

```
#不同种类不同颜色的线并添加图例
x=np.linspace(0,10,100)
plt.plot(x,x+0,'-g',label='-g')       #实线   绿色
plt.plot(x,x+1,'--c',label='--c')     #虚线  浅蓝色
plt.plot(x,x+2,'-.k',label='-.k')     #点划线  黑色
plt.plot(x,x+3,'-r',label='-r')       #实线   红色
plt.plot(x,x+4,'o',label='o')         #点     默认是蓝色
plt.plot(x,x+5,'x',label='x')         #叉叉   默认是蓝色
plt.plot(x,x+6,'dr',label='dr')       #砖石   红色
#添加图例右下角 lower right   左上角 upper left  边框   透明度   阴影   边框宽度
plt.legend(loc='lower right',fancybox=True,framealpha=1,shadow=True,borderpad=1)
plt.show()
```

运行结果如图 2-11 所示：

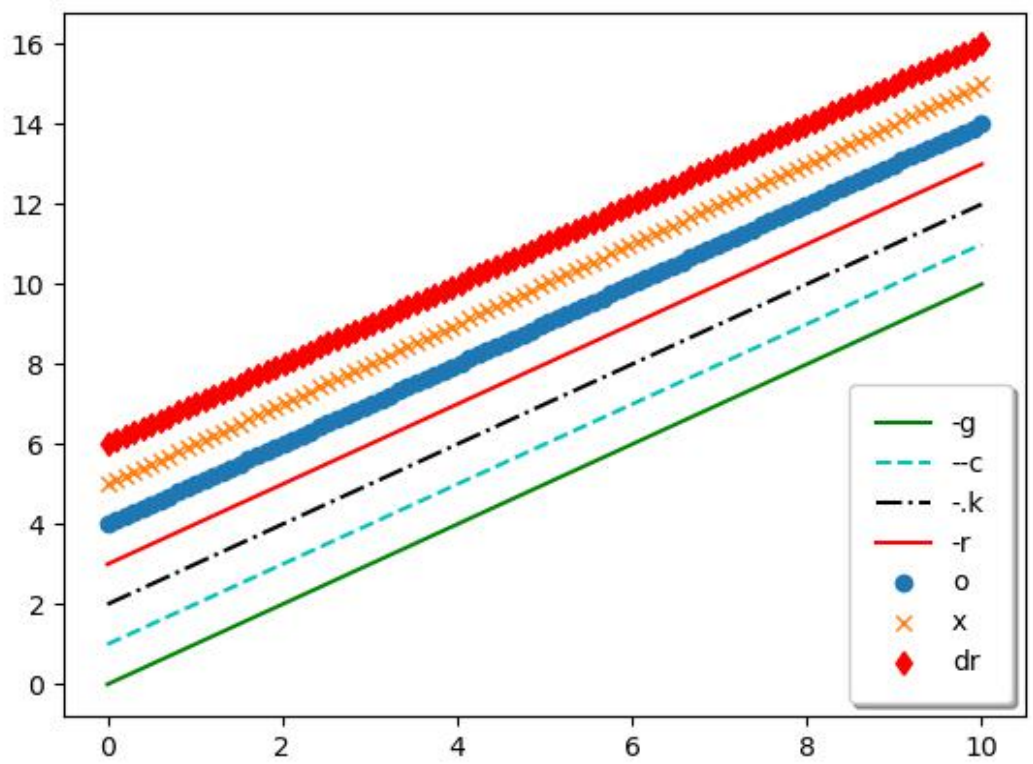


图 2-11 示例 2-11 运行效果图

2.4 绘制柱状图



使用 bar 函数可以绘制柱状图。柱状图需要水平的 x 坐标值，以及每一个 x 坐标值对应的 y 坐标值，从而形成柱状的图。柱状图主要用来纵向对比和横向对比的。例如，根据年份对销售收据进行纵向对比，x 坐标值就表示年份，y 坐标值表示销售数据。

【示例 2-12】使用 bar()绘制柱状图，并设置柱的宽度

```
import matplotlib.pyplot as plt
import numpy as np
x=[1980,1985,1990,1995]
x_labels=['1980 年','1985 年','1990 年','1995 年']
y=[1000,3000,4000,5000]
plt.bar(x,y,width=3)
plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签
```

```
plt.xticks(x,x_labels)
plt.xlabel('年份')
plt.ylabel('销量')
plt.title('根据年份销量对比图')
plt.show()
```

运行结果如图 2-12 所示：

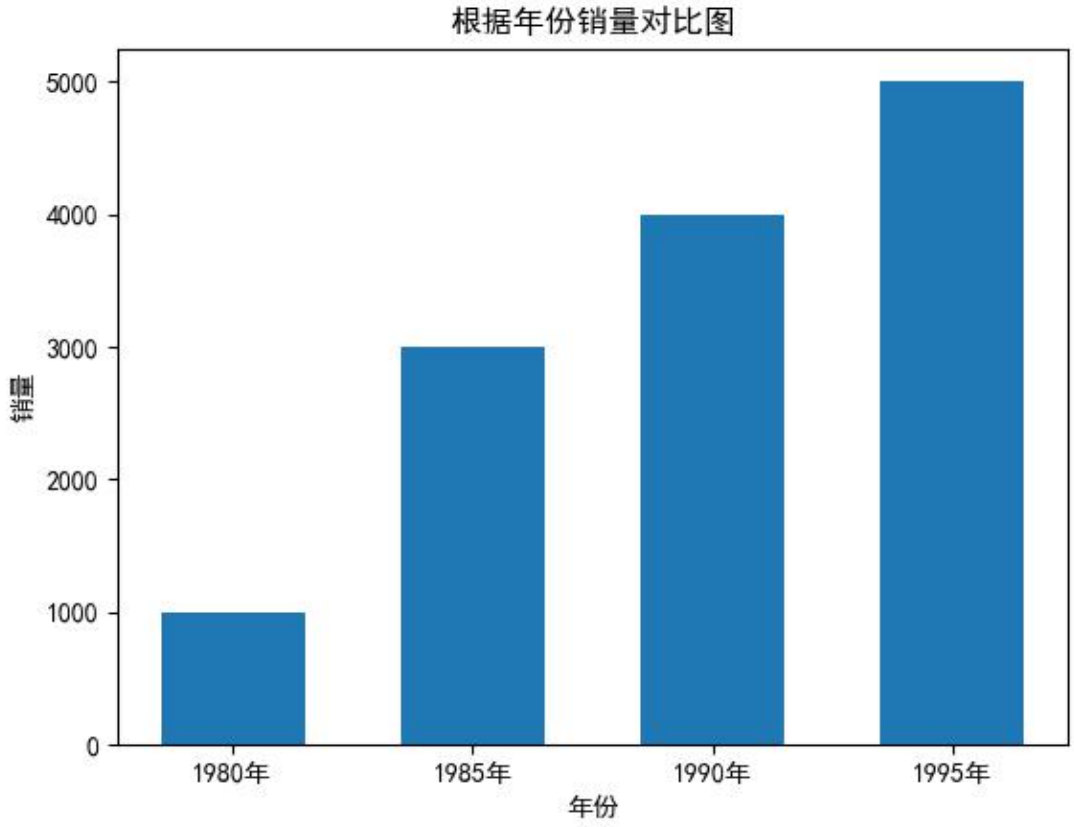


图 2-12 示例 2-12 运行效果图

需要注意的是 bar 函数的宽度并不是像素宽度。bar 函数会根据二维坐标系的尺寸，以及 x 坐标值的多少，自动确定每一个柱的宽度，而 width 指定的宽度就是这个标准柱宽度的倍数。该参数值可以是浮点数，如 0.5，表示柱的宽度是标准宽度的 0.5 倍。

【示例 2-13】使用 bar()和 barh()函数绘制柱状图

```
import matplotlib.pyplot as plt
import numpy as np
np.random.seed(0)
x=np.arange(5)
y=np.random.randint(-5,5,5)
print(x,y)
```

```
# 将画布分隔成一行两列
plt.subplot(1,2,1)
#在第一列中画图
v_bar=plt.bar(x,y)
#在第一列的画布中 0 位置画一条蓝线
plt.axhline(0,color='blue',linewidth=2)
plt.subplot(1,2,2)
#barh 将 y 和 x 轴对换 竖着方向为 x 轴
h_bar=plt.barh(x,y,color='red')
#在第二列的画布中 0 位置处画蓝色的线
plt.axvline(0,color='red',linewidth=2)
plt.show()
```

运行结果如图 2-13 所示：

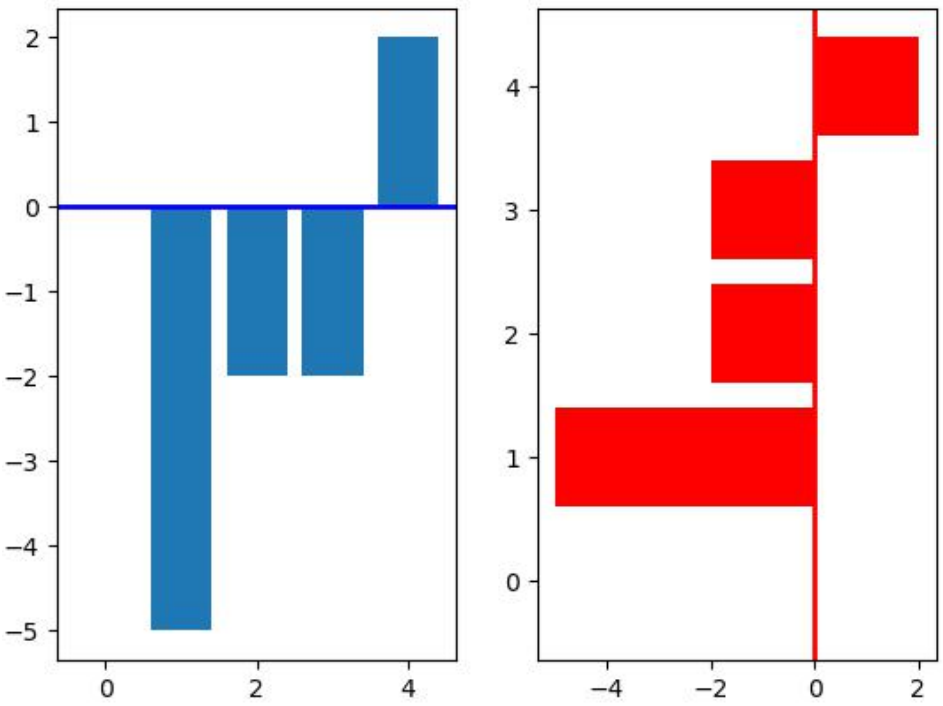


图 2-13 示例 2-13 运行效果图

【示例 2-14】对柱状图的部分柱状设置颜色

```
import matplotlib.pyplot as plt
import numpy as np
np.random.seed(0)
x=np.arange(5)
y=np.random.randint(-5,5,5)
```

```
v_bar=plt.bar(x,y,color='lightblue')
for bar,height in zip(v_bar,y):
    if height<0:
        bar.set(edgecolor='darkred',color='lightgreen',linewidth='3')
plt.show()
```

运行结果如图 2-14 所示：

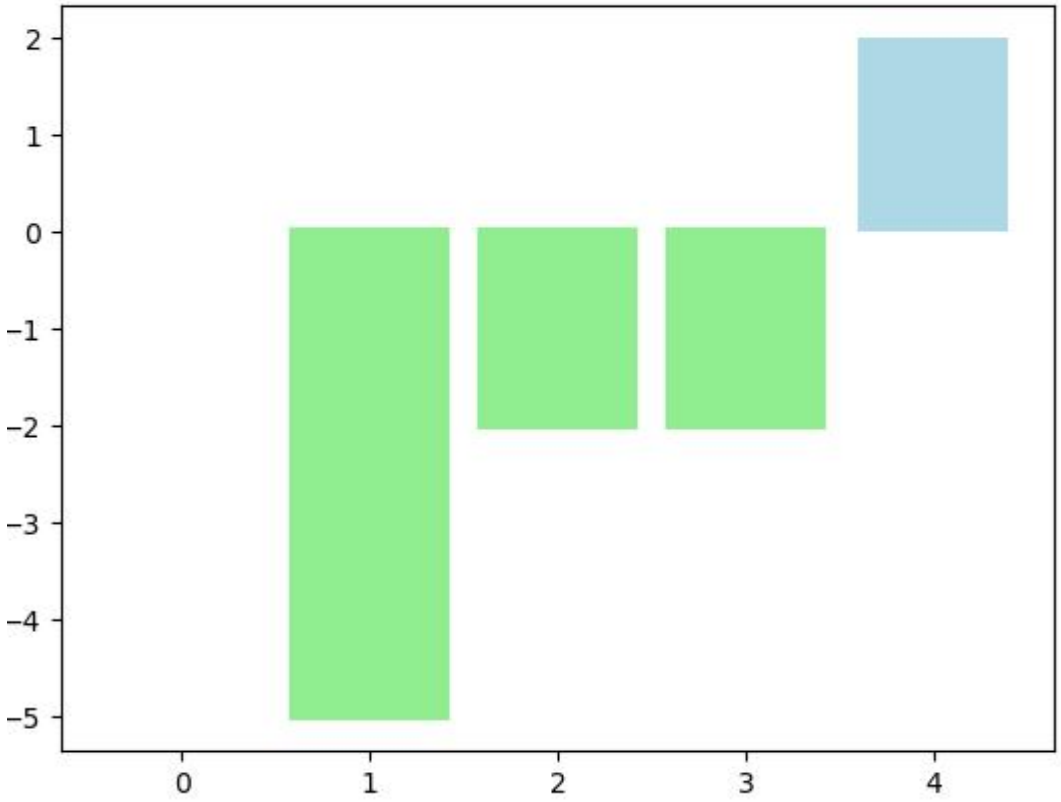


图 2-14 示例 2-14 运行效果图

【示例 2-15】使用 bar()绘制三天中三部电影的票房变化

```
import matplotlib.pyplot as plt
import numpy as np
#三天中三部电影的票房变化
real_names=['千与千寻','玩具总动员 4','黑衣人：全球追缉']
real_num1=[5453,7548,6543]
real_num2=[1840,4013,3421]
real_num3=[1080,1673,2342]
#生成 x  第 1 天  第 2 天  第 3 天
x=np.arange(len(real_names))
x_label=['第{}天'.format(i+1) for i in range(len(real_names))]
```

```
#绘制柱状图
#设置柱的宽度
width=0.3
plt.bar(x,real_num1,color='g',width=width,label=real_names[0])
plt.bar([i+width for i in x],real_num2,color='b',width=width,label=real_names[1])
plt.bar([i+2*width for i in x],real_num3,color='r',width=width,label=real_names[2])
plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签
#修改 x 坐标
plt.xticks([i+width for i in x],x_label)
#添加图例
plt.legend()
#添加标题
plt.title('3 天的票房数')
plt.show()
```

运行结果如图 2-15：

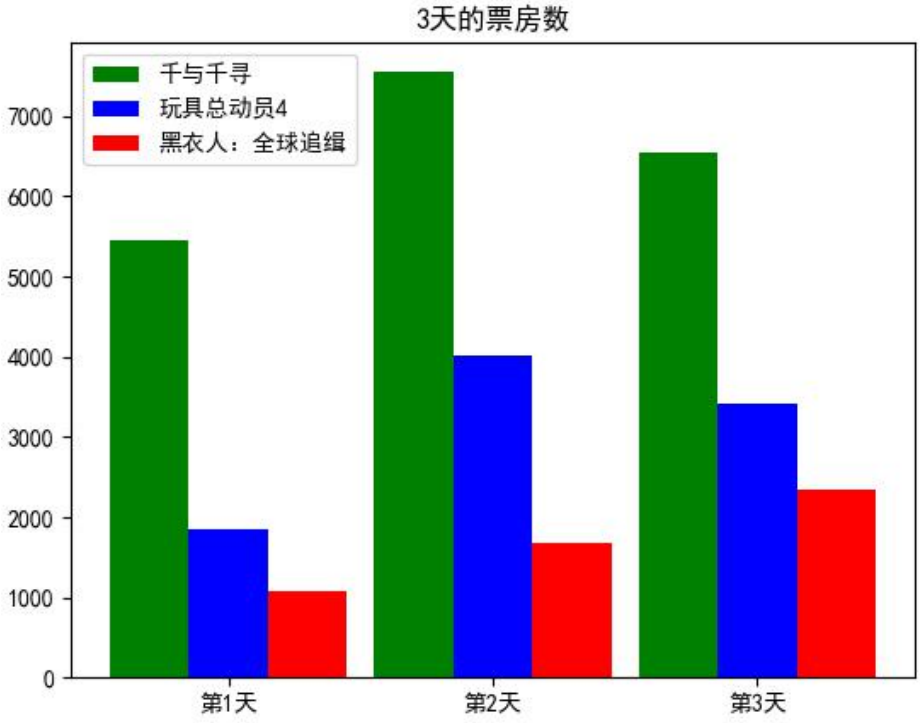


图 2-15 示例 2-15 运行效果图

2.5 绘制饼状图

pie 函数可以绘制饼状图，饼图主要是用来呈现比例的。只要传入比例数据即



可。

【示例 2-16】绘制饼状图

```
#导入模块
import matplotlib.pyplot as plt
import numpy as np
#准备男、女的人数及比例
man=71351
woman=68187
man_perc=man/(woman+man)
woman_perc=woman/(woman+man)
#添加名称
labels=['男','女']
#添加颜色
colors=['blue','red']
#绘制饼状图　pie
plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签
# labels 名称 colors：颜色，explode=分裂　autopct 显示百分比
paches,texts,autotexts=plt.pie([man_perc,woman_perc],labels=labels,colors=colors,explode=(0,0.
05),autopct='%0.1f%%')

#设置饼状图中的字体颜色
for text in autotexts:
    text.set_color('white')

#设置字体大小
for text in texts+autotexts:
    text.set_fontsize(20)
plt.show()
```

运行结果如图 2-16 所示：

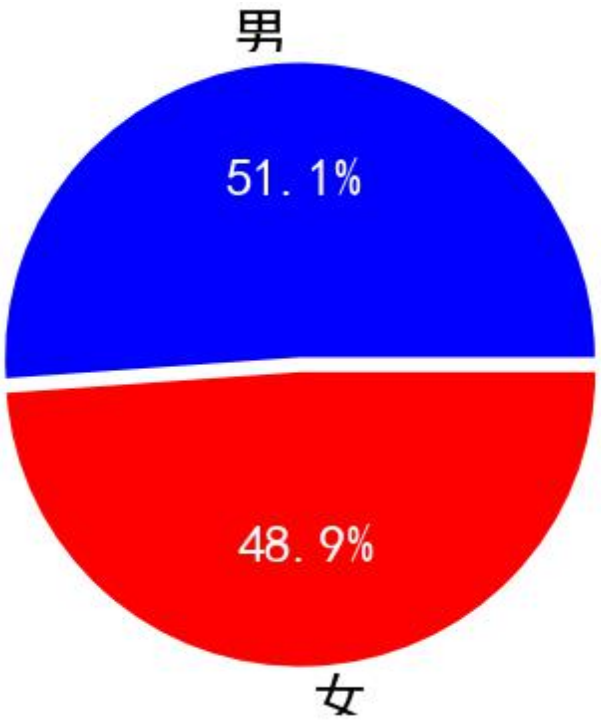
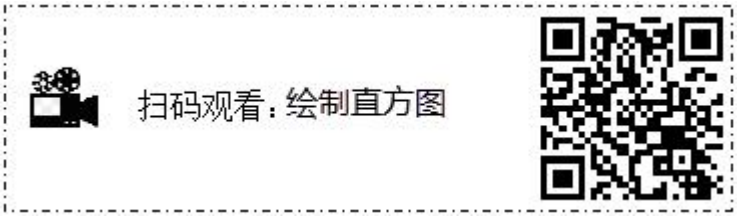


图 2-16 示例 2-16 运行效果图

2.6 绘制直方图



直方图与柱状图的分格类似，都是由若干个柱组成，但直方图和柱状图的含义却有很大的差异。直方图是用来观察分布状态的，而柱状图是用来看每一个 X 坐标对应的 Y 的值的。也就是说，直方图关注的是分布，并不关心具体的某个值，而柱状图关心的是具体的某个值。使用 hist 函数绘制直方图。

【示例 2-17】绘制直方图

```
import numpy as np
import matplotlib.pyplot as plt
#频次直方图，均匀分布
#正太分布
x=np.random.randn(1000)
#画正太分布图
# plt.hist(x)
plt.hist(x,bins=100) #装箱的操作，将 10 个柱装到一起及修改柱的宽度
plt.show()
```

运行结果如图 2-17 所示：

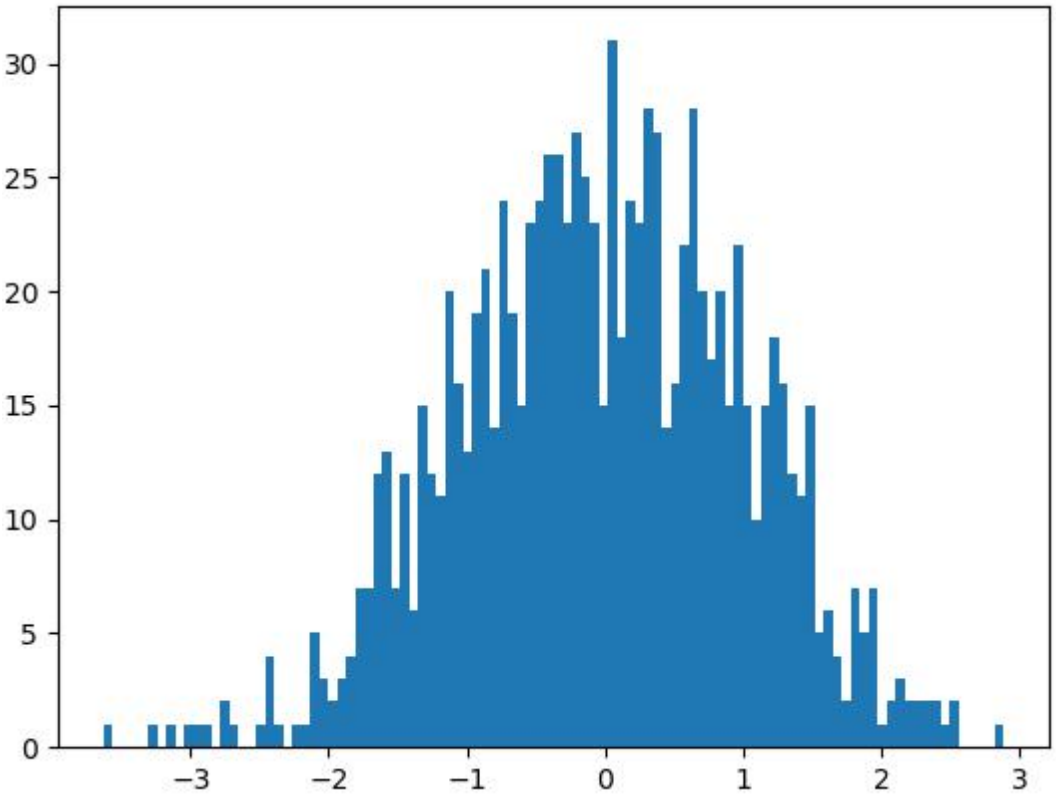


图 2-17 示例 2-17 运行效果图

【示例 2-18】同一画布绘制三个直方图

```
import numpy as np
import matplotlib.pyplot as plt
#几个直方图画到一个画布中,第一个参数期望　第二个均值
x1=np.random.normal(0,0.8,1000)
x2=np.random.normal(-2,1,1000)
x3=np.random.normal(3,2,1000)
#参数分别是 bins：装箱，alpha：透明度
kwargs=dict(bins=100,alpha=0.4)
plt.hist(x1,**kwargs)
plt.hist(x2,**kwargs)
plt.hist(x3,**kwargs)
plt.show()
```

运行结果如图 2-18 所示：

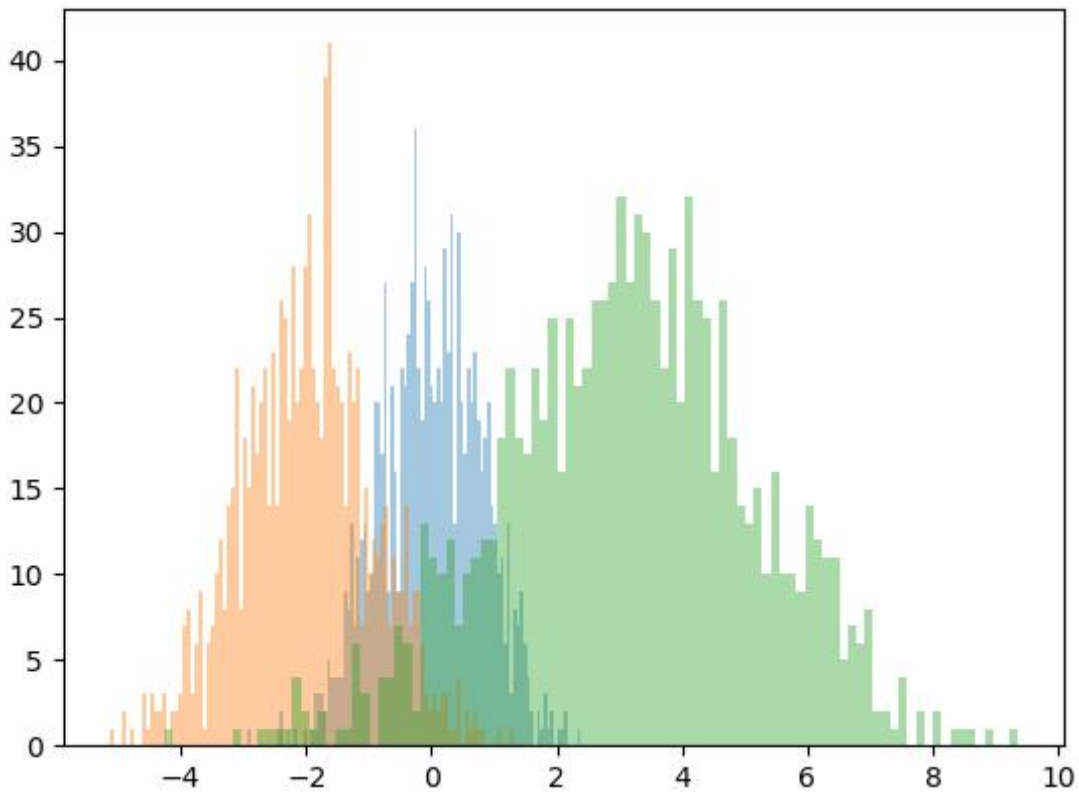


图 2-18 示例 2-18 运行效果图

2.7 绘制等高线图



扫码观看：绘制等高线图



【示例 2-19】使用 matplotlib 绘制等高线图

```
#导入模块
import matplotlib.pyplot as plt
import numpy as npaa
x=np.linspace(-10,10,100)
y=np.linspace(-10,10,100)
#计算 x 和 y 的相交点 a
X,Y=np.meshgrid(x,y)
# 计算 Z 的坐标
Z=np.sqrt(X**2+Y**2)
plt.contourf(X,Y,Z)
plt.contour(X,Y,Z)
# 颜色越深表示值越小，中间的黑色表示 z=0.
```

```
plt.show()
```

运行结果如图 2-19 所示：

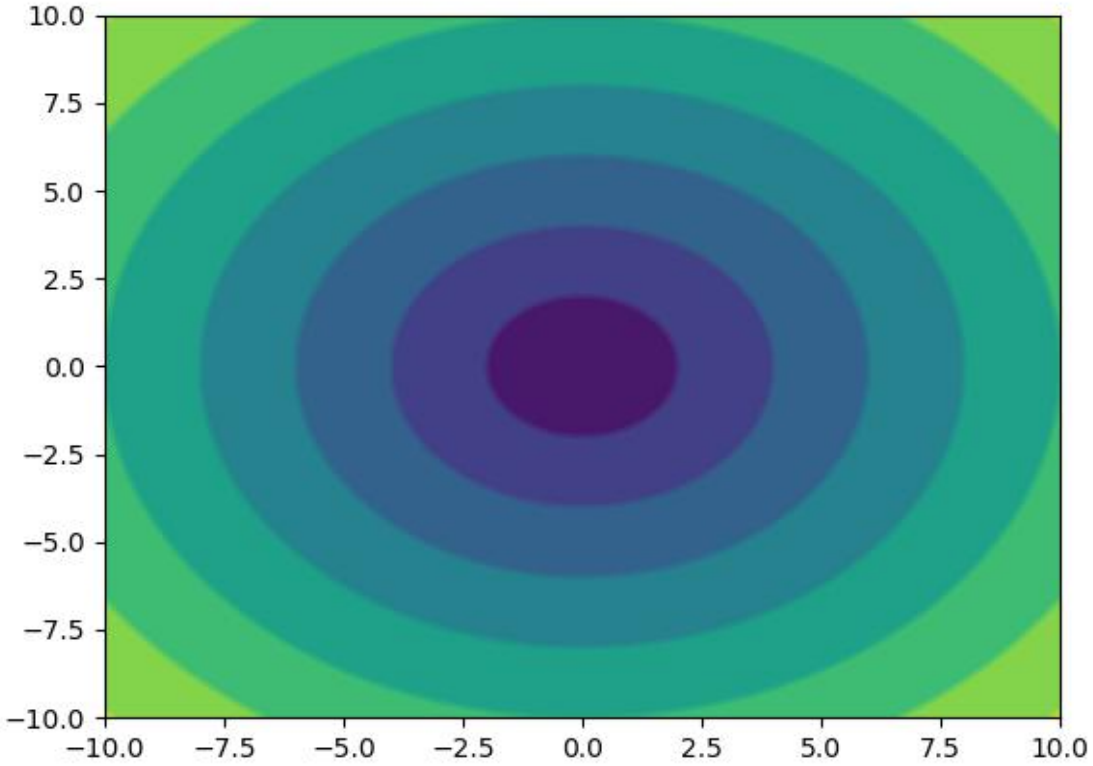


图 2-19 示例 2-19 运行效果图

2.8 绘制三维图

使用 pyplot 包和 Matplotlib 绘制三维图。

【示例 2-20】使用 Matplotlib 绘制三维图

```
import matplotlib.pyplot as plt
#导入 3D 包
from mpl_toolkits.mplot3d import Axes3D
#创建 X、Y、Z 坐标
X=[1,1,2,2]
Y=[3,4,4,3]
Z=[1,100,1,1]
fig = plt.figure()
#  创建了一个 Axes3D 的子图放到 figure 画布里面
ax = Axes3D(fig)
ax.plot_trisurf(X, Y, Z)
```

```
plt.show()
```

运行结果如图 2-20 所示：

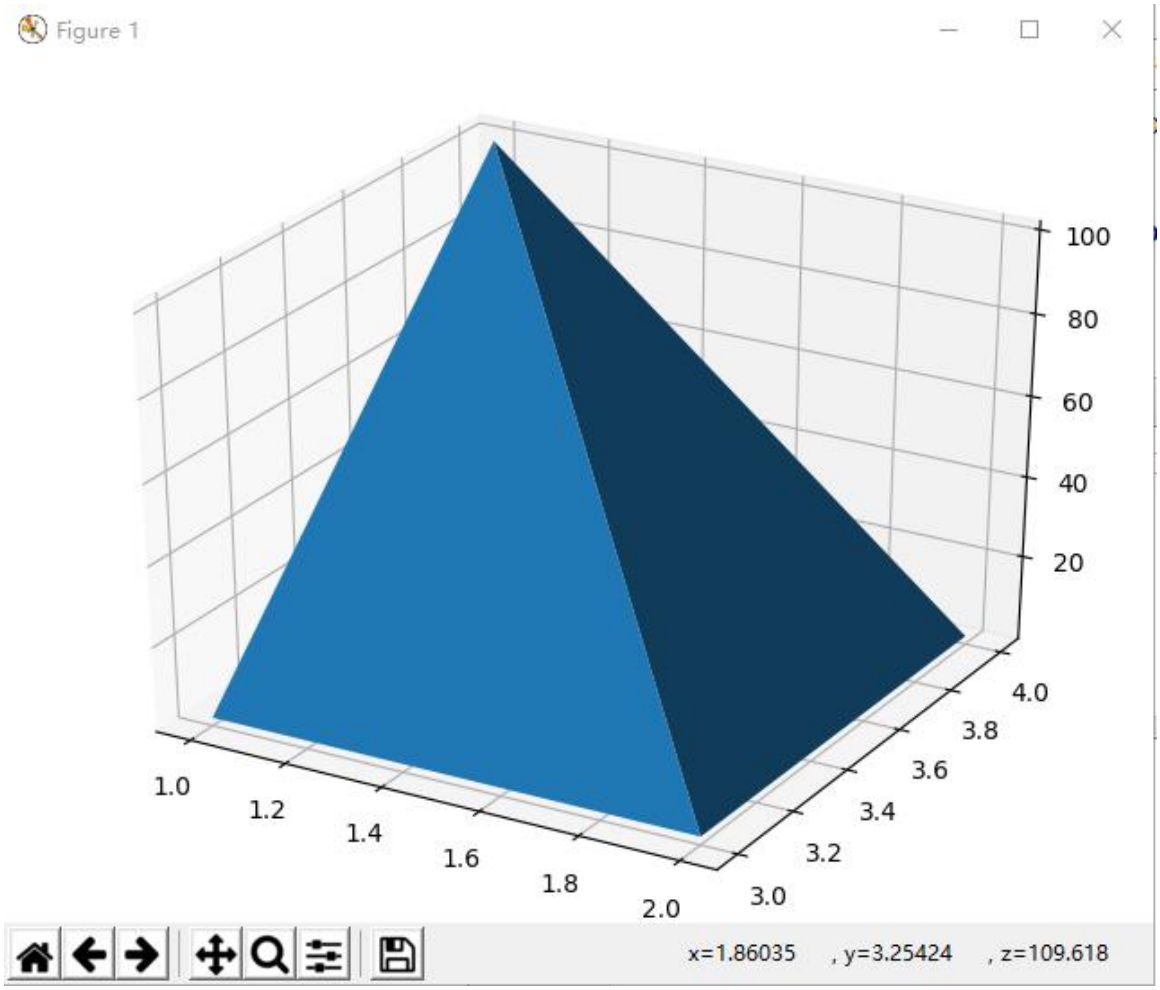


图 2-20 示例 2-20 运行效果图

习题

一、选择题

1. 绘制散点图的方法是（）

A. plt.plot()

B. plt.scatter()

C. plt.bar()

D. plt.hist()

2. 下列叙述错误的是（）

A. title()设置图表的名称

B. xlabel()设置 x 轴名称

C. yticks()设置 y 轴的刻度

D. plot()可以绘制线性图表，不能绘制散点图

3. 实现 x=(0,10)间 sin 的散点图像的是（）（多选）

A. plt.scatter(x, np.sin(x))

B. plt.plot(x, np.sin(x), 'o')

C. plt.plot(x, np.sin(x))

D. plt.scatter(x, np.sin(x), 'o')

4. 下列能绘制一个水平方向柱状图的是（）

A. plt.bar(x,y,tick_label = label)

B. plt.barh(x,y,tick_label = label)

C. plt.errorbar(x, y,tick_label = label）

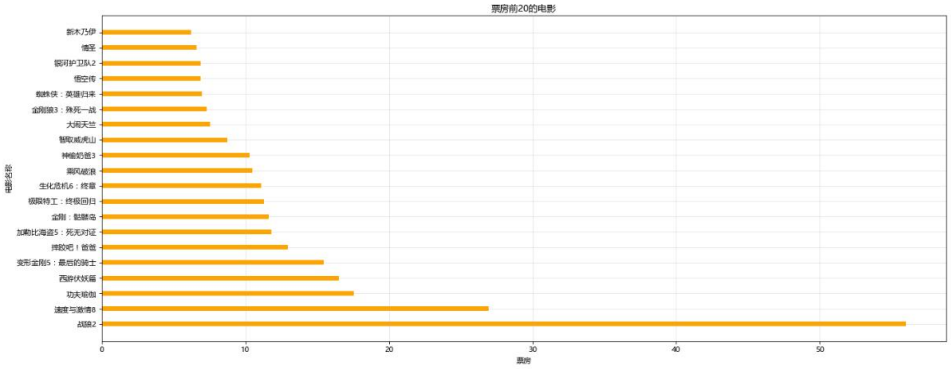
D. plt.hist(x,y,tick_label = label)

二、解答题

1. 搭建 matplotlib 开发环境。
2. pyplot 的基本方法。
3. Matplotlib 绘制基本图形及使用方法。
4. 如何绘制子图。

三、编码题

1. 编写程序，用 matplotlib 绘制余切曲线并保存成图片。
2. 编写程序，用 matplotlib 绘制一个饼状图（数据可以自定指定）。
3. 编写程序，用 matplotlib 绘制一个柱状图分析 3 部电影 3 天的票房。
4. 编写程序，用 matplotlib 绘制实现如下图的效果：



1) 内地电影票房前 20 的电影（列表 a）a = ["战狼 2","速度与激情 8","功夫瑜伽","西游伏妖篇","变形金刚 5：最后的骑士","摔跤吧！爸爸","加勒比海盗 5：死无对证","金刚：骷髅岛","极限特工：终极回归","生化危机 6：终章","乘风破浪","神偷奶爸 3","智取威虎山","大闹天竺","金刚狼 3：殊死一战","蜘蛛侠：英雄归来","悟空传","银河护卫队 2","情圣","新木乃伊",]

2) 电影票房数据（列表 b）b= [56.01,26.94,17.53,16.49,15.45,12.96,11.8,11.61,11.28,11.12,10.49,10.3,8.75,7.55,7.32,6.99,6.88,6.86,6.58,6.23]

第三章 Pandas 数据分析库

Pandas 是基于 Numpy 的一套数据分析工具，该工具是为了解决数据分析任务而创建的。Pandas 纳入了大量标准的数据模型，提供了高效地操作大型数据集所需的工具。Pandas 提供了大量能使我们快速便捷地处理数据的函数和方法。它是使 Python 成为强大而高效的数据分析环境的重要因素之一。

通过阅读本章，您可以：

- 掌握 Anaconda 环境的安装及使用
- 了解什么是 Pandas
- 掌握 Series 对象基本操作
- 掌握 DataFrame 对象的基本操作
- 掌握缺值处理
- 掌握 Series 对象和 DataFrame 的拼接
- 掌握 merge 的使用

3.1 Pandas 开发环境搭建

Pandas 是第三方程序库，所以在使用 Pandas 之前必须安装 Pandas 库。但是如果使用 Anaconda Python 开发环境，那么 Pandas 已经集成到 Anaconda 环境中，不需要再安装。

3.1.1 Anaconda 下载和安装

Anaconda 已经自动的安装了 Jupter Notebook 及其他工具，还有 python 中超过 180 个科学包及其依赖项。现在就看一下 Anaconda 的具体安装流程。



1) 进入官网（https://www.anaconda.com/products/individual），单击 Download 按钮，如图 3-1 所示。

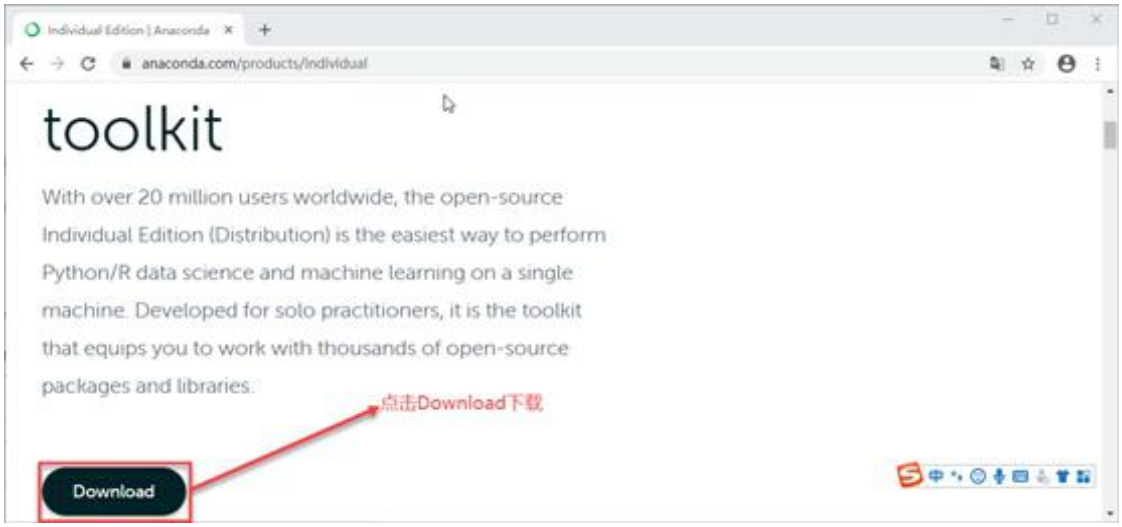


图 3-1 Anaconda 官网

2) 根据电脑系统类型及 Python 环境下载对应版本，因为在 2020 年之后官方就不再支持 Python2 了，所以建议大家选择 Python3，本书的代码也是基于 Python3，然后根据电脑的操作系统位数（32Bit/64Bit）选择对应版本，如图 3-2 所示。



图 3-2 Anaconda 下载

3) 下载后保存到电脑里，双击安装包打开后进行安装，如图 3-3、3-4、3-5 依次单击相应按钮。

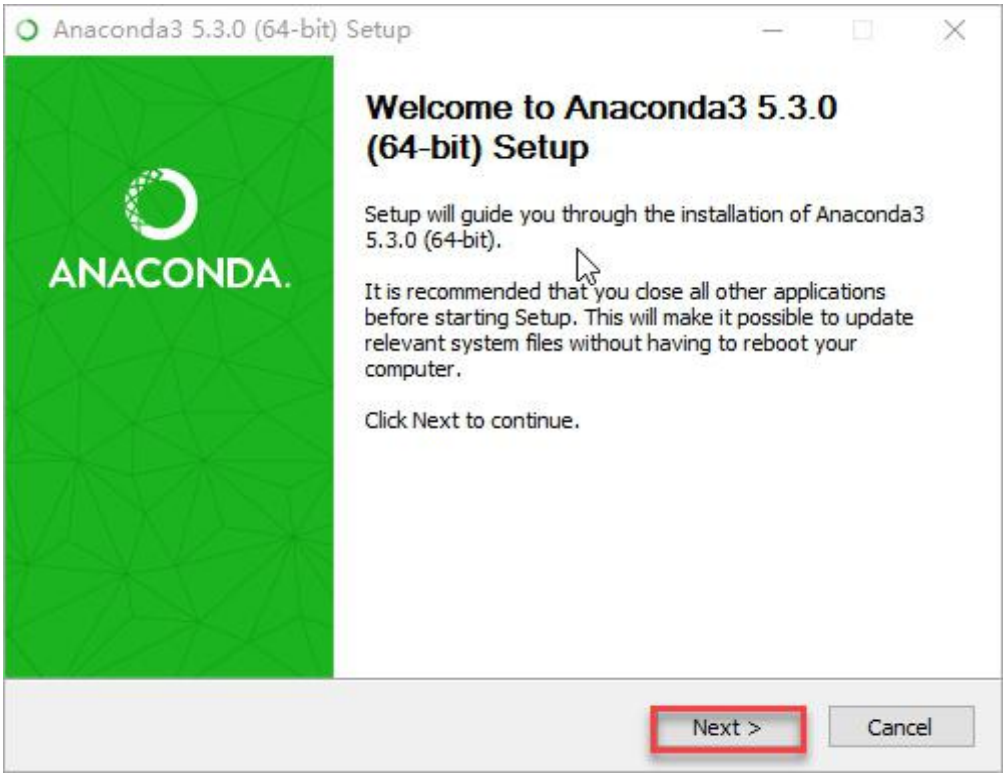


图 3-3 双击安装文件

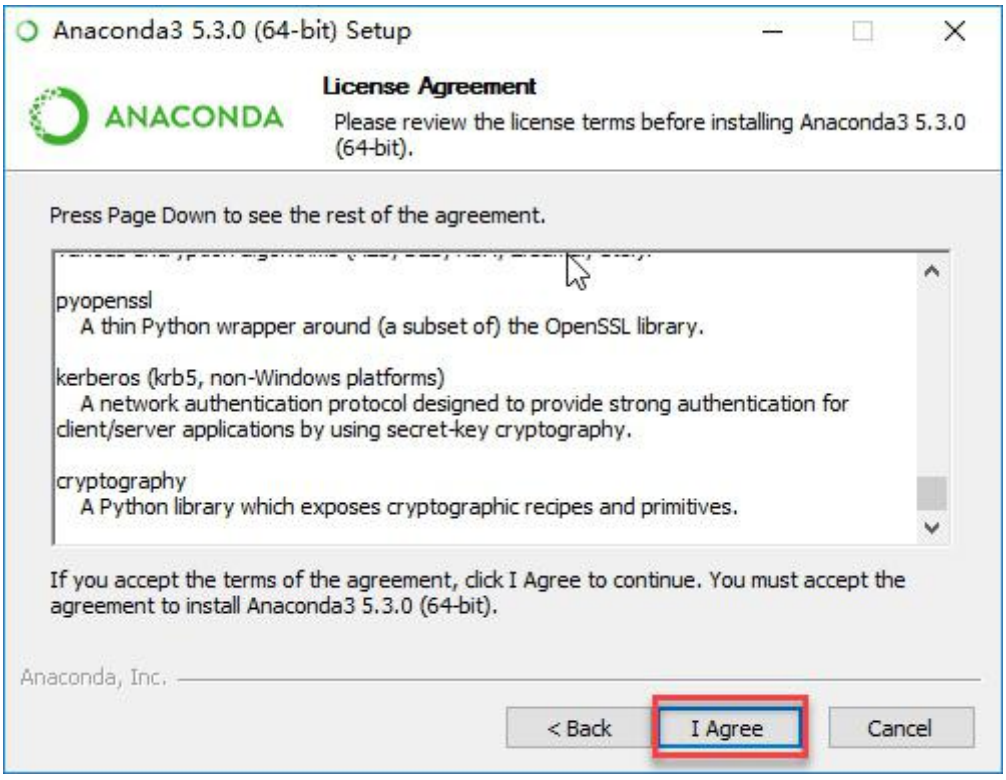


图 3-4 同意协议

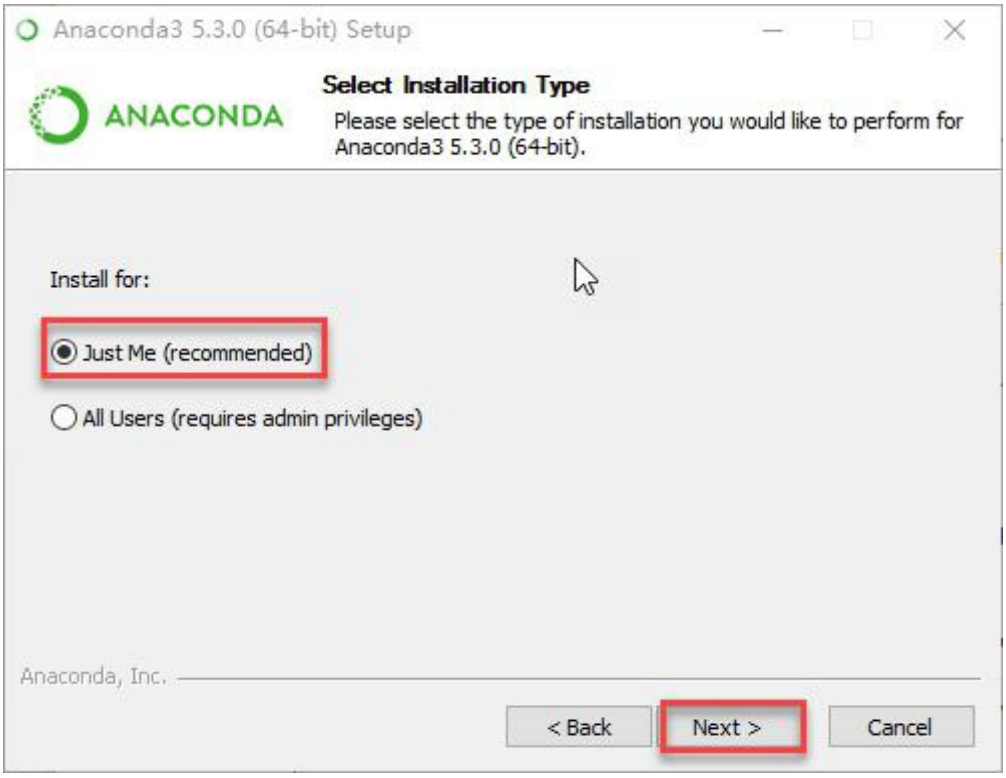


图 3-5 选择用户

4) 选择安装路径，不需要添加环境变量，然后点击 Next 按钮，并在弹出的对话框中勾选相应选项即可。

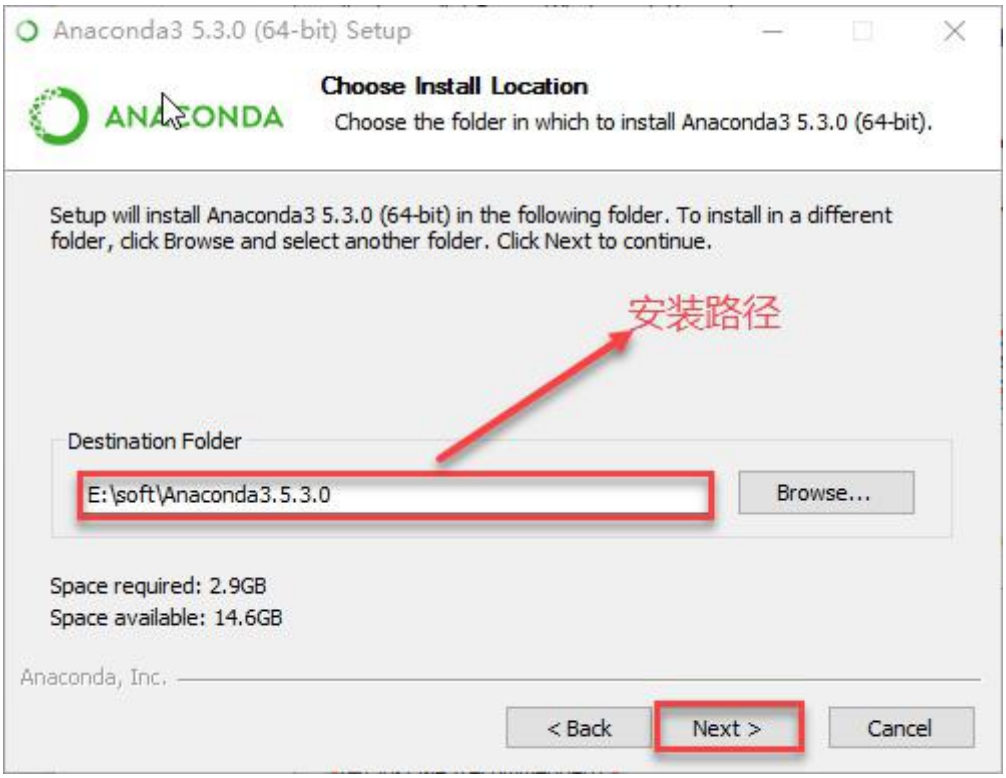


图 3-6 选择安装路径

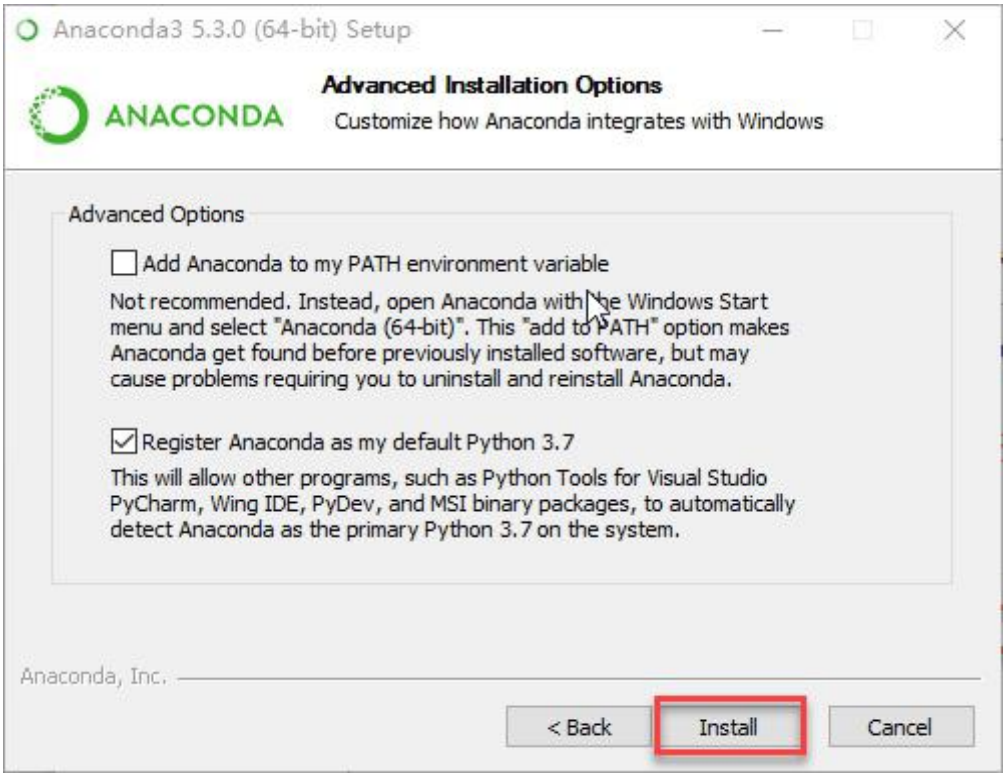


图 3-7 不添加环境变量

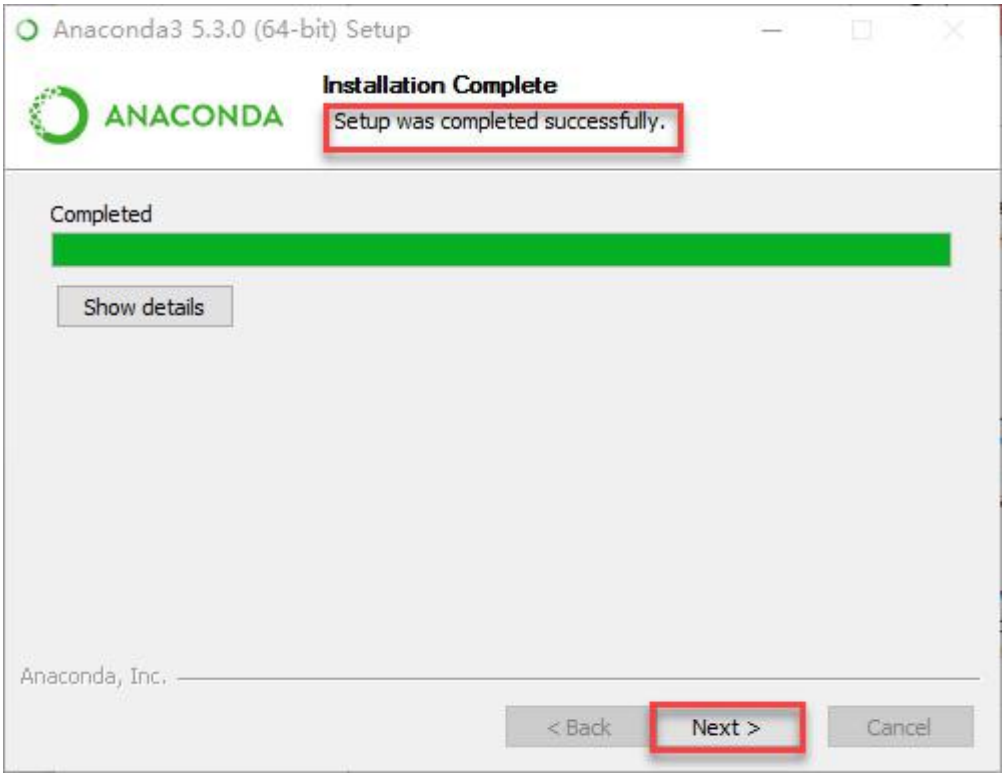


图 3-8 安装成功

5) 单击并勾选如图 3-9、3-10 所示按钮。

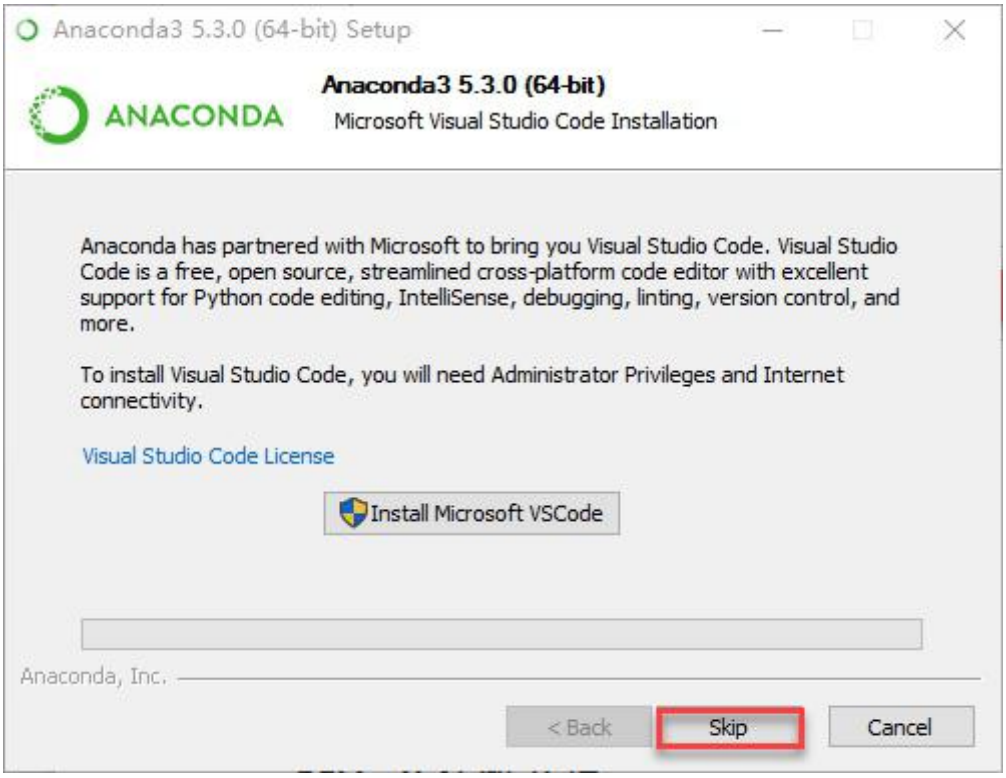


图 3-9 跳过

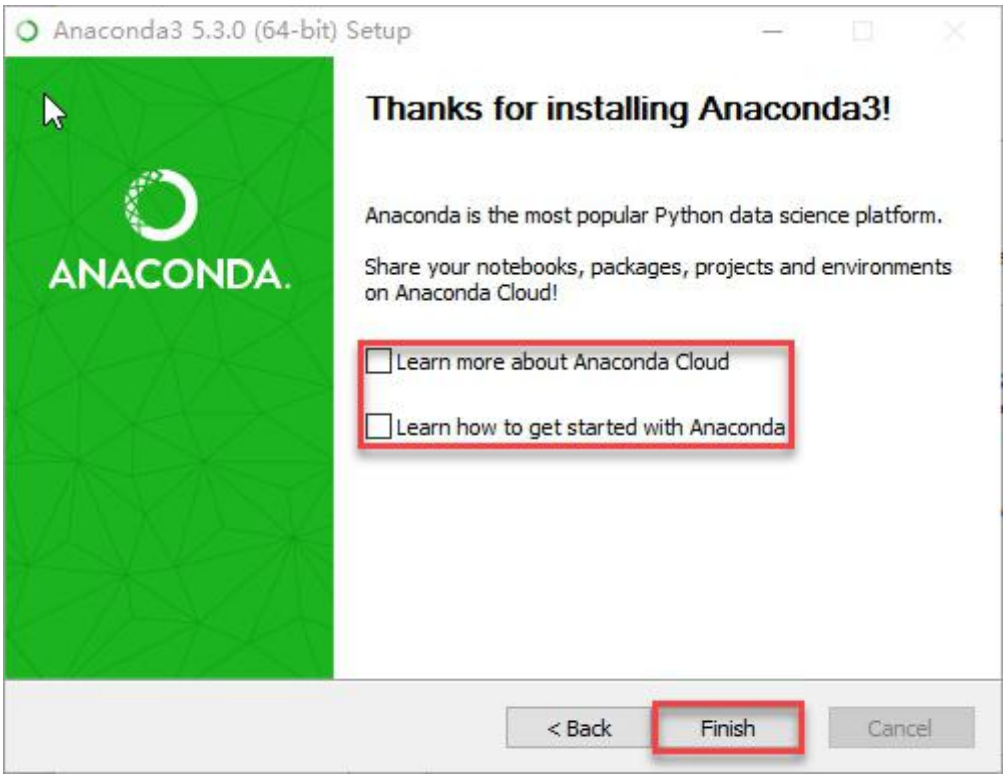


图 3-10 安装完成

6) 完成上述操作后，开始界面就会看到如图 3-11 所示新添加的程序，这就表示 Anoconda 安装好了。单击 Jupyter Notebook 打开，会弹出一个黑框，按 Enter 键后会让你选择哪个浏览器打开，建议选择 Chrome 浏览器。

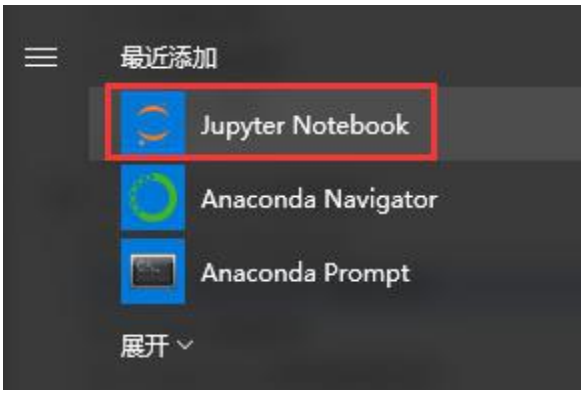


图 3-11 开始界面新增程序

7) 当看到如图 3-12 所示界面时，表示环境已经配置好了。

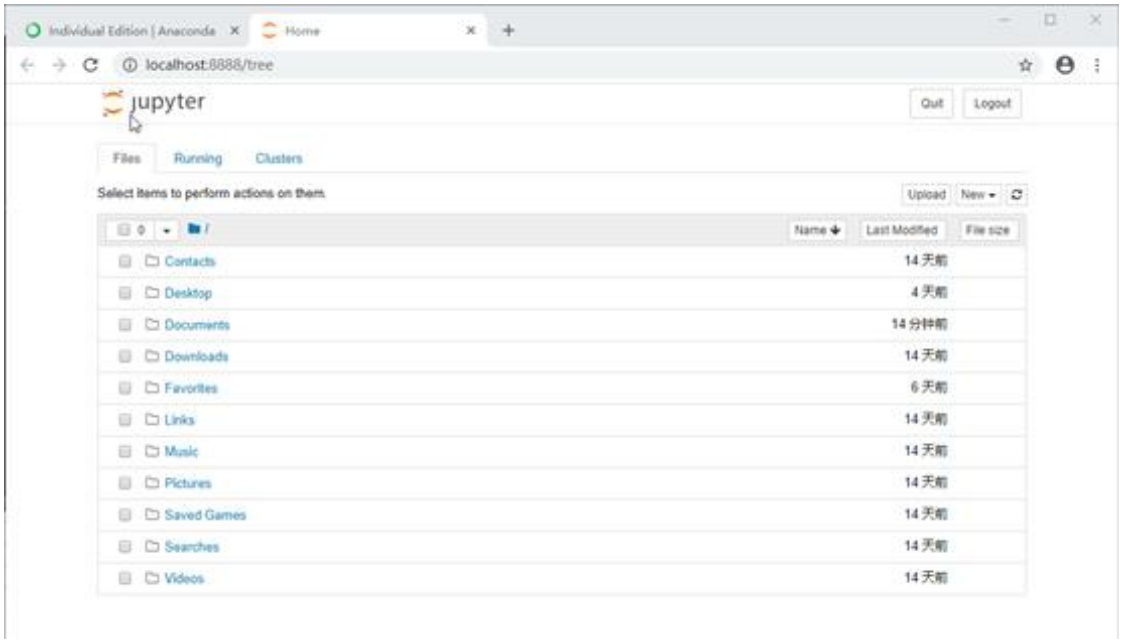


图 3-12 启动 Jupyter Notebook 界面

3.1.2 新建 Jupyter Notebook 文件

打开 Jupyter Notebook 后单击右上角的 New 按钮，在下拉列表中选择 Python3 选项来创建一个 Python 文件，如图 3-13 所示。

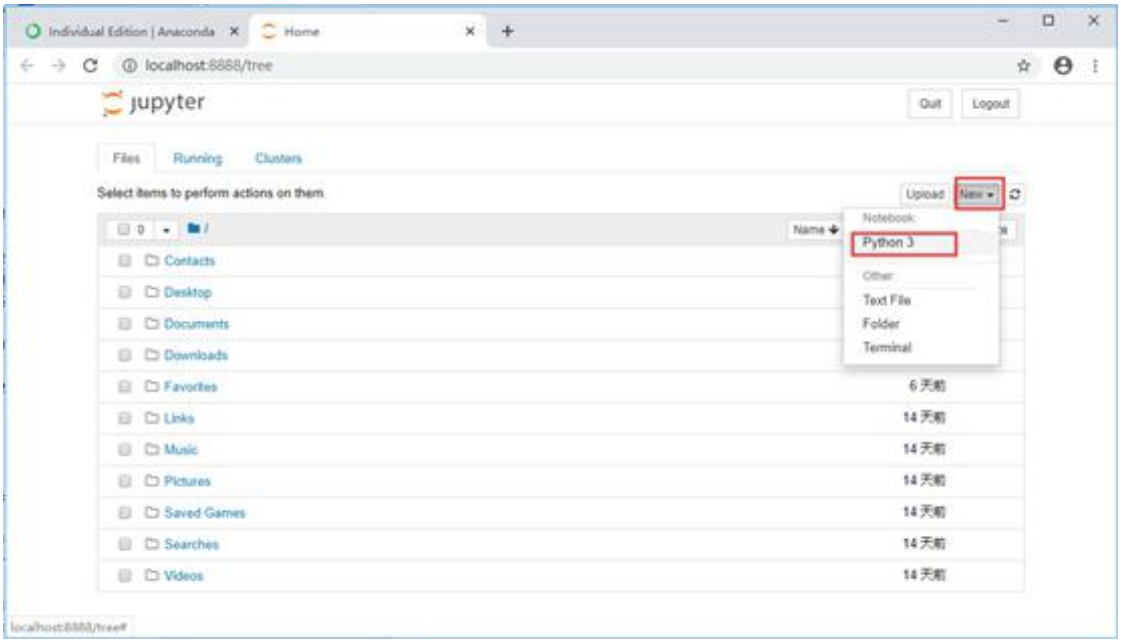


图 3-13 新建 Jupyter Notebook 文件

当你看到如图 3-14 所示的界面就表示新建了一个 Jupyter Notebook 文件。

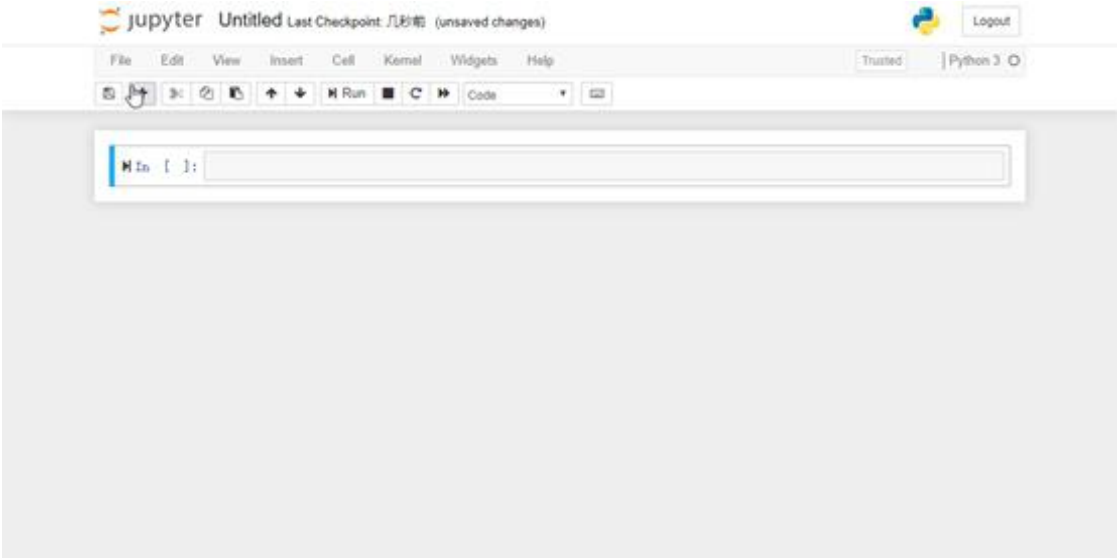


图 3-14 Jupyter Notebook 文件

3.1.3 运行第一段代码

在图 3-14 的代码框中输入 print(“hello Python”)，然后单击 Run 按钮，或者按 Ctrl+Enter 组合键，就会输出 hello Python，这就表示你的第一段代码运行成功。当你想换一个代码框输入代码时，你可以通过单击左上角的“+”按钮来新增代码框。执行效果如图 3-15 所示。

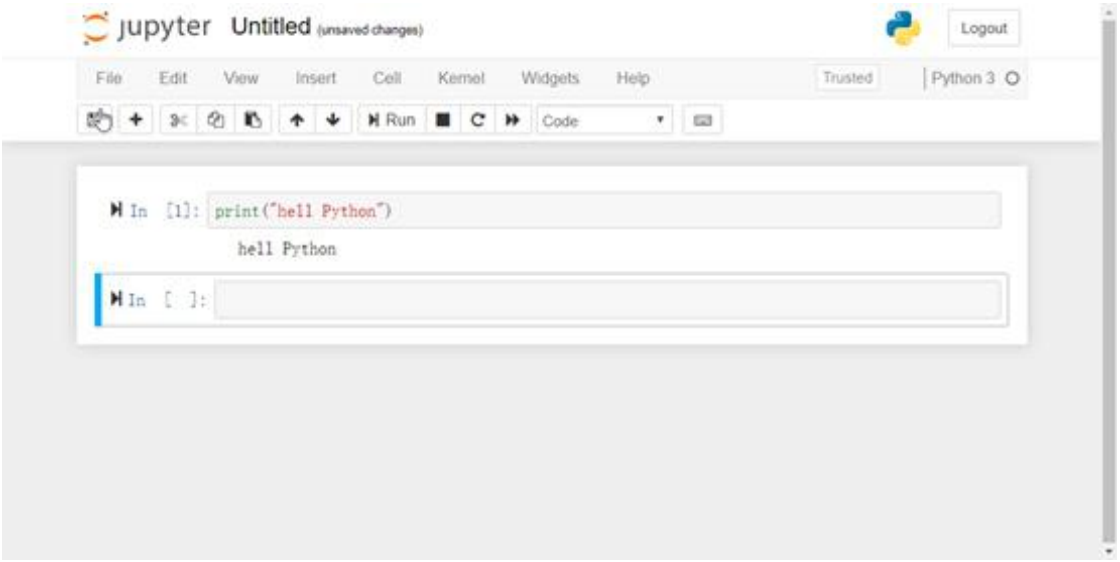


图 3-15 运行第一段代码

3.1.4 重命名 Jupyter Notebook 文件

当新建一个 Jupyter Notebook 文件时，该文件名默认为 Untitled（类似于 Excel 中的工作簿），可以单击 File→Rename 对该文件进行重命名，如图 3-16 所示。

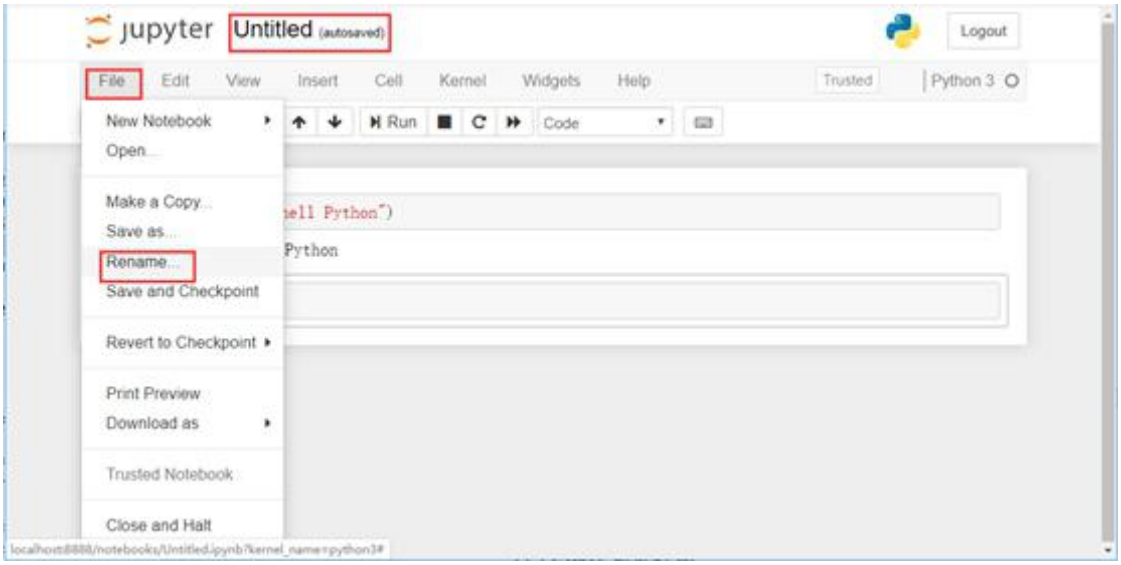


图 3-16 重命名 Jupyter Notebook 文件

3.1.5 保存 Jupyter Notebook 文件

保存文件有两种方式：

- 单击 File→Save and Checkpoint 保存文件，但是这种方法会将文件保存到默认路径下，且文件默认格式为 ipynb，ipynb 是 Jupyter Notebook 的专属文件格式。
- 选择 Download as 选项对文件进行保存，它相当于 Excel 中的“另存为”，可以自己选择保存路径及保存格式，如图 3-17 所示。

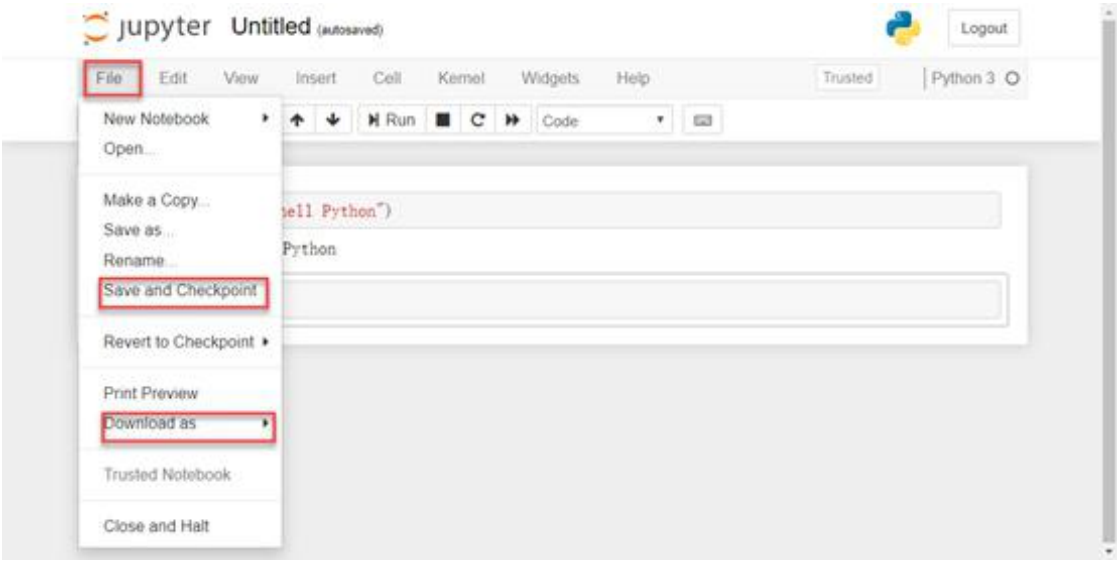


图 3-17 保存 Jupyter Notebook 文件

3.2 Pandas 数据类型

Pandas 中两个重要的数据类型：Series 和 DataFrame。Series 表示数据列表，DataFrame 表示二维数据集。

3.2.1 Series 对象创建



【示例 3-1】使用列表创建 Series 对象

```
import pandas as pd
data=pd.Series([4,3,5,6,1])
data
```

运行结果如图 3-18 所示：

```
In [2]: import pandas as pd

In [3]: data=pd.Series([4,3,5,6,1])
        data

Out[3]: 0    4
        1    3
        2    5
        3    6
        4    1
        dtype: int64
```

图 3-18 示例 3-1 运行效果图

series 对象包装的是 numpy 中的一维数组，实际上是将一个一维数组与一个索引名称捆绑在一起了。

pandas 中两个重要的属性 values 和 index，values 是 Series 对象的原始数据。index 对应了 Series 对象的索引对象。

【示例 3-2】Series 对象中两个重要的属性 values 和 index

```
data.values
data.index
```

运行结果如图 3-19 所示：

```
data.index
RangeIndex(start=0, stop=5, step=1)
data.values
array([4, 3, 5, 6, 1], dtype=int64)
```

图 3-19 示例 3-2 运行效果图

【示例 3-3】创建 Series 对象时候，指定 index 属性

```
data=pd.Series([5,4,6,3,1],index=['one','two','three','four','five'])
data
```

运行结果如图 3-20 所示：

```
In [10]: data=pd.Series([5,4,6,3,1],index=['one','two','three','four','five'])
         data
Out[10]: one      5
         two      4
         three    6
         four     3
         five     1
         dtype: int64
```

图 3-20 示例 3-3 运行效果图

【示例 3-4】创建 Series 对象时候，使用 list 列表指定 index 属性

```
data=pd.Series([4,3,2,1],index=list('abcd'))
data
```

运行结果如图 3-21 所示：

```
Out[14]: a    4
         b    3
         c    2
         d    1
         dtype: int64
```

图 3-21 示例 3-4 运行效果图

【示例 3-5】使用字典创建 Series 对象，默认将 key 作为 index 属性

```
population_dict={'bj':3000,'gz':1500,'sh':2800,'sz':1200}
population_series=pd.Series(population_dict)
population_series
```

运行结果如图 3-22 所示：

```
In  [2]: population_dict={'bj':3000,'gz':1500,'sh':2800,'sz':1200}
         population_series=pd.Series(population_dict)
         population_series

Out[2]: bj    3000
        gz    1500
        sh    2800
        sz    1200
        dtype: int64
```

图 3-22 示例 3-5 运行效果图

【示例 3-6】使用字典创建 Series 对象，又指定了 index 属性值，如果 key 不存在，则值为 NaN

```
sub_series=pd.Series(population_dict,index=['bj','sh']) #如果存在取交集
sub_series
sub_series=pd.Series(population_dict,index=['bj','xa']) #如果不存在则值为 NaN
sub_series
```

运行结果如图 3-23 所示：

```
In [22]: sub_series=pd.Series(population_dict,index=['bj','sh'])
         sub_series
Out[22]: bj    3000
         sh    2800
         dtype: int64
```

```
In [23]: sub_series=pd.Series(population_dict,index=['bj','xa'])
         sub_series
Out[23]: bj    3000.0
         xa       NaN
         dtype: float64
```

图 3-23 示例 3-6 运行效果图

【示例 3-7】标量与 index 属性创建 Series

```
data=pd.Series(10,index=[4,3,2,5])
data
```

运行结果如图 3-24 所示：

```
In [25]: data=pd.Series(10,index=[4,3,2,5])
         data
Out[25]: 4    10
         3    10
         2    10
         5    10
         dtype: int64
```

图 3-24 示例 3-7 运行效果图

3.2.2 DataFrame 对象创建

将两个 series 对象作为字典的值，就可以创建一个 DataFrame 对象。



【示例 3-8】创建 DataFrame 对象

```
population_dict={'beijing':3000,'shanghai':1200,'guangzhou':1800}
area_dict={'beijing':300,'guangzhou':200,'shanghai':180}
```

```
import pandas as pd
population_series=pd.Series(population_dict)
area_series=pd.Series(area_dict)
citys=pd.DataFrame({'area':area_series,'population':population_series})
citys
```

运行结果如图 3-25 所示：

Out[4]:

	area	population
beijing	300	3000
guangzhou	200	1800
shanghai	180	1200

图 3-25 示例 3-8 运行效果图

【示例 3-9】查看 DataFrame 对象的 values、index 和 columns 属性

```
citys.index
citys.values
citys.columns
```

运行结果如图 3-26 所示：

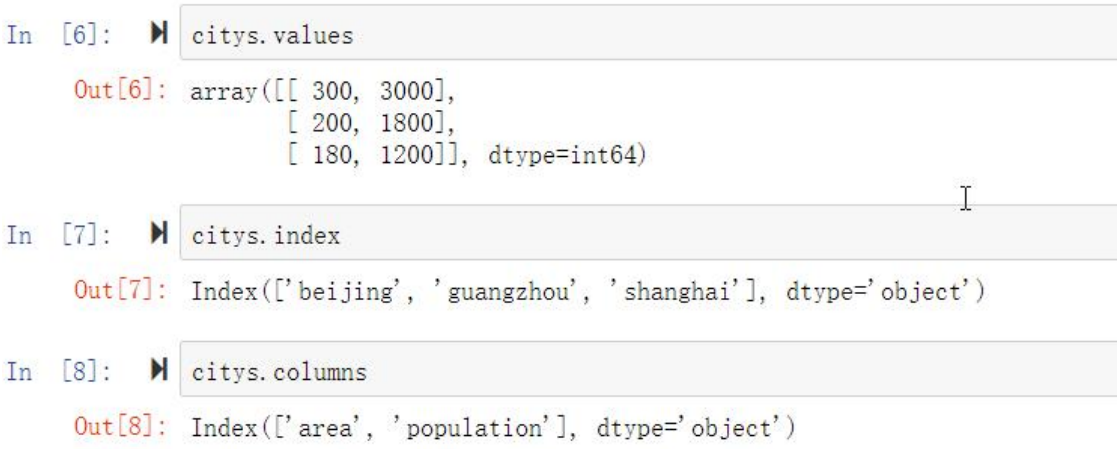


图 3-26 示例 3-9 运行效果图

【示例 3-10】使用列表创建 DataFrame 对象

```
population_dict={'beijing':3000,'shanghai':1200,'guangzhou':1800}
area_dict={'beijing':300,'shanghai':180,'guangzhou':200}
data=pd.DataFrame([population_dict,area_dict])
```

```
print(data) #将‘beijing’ ‘shanghai’ ‘guangzhou’作为表头
```

运行结果如图 3-27 所示：

	beijing	guangzhou	shanghai
0	3000	1800	1200
1	300	200	180

图 3-27 示例 3-10 运行效果图

【示例 3-11】创建 DataFrame 对象，指定 index 属性

```
population_dict={'beijing':3000,'guangzhou':1800,'shanghai':1200}
area_dict={'beijing':300,'shanghai':180,'guangzhou':200}
data=pd.DataFrame([area_dict,population_dict],index=['area','population'])
data
```

运行结果如图 3-28 所示：

	beijing	guangzhou	shanghai
area	300	200	180
population	3000	1800	1200

图 3-28 示例 3-11 运行效果图

【示例 3-12】创建 DataFrame 对象，指定列索引 columns

```
population_series=pd.Series(population_dict)
pd.DataFrame(population_series,columns=['population'])
```

运行结果如图 3-29 所示：

	population
beijing	3000
guangzhou	1800
shanghai	1200

图 3-29 示例 3-12 运行效果图

【示例 3-13】使用列表创建 Dataframe 对象

```
pd.DataFrame([{'a':i,'b':i*2} for i in range(3)])
```

运行结果如图 3-30 所示：

```
In [23]: pd.DataFrame([{'a':i,'b':i*2} for i in range(3)])
```

Out[23]:

	a	b
0	0	0
1	1	2
2	2	4

图 3-30 示例 3-13 运行效果图

【示例 3-14】使用一个二维数组并指定 columns 和 index 创建 DataFrame 对象

```
import numpy as np
pd.DataFrame(np.random.randint(0,10,(3,2)),columns=list('ab'),index=list('efg'))
```

运行结果如图 3-31 所示：

```
In [24]: import numpy as np
         pd.DataFrame(np.random.randint(0,10,(3,2)),columns=list('ab'),index=list('efg'))
```

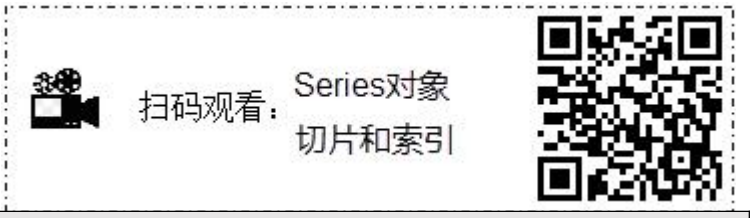
Out[24]:

	a	b
e	4	9
f	6	0
g	3	1

图 3-31 示例 3-14 运行效果图

3.2.3 获取 Series 对象的值

【示例 3-15】Series 对象的切片、索引




```
import numpy as np
import pandas as pd
data=pd.Series([4,3,25,2,3],index=list('abcde'))
display('根据 key 获取：',data['a'])
display('切片获取：',data['a':'d'])
display('索引获取：',data[1])
display('索引切片：',data[2:4])
```

运行结果如图 3-32 所示：

```
'根据key获取：'
4
'切片获取：'
a     4
b     3
c    25
d     2
dtype: int64
'索引获取：'
3
'索引切片：'
c    25
d     2
dtype: int64
```

图 3-32 示例 3-15 运行效果图

从示例 3-15 运行结果如图 3-32 所示，可以看出 Series 与 ndarray 数组都可以通过索引访问元素，Series 对象的索引分为位置索引和标签索引。不同之处，标签索引进行切片时候是左闭右闭，而位置索引是左闭右开。

【示例 3-16】位置索引与标签索引相同的问题

```
data=pd.Series([5,6,7,8],index=[1,2,3,4])
data[1]
```

如示例 3-16 所示，位置索引与标签索引有相同值 1，这时候 data[1]就不知道是按哪个来获取，此时要使用 loc、iloc。其中 loc 表示的是标签索引，iloc 表示的是位置索引。

【示例 3-17】Series 对象中 loc 与 iloc 的使用

```
data=pd.Series([5,3,2,5,9],index=[1,2,3,4,5])
data.loc[1]
data.iloc[1]
```

运行结果如图 3-33 所示：

```
In [44]: data=pd.Series([5,3,2,5,9],index=[1,2,3,4,5])
         data
Out[44]: 1    5
         2    3
         3    2
         4    5
         5    9
         dtype: int64

In [45]: data.loc[1]
Out[45]: 5

In [46]: data.iloc[1]
Out[46]: 3
```

图 3-33 示例 3-17 运行效果图

3.2.4 获取 DataFrame 的值

1) 选择某一列/某几列



DataFrame 对象非常容易获取数据集中指定列的数据。只需要在表 df 后面的括号中指明要选择的列名即可。如果要获取一列，则只需要传入一个列名；如果是同时选择多列，则传入多个列名即可，多个列名用一个 list 存放。

【示例 3-18】创建 DataFrame 对象

```
import numpy as np
import pandas as pd
data=pd.DataFrame(np.arange(12).reshape(3,4),index=list('abc'),columns=list('ABCD'))
```

运行结果如图 3-34 所示：

```
In [3]: import numpy as np
        import pandas as pd
        data=pd.DataFrame(np.arange(12).reshape(3,4),index=list('abc'),columns=list('ABCD'))
        data
```

Out[3]:

	A	B	C	D
a	0	1	2	3
b	4	5	6	7
c	8	9	10	11

图 3-34 示例 3-18 运行效果图

【示例 3-19】获取 DataFrame 对象中某一列/某几列

```
print('获取‘B’列：')
print(data['B'])
print('获取‘A’ ‘C’两列：')
print(data[['A','C']])
```

运行结果如图 3-35 所示：

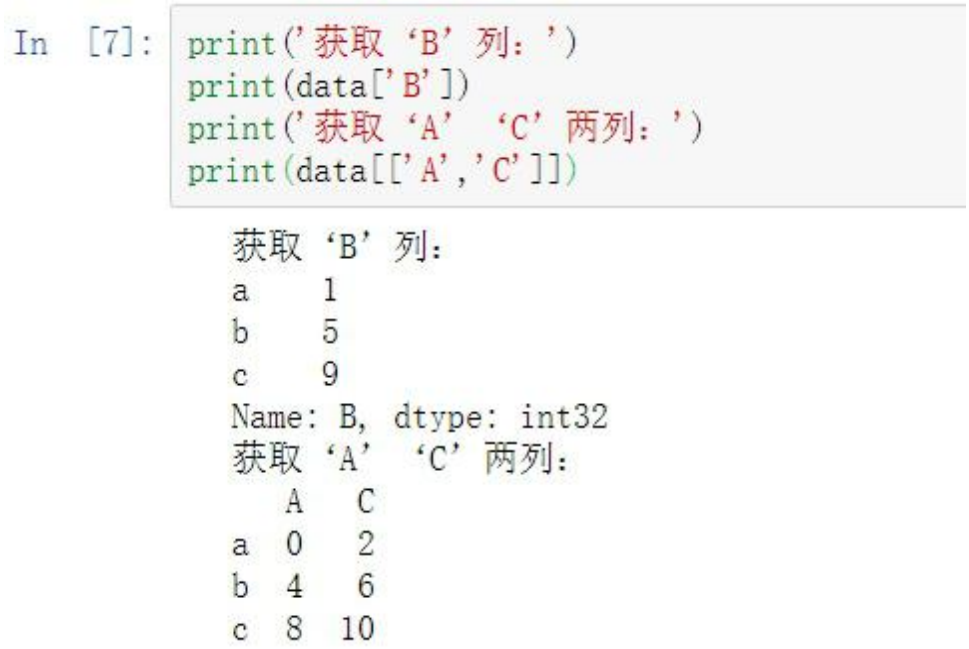


图 3-35 示例 3-19 运行效果图

DataFrame 对象获取列，除了传入具体的列名，还可以传入具体列的位置，通过传入位置来获取数据时需要用到 iloc 方法。

【示例 3-20】通过传入位置获取 DataFrame 对象中某一列/某几列

```
print('获取第 1 列：')
print(data.iloc[:,0])
print('获取第 1 列和第 3 列：')
print(data.iloc[:,[0,2]])
```

运行结果如图 3-36 所示：

```
[10]: print('获取第1列：')
      print(data.iloc[:,0])
      print('获取第1列和第3列：')
      print(data.iloc[:,[0,2]])
```

```
获取第1列：
a    0
b    4
c    8
Name: A, dtype: int32
获取第1列和第3列：
   A   C
a  0   2
b  4   6
c  8  10
```

图 3-36 示例 3-20 运行效果图

从上面的示例中可以看到，iloc 后的方括号中逗号之前的部分表示要获取的行的位置。只输入一个冒号，不输入任何数值表示获取所有的行；逗号之后的方括号表示要获取的列的位置，列的位置同样也是从 0 开始计数。

2) 选择连续的某几列

我们将通过列名选择数据的方式叫做普通索引，传入列的位置选择数据的方式叫做位置索引。获取连续的某几列，用普通索引和位置索引都可以做到。因为要获取的列是连续的，所以直接对列进行切片。

【示例 3-21】获取 DataFrame 对象中连续几列

```
print('获取 B C D 三列，使用普通索引获取：')
print(data.loc[:,'B':'D'])
print('获取 B C D 三列，使用位置索引获取：')
print(data.iloc[:,1:4])
```

运行结果如图 3-37 所示：

```
print('获取B C D三列，使用普通索引获取：')
print(data.loc[:,'B':'D'])
print('获取B C D三列，使用位置索引获取：')
print(data.iloc[:,1:4])
```

```
获取B C D三列，使用普通索引获取：
   B   C   D
a  1   2   3
b  5   6   7
c  9  10  11
获取B C D三列，使用位置索引获取：
   B   C   D
a  1   2   3
b  5   6   7
c  9  10  11
```

图 3-37 示例 3-21 运行效果图

从上面的示例可以看到，loc 和 iloc 后的方括号中逗号之前的表示选择的行，当只传入一个冒号时，表示选择所有行，逗号后面表示要选择列。data.loc[:,'B':'D']表示选择从 B 列开始到 D 列之间的值（包括 B 列也包括 D 列），data.iloc[:,1:4]表示选择第 2 列到第 5 列之间的值（包括第 1 列但不包括第 5 列）。

3) 选择某一行/某几行

获取行的方式主要有两种，一种是普通索引，即传入具体行索引的名称，需要用到 loc 方法；另外一种是位置索引，即传入具体的行数，需要用到 iloc 方法。



【示例 3-22】DataFrame 对象中选择某一行/某几行

```
print('获取 a 行,普通索引获取：')
print(data.loc['a'])
print('获取 a c 行,普通索引获取：')
print(data.loc[['a','c']])
print('获取第 1 行，位置索引获取：')
print(data.iloc[0])
print('获取第 1 行第 3 行，位置索引获取：')
print(data.iloc[[0,2]])
```

运行结果如图 3-38 所示：

```
获取a行,普通索引获取：
A    0
B    1
C    2
D    3
Name: a, dtype: int32
获取a c行,普通索引获取：
   A  B   C   D
a  0  1   2   3
c  8  9  10  11
获取第1行，位置索引获取：
A    0
B    1
C    2
D    3
Name: a, dtype: int32
获取第1行第3行，位置索引获取：
   A  B   C   D
a  0  1   2   3
c  8  9  10  11
```

图 3-38 示例 3-22 运行效果图

4) 选择连续的某几行

选择连续的某几行和选择连续某几列类似，只要把连续行的位置用一个区间表示即可。

【示例 3-23】DataFrame 对象中选择某一行/某几行

```
print('选择 a 行  b 行，使用普通索引：')
print(data.loc['a':'b'])
print('选择第 1 行  第 2 行，使用位置索引：')
print(data.iloc[0:2])
```

运行结果如图 3-39 所示：

```
选择a行 b行，使用普通索引：
   A  B  C  D
a  0  1  2  3
b  4  5  6  7
选择第1行 第2行，使用位置索引：
   A  B  C  D
a  0  1  2  3
b  4  5  6  7
```

图 3-39 示例 3-23 运行效果图

5) 行列同时选择

【示例 3-24】同时选择连续的部分行和部分列

```
print('同时获取 a b 行，A B 列，使用普通索引：')
print(data.loc['a':'b','A':'B'])
print('同时获取 a b 行，A B 列，使用位置索引：')
print(data.iloc[0:2,0:2])
```

运行结果如图 3-40 所示：

```
同时获取a b行，A B列，使用普通索引：
   A  B
a  0  1
b  4  5
同时获取a b行，A B列，使用位置索引：
   A  B
a  0  1
b  4  5
```

图 3-40 示例 3-24 运行效果图

【示例 3-25】同时选择不连续的部分行和部分列

```
print('同时获取 a c 行，ABD 列，使用普通索引：')
print(data.loc[['a','c'],['A','B','D']])
print('同时获取 a c 行，ABD 列，使用位置索引：')
print(data.iloc[[0,2],[0,1,3]])
```

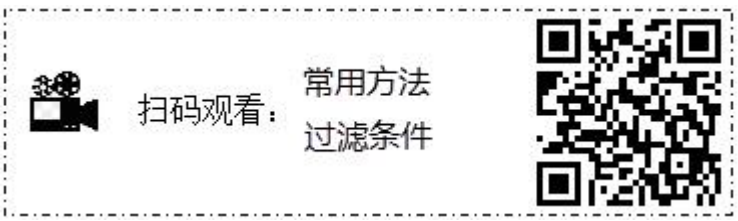
运行结果如图 3-41 所示：

```
同时获取a c行，ABD列，使用普通索引：
   A  B   D
a  0  1   3
c  8  9  11
同时获取a c行，ABD列，使用位置索引：
   A  B   D
a  0  1   3
c  8  9  11
```

图 3-41 示例 3-25 运行效果图

3.2.5 Series 的方法

Series 对象中有很多常用的方法可以对数据进行



各种处理。例如，mean 方法可以对某一列数据取平均数，min 方法获取最小值，max 方法获取最大值，std 方法获取标准差。

【示例 3-26】使用 mean、min、max、std 等方法对数据集进行各种运算，最后对数据集进行排序操作

```
import pandas as pd
data = pd.DataFrame({
    'Name':['zs','lisi','ww'],
    'Sno':['1001','1002','1003'],
    'Sex':['man','woman','man'],
    'Age':[17,18,19],
    'Score':[80,97,95]
},columns=['Sno','Sex','Age','Score'],index=['zs','lisi','ww'])
display('数据集',data)
ages = data['Age']
display('获取数据集中 Age 列的所有',ages)
print('计算 Age 列的平均值：',ages.mean())
print('计算 Age 列的最大值：',ages.max())
print('计算 Age 列的最小值：',ages.min())
print('计算 Age 列的标准差：',ages.std())
display('对 Age 进行降序排序：',ages.sort_values(ascending=False))
```

运行结果如图 3-42 所示：

'数据集'

	Sno	Sex	Age	Score
zs	1001	man	17	80
lisi	1002	woman	18	97
ww	1003	man	19	95

'获取数据集中Age列的所有'

```
zs      17
lisi    18
ww      19
Name: Age, dtype: int64

计算Age列的平均值：  18.0
计算Age列的最大值：  19
计算Age列的最小值：  17
计算Age列的标准差：  1.0

'对Age进行降序排序：'

ww      19
lisi    18
zs      17
Name: Age, dtype: int64
```

图 3-42 示例 3-26 运行效果图

3.2.6 Series 的条件过滤

Series 对象也可以像 SQL 语句一样，通过指定条件来过滤数据。

【示例 3-27】Series 对象指定条件来过滤数据

```
import pandas as pd
data = pd.DataFrame({
    'Name':['zs','lisi','ww'],
    'Sno':['1001','1002','1003'],
    'Sex':['man','woman','man'],
    'Age':[17,18,19],
    'Score':[80,97,95]
},columns=['Sno','Sex','Age','Score'],index=['zs','lisi','ww'])
```

```
display('数据集',data)
scores = data['Score']
display('筛选出成绩大于平均值的数据：',scores[scores>scores.mean()])
```

运行结果如图 3-43 所示：

'数据集'

	Sno	Sex	Age	Score
zs	1001	man	17	80
lisi	1002	woman	18	97
ww	1003	man	19	95

'筛选出成绩大于平均值的数据：'

```
lisi    97
ww      95
Name: Score, dtype: int64
```

图 3-43 示例 3-27 运行效果图

3.2.7 DataFrame 的条件过滤

DataFrame 与 Series 类似，也可以使用条件进行过滤。

【示例 3-28】DataFrame 对象指定条件来过滤数据

```
import pandas as pd
data = pd.DataFrame({
    'Name':['zs','lisi','ww'],
    'Sno':['1001','1002','1003'],
    'Sex':['man','woman','man'],
    'Age':[17,18,19],
    'Score':[80,97,95]
},columns=['Sno','Sex','Age','Score'],index=['zs','lisi','ww'])
display('数据集',data)
scores = data['Score']
display('输出数据中所有成绩大于平均值的记录',data[scores>scores.mean()])
display('获取成绩大于平均值得所有记录，只显示 Sno Age Score 三列：',data[scores>scores.mean()].loc[:,['Sno','Age','Score']])
```

运行结果如图 3-44 所示：

'数据集'

	Sno	Sex	Age	Score
zs	1001	man	17	80
lisi	1002	woman	18	97
ww	1003	man	19	95

'输出数据中所有成绩大于平均值的记录'

	Sno	Sex	Age	Score
lisi	1002	woman	18	97
ww	1003	man	19	95

'获取成绩大于平均值得所有记录，只显示Sno Age Score三列：'

	Sno	Age	Score
lisi	1002	18	97
ww	1003	19	95

图 3-44 示例 3-28 运行效果图

3.3 处理缺失值

缺失值就是由某些原因导致部分数据为空，对于为空的这部分数据我们一般有两种处理方式，一种是删除，即把含有缺失值的数据删除；另外一种是填充，即把缺失的那部分数据用某个值代替。

3.3.1 缺失值查看

对缺失值进行处理，首先要把缺失值找出来，也就是查看哪列有缺失值。直接调用 info()方法就会返回每一列的缺失情况。



【示例 3-29】缺失值查看

```
df=pd.DataFrame([[1,2,np.nan],[4,np.nan,6],[5,6,7]])
df.info()
```

运行结果如图 3-45 所示：

```
df=pd.DataFrame([[1, 2, np.nan], [4, np.nan, 6], [5, 6, 7]])
df
```

	0	1	2
0	1	2.0	NaN
1	4	NaN	6.0
2	5	6.0	7.0

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 3 columns):
0    3 non-null int64
1    2 non-null float64
2    2 non-null float64
dtypes: float64(2), int64(1)
memory usage: 152.0 bytes
```

图 3-45 示例 3-29 运行效果图

Pandas 中缺失值用 NaN 表示，从用 info()方法的结果来看，索引 1 这一列是 1　　2 non-null float64，表示这一列有 2 个非空值，而应该是 3 个非空值，说明这一列有 1 个空值。

还可以用 isnull()方法来判断哪个值是缺失值，如果是缺失值则返回 True，如果不是缺失值返回 False。

【示例 3-30】获取所有缺失值

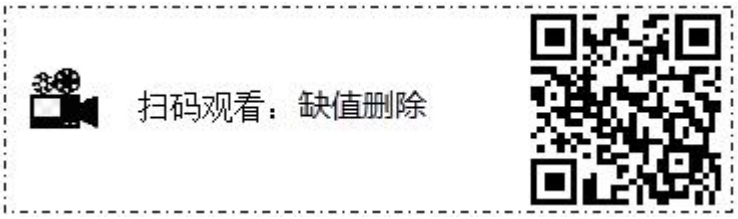
```
data=pd.Series([3,4,np.nan,1,5,None])
print('isnull()方法判断是否是缺值：')
print(data.isnull())
print('获取缺值：')
print(data[data.isnull()])
print('获取非空值')
print(data[data.notnull()])
```

运行结果如图 3-46 所示：

```
isnull()方法判断是否是缺值：
0    False
1    False
2     True
3    False
4    False
5     True
dtype: bool
获取缺值：
2   NaN
5   NaN
dtype: float64
获取非空值
0    3.0
1    4.0
3    1.0
4    5.0
dtype: float64
```

图 3-46 示例 3-30 运行效果图

3.3.2 缺失值删除



缺失值分为两种，一种是一行中某个字段是缺失值；另一种是一行中的字段全部为缺失值，即为一个空白行。调用 dropna()方法删除缺失值，dropna()方法默认删除含有缺失值的行，也就是只要某一行有缺失值就把这一行删除。如果想按列为单位删除缺失值，需要传入参数 axis=’columns’。

【示例 3-31】删除缺失值

```
df=pd.DataFrame([[1,2,np.nan],[4,np.nan,6],[5,6,7]])
print('默认为以行为单位剔除:')
display(df.dropna())
print('以列为单位剔除:')
display(df.dropna(axis='columns'))
```

运行结果如图 3-47 所示：

默认为以行为单位剔除：

	0	1	2
2	5	6.0	7.0

以列为单位剔除：

	0
0	1
1	4
2	5

图 3-47 示例 3-31 运行效果图

如果想删除空白行，需要给 dropna()方法中传入参数 how=’all’即可，默认值是 any。

【示例 3-32】删除空白行

```
df=pd.DataFrame([[1,2,np.nan],[4,np.nan,6],[5,6,7]])
print('所有为 nan 时候才剔除:')
display(df.dropna(how='all'))
print('默认情况，只要有就剔除')
display(df.dropna(how='any'))
```

运行结果如图 3-48 所示：

所有为nan时候才剔除：

	0	1	2
0	1	2.0	NaN
1	4	NaN	6.0
2	5	6.0	7.0

默认情况，只要有就剔除

	0	1	2
2	5	6.0	7.0

图 3-48 示例 3-32 运行效果图

3.3.3 缺失值填充

上面介绍了缺失值的删除，但是数据是宝贵的，一般情况下只要数据缺失比例不高（不大于 30%），尽量别删除，而是选择填充。



调用 fillna()方法对数据表中的所有缺失值进行填充，在 fillna()方法中输入要填充的值。还可以通过 method 参数使用前一个数和后一个数来进行填充。

【示例 3-33】Series 对象缺失值填充

```
data=pd.Series([3,4,np.nan,1,5,None])
print('以 0 进行填充:')
display(data.fillna(0))
print('以前一个数进行填充:')
display(data.fillna(method='ffill'))
print('以后一个数进行填充:')
display(data.fillna(method='bfill'))
print('先按前一个，再按后一个')
display(data.fillna(method='bfill').fillna(method='ffill'))
```

运行结果如图 3-49 所示：

```
以0进行填充:

0    3.0
1    4.0
2    0.0
3    1.0
4    5.0
5    0.0
dtype: float64

以前一个数进行填充:

0    3.0
1    4.0
2    4.0
3    1.0
4    5.0
5    5.0
dtype: float64

以后一个数进行填充:

0    3.0
1    4.0
2    1.0
3    1.0
4    5.0
5    NaN
dtype: float64
```

图 3-49 示例 3-33 运行效果图

【示例 3-34】DataFrame 对象缺失值填充

```
df=pd.DataFrame([[1,2,np.nan],[4,np.nan,6],[5,6,7]])
print('使用数值 0 来填充：')
display(df.fillna(0))
print('使用行的前一个数来填充：')
display(df.fillna(method='ffill'))
print('使用列的后一个数来填充：')
display(df.fillna(method='bfill' ,axis=1))
```

运行结果如图 3-50 所示：

使用数值0来填充：

	0	1	2
0	1	2.0	0.0
1	4	0.0	6.0
2	5	6.0	7.0

使用行的前一个数来填充：

	0	1	2
0	1	2.0	NaN
1	4	2.0	6.0
2	5	6.0	7.0

使用列的后一个数来填充：

	0	1	2
0	1.0	2.0	NaN
1	4.0	6.0	6.0
2	5.0	6.0	7.0

图 3-50 示例 3-34 运行效果图

【示例 3-35】列的平均值来填充

```
df=pd.DataFrame([[1,2,np.nan],[4,np.nan,6],[5,6,7]])
for i in df.columns:
    df[i]=df[i].fillna(np.nanmean(df[i]))
df
```

运行结果如图 3-51 所示：

	0	1	2
0	1	2.0	6.5
1	4	4.0	6.0
2	5	6.0	7.0

图 3-51 示例 3-35 运行效果图

3.4 拼接

扫码观看：拼接

【示例 3-36】Series 对象拼接

```
ser1=pd.Series([1,2,3],index=list('ABC'))
ser2=pd.Series([4,5,6],index=list('DEF'))
pd.concat([ser1,ser2])
```

运行结果如图 3-52 所示：

```
In [4]: ser1=pd.Series([1,2,3],index=list('ABC'))
        ser2=pd.Series([4,5,6],index=list('DEF'))
        pd.concat([ser1,ser2])

Out[4]: A    1
        B    2
        C    3
        D    4
        E    5
        F    6
        dtype: int64
```

图 3-52 示例 3-36 运行效果图

【示例 3-37】两个 df 对象拼接，默认找相同的列索引进行合并

```
def make_df(cols,index):
    data={c:[str(c)+str(i) for i in index] for c in cols}
    return pd.DataFrame(data,index=index)
df1=make_df('AB',[1,2])
df2=make_df('AB',[3,4])
pd.concat([df1,df2])
```

运行结果如图 3-53 所示：

```
In [5]: df1=make_df('AB',[1,2])
        df2=make_df('AB',[3,4])
        pd.concat([df1,df2])
```

Out[5]:

	A	B
1	A1	B1
2	A2	B2
3	A3	B3
4	A4	B4

图 3-53 示例 3-37 运行效果图

【示例 3-38】两个 df 对象拼接，添加 axis 参数

```
df1=make_df('AB',[1,2])
df2=make_df('AB',[3,4])
pd.concat([df1,df2],axis=1) #或者 pd.concat([df1,df2],axis='columns')
```

运行结果如图 3-54 所示：

```
In [6]: pd.concat([df1,df2],axis=1)
```

Out[6]:

	A	B	A	B
1	A1	B1	NaN	NaN
2	A2	B2	NaN	NaN
3	NaN	NaN	A3	B3
4	NaN	NaN	A4	B4

```
In [7]: pd.concat([df1,df2],axis='columns')
```

Out[7]:

	A	B	A	B
1	A1	B1	NaN	NaN
2	A2	B2	NaN	NaN
3	NaN	NaN	A3	B3
4	NaN	NaN	A4	B4

图 3-54 示例 3-38 运行效果图

【示例 3-39】两个 df 对象拼接，索引重复问题

```
x=make_df('AB',[1,2])
y=make_df('AB',[1,2])
pd.concat([x,y])
```

运行结果如图 3-55 所示：

```
In [9]: x=make_df('AB',[1,2])
        y=make_df('AB',[1,2])
        pd.concat([x,y])
```

Out[9]:

	A	B
1	A1	B1
2	A2	B2
1	A1	B1
2	A2	B2

图 3-55 示例 3-39 运行效果图

【示例 3-40】两个 df 对象拼接，解决索引重复问题加 ignore_index 属性

```
x=make_df('AB',[1,2])
y=make_df('AB',[1,2])
pd.concat([x,y],ignore_index=True)
```

运行结果如图 3-56 所示：

```
In [10]: pd.concat([x,y],ignore_index=True)
Out[10]:
```

	A	B
0	A1	B1
1	A2	B2
2	A1	B1
3	A2	B2

图 3-56 示例 3-40 运行效果图

【示例 3-41】两个 df 对象拼接，解决索引重复问题，加 keys 属性

```
x=make_df('AB',[1,2])
y=make_df('AB',[1,2])
pd.concat([x,y],keys=list('xy'))
```

运行结果如图 3-57 所示：

```
In [12]: pd.concat([x,y],keys=list('xy'))
Out[12]:
```

		A	B
x	1	A1	B1
	2	A2	B2
y	1	A1	B1
	2	A2	B2

图 3-57 示例 3-41 运行效果图

【示例 3-42】两个 df 对象拼接，join 内连接做交集

```
a=make_df('ABC',[1,2])
b=make_df('BCD',[3,4])
pd.concat([a,b],join='inner')
```

运行结果如图 3-58 所示：

```
In [18]: pd.concat([a,b],join='inner')
Out[18]:
```

	B	C
1	B1	C1
2	B2	C2
3	B3	C3
4	B4	C4

图 3-58 示例 3-42 运行效果图

3.5 merge 的使用

pandas 中的 merge 和 concat 类似，但主要是用于两组有 key column 的数据，统一索引的数据。通常也被用在 Database 的处理当中。



合并时有 4 种方式 how = ['left', 'right', 'outer', 'inner']，默认值 how='inner'。

【示例 3-43】merge 的使用，默认以 how=’inner’进行合并

```
left=pd.DataFrame({'key':['k0','k1','k2','k3'],
                   'A':['A0','A1','A2','A3'],
                   'B':['B0','B1','B2','B3'],
                  })
right=pd.DataFrame({'key':['k0','k1','k4','k3'],
                   'C':['C0','C1','C2','C3'],
                   'D':['D0','D1','D2','D3'],
                  })
result=pd.merge(left,right)
```

运行结果如图 3-59 所示：

```
left=pd.DataFrame({'key':['k0','k1','k2','k3'],
                   'A':['A0','A1','A2','A3'],
                   'B':['B0','B1','B2','B3'],
                  })
right=pd.DataFrame({'key':['k0','k1','k4','k3'],
                   'C':['C0','C1','C2','C3'],
                   'D':['D0','D1','D2','D3'],
                  })
result=pd.merge(left,right,how='inner')
```

```
result
```

]:

	key	A	B	C	D
0	k0	A0	B0	C0	D0
1	k1	A1	B1	C1	D1
2	k3	A3	B3	C3	D3

图 3-59 示例 3-43 运行效果图

【示例 3-44】merge 的使用，参数 how=’outer’进行合并

```
left=pd.DataFrame({'key':['k0','k1','k2','k3'],
                   'A':['A0','A1','A2','A3'],
                   'B':['B0','B1','B2','B3'],
                  })
right=pd.DataFrame({'key':['k0','k1','k4','k3'],
                   'C':['C0','C1','C2','C3'],
                   'D':['D0','D1','D2','D3'],
                  })
pd.merge(left,right,how='outer')
```

运行结果如图 3-60 所示：

```
left=pd.DataFrame({'key':['k0','k1','k2','k3'],
                   'A':['A0','A1','A2','A3'],
                   'B':['B0','B1','B2','B3'],
                  })
right=pd.DataFrame({'key':['k0','k1','k4','k3'],
                   'C':['C0','C1','C2','C3'],
                   'D':['D0','D1','D2','D3'],
                  })
pd.merge(left,right,how='outer')
```

]:

	key	A	B	C	D
0	k0	A0	B0	C0	D0
1	k1	A1	B1	C1	D1
2	k2	A2	B2	NaN	NaN
3	k3	A3	B3	C3	D3
4	k4	NaN	NaN	C2	D2

图 3-60 示例 3-44 运行效果图

【示例 3-45】merge 的使用，参数 how=’left’进行合并

```
left=pd.DataFrame({'key':['k0','k1','k2','k3'],
                   'A':['A0','A1','A2','A3'],
                   'B':['B0','B1','B2','B3'],
                  })
right=pd.DataFrame({'key':['k0','k1','k4','k3'],
                   'C':['C0','C1','C2','C3'],
                   'D':['D0','D1','D2','D3'],
                  })
pd.merge(left,right,how='left')
```

运行结果如图 3-61 所示：

```
left=pd.DataFrame({'key':['k0','k1','k2','k3'],
                   'A':['A0','A1','A2','A3'],
                   'B':['B0','B1','B2','B3'],
                  })
right=pd.DataFrame({'key':['k0','k1','k4','k3'],
                   'C':['C0','C1','C2','C3'],
                   'D':['D0','D1','D2','D3'],
                  })
pd.merge(left,right,how='left')
```

i]:

	key	A	B	C	D
0	k0	A0	B0	C0	D0
1	k1	A1	B1	C1	D1
2	k2	A2	B2	NaN	NaN
3	k3	A3	B3	C3	D3

图 3-61 示例 3-45 运行效果图

【示例 3-46】行索引不同， how=’right’进行合并

```
left=pd.DataFrame({'key':['k0','k1','k2','k3'],
                   'A':['A0','A1','A2','A3'],
                   'B':['B0','B1','B2','B3'],
                  })
right=pd.DataFrame({'key':['k0','k1','k4','k3'],
                   'C':['C0','C1','C2','C3'],
                   'D':['D0','D1','D2','D3'],
                  })
pd.merge(left,right,how=’right’)
```

运行结果如图 3-62 所示：

```
left=pd.DataFrame({'key':['k0','k1','k2','k3'],
                   'A':['A0','A1','A2','A3'],
                   'B':['B0','B1','B2','B3'],
                  })
right=pd.DataFrame({'key':['k0','k1','k4','k3'],
                   'C':['C0','C1','C2','C3'],
                   'D':['D0','D1','D2','D3'],
                  })
pd.merge(left,right,how='right')
```

]:

	key	A	B	C	D
0	k0	A0	B0	C0	D0
1	k1	A1	B1	C1	D1
2	k3	A3	B3	C3	D3
3	k4	NaN	NaN	C2	D2

图 3-62 示例 3-46 运行效果图

习题

一、简答题

1. pandas 库的重要特征是什么？
2. 如何计算 Series 的标准偏差？
3. 在 pandas 中创建 DataFrame 有哪些不同方式？
4. 如何获得数字序列的最小值，第 25，中位数，第 75 和最大值？
5. 如何将 DataFrame 转换为 Excel 文件？

二、编码题

1. 设有如下 data 数据，完成操作：

```
data =  {‘xiaoming’:10,’xiaolan’:80,’black’:77}
```

a. 创建 Series 对象。

b. 获取 values、index 属性。

c. 获取’xiaolan’的值，分别用 loc 和 iloc 获取。

d. 使用切片获取值为 10,80 的数据。

2. 设有如下 data 数据，完成操作：

```
data = {'animal': ['cat', 'cat', 'snake', 'dog', 'dog', 'cat', 'snake', 'cat', 'dog', 'dog'],
        'age': [2.5, 3, 0.5, np.nan, 5, 2, 4.5, np.nan, 7, 3],
        'visits': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
        'priority': ['yes', 'yes', 'no', 'yes', 'no', 'no', 'no', 'yes', 'no', 'no']}
labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
```

a. 创建 DataFrame 对象。

b. 获取前三行的数据。

c. 获取所有行，'animal'和'age'列。

d. 获取 3,4,8 行，'animal'和'age'列。

e. 获取 'visits'的值大于等于 3 的数据。

f. 获取'animal'的值是’cat’的数据。

g. 获取'age'不为空的数据。

第四章 网络爬虫基础

在数据量爆发式增长的互联网时代，网站与用户的沟通本质上是数据的交换：搜索引擎从数据库中提取搜索结果，将其展现在用户面前；电商将产品的描述、价格展现在网站上，以供买家选择心仪的产品；社交媒体在用户生态圈的自我交互下产生大量文本、图片和视频数据等。这些数据如果得以分析利用，不仅能够帮助第一方企业（拥有这些数据的企业）做出更好的决策，对于第三方企业也是有益的。而网络爬虫技术，则是大数据分析领域的第一个环节。

4.1 初识网络爬虫

4.1.1 什么是网络爬虫



如果我们把互联网比作一张大的蜘蛛网，那一台计算机上的数据便是蜘蛛网上的一个猎物，而爬虫程序就是一只小蜘蛛，沿着蜘蛛网抓取自己想要的猎物/数据。

通俗理解：爬虫是一个模拟人类请求网站行为的程序。可以自动请求网页、并把数据抓取下来，然后使用一定的规则提取有价值的数据。

比如百度搜索引擎的爬虫叫做百度蜘蛛（Baiduspider）。百度蜘蛛每天会在海量的互联网信息中爬取，爬取优质信息并搜寻，当用户在百度搜索引擎上检索对应关键词时，百度将对关键词进行分析处理，从收录的网页中找出相关网页，按照一定的排名规则进行排序并将结果展现给用户。

除了百度搜索引擎离不开爬虫以外，其他搜索引擎也离不开爬虫，它们也拥有自己的爬虫。比如 360 的爬虫叫 360Spider，搜狗的爬虫叫 Sogouspider，必应的爬虫叫 Bingbot。

大数据时代也离不开爬虫，比如在进行大数据分析或数据挖掘时，可以去一些比较大型的官方站点下载数据源。但这些数据源比较有限，那么如何才能获取更多更高质量的数据源呢？此时，我们可以编写自己的爬虫程序，从互联网中进行数据信息的获取。

4.1.2 能从网络上爬取什么数据

简单来说，平时在浏览网站时，所有能见到的数据都可以通过爬虫程序保存下来。从社交媒体的每一条发帖到团购网站的价格及点评，再到招聘网站的招聘信息，这些数据都可以存储下来。

4.1.3 应不应该学爬虫

应不应该学爬虫？这也是我之前问自己的一个问题，我认为对于任何一个与互联网有关的从业人员，无论是非技术的产品、运营或营销人员，还是前断、后端的程序员，都应该学习网络爬虫技术。

一方面，网络爬虫简单易学、门槛很低。没有任何编程基础的人在认真看完本章的爬虫基础内容后，都能够自己完成简单的网络爬虫任务，从网站上自动获取需要的数据。

另一方面，网络爬虫不仅仅能使你学会一项新的技术，还能让你在工作的时候节省大量的时间。如果你对网络爬虫的世界有兴趣，就算你不懂编程也不要担心，本章将深入浅出讲解网络爬虫。

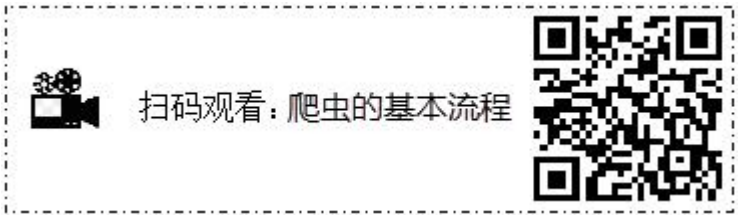
4.1.4 网络爬虫是否合法

网络爬虫领域目前还属于早期的拓荒阶段，虽然互联网世界已经通过自身的协议建立起一定的道德规范（Robots 协议），但法律部分还在建立和完善中。从目前情况来看，如果抓取的数据属于个人使用或科研范畴，基本不存在问题；而如果数据属于商业盈利范畴，就要就事论事，有可能属于违法行为，也有可能不违法。

除了上述 Robots 协议之外，我们使用网络爬虫的时候还要对自己进行约束：过于快速或者频密的网络爬虫都会对服务器产生巨大的压力，网站可能封锁你的 IP，甚至采取进一步的法律行动。因此，需要约束自己的网络爬虫行为，将请求的速度限定在一个合理的范围之内。

4.2 爬虫的基本流程

网络爬虫的基本工作流程如下：



1) 浏览器通过 DNS 服务器查找域名对应的 IP 地址。
2) 向 IP 地址对应的 Web 服务器发送请求 。
3) Web 服务器响应请求，发回 HTML 页面 。
4) 浏览器解析 HTML 内容，并显示出来。

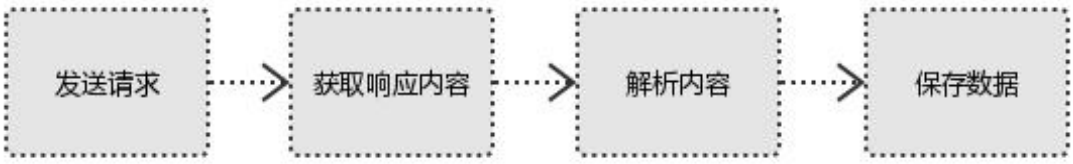


图 4-1 网络爬虫的基本工作流程

(1) DNS

- DNS 是计算机域名系统 (Domain Name System (Domain Name System 或

Domain Name Service) 的缩写，由解析器和域名服务组成的。

- 域名服务器是指保存有该网络中所主机的和对应 IP 地址，并具有将域名转换为 IP 地址功能的服务器。
- 一个域名的 DNS 解析时间在 10~60 毫秒之间。
- 一个域名必须对应 IP 地址，而一个 IP 地址不一定会有域名

(2) HTTP 和 HTTPS

- HTTP 协议（HyperText Transfer Protocol，超文本传输协议）是一种发布和接收 HTML 页面的方法。
- HTTPS 协议（HyperText Transfer Protocol over Secure Socket Layer）简单的讲就是 HTTP 的安全版，在 HTTP 下加入 SSL 层。
- SSL（Secure Socket Layer 安全套接层）主要用于 Web 的安全传输协议，在传输层对网络连接进行加密，保障在 Internet 上数据传输的安全。
- HTTP 的端口号是 80，HTTPS 的端口号是 443。

(3) URI 与 URL

- URI（Uniform Resource Identifier）统一资源标志符。
- URL（Universal Resource Locator）统一资源定位符，用于完整地描述 Internet 上网页和其他资源的地址的一中标识方法。
- URL 的基本格式：scheme://host[:port]/path/.../[?query-string][#anchor]。

4.3 HTTP 协议



HTTP 协议（HyperText Transfer Protocol，超文本传输协议）是用于 WWW 服务器传输超文本到本地浏览器的传送协议。它可以使浏览器更加高效，减少网络传输。它不仅保证计算机正确快速传输超文本文档，还确定传输文档中的哪一部分，以及哪部分内容首先显示（如文本先于图形）等。之后的 Python 爬虫开发，主要就是和 HTTP 协议打交道。

4.3.1 HTTP 请求过程

HTTP 协议采取的是请求响应模型，HTTP 协议永远都是客户端发起请求，服务器回送响应。模型如图 4-2 所示。

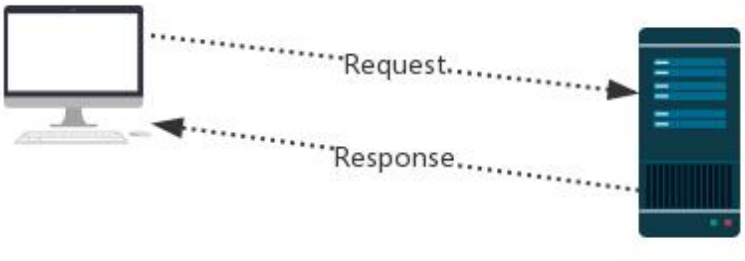


图 4-2 请求响应模型

HTTP 协议是一个无状态的协议，同一个客户端的这次请求和上次请求没有对应关系。一次 HTTP 操作称为一个事物，其执行过程可分为四步：

- 首先客户端与服务器建立连接，例如单击某个超链接。
- 建立连接后，客户端发送一个请求给服务器，请求方式的格式为：统一资源标识符（URL）、协议版本号，后边是 MIME 信息，包括请求修饰符、客户机信息和可能的内容。
- 服务器接到请求后，给予相应的响应信息，其格式为一个状态行，包括信息的协议版本号、一个成功或错误的代码，后边是 MIME 信息，包括服务器信息、实体信息和可能的内容。
- 客户端接收服务器所返回的信息，通过浏览器将信息显示在用户的显示屏上，然后客户端与服务器端断开连接。

4.3.2 HTTP 状态码

当浏览者访问一个网页时，浏览者的浏览器会向网页所在的服务器发出请求。在浏览器接收并显示网页前，此网页所在的服务器会返回一个包含 HTTP 状态码的信息头（server header）用以响应浏览器的请求。HTTP 状态码主要是为了标识此次 HTTP 请求的运行状态。下面是常见的 HTTP 状态码：

- 200———请求成功。
- 301———资源被永久转移到其他 URL。
- 404———请求的资源不存在。
- 500———内部服务器错误。

HTTP 状态码由三个十进制数字组成，第一个十进制数字定义了状态码的类型。HTTP 状态码分为 5 中类型，如表 4-1 所示。

表 4-1 HTTP 状态码

分类	分类描述
1**	信息，服务器收到请求，需要请求者继续执行操作
2**	成功，操作被成功接收并处理
3**	重定向，需要进一步的操作以完成请求

分类	分类描述
4**	客户端错误，请求包含语法错误或无法完成请求
5**	服务器错误，服务器在处理请求的过程中发生了错误

4.3.3 HTTP 头部信息

HTTP 头部信息由众多的头域组成，每个头域由一个域名、冒号（：）和域值三部分组成。域名是大小写无关的，域值前可以添加任何数量的空格符，头域可以被扩展为多行，在每行开始处，使用至少一个空格或制表符。

通过浏览器访问百度首页时，使用 F12 打开开发者工具，里面可以监控整个 HTTP 访问的过程。下面就以访问百度的 HTTP 请求进行分析，首先是浏览器发出请求，请求头的数据如下：

```
GET / HTTP/1.1
Host: www.baidu.com
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.113 Mobile Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9
```

在请求头中包含以下内容：

- GET 代表的是请求方式，HTTP/1.1 表示使用 HTTP1.1 协议标准。
- Host 头域，用于指定请求资源的 Intenet 主机和端口号，必须表示请求 URL 的原始服务器或网关的位置。HTTP/1.1 请求必须包含主机头域，否则系统会以 400 状态码返回。
- User-Agent 头域，里面包含发出请求的用户信息，其中有使用的浏览器型号、版本和操作系统的信息。这个头域经常用来作为反爬虫的措施。
- Accept 请求报头域，用于指定客户端接受哪些类型的信息。例如：Accept：image/gif，表明客户端希望接受 GIF 图像格式的资源；Accept：text/html，表明客户端希望接受 html 文本。
- Accept-Language 请求报头域，类似于 Accept，但是它用于指定一种自然语言。例如：Accept-Language：zh-ch。如果请求消息中没有设置这个报头域，服务器假定客户端对各种语言都可以接受。

- Accept-Encoding 请求报头域，类似于 Accept，但是它用于指定可接受的内容编码。
- Connection 报头域允许发送用于指定连接的选项。例如指定连接的状态是连续，或者指定“close”选项，通知服务器，在响应完成后，关闭连接。

请求发送成功后，服务器进行响应，响应的信息，数据如下：

```
HTTP/1.1 200 OK
Cache-Control: no-cache
Connection: keep-alive
Content-Encoding: gzip
Content-Type: text/html;charset=utf-8
Coremonitorno: 0
Date: Mon, 18 May 2020 09:00:11 GMT
Vary: Accept-Encoding
Transfer-Encoding: chunked
```

- HTTP/1.1 表示使用 HTTP1.1 协议标准，200 OK 说明请求成功。
- Date 表示消息产生的日期和时间。
- Content-Type 实体报头域用于指明发送给接收者的实体正文的媒体类型。text/html;charset=utf-8 代表 HTML 文本文档，UTF-8 编码。
- Transfer-Encoding: chunked 表示输出的内容长度不能确定。
- Connection 报头域允许发送用于指定连接的选项。例如指定连接的状态是连续，或者指定“close”选项，通知服务器，在响应完成后，关闭连接。
- Vary 头域指定了一些请求头域，这些请求头域用来决定当缓存中存在一个响应，并且该缓存没有过期失效时，是否被允许利用此响应去回复后续请求而不需要重复验证。
- Cache-Control 用于指定缓存指令，缓存指令是单向的，且是独立的。
- Content-Encoding 实体报头域被用作媒体类型的修饰符，它的值表示了已经被应用到实体正文的附加内容的编码，因而要获得 Content-Type 报头域中所引用的媒体类型，必须采用响应的解码机制。

4.3.4 HTTP 请求方式

HTTP 协议请求主要分为 6 中类型，各类型的主要作用如下：



1) GET 请求：GET 请求会通过 URL 网址传递信息，可以直接在 URL 中写上要传递的信息，也可以由表单进行传递。如果使用表单进行传递，这表单中的信息会自动转为 URL 地址中的数据，通过 URL

地址传递。

2) POST 请求：可以向服务器提交数据，是一种比较主流也比较安全的数据传递方式，比如在登录时，经常使用 POST 请求发送数据。

3) PUT 请求：请求服务器存储一个资源，通常要指定存储的位置。

4) DELETE 请求：请求服务器删除一个资源。

5) HEAD 请求：请求获取对应的 HTTP 报头信息。

6) OPTIONS 请求：可以获得当前 URL 所支持的请求类型。

除此之外，还有 TRACE 请求与 CONNECT 请求等，TRACE 请求主要用于测试或诊断。由于用得非常少，所以这里不再提及。

接下来，我们将通过实例讲解 HTTP 协议请求中的 GET 请求和 POST 请求，这两种是最常用的请求方式。

a) GET 请求实例分析

有时候在百度上查询一个关键词，我们会打开百度首页，并输入该关键词进行查询，比如输入“Python”，然后按回车键，此时会出现对应的查询结果，观察一下 URL 的变化，如下所示。

```
https://www.baidu.com/s?wd=Python&rsv_spt=1&rsv_iqid=0x8a7b64fb0001c521&issp=1&f=8&rsv_bp=1&rsv_idx=2&ie=utf-8&rqlang=cn&tn=baiduhome_pg&rsv_enter=1&rsv_dl=tb&oq=%25E7%25BD%2591%25E7%25BB%259C%25E7%2588%25AC%25E8%2599%25AB%25E7%259A%2584%25E5%25A5%25BD%25E5%25A4%2584&rsv_t=87f3Kz6nUiM7HXHQTzBpxWSGowc4u4IuzpZOSjJlWWAjkruEp0ZptPDzt43dqEfbewEw&rsv_btype=t&inputT=3469&rsv_pq=b71284a20002123b&rsv_sug3=39&rsv_sug1=31&rsv_sug7=100&rsv_sug2=0&rsv_sug4=69751414&rsv_sug=2
```

可以发现，对应的查询信息是通过 URL 传递的，这里采用的是 HTTP 请求中的 GET 方法。其中字段 ie 的值为 utf-8，代表的是编码信息，而字段 wd 的值为 Python，代表的是用户检索的关键词。

b) POST 请求实例分析

POST 请求则是常用在登录、注册等操作的时候，提交的数据放置在实体区内提交。下面展示一个完整的 POST 请求。图 4-3 是为大家提供的一个 POST 表单测试网页，输入用户名和密码后点击登录。使用 F12 打开开发者工具，捕获的请求数据如图 4-4 所示：



表 4-3 POST 表单测试网页



表 4-4 POST 请求数据

c) POST 请求与 GET 请求的区别

1、GET 请求，请求的数据会附加在 URL 之后，以?分割 URL 和传输数据，多个参数用&连接。POST 请求：POST 请求会把请求的数据放置在 HTTP 请求包的包体中。因此，GET 请求的数据会暴露在地址栏中，而 POST 请求则不会。

2、传输数据的大小

对于 GET 请求方式提交的数据最多只能有 1024 字节，而 POST 请求没有限制。

3、安全性问题

使用 GET 请求的时候，参数会显示在地址栏上，而 POST 请求不会。所以，如果这些数据是非敏感数据，那么使用 GET；如果用户输入的数据包含敏感数据，那么还是使用 POST 为好。

4.3.5 请求

Requests 是用 python 语言基于 urllib 编写的，采用的是 Apache2 Licensed 开源协议的 HTTP 库。

1) Requests 库安装

以“管理员身份运行 cmd”，直接使用 pip 进行安装，语法格式如下：

```
pip install requests
```

2) Requests 库的 7 个主要方法

表 4-2 Requests 库的主要方法

方法	说明
requests.request()	构造一个请求，支撑一下个方法的基础方法。
requests.get()	获取 HTML 网页的主要方法，对应 HTTP 的 GET
requests.head()	获取 HTML 网页投信息的方法，对应 HTTP 的 HEAD
requests.post()	向 HTML 网页提交 POST 请求的方法，对应 HTTP 的 POST
requests.put()	向 HTML 网页提交 PUT 请求的方法，对应 HTTP 的 PUT
requests.patch()	向 HTML 网页提交局部修改请求，对应 HTTP 的 PATCH
requests.delete()	向 HTML 网页提交删除请求，对应 HTTP 的 DELETE

3) requests.request()

```
requests.request(method,url,**kwargs)
```

其中参数 method 表示请求方式，对应 get/put/post 等 7 种；url 表示获取页面的 url 链接；

**kwargs 表示控制访问的参数，均为可选项。

4) Requests 库的 get()方法

```
requests.get(url,params=None,**kwargs)
```

其中参数 url 表示获取页面的 url 链接；params 表示 url 中额外参数，字典或字节流格式，是可选参数；**kwargs 表示 12 个控制访问的参数。

4.3.6 响应

Response 对象包含服务器返回的所有信息，分为三部分：响应状态码、响应头、响应内容。

表 4-3 Response 对象的属性

属性	说明
r.status_code	HTTP 请求的返回状态，200 表示连接成功，404 表示失败
r.text	HTTP 响应内容的字符串形式，即，url 对应的页面内容
r.encoding	从 HTTP header 中猜测的相应内容编码方式
r.apparent_encoding	从内容中分析出的相应内容编码方式（备选编码方式）
r.content	HTTP 响应内容的二进制形式
r.encoding	如果 header 中不存在 charset，则认为编码为 ISO-8859-1
r.apparent_encoding	根据网页内容分析出的编码方式可以 看作是 r.encoding 的备选

Response 的编码：

1) r.encoding：从 HTTP header 中猜测的响应内容的编码方式；如果 header 中不存在 charset，则认为编码为 ISO-8859-1，r.text 根据 r.encoding 显示网页内容。
2) r.apparent_encoding：根据网页内容分析出的编码方式，可以看作 r.encoding 的备选。

【示例 4-1】第一个简单的爬虫

```
import requests
r = requests.get('http://www.baidu.com')
print('content:',r.content)
print('text:',r.text)
```

执行结果如下所示：

```
<!DOCTYPE html>
<!--STATUS OK--><html> <head><meta http-equiv=content-type content=text/html;charset=utf-8><meta http-equiv=X-UA-Compatible content=IE=Edge><meta content=always name=referrer><link rel=stylesheet type=text/css href=http://s1.bdstatic.com/r/www/cache/bdorz/baidu.min.css><title>ç™¾åº¦ä¸€ä¸‹ï¼Œä½ å°±
```

```
¥é “</title></head> <body link=#0000cc> <div id=wrapper> <div id=head> <div class=head_wrapper> <div class=s_form> <div class=s_form_wrapper> <div id=lg> <img hidefocus=true src=//www.baidu.com/img/bd_logo1.png width=270 height=129> </div> <form id=form name=f action=//www.baidu.com/s class=fm> <input type=hidden name=bdorz_come value=1> <input type=hidden name=ie value=utf-8> <input type=hidden name=f value=8> <input type=hidden name=rsv_bp value=1> <input type=hidden name=rsv_idx value=1> <input type=hidden name=tn value=baidu><span class="bg s_ipt_wr"><input id=kw name=wd class=s_ipt value maxlength=255 autocomplete=off autofocus></span><span class="bg s_btn_wr"><input type=submit id=su value=ç™¾å°ä¸‚ ä¸‹ class="bg s_btn"></span> </form> </div> </div> <div id=u1> <a href=http://news.baidu.com name=tj_trnews class=mnav>æ–°é—»</a> <a href=http://www.hao123.com name=tj_trhao123 class=mnav>hao123</a> <a href=http://map.baidu.com name=tj_trmap class=mnav>åœ°å›¾</a> <a href=http://v.baidu.com name=tj_trvideo class=mnav>è§†é¢‘</a> <a href=http://tieba.baidu.com name=tj_trtieba class=mnav>è´´å§</a> <noscript> <a href=http://www.baidu.com/bdorz/login.gif?login&amp;tpl=mn&amp;u=http%3A%2F%2Fwww.baidu.com%2f%3fbdorz_come%3d1 name=tj_login class=lb>ç™»å½•</a> </noscript> <script>document.write('<a href="http://www.baidu.com/bdorz/login.gif?login&tpl=mn&u='+ encodeURIComponent(window.location.href+ (window.location.search === "" ? "?" : "&")+ "bdorz_come=1")+ '" name="tj_login" class="lb">ç™»å½•</a>');</script> <a href=//www.baidu.com/more/ name=tj_briicon class=bri style="display: block;">æ›´å¤šäº§å“ </a> </div> </div> </div> <div id=ftCon> <div id=ftConw> <p id=lh> <a href=http://home.baidu.com>å…³äº ç™¾å°ä¸‚</a> <a href=http://ir.baidu.com>About Baidu</a> </p> <p id=cp>&copy;2017 Baidu <a href=http://www.baidu.com/duty/>ä½¿ç”¨ç™¾å°ä¸‚å‰ å¿…è¯»</a>  <a href=http://jianyi.baidu.com/
class=cp-feedback>æ„ è§ å é¦ˆ</a> äº¬ICPè¯ 030173å ·  <img src=//www.baidu.com/img/gs.gif> </p> </div> </div> </div> </body> </html>
```

上述代码是获取百度首页的 HTML 代码，首先导入包 requests，之后获取网页内容。r 是 requests 的 Response 回复对象，调用该对象的 text 属性可以获取网页内容代码。但发现获取的内容乱码，需要修改编码，示例如下。

【示例 4-2】Response 设置编码

```
import requests
r = requests.get('http://www.baidu.com')
```

```
print('content:',r.content)
r.encoding='utf-8'
print('new text：',r.text)
```

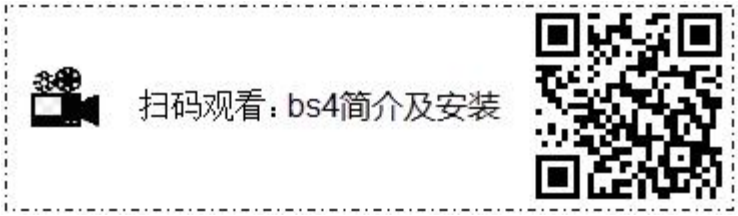
执行结果如下所示：

```
new text： <!DOCTYPE html>
<!--STATUS OK--><html> <head><meta http-equiv=content-type content=text/html;charset=utf-8><meta http-equiv=X-UA-Compatible content=IE=Edge><meta content=always name=referrer><link rel=stylesheet type=text/css href=http://s1.bdstatic.com/r/www/cache/bdorz/baidu.min.css><title>百度一下，你就知道</title></head> <body link=#0000cc> <div id=wrapper> <div id=head> <div class=head_wrapper> <div class=s_form> <div class=s_form_wrapper> <div id=lg> <img hidefocus=true src=//www.baidu.com/img/bd_logo1.png width=270 height=129> </div> <form id=form name=f action=//www.baidu.com/s class=fm> <input type=hidden name=bdorz_come value=1> <input type=hidden name=ie value=utf-8> <input type=hidden name=f value=8> <input type=hidden name=rsv_bp value=1> <input type=hidden name=rsv_idx value=1> <input type=hidden name=tn value=baidu><span class="bg s_ipt_wr"><input id=kw name=wd class=s_ipt value maxlength=255 autocomplete=off autofocus></span><span class="bg s_btn_wr"><input type=submit id=su value=百度一下 class="bg s_btn"></span> </form> </div> </div> <div id=u1> <a href=http://news.baidu.com name=tj_trnews class=mnav>新闻</a> <a href=http://www.hao123.com name=tj_trhao123 class=mnav>hao123</a> <a href=http://map.baidu.com name=tj_trmap class=mnav>地图</a> <a href=http://v.baidu.com name=tj_trvideo class=mnav>视频</a> <a href=http://tieba.baidu.com name=tj_trtieba class=mnav>贴吧</a> <noscript> <a href=http://www.baidu.com/bdorz/login.gif?login&amp;tpl=mn&amp;u=http%3A%2F%2Fwww.baidu.com%2f%3fbdorz_come%3d1 name=tj_login class=lb>登录</a> </noscript> <script>document.write('<a href="http://www.baidu.com/bdorz/login.gif?login&tpl=mn&u='+ encodeURIComponent(window.location.href+ (window.location.search === "" ? "?" : "&")+ "bdorz_come=1")+ '" name="tj_login" class="lb">登录</a>');</script> <a href=//www.baidu.com/more/ name=tj_briicon class=bri style="display: block;">更多产品</a> </div> </div> </div> <div id=ftCon> <div id=ftConw> <p id=lh> <a href=http://home.baidu.com>关于百度</a> <a href=http://ir.baidu.com>About Baidu</a> </p> <p id=cp>&copy;2017 Baidu <a href=http://www.baidu.com/duty/>使用百度前必读</a>  <a href=http://jianyi.baidu.com/ class=cp-feedback>意见反馈</a> 京 ICP 证 030173 号  <img src=//www.baidu.com/img/gs.gif> </p> </div> </div> </div> </body> </html>
```

4.4Beautiful Soup 库

4.4.1 Beautiful Soup 简介

Beautiful Soup 提供一些简单的、python 式的函数用来处理导航、搜索、修改分析“标签树”等功能。它是一个工具箱，通过解析文档为用户提供需要抓取的数据，因为简单，所以不需要多少代码就可以写出一个完整的应用程序。



Beautiful Soup 自动将输入文档转换为 Unicode 编码，输出文档转换为 utf-8 编码。你不需要考虑编码方式，除非文档没有指定一个编码方式，这时，Beautiful Soup 就不能自动识别编码方式了。然后，你仅仅需要说明一下原始编码方式就可以了。

Beautiful Soup 已成为和 lxml、html6lib 一样出色的 python 解释器，为用户灵活地提供不同的解析策略或强劲的速度。

4.4.2 Beautiful Soup 安装

目前，Beautiful Soup 的最新版本是 4.x 版本，之前的版本已经停止开发，这里推荐使用 pip 来安装，安装命令如下：

```
pip install beautifulsoup4
```

【示例 4-3】查看 Beautiful Soup 安装是否成功

```
from bs4 import BeautifulSoup
soup = BeautifulSoup('<p>Hello</p>','html.parser')
print(soup.p.string)
```

执行结果如图 4-5 所示：



图 4-5 示例 4-3 运行效果图

注意：

- 这里虽然安装的是 beautifulsoup4 这个包，但是引入的时候却是 bs4，因为这个包源代码本身的库文件名称就是 bs4，所以安装完成后，这个库文件就被移入到本机 Python3

的 lib 库里，识别到的库文件就叫作 bs4。

□ 因此，包本身的名称和我们使用时导入包名称并不一定是一致的。

4.4.3Beautiful Soup 库解析器

Beautiful Soup 在解析时实际上依赖解析器，它除了支持 Python 标准库中的 HTML 解析器外，还支持一些第三方解析器（比如 lxml）。表 4-4 列出了 Beautiful Soup 支持的解析器。

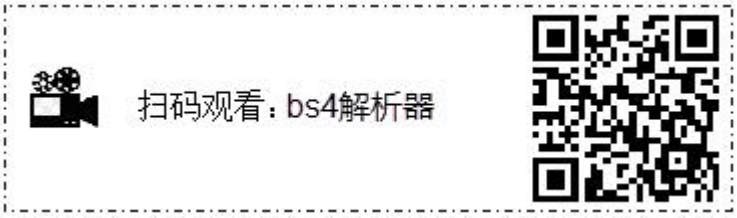


表 4-4 BeautifulSoup 库解析器

解析器	使用方法	条件	优势
bs4 的 HTML 解析器	BeautifulSoup(mk,'html.parser')	安装 bs4 库	Python 的内置标准库、执行速度适中、文档容错能力强
lxml 的 HTML 解析器	BeautifulSoup(mk,'lxml')	pip install lxml	速度快、文档容错能力强
lxml 的 XML 解析器	BeautifulSoup(mk,'xml')	pip install lxml	速度快、唯一支持 XML 的解析器
html5lib 的解析器	BeautifulSoup(mk,'htmlslib')	pip install html5lib	最好的容错性、以浏览器的方式解析文档、生成HTML5格式的文档

【示例 4-4】初始化 BeautifulSoup 使用 lxml，把第二个参数改为 lxml

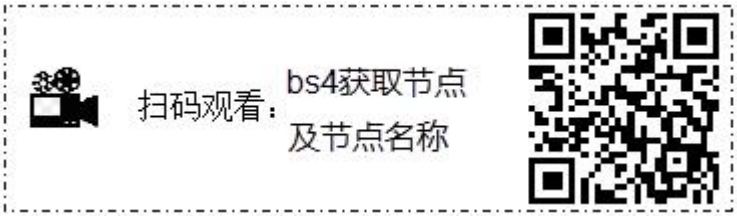
```
from bs4 import BeautifulSoup
bs = BeautifulSoup('<p>Python</p>','lxml')
print(bs.p.string)
```

执行结果如图 4-6 所示：



图 4-6 示例 4-4 运行效果图

4.4.4 BeautifulSoup



1) 基本用法

表 4-5 BeautifulSoup 类的基本元素

基本元素	说明
Tag	标签，基本信息组织单元，分别用<>和</>标明开头和结尾
Name	标签的名字， 的名字是‘p’，格式：<tag>.name
Attributes	标签的属性，字典形式组织，格式：<tag>.attrs
NavigableString	标签内非属性字符串，<>...<>中字符串，格式：<tag>.string
Comment	标签内字符串的注释部分，一种特殊的 Comment 类型

【示例 4-5】获取 title 节点，查看它的类型

```
html = """
<html><head><title>The Dormouse's story</title></head>
<body>
<p class="title" name="dromouse"><b>The Dormouse's story</b></p>
<p class="story">Once upon a time there were three little sisters; and their names were
<a href="http://example.com/elsie" class="sister" id="link1"><!-- Elsie --></a>,
<a href="http://example.com/lacie" class="sister" id="link2">Lacie</a> and
<a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>;
and they lived at the bottom of a well.</p>
<p class="story">...</p>
</body>
</html>
"""
from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'lxml')
print(soup.prettify())
print(soup.title.string)
```

执行结果如下所示：

```
<html>
 <head>
```

```
    <title>
      The Dormouse's story
    </title>
  </head>
  <body>
    <p class="title" name="dromouse">
      <b>
        The Dormouse's story
      </b>
    </p>
    <p class="story">
      Once upon a time there were three little sisters; and their names were
      <a class="sister" href="http://example.com/elsie" id="link1">
        <!-- Elsie -->
      </a>
      ,
      <a class="sister" href="http://example.com/lacie" id="link2">
        Lacie
      </a>
      and
      <a class="sister" href="http://example.com/tillie" id="link3">
        Tillie
      </a>
      ;
and they lived at the bottom of a well.
    </p>
    <p class="story">
      ...
    </p>
  </body>
</html>

The Dormouse's story
```

上述示例首先声明变量 html，它是一个 HTML 字符串。接着将它当作第一个参数传给 BeautifulSoup 对象，该对象的第二个参数为解析器的类型（这里使用 lxml），此时就完成

了 BeaufulSoup 对象的初始化。

接着调用 soup 的各个方法和属性解析这串 HTML 代码了。

调用 prettify()方法。可以把要解析的字符串以标准的缩进格式输出。这里需要注意的是，输出结果里面包含 body 和 html 节点，也就是说对于不标准的 HTML 字符串 BeautifulSoup，可以自动更正格式。

调用 soup.title.string，输出 HTML 中 title 节点的文本内容。所以，soup.title 可以选出 HTML 中的 title 节点，再调用 string 属性就可以得到里面的文本了。

【示例 4-6】选择元素

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'lxml')
print(soup.head)
print(soup.p)
```

执行结果如图 4-7 所示：

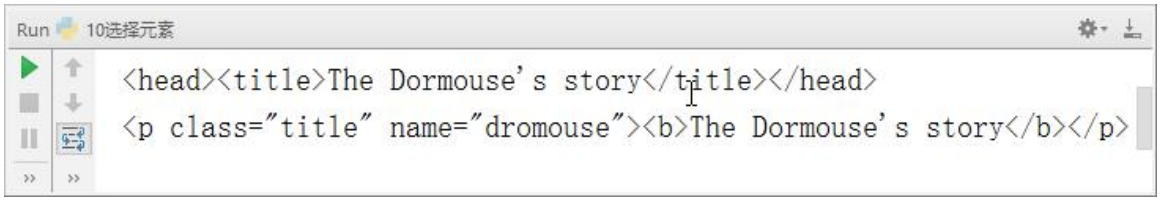


图 4-7 示例 4-6 运行效果图

从上述示例运行结果可以看到，获取 head 节点的结果是节点加其内部的所有内容。最后，选择了 p 节点。不过这次情况比较特殊，我们发现结果是第一个 p 节点的内容，后面的几个 p 节点并没有选到。也就是说，当有多个节点时，这种选择方式只会选择到第一个匹配的节点，其他的后面节点都会忽略。

【示例 4-7】调用 name 属性获取节点的名称

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'lxml')
print(soup.title.name)
```

执行结果如图 4-8 所示：

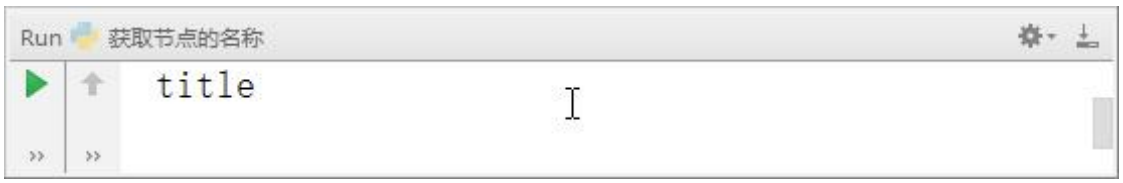
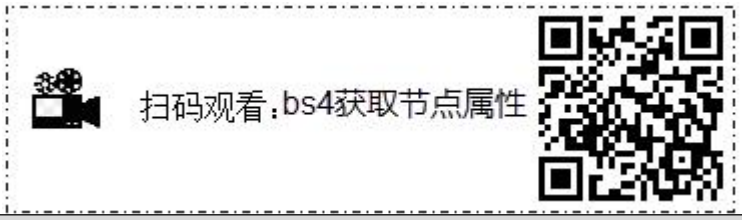


图 4-8 示例 4-7 运行效果图

【示例 4-8】调用 attrs 获取所有属性




```
from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'lxml')
print(soup.p.attrs)
print(soup.p.attrs['name'])
```

执行结果如图 4-9 所示：

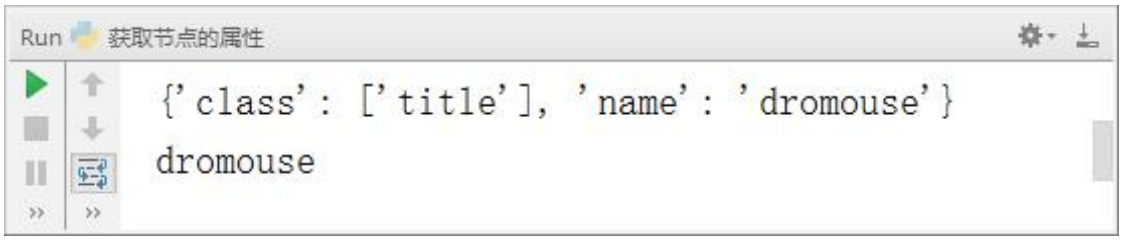


图 4-9 示例 4-8 运行效果图

从上述运行结果可以看到，attrs 的返回结果是字典形式，它把选择节点的所有属性和属性值组合成一个字典。如果要获取 name 属性，就相当于从字典中获取某个键值，只需要用中括号加属性名就可以了。例如，要获取 name 属性，就可以通过 attrs[‘name’]来得到。

【示例 4-9】简单获取属性的方式

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'lxml')
print(soup.p.attrs)
print(soup.p.attrs['name'])
print('简单方式获取属性：')
print(soup.p['name'])
print(soup.p['class'])
```

执行结果如图 4-10 所示：

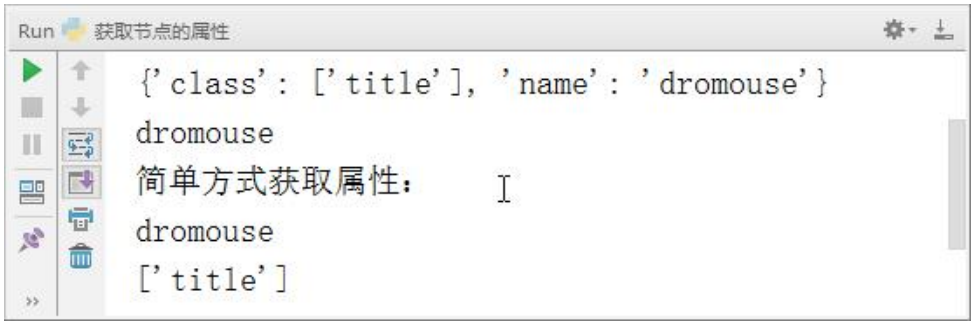


图 4-10 示例 4-9 运行效果图

这里需要注意的是，获取属性有的返回结果是字符串，有的返回结果是字符串组成的列表。比如，name 属性的值是唯一的，返回的结果就是单个字符串。而对于 class，一个节点元素可能有多个 class，所以返回的是列表。

【示例 4-10】调用 string 属性获取节点元素包含的文本内容

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'lxml')
print(soup.p.string)
```

执行结果如图 4-11 所示：

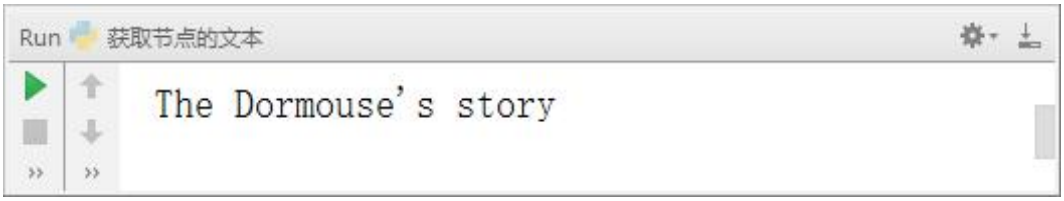


图 4-11 示例 4-10 运行效果图

【示例 4-11】嵌套选择

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'lxml')
print(soup.head.title)
print(type(soup.head.title))
print(soup.head.title.string)
```

执行结果如图 4-12 所示：

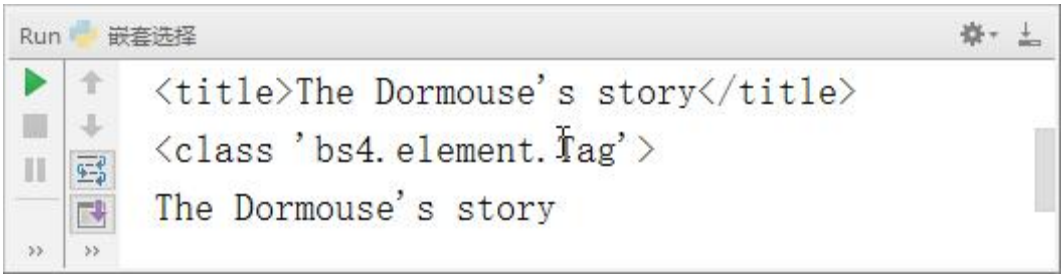


图 4-12 示例 4-11 运行效果图

从上述示例运行结果可以看到，调用 head 之后再次调用 title 可以选择 title 节点元素。输出了它的类型可以看到，它仍然是 bs4.element.Tag 类型。也就是说，我们在 Tag 类型的基础上再次选择得到的依然还是 Tag 类型，每次返回的结果都相同。

【示例 4-12】调用 children 属性，获取它的直接子节点



扫码观看：获取子节点_父节点_兄弟节点



```
html = """
<html>
    <head>
        <title>The Dormouse's story</title>
    </head>
    <body>
        <p class="story">
            Once upon a time there were three little sisters; and their names were
            <a href="http://example.com/elsie" class="sister" id="link1">
                <span>Elsie</span>
            </a>
            <a href="http://example.com/lacie" class="sister" id="link2">Lacie</a>
            and
            <a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>
            and they lived at the bottom of a well.
        </p>
        <p class="story">...</p>
"""
from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'lxml')
print(soup.p.children)
for i, child in enumerate(soup.p.children):
    print(i, child)
```

执行结果如下所示：

```
<list_iterator object at 0x00000246ACC46400>
0
            Once upon a time there were three little sisters; and their names were


1 <a class="sister" href="http://example.com/elsie" id="link1">
<span>Elsie</span>
</a>
```

```
2


3 <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>
4
                and

5 <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>
6
                and they lived at the bottom of a well.
```

从上述示例运行结果可以看到，调用 children 属性，返回结果是生成器类型。用 for 循环输出相应的内容。

【示例 4-13】调用 parent 属性，获取某个节点元素的父节点

```
html = """
<html>
    <head>
        <title>The Dormouse's story</title>
    </head>
    <body>
        <p class="story">
            Once upon a time there were three little sisters; and their names were
            <a href="http://example.com/elsie" class="sister" id="link1">
                <span>Elsie</span>
            </a>
        </p>
        <p class="story">...</p>
"""
from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'lxml')
print(soup.a.parent)
```

执行结果如下所示：

```
<p class="story">
            Once upon a time there were three little sisters; and their names were
            <a class="sister" href="http://example.com/elsie" id="link1">
```

```
<span>Elsie</span>
</a>
</p>
```

从上述示例运行结果可以看到，我们选择的是第一个 a 节点的父节点元素，它的父节点是 p 节点，输出结果便是 p 节点及其内部的内容。

需要注意的是，这里输出的仅仅是 a 节点的直接父节点，而没有再向外寻找父节点的祖先节点。如果想获取所有的祖先节点，可以调用 parents 属性。

【示例 4-14】调用 parents 属性，获取某个节点元素的祖先节点

```
html = """
<html>
    <body>
        <p class="story">
            <a href="http://example.com/elsie" class="sister" id="link1">
                <span>Elsie</span>
            </a>
        </p>
"""
from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'lxml')
print(type(soup.a.parents))
print(list(enumerate(soup.a.parents)))
```

执行结果如下所示：

```
<class 'generator'>
[(0, <p class="story">
<a class="sister" href="http://example.com/elsie" id="link1">
<span>Elsie</span>
</a>
</p>), (1, <body>
<p class="story">
<a class="sister" href="http://example.com/elsie" id="link1">
<span>Elsie</span>
</a>
</p>
</body>), (2, <html>
```

```
<body>
<p class="story">
<a class="sister" href="http://example.com/elsie" id="link1">
<span>Elsie</span>
</a>
</p>
</body></html>), (3, <html>
<body>
<p class="story">
<a class="sister" href="http://example.com/elsie" id="link1">
<span>Elsie</span>
</a>
</p>
</body></html>)]
```

【示例 4-15】调用 next_sibling 和 previous_sibling 分别获取节点的下一个和上一个兄弟元素

```
html = """
<html>
    <body>
        <p class="story">
            Once upon a time there were three little sisters; and their names were
            <a href="http://example.com/elsie" class="sister" id="link1">
                <span>Elsie</span>
            </a>
            Hello
            <a href="http://example.com/lacie" class="sister" id="link2">Lacie</a>
            and
            <a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>
            and they lived at the bottom of a well.
        </p>
"""
from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'lxml')
```

```
print('Next Sibling', soup.a.next_sibling)
print('Prev Sibling', soup.a.previous_sibling)
```

执行结果如下所示：

```
Next Sibling
                Hello


Prev Sibling
                Once upon a time there were three little sisters; and their names were
```

2) 方法选择器



上面所讲的选择方法都是通过属性来选择的，这种方法非常快，但是如果进行比较复杂的选择的话，它就比较烦琐，不够灵活了。Beautiful Soup 还提供了一些查询方法，例如 find_all()和 find()等。

find_all 是查询所有符合条件的元素。给它传入一些属性或文本，就可以得到符合条件的元素，它的功能十分强大，语法格式如下：

```
find_all(name , attrs , recursive , text , **kwargs)
```

【示例 4-16】find_all 方法传入 name 参数，根据节点名来查询元素

```
html='''
<div class="panel">
    <div class="panel-heading">
        <h4>Hello</h4>
    </div>
    <div class="panel-body">
        <ul class="list" id="list-1">
            <li class="element">Foo</li>
            <li class="element">Bar</li>
            <li class="element">Jay</li>
        </ul>
        <ul class="list list-small" id="list-2">
            <li class="element">Foo</li>
            <li class="element">Bar</li>
```

```
            </ul>
        </div>
    </div>
    '''
    from bs4 import BeautifulSoup
    soup = BeautifulSoup(html, 'lxml')
    for ul in soup.find_all(name='ul'):
        print(ul.find_all(name='li'))
        for li in ul.find_all(name='li'):
            print(li.string)
```

执行结果如图 4-13 所示：



图 4-13 示例 4-16 运行效果图

从上述示例可以看到，调用 find_all()方法，name 参数值为 ul。返回结果是查询到的所有 ul 节点列表类型，长度为 2，每个元素依然都是 bs4.element.Tag 类型。因为都是 Tag 类型，所以依然可以进行嵌套查询。再继续查询其内部的 li 节点，返回结果是 li 节点列表类型，遍历列表中的每个 li，获取它的文本。

【示例 4-17】find_all 方法传入 attrs 参数，根据属性来查询

```
    html='''
    <div class="panel">
        <div class="panel-heading">
            <h4>Hello</h4>
```

```
        </div>
        <div class="panel-body">
            <ul class="list" id="list-1" name="elements">
                <li class="element">Foo</li>
                <li class="element">Bar</li>
                <li class="element">Jay</li>
            </ul>
            <ul class="list list-small" id="list-2">
                <li class="element">Foo</li>
                <li class="element">Bar</li>
            </ul>
        </div>
    </div>
    '''
    from bs4 import BeautifulSoup
    soup = BeautifulSoup(html, 'lxml')
    print(soup.find_all(attrs={'id': 'list-1'}))
    print(soup.find_all(attrs={'name': 'elements'}))
```

执行结果如图 4-14 所示：

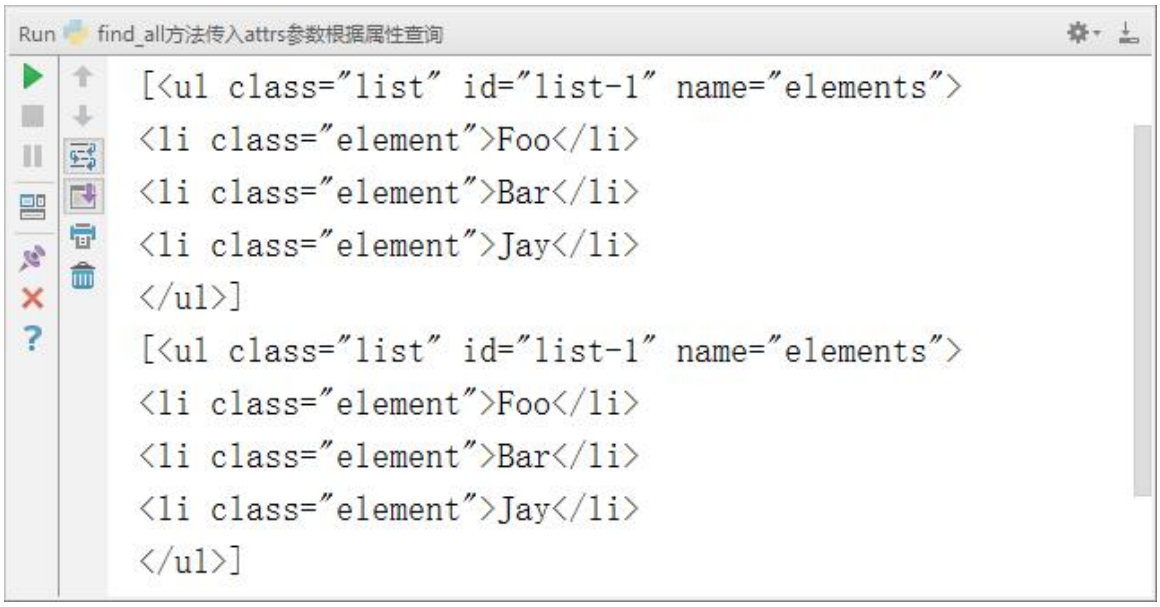


图 4-14 示例 4-17 运行效果图

从上述示例可以看到，传入 attrs 参数，参数的类型是字典类型。比如，要查询 id 为 list-1 的节点，可以传入 attrs={‘id’：‘list-1’}的查询条件，得到的结果是列表形式，包含的内容就是符合 id 为 list-1 的所有节点。符合条件的元素个数是 1，长度为 1 的列表。

对于一些常用的属性，比如 id 和 class 等，可以不用 attrs 来传递。比如，要查询 id 为 list-1 的节点，可以直接传入 id 这个参数。示例如下：

【示例 4-18】find_all 方法根据属性来查询

```
html='''
<div class="panel">
    <div class="panel-heading">
        <h4>Hello</h4>
    </div>
    <div class="panel-body">
        <ul class="list" id="list-1">
            <li class="element">Foo</li>
            <li class="element">Bar</li>
            <li class="element">Jay</li>
        </ul>
        <ul class="list list-small" id="list-2">
            <li class="element">Foo</li>
            <li class="element">Bar</li>
        </ul>
    </div>
</div>
'''
from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'lxml')
print(soup.find_all(id='list-1'))
print(soup.find_all(class_='element'))
```

执行结果如图 4-15 所示：



图 4-15 示例 4-18 运行效果图

上述示例直接传入 id=’list-1’，就可以查询 id 为 list-1 的节点元素了。而对于 class 来说，由于 class 在 Python 里是一个关键字，所以后面需要加一个下划线，即 class_=’element’，返回的结果依然还是 Tag 组成的列表。

find_all 方法传入 text 参数可用来匹配节点的文本，传入的形式可以是字符串，可以是正则表达式对象。

【示例 4-19】find_all 方法根据文本来查询

```
import re
html='''
<div class="panel">
    <div class="panel-body">
        <a>Hello, this is a link</a>
        <a>Hello, this is a link, too</a>
    </div>
</div>
'''
from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'lxml')
print(soup.find_all(text=re.compile('link')))
```

执行结果如图 4-16 所示：

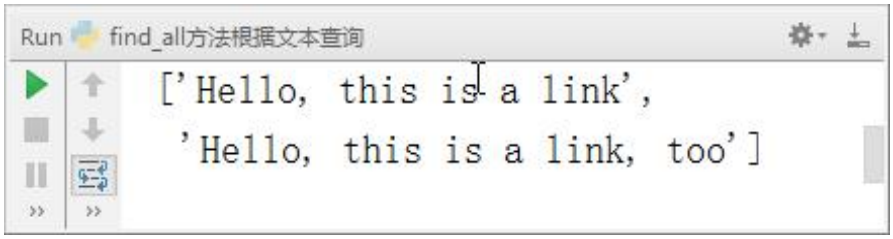


图 4-16 示例 4-19 运行效果图

上述示例有两个 a 节点，其内部包含文本信息。这里在 find_all()方法中传入 text 参数，该参数为正则表达式对象，结果返回所有匹配正则表达式的节点文本组成的列表。

除了 find_all()方法，还有 find()方法，不过后者返回的是单个元素，也就是第一个匹配的元素，而前者返回的是所有匹配的元素组成的列表。

【示例 4-20】find 方法查询第一个匹配的元素

```
html='''
<div class="panel">
    <div class="panel-heading">
        <h4>Hello</h4>
    </div>
    <div class="panel-body">
        <ul class="list" id="list-1">
            <li class="element">Foo</li>
            <li class="element">Bar</li>
            <li class="element">Jay</li>
        </ul>
        <ul class="list list-small" id="list-2">
            <li class="element">Foo</li>
            <li class="element">Bar</li>
        </ul>
    </div>
</div>
'''
from bs4 import BeautifulSoup
soup = BeautifulSoup(html, 'lxml')
print(soup.find(name='ul'))
print(type(soup.find(name='ul')))
print(soup.find(class_='list'))
```

执行结果如图 4-17 所示：

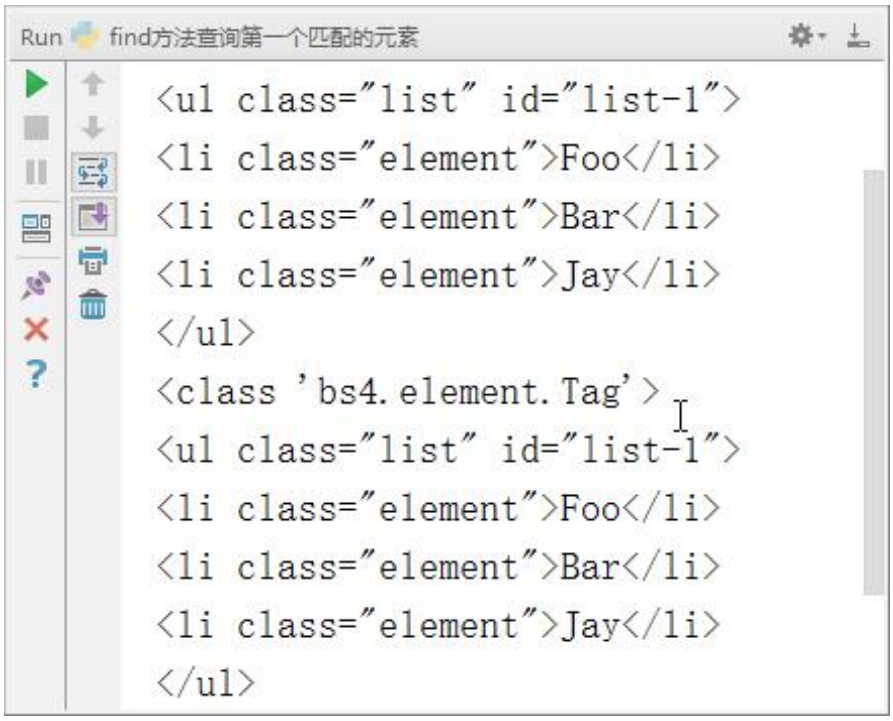


图 4-17 示例 4-20 运行效果图

上述示例使用 find 方法返回结果不再是列表形式，而是第一个匹配的节点元素，类型依然是 Tag 类型。

3) CSS 选择器

Beautiful Soup 还提供了另外一种选择器，那就是 CSS 选择器。使用 CSS 选择器时，只需要调用 select()方法，传入相应的 CSS 选择器即可。

【示例 4-21】CSS 选择器的使用

```
html='''
<div class="panel">
    <div class="panel-heading">
        <h4>Hello</h4>
    </div>
    <div class="panel-body">
        <ul class="list" id="list-1">
            <li class="element">Foo</li>
            <li class="element">Bar</li>
            <li class="element">Jay</li>
        </ul>
        <ul class="list list-small" id="list-2">
```

```
                    <li class="element">Foo</li>
                    <li class="element">Bar</li>
                </ul>
            </div>
        </div>
        '''
        from bs4 import BeautifulSoup
        soup = BeautifulSoup(html, 'lxml')
        print(soup.select('.panel .panel-heading'))
        print(soup.select('ul li'))
        lis = soup.select('#list-2 .element')
        for l in lis:
            print('Get Text:',l.get_text())
            print('String:',l.string)
```

执行结果如图 4-18 所示：

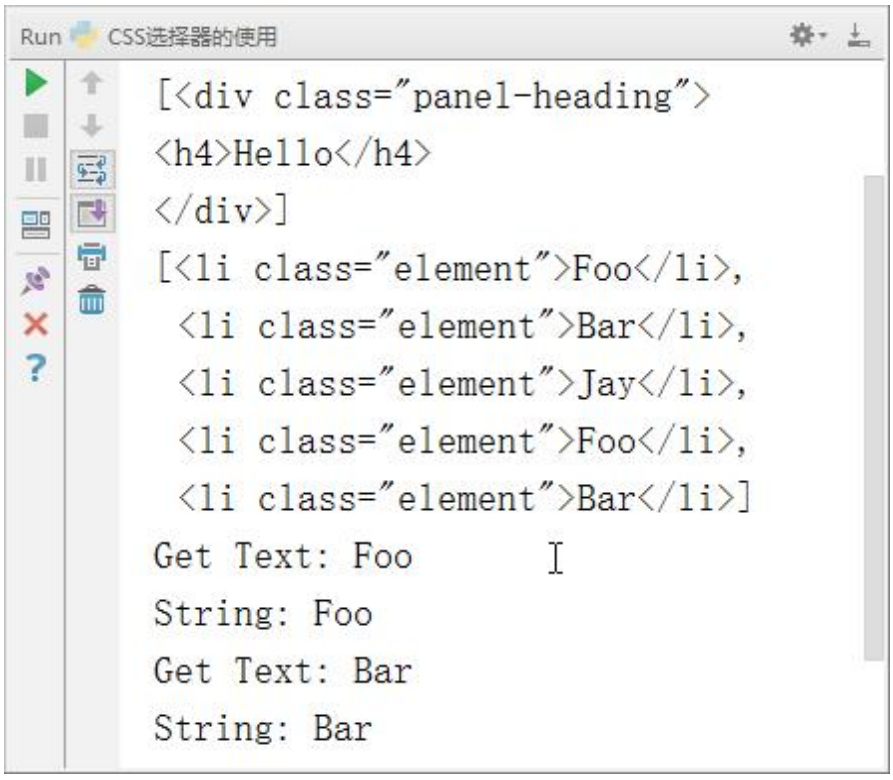
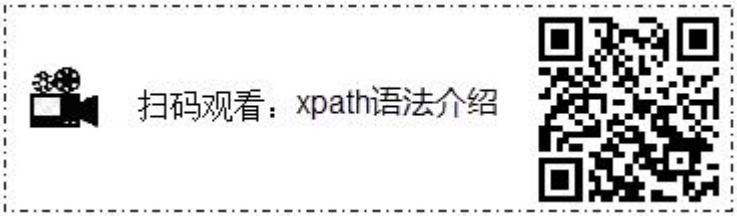


图 4-18 示例 4-21 运行效果图

上述示例，用了 3 次 CSS 选择器，返回的结果均是符合 CSS 选择器的节点组成的列表。例如，select(‘ul li’)则是选择所有 ul 节点下面的所有 li 节点，结果便是所有的 li 节点组成的列表。要获取文本，当然也可以用前面所讲的 string 属性。此外，还有一个方法，那就是 get_text()。

4.5XPath

4.5.1 XPath 简介



XPath，全称 XML Path Language，即 XML 路径语言，它是一门在 XML 文档中查找信息的语言。最初是用来搜寻 XML 文档的，但同样适用于 HTML 文档的搜索。所以在做爬虫时完全可以使用 XPath 做相应的信息抽取。

XPath 的选择功能十分强大，它提供了非常简洁明了的路径选择表达式。另外，它还提供了超过 100 个内建函数，用于字符串、数值、时间的匹配以及节点、序列的处理等，几乎所有想要定位的节点都可以用 XPath 来选择。官方文档：https://www.w3.org/TR/xpath/。

4.5.2 安装

```
pip install lxml
```

4.5.3 XPath 语法

XPath 使用路径表达式来选取 XML 文档中的节点或节点集。节点是沿着路径（path）或者步（steps）来选取的。下面列举了一些常用的路径表达式进行节点的选取，如表 4-6 所示。

表 4-6 路径表达式

表达式	描述
nodename	选取此节点的所有子节点
/	从当前节点选区直接子节点
//	从当前节点选取子孙节点
.	选取当前节点
..	选取当前节点的父节点

XPath 的常用匹配规则如下所示：

```
    //title[@lang='eng']
```

上述 XPath 匹配规则代表的是选择所有名称为 title，同时属性 lang 的值为 eng 的节点，后面会通过 Python 的 lxml 库，利用 XPath 进行 HTML 的解析。

上面路径都是选取了所有符合条件的节点，是否能选取某个特定的节点或者包含某一个指定值得节点，这就需要用到谓语。谓语被嵌在方括号中，用来查找某个特定的节点或者包含某个指定的值的节点。

表 4-7 谓语示例

路径表达式	结果
/bookstore/book[1]	选取属于 bookstore 子元素的第一个 book 元素。
/bookstore/book[last()]	选取属于 bookstore 子元素的最后一个 book 元素。
/bookstore/book[last()-1]	选取属于 bookstore 子元素的倒数第二个 book 元素。
/bookstore/book[position()<3]	选取最前面的两个属于 bookstore 元素的子元素的 book 元素。
//title[@lang]	选取所有拥有名为 lang 的属性的 title 元素。
//title[@lang='eng']	选取所有 title 元素，且这些元素拥有值为 eng 的 lang 属性。
/bookstore/book[price>35.00]	选取 bookstore 元素的所有 book 元素，且其中的 price 元素的值须大于 35.00。
/bookstore/book[price>35.00]/title	选取 bookstore 元素中的 book 元素的所有 title 元素，且其中的 price 元素的值须大于 35.00。

表 4-8 选取未知节点

通配符	描述
*	匹配任何元素节点。
@*	匹配任何属性节点。
node()	匹配任何类型的节点。

表 4-9 通配符选取示例

路径表达式	结果
/bookstore/*	选取 bookstore 元素的所有子元素。
//*	选取文档中的所有元素。
//title[@*]	选取所有带有属性的 title 元素。

通过在路径表达式中使用“|”运算符，您可以选取若干个路径。

表 4-10 选取若干路径示例

路径表达式	结果
//book/title|//book/price	选取 book 元素的所有 title 和 price 元素。
//title | //price	选取文档中的所有 title 和 price 元素。
/bookstore/book/title | //price	选取属于 bookstore 元素的 book 元素的所有 title 元素，以及文档中所有的 price 元素。

4.5.4 XPath 使用



首先给出XPath_test.html，实例分析就按照这个文档来进行，文档内容如下：

```
<html><body><div>
<ul>
<li class="item-0"><a href="link1.html">first item</a></li>
<li class="item-1"><a href="link2.html">second item</a></li>
<li class="item-inactive"><a href="link3.html">third item</a></li>
<li class="item-1"><a href="link4.html">fourth item</a></li>
<li class="item-0"><a href="link5.html">fifth item</a>
</li></ul>
</div>
</body></html>
```

【示例 4-22】 用以 // 开头的 XPath 规则来选取所有符合要求的节点

```
from lxml import etree
html = etree.parse('./XPath_test.html', etree.HTMLParser())
result = html.xpath('//*')
print(result)
```

执行结果如图 4-19 所示：

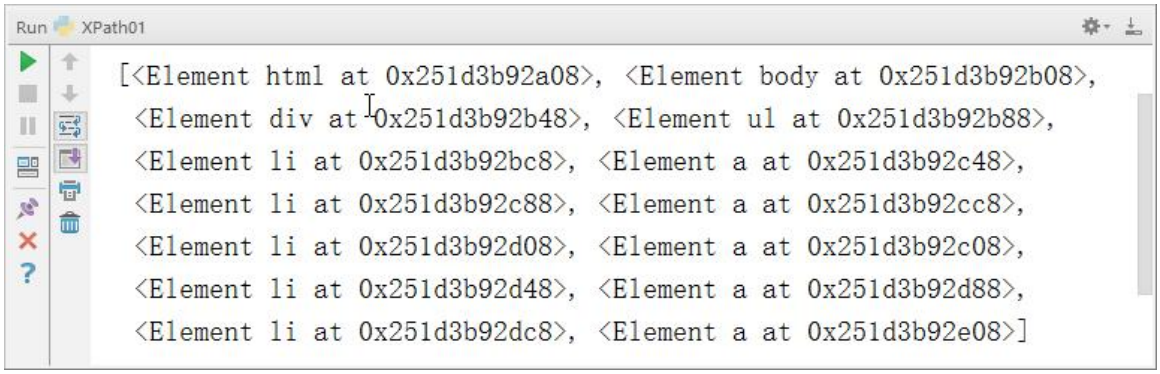


图 4-19 示例 4-22 运行效果图

上述示例中 * 代表匹配所有节点，返回的结果是一个列表，每个元素都是一个 Element 类型，后跟节点名称。也可以指定匹配的节点名称，示例如下。

【示例 4-23】 指定匹配的节点名称

```
from lxml import etree
html = etree.parse('./XPath_test.html', etree.HTMLParser())
result = html.xpath('//li')
print(result)
```

执行结果如图 4-20 所示：

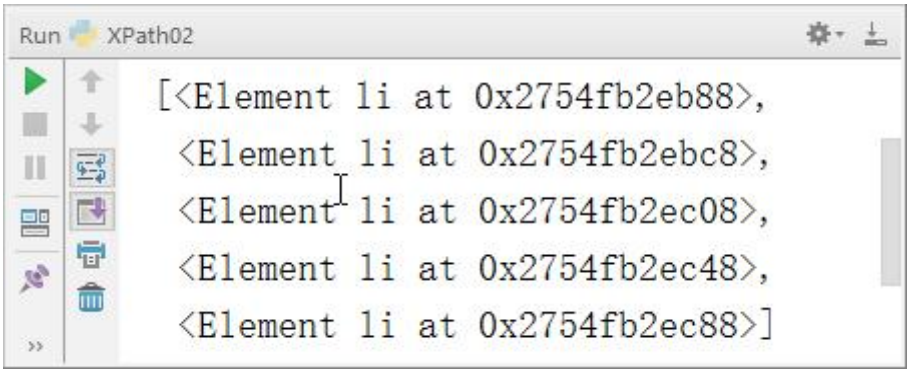


图 4-20 示例 4-23 运行效果图

通过 / 或 // 即可查找元素的子节点或子孙节点。选择 li 节点的所有直接 a 子节点。

【示例 4-24】 选择 li 节点的所有直接 a 子节点

```
from lxml import etree
html = etree.parse('./XPath_test.html', etree.HTMLParser())
result = html.xpath('//li/a')
print(result)
```

执行结果如图 4-21 所示：

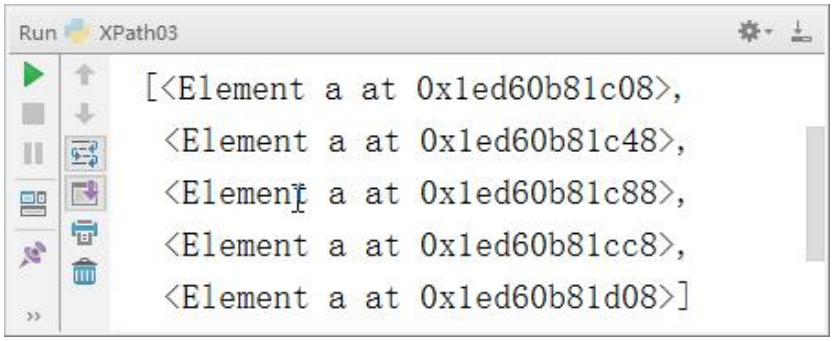


图 4-21 示例 4-24 运行效果图

【示例 4-25】 用@符号进行属性过滤

```
from lxml import etree
html = etree.parse('./XPath_test.html', etree.HTMLParser())
```

```
result = html.xpath('//li[@class="item-inactive"]')
print(result)
```

执行结果如图 4-22 所示：

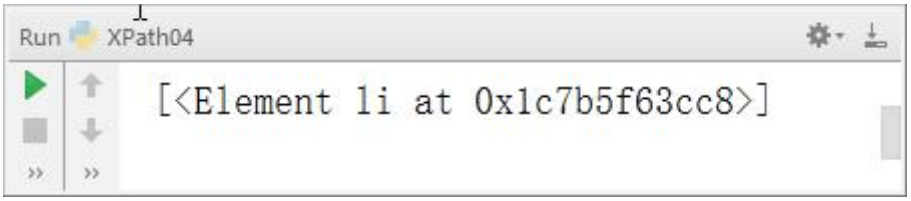


图 4-22 示例 4-25 运行效果图

【示例 4-26】 调用 text()方法获取文本

```
from lxml import etree
html = etree.parse('./XPath_test.html', etree.HTMLParser())
result = html.xpath('//li[@class="item-0"]/a/text()')
print(result)
```

执行结果如图 4-23 所示：

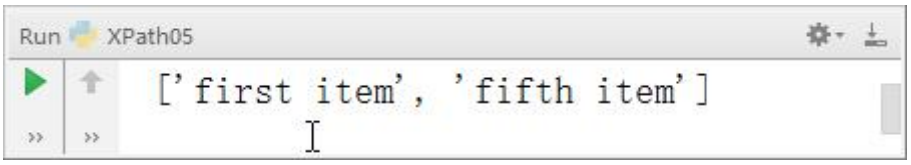


图 4-23 示例 4-26 运行效果图

【示例 4-27】@符号跟属性名直接获取节点的属性值

```
from lxml import etree
html = etree.parse('./XPath_test.html', etree.HTMLParser())
result = html.xpath('//li/a/@href')
print(result)
```

执行结果如图 4-24 所示：



图 4-24 示例 4-27 运行效果图

【示例 4-28】多属性匹配

```
from lxml import etree
text = '''
<li class="li li-first" name="item"><a href="link.html">first item</a></li>
'''
html = etree.HTML(text)
result = html.xpath('//li[contains(@class, "li") and @name="item"]/a/text()')
print(result)
```

执行结果如图 4-25 所示：

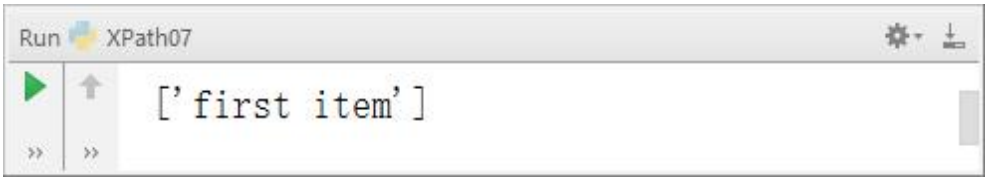


图 4-25 示例 4-28 运行效果图

习题

一、简答题

1. 什么是爬虫。
2. 爬虫的基本流程。
3. HTTP 请求方式有哪些。
4. GET 请求和 POST 请求有什么区别。

二、编码题

1. 爬取一张图片并保存。
2. 使用 BeautifulSoup4 解析器，将招聘网页上的职位名称、职位类别、招聘人数、工作地点、时间、以及每个职位详情的点击链接存储出来。
3. 从网址“https://bj.zu.ke.com/zufang”，获取到房源的朝向，大小，租金，名字，租赁方式及厅室情况等相关信息。
4. 用 XPath 来做一个简单的爬虫，爬取某个贴吧里的所有帖子，并且将该这个帖子里每个楼层发布的图片下载到本地。
5. 用 XPath 爬取一下豆瓣音乐专区数据。

第五章 Python 操作 Excel 存储数据

5.1 openpyxl 模块



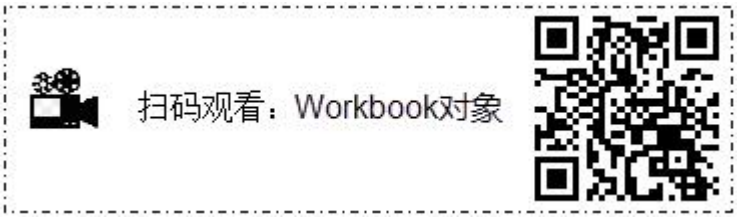
openpyxl 模块是一个读写 Excel 2010 文档的 Python 库，如果要处理更早格式的 Excel 文档，需要用到额外的库，openpyxl 是一个比较综合的工具，能够同时读取和修改 Excel 文档。

5.1.1 安装 openpyxl 模块

要想使用 openpyxl 模块，必须先安装此模块。直接使用 pip 就可以进行安装，命令如下：

```
pip install openpyxl
```

5.1.2 Excel 文件的三个对象



想要操作 Excel 首先要了解 Excel 基本概念，Excel 中列以字母命名，行以数字命名，比如左上角第一个单元格的坐标为 A1，下面的为 A2，右边的 B1。

openpyxl 中有三个不同层次的类，Workbook 是对工作簿的抽象，Worksheet 是对表格的抽象，Cell 是对单元格的抽象，每一个类都包含了许多属性和方法。

打开或者创建一个 Excel 需要创建一个 Workbook 对象。获取一个表则需要先创建一个 Workbook 对象，然后使用该对象的方法来得到一个 Worksheet 对象。如果要获取表中的数据，那么得到 Worksheet 对象以后再从中获取代表单元格的 Cell 对象。

1) Workbook 对象

一个 Workbook 对象代表一个 Excel 文档，因此在操作 Excel 之前，都应该先创建一个 Workbook 对象。对于创建一个新的 Excel 文档，直接进行 Workbook 类的调用即可，对于一个已经存在的 Excel 文档，可以使用 openpyxl 模块的 load_workbook 函数进行读取，该函数包涵多个参数，但只有 filename 参数为必传参数。filename 是一个文件名，也可以是一个打开的文件对象。

【示例 5-1】创建 Workbook 对象

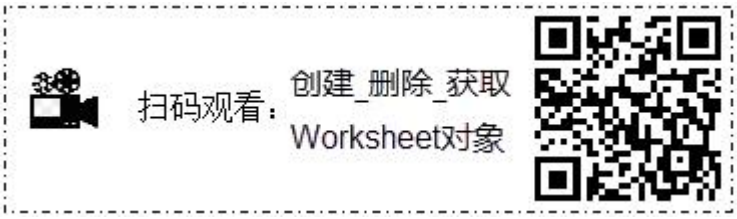
```
import openpyxl
excel = openpyxl.Workbook()  # 创建本地工作簿
excel = openpyxl.load_workbook("abc.xlsx")  # 加载本地已存在的工作簿
# 操作工作簿完毕后需要保存工作簿
excel.save("workbook_test.xlsx")
```

Workbook 对象提供了很多属性和方法，其中，大部分方法都与 sheet 有关，常用属性和方法如表 5-1 所示：

表 5-1 Workbook 对象常用属性和方法

名称	说明
excel.active	获取当前活跃的 Worksheet 对象
excel.worksheets	以列表的形式返回所有的 Worksheet 对象
excel.sheetnames	获取工作簿中的表名[name1, name2, name3]
excel.encoding	获取文档的字符集编码
excel.properties	获取文档的元数据，如标题，创建者，创建日期等
excel_sheet = excel["Sheet1"]	通过工作表名获取到 worksheet 对象
excel.remove(excel_sheet)	删除一个表格，参数为 worksheet 对象
excel.create_sheet()	创建一个空的表格，参数为表名和 index

2) Worksheet 对象



通过 Worksheet 对象获取表格的属性，得到单元格中的数据，修改表格中的内容。openpyxl 提供了非常灵活的方式来访问表格中的单元格和数据，常用的 Worksheet 属性和方法如表 5-2 和表 5-3 所示：

表 5-2 Worksheet 对象常用属性

属性名	说明
excel_sheet.title	表格的标题(可读可写)
excel_sheet.dimensions	表格的大小，这里的大小是指含有数据的表格的大小，即：左上角的坐标:右下角的坐标
excel_sheet.max_row	表格的最大行数
excel_sheet.min_row	表格的最小行数
excel_sheet.max_column	表格的最大列数
excel_sheet.min_column	表格的最小列数

属性名	说明
excel_sheet.rows	按行获取单元格(Cell 对象)
excel_sheet.columns	按列获取单元格(Cell 对象)
excel_sheet.values	按行获取表格的内容(数据)

表 5-3 Worksheet 对象常用方法

属性名	说明
excel_sheet.iter_rows()	按行获取所有单元格，内置属性有 (min_row,max_row,min_col,max_col)
excel_sheet.iter_cols()	按列获取所有的单元格
excel_sheet["A"]	返回 A 列中所有的单元格 cell 对象
excel_sheet["1"]	返回第一行中所有的单元格 cell 对象
excel_sheet["A1"]	返回该单元格对象
excel.cell()	设置单元格的值
excel.append(iterable)	逐行写

3) Cell

Cell 对象比较简单，常用的属性如表 5-4 所示：

表 5-4 Cell 对象常用属性

属性名	说明
row	单元格所在的行
column	单元格所在的列
value	单元格的值
coordinate	单元格的坐标

5.2 Python 操作 Excel

5.2.1 Python 操作 Excel 之读取

【示例 5-2】打开本地工作簿，获取所有工作表名称

```
import openpyxl
wb = openpyxl.load_workbook('excelTest.xlsx')
#获取工作表的名称
print(wb.sheetnames)
#遍历获取工作表的名称
for sheet in wb:
```

```
        print(sheet.title)
```

执行结果如图 5-1 所示：

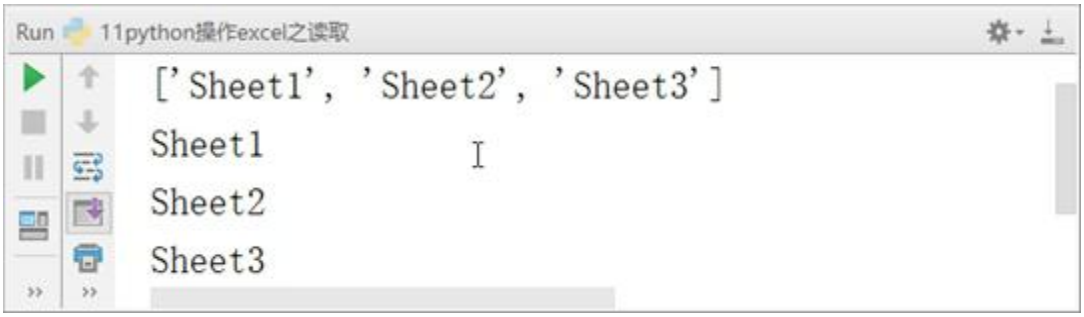


图 5-1 示例 5-2 运行效果图

【示例 5-3】创建工作表

```
import openpyxl
wb = openpyxl.load_workbook('excelTest.xlsx')
#创建工作表
mySheet = wb.create_sheet('mySheet')
print(wb.sheetnames)
#遍历获取工作表的名称
for sheet in wb:
    print(sheet.title)
```

执行结果如图 5-2 所示：

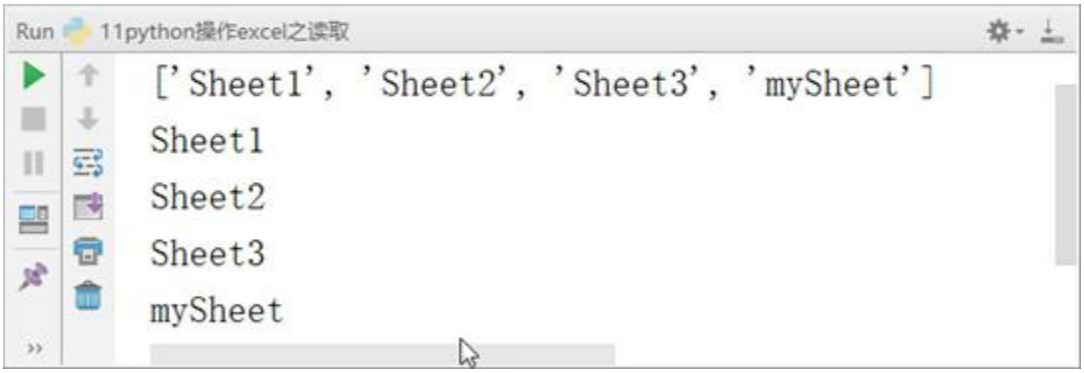


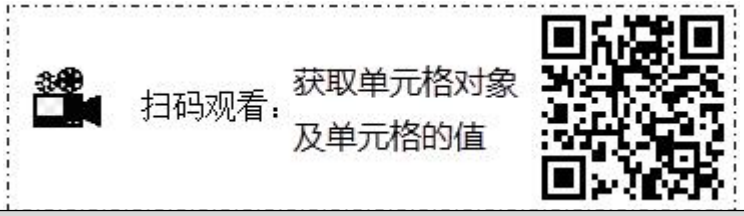
图 5-2 示例 5-3 运行效果图

【示例 5-4】根据工作表名称获取工作表

```
import openpyxl
wb = openpyxl.load_workbook('excelTest.xlsx')
#根据名称获取表单
sheet3 = wb.get_sheet_by_name('Sheet2')
```

```
#或者
sheet4 = wb['mySheet']
```

【示例 5-5】获取单元格对象及单元格的值




```
import openpyxl
wb = openpyxl.load_workbook('excelTest.xlsx')
#获取当前激活的工作表
ws = wb.active    #返回的是工作表对象
print(ws)    #<Worksheet "Sheet1">
#获取 Cell 对象及单元格的值
print(ws['A1'])    #<Cell 'Sheet1'.A1>
print(ws['A1'].value)
```

【示例 5-6】获取单元格的行、列及值

```
import openpyxl
wb = openpyxl.load_workbook('excelTest.xlsx')
ws = wb.active
c = ws['B1']
print('Row:{} Column:{} is {}'.format(c.row,c.column,c.value))
print('cell:{}    is {}'.format(c.coordinate,c.value))
print('ws.cell(row=1,column=2).value：',ws.cell(row=1,column=2).value)
#for 循环获取
print('循环遍历获取'.center(20,'*'))
for r in ws.rows:
    for c in r:
        print(c.value,end='\t')
    print()
```

执行结果如图 5-3 所示：

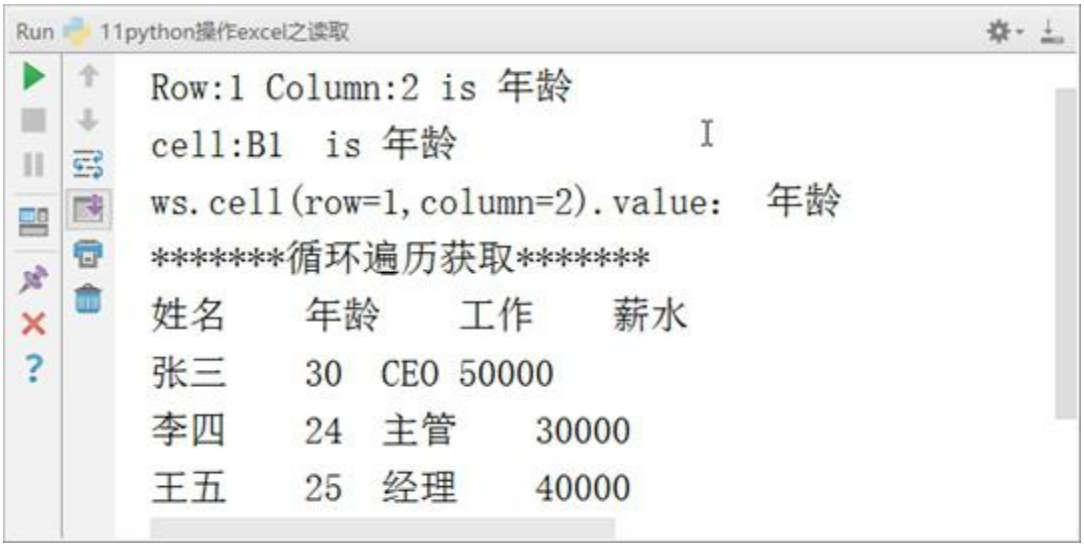
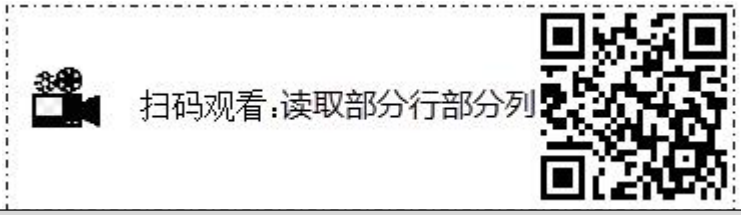


图 5-3 示例 5-6 运行效果图

【示例 5-7】读取整行、整列及部分几行




```
import openpyxl
wb = openpyxl.load_workbook('excelTest.xlsx')
ws = wb.active
print('获取整列'.center(20,'*'))
colc = ws['C']   #获取到 C 列
print(colc)
print('遍历获取整列的数据'.center(20,'*'))
for c in colc:
    print(c.value,end='\t')
print()
print('获取整行数据'.center(20,'*'))
row6 = ws[6] #获取第 6 行
print(row6)
print('切片获取部分几行'.center(20,'*'))
row_range = ws[2:4]
print(row_range)
print('循环遍历部分几行数据'.center(20,'*'))
for row in row_range:
    for c in row:
        print(c.value,end='\t')
```

```
        print()
```

执行结果如图 5-4 所示：



图 5-4 示例 5-7 运行效果图

【示例 5-8】使用 iter_rows()逐行读取

```
import openpyxl
wb = openpyxl.load_workbook('excelTest.xlsx')
ws = wb.active
print('共{}行， 共{}列'.format(ws.max_row,ws.max_column))
print('ws.iter_rows()设置最小列、最大列、最小行和最大行读取'.center(20,'*'))
for row in ws.iter_rows(min_row=1,max_row=3,max_col=3,min_col=2):
    for cell in row:
        print(cell.value,end=',')
    print()
```

执行结果如图 5-5 所示：

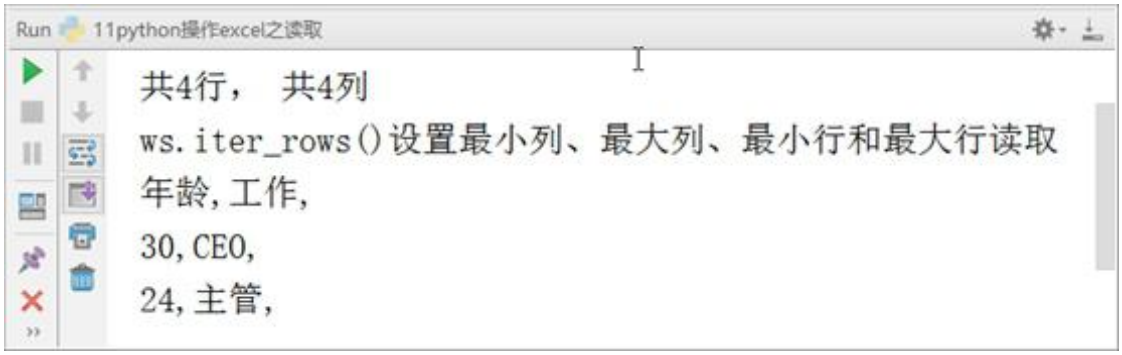


图 5-5 示例 5-8 运行效果图

【示例 5-9】部分行部分列切片读取

```
import openpyxl
wb = openpyxl.load_workbook('excelTest.xlsx')
ws = wb.active
print('共{}行， 共{}列'.format(ws.max_row,ws.max_column))
print('部分行部分列切片读取'.center(20,'*'))
cell_range = ws['A1:C3']
for rowObject in cell_range:
    for cellObject in rowObject:
        print(cellObject.coordinate,cellObject.value,end='\t')
    print()
```

执行结果如图 5-6 所示：

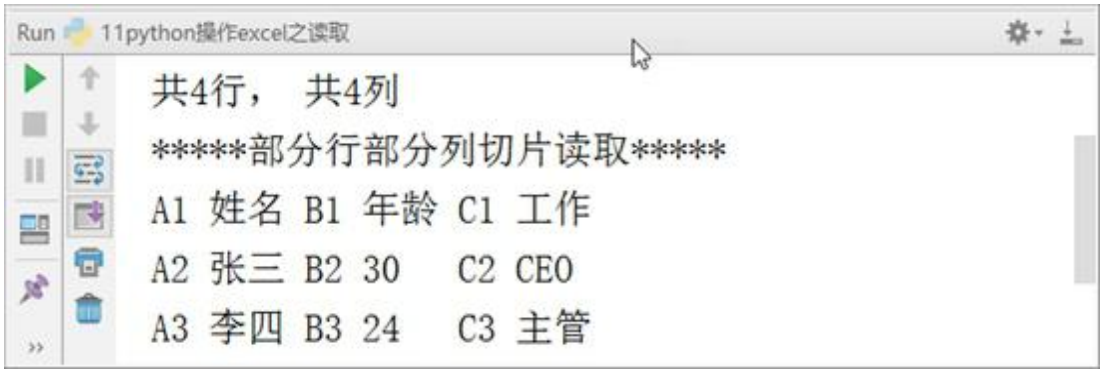


图 5-6 示例 5-9 运行效果图

【示例 5-10】列数字与字母的对应转换

```
from openpyxl.utils import get_column_letter,column_index_from_string
print('第 2 列对应的字母：',get_column_letter(2))
print('第 34 列对应的字母：',get_column_letter(34))
print('第 100 列对应的字母：',get_column_letter(100))
print('AAH 字母列对应的数字：',column_index_from_string('AAH'))
print('B 字母列对应的数字：',column_index_from_string('B'))
```

执行结果如图 5-7 所示：

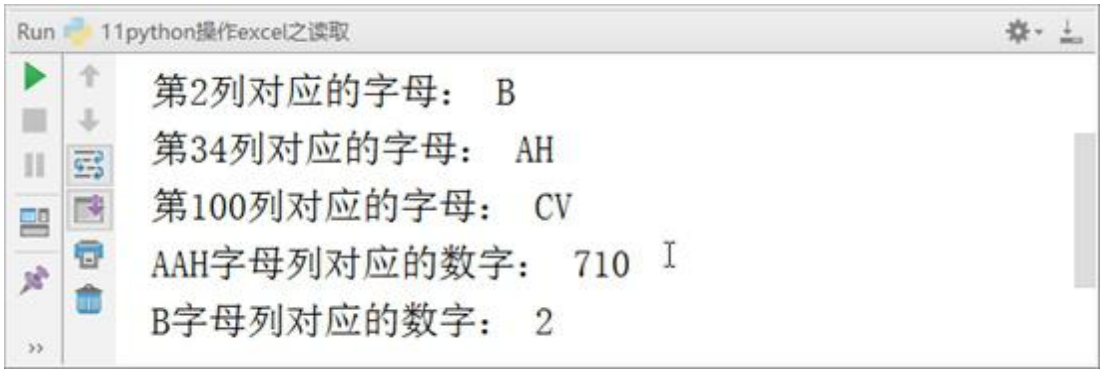
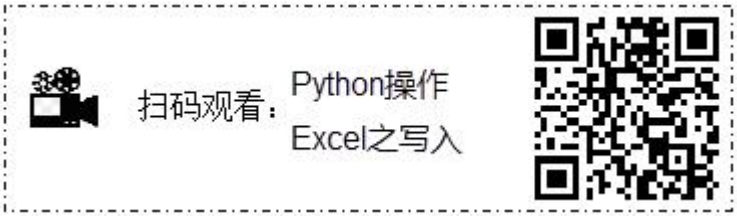


图 5-7 示例 5-10 运行效果图

5.2.2 Python 操作 Excel 之写



【示例 5-11】创建、删除工作表

```
import openpyxl
#python 操作 excel 之写
wb = openpyxl.Workbook()
sheet = wb.active
print(sheet.title)
sheet.title = 'PythonSheet'
print(wb.get_sheet_names())
wb.create_sheet(index=0,title='First Sheet')
wb.create_sheet(index=2,title='Three Sheet')
wb.create_sheet(index=1,title='Two Sheet')
print(wb.get_sheet_names())

#删除工作表
wb.remove_sheet(wb.get_sheet_by_name('Two Sheet'))
print(wb.get_sheet_names())
wb.save('example_copy.xlsx')
```

执行结果如图 5-8 所示：

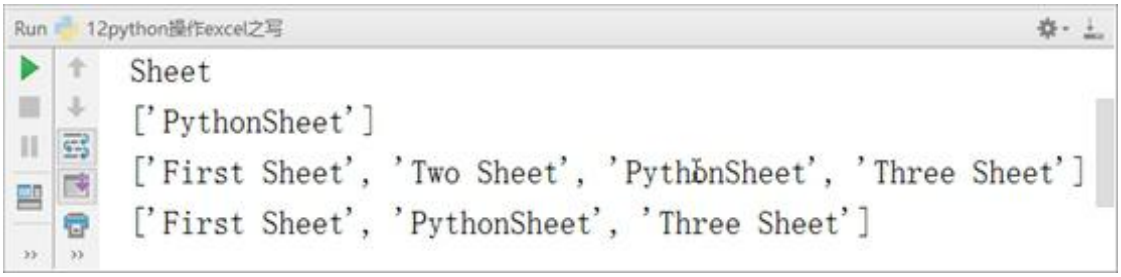


图 5-8 示例 5-11 运行效果图

【示例 5-12】使用 list 写入

```
import openpyxl
#向工作单元写内容
wb = openpyxl.Workbook()
sheet = wb.active
sheet['A1'] = 'Hello python'
print(sheet['A1'].value)

print('使用列表 list 写入'.center(20,'*'))
ws = wb.create_sheet('List sheet')
rows =[['Number','Batch 1','Batch2'],
        [2,40,30],
        [3,40,25],
        [4,30,24],
        [5,26,28],
        [6,32,21],
        [7,28,31]]
for row in rows:
    ws.append(row)
```

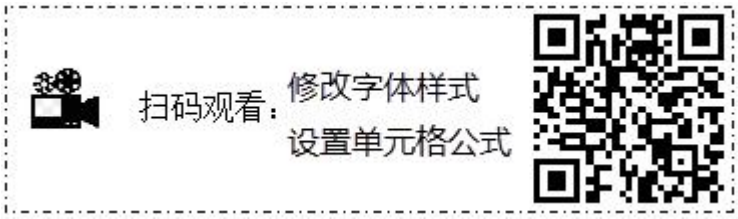
【示例 5-13】使用 range 方式写入

```
import openpyxl
#向工作单元写内容
wb = openpyxl.Workbook()
ws2 = wb.create_sheet('range names')
for row in range(1,40):
    ws2.append(range(17))
```

【示例 5-14】使用 cell()方法写入

```
import openpyxl
from openpyxl.utils import get_column_letter
#向工作单元写内容
wb = openpyxl.Workbook()
ws3 = wb.create_sheet(title='Data')
for row in range(5,30):
    for col in range(15,54):
        ws3.cell(column=col,row=row,value=get_column_letter(col))
wb.save('example_copy.xlsx')
```

5.2.3 Python 操作 Excel 之修改样式



【示例 5-15】修改字体样式

```
import openpyxl
from openpyxl.styles import Font
from openpyxl.styles import colors
wb = openpyxl.Workbook()
#对字体对象的修改
ws = wb.active
ws.title = 'Font'

#创建字体对象
font = Font(size=24,italic=True)
ws['B3'].font = font
ws['B3'] = '24pt Italic'

boldRedFont = Font(name='微软雅黑',bold=True,color=colors.RED)
ws['A1'].font = boldRedFont
ws['A1'] = 'Bold Red  微软雅黑'
wb.save('style_excelTest.xlsx')
```

执行结果如图 5-9 所示：



图 5-9 示例 5-15 运行效果图

【示例 5-16】设置单元格公式

```
import openpyxl
from openpyxl.styles import Font
from openpyxl.styles import colors
wb = openpyxl.Workbook()
ws = wb.active
#设置单元格公式
ws = wb.create_sheet('Formula')
ws['A1'] = 20
ws['A2'] = 30
ws['A3'] = '=SUM(A1:A2)'
wb.save('style_excelTest.xlsx')
```

执行结果如图 5-10 所示：

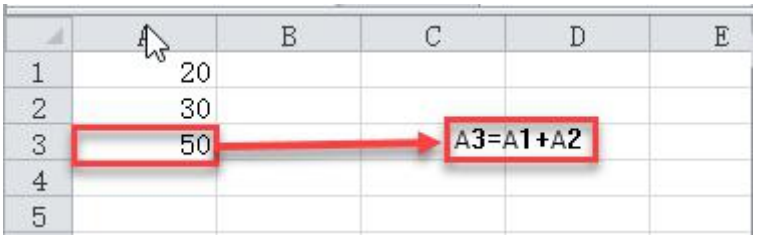
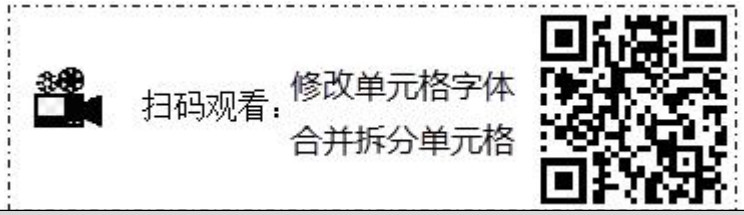


图 5-10 示例 5-16 运行效果图

【示例 5-17】设置行高和列宽




```
import openpyxl
from openpyxl.styles import Font
from openpyxl.styles import colors
wb = openpyxl.Workbook()
ws = wb.active
```

```
#设置行高和列宽
ws = wb.create_sheet('dimesions')
ws['A1'] = 'Tall row'
ws.row_dimensions[1].height = 70
ws['B2'] = 'Wide column'
ws.column_dimensions['B'].width = 20
wb.save('style_excelTest.xlsx')
```

执行结果如图 5-11 所示：



图 5-11 示例 5-17 运行效果图

【示例 5-18】合并单元格

```
import openpyxl
from openpyxl.styles import Font
from openpyxl.styles import colors
wb = openpyxl.Workbook()
ws = wb.active
#合并单元格
ws = wb.create_sheet('merged')
ws.merge_cells('A1:D3')
ws['A1'] = 'Twelve cells merged together'
ws.merge_cells('C5:D5')
ws['C5'] ='Two merged cells'
wb.save('style_excelTest.xlsx')
```

执行结果如图 5-12 所示：

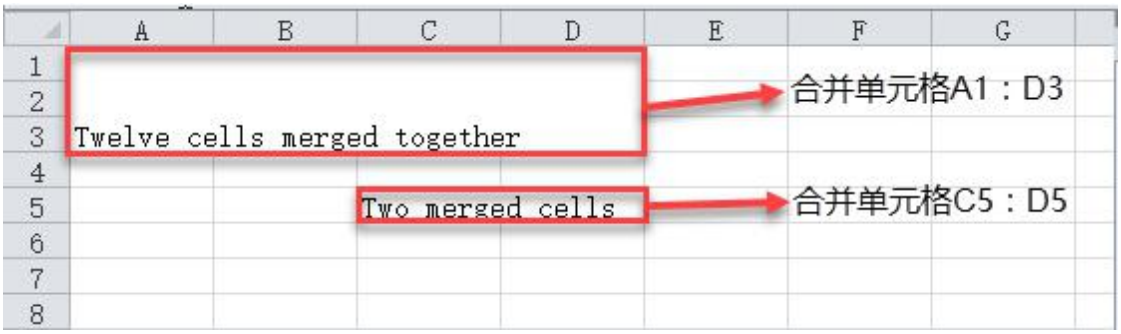


图 5-12 示例 5-18 运行效果图

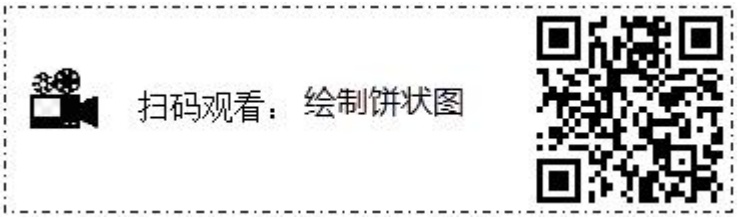
【示例 5-19】拆分单元格

```
import openpyxl
from openpyxl.styles import Font
from openpyxl.styles import colors
wb = openpyxl.Workbook()
ws = wb.active
#拆分单元格
ws = wb.copy_worksheet(wb.get_sheet_by_name('merged'))
ws.title = 'unmerged'
ws.unmerge_cells('A1:D3')
ws.unmerge_cells('C5:D5')
wb.save('style_excelTest.xlsx')
```

5.2.4 Python 操作 Excel 之图表

1) 饼图

饼图将数据绘制为一个圆的切片，每个切片代表整个百分比。切片按顺时针方向绘制，圆的顶部为 0°。



【示例 5-20】饼图

```
import openpyxl
from openpyxl.chart import (
PieChart,
Reference,
)
```

```
#绘制饼状图
wb = openpyxl.Workbook()
ws = wb.active
ws.title = 'pieChart'
data = [
    ['Pie','Sold'],
    ['Apple',50],
    ['Cherry',30],
    ['Pumpkin',10],
    ['Chocolate',40],
]
#单元格写入数据
for row in data:
    ws.append(row)
#创建饼图对象
pie = PieChart()
labels = Reference(ws,min_col=1,min_row=2,max_row=5)
data = Reference(ws,min_col=2,min_row=2,max_row=5)
pie.add_data(data)
pie.set_categories(labels)
pie.title = 'Pies sold by category'
ws.add_chart(pie,'D1')
wb.save('char_excel_text.xlsx')
```

执行结果如图 5-13 所示：

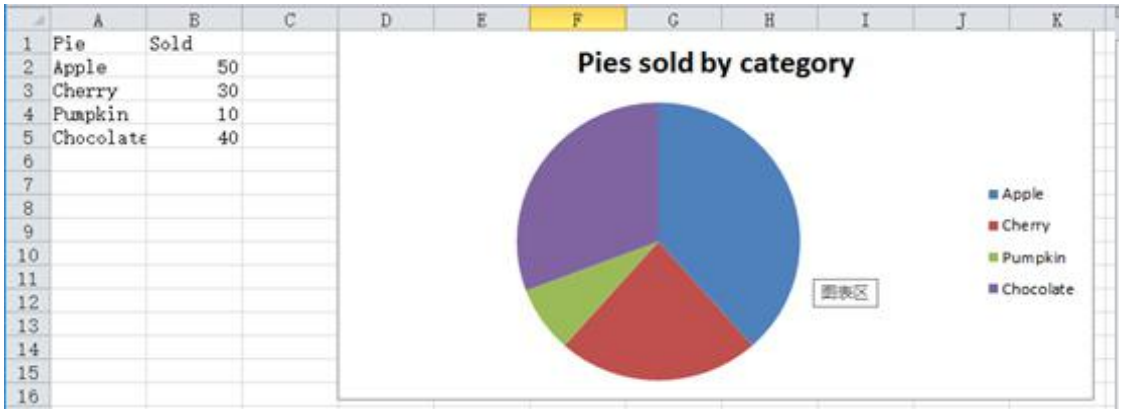
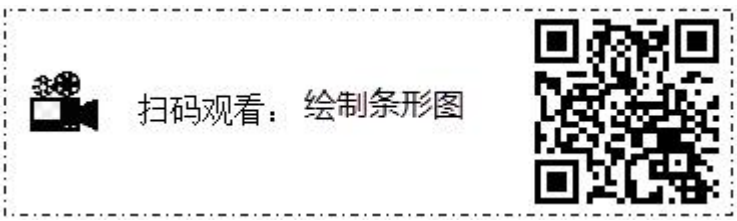


图 5-13 示例 5-20 运行效果图

2) **条形图**

在条形图中，值被绘制



为水平条或垂直列。可以通过 type 属性来设置。绘制垂直的条形图则使用如下：

```
chart1.type = 'col'
```

绘制成水平条形图示例如下：

```
chart1.type = 'bar'
```

【示例 5-21】条形图

```
import openpyxl
from openpyxl.chart import (
BarChart,
Reference,
)
rows =[['Number','Batch1','Batch2'],
        [2,40,30],
        [3,40,25],
        [4,30,24],
        [5,26,28],
        [6,32,21],
        [7,28,31]]
ws = wb.create_sheet('barChart')
for row in rows:
    ws.append(row)

chart1 = BarChart()
chart1.type = 'col'    #bar    垂直或水平
chart1.style = 15
chart1.title = 'Bar Chart'
chart1.y_axis.title = 'Sample length(mm)'
chart1.x_axis.title = 'Test number'
cats = Reference(ws,min_col=1,min_row=2,max_row=7)
data = Reference(ws,min_col=2,max_col=3,min_row=1,max_row=7)
chart1.add_data(data,titles_from_data=True)
chart1.set_categories(cats)
ws.add_chart(chart1,'E1')
wb.save('char_excel_text.xlsx')
```

执行结果如图 5-14 所示：

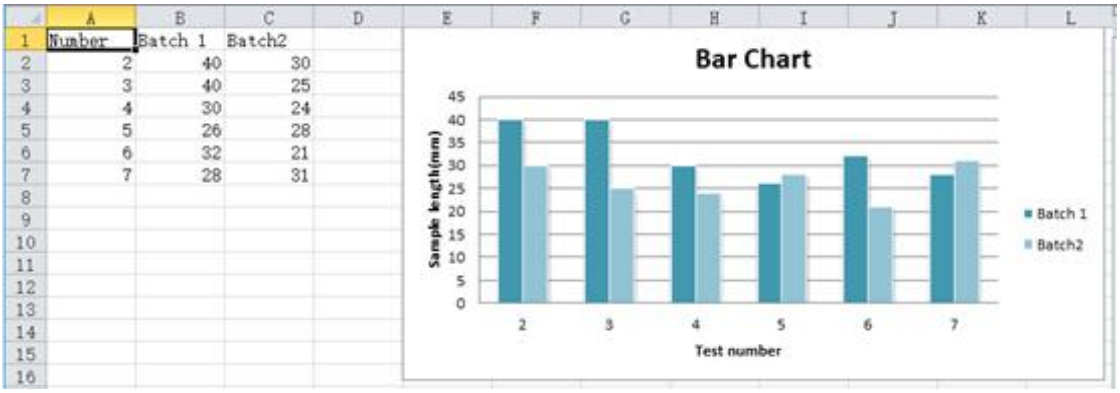
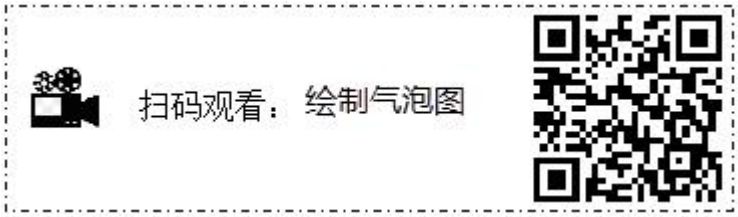


图 5-14 示例 5-21 运行效果图

3) 气泡图

气泡图类似于散点图，但使用第三维来确定气泡的大小。图表可以包括多个系列。



【示例 5-22】气泡图

```
import openpyxl
#绘制气泡图
from openpyxl.chart import (
Series,
BubbleChart,
Reference,
)
rows =  [
    ('商品数量','在 USD 销售','市场份额'),
    (14,12200,15),
    (20,60000,33),
    (18,24400,10),
    (22,32000,42),
    (),
    (12,8200,18),
    (15,50000,30),
    (19,22400,15),
    (25,25000,50)
]
```

```
ws = wb.create_sheet('BubbleChart')
for row in rows:
    ws.append(row)
#创建气泡图
chart = BubbleChart()
chart.style = 18
#添加第一组数据
xvalues = Reference(ws,min_col=1,min_row=2,max_row=5)
yvalues = Reference(ws,min_col=2,min_row=2,max_row=5)
size = Reference(ws,min_col=3,min_row=2,max_row=5)
series = Series(values=yvalues,xvalues=xvalues,zvalues=size,title='2013')
chart.series.append(series)

#添加第二组数据
xvalues = Reference(ws,min_col=1,min_row=7,max_row=10)
yvalues = Reference(ws,min_col=2,min_row=7,max_row=10)
size = Reference(ws,min_col=3,min_row=7,max_row=10)
series = Series(values=yvalues,xvalues=xvalues,zvalues=size,title='2014')
chart.series.append(series)
ws.add_chart(chart,'E1')
```

执行结果如图 5-15 所示：

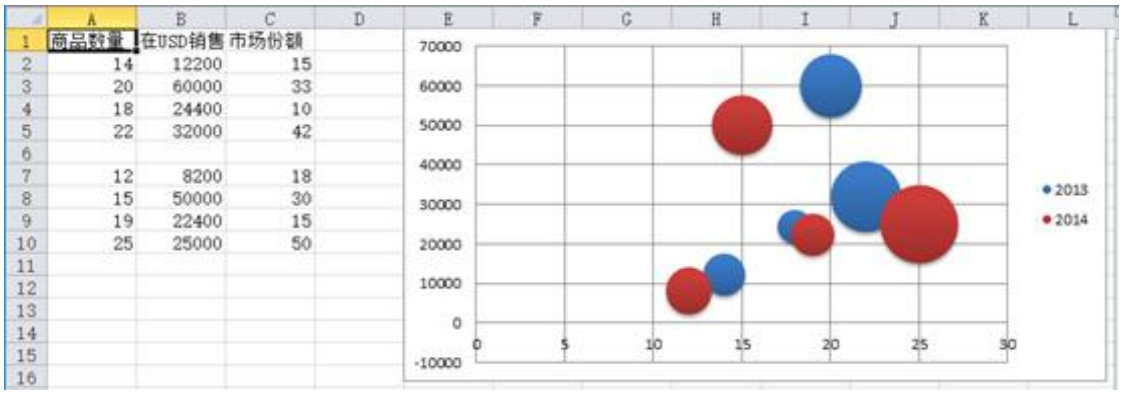
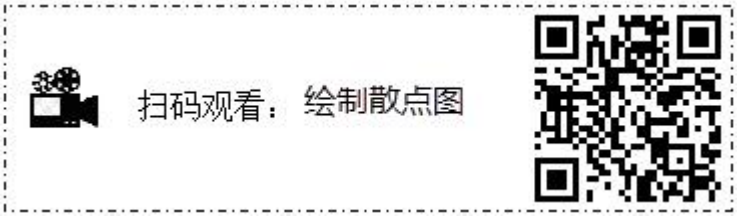


图 5-15 示例 5-22 运行效果图

4) 散点图

散点图或 xy 图类似于某些折线图。主要的区别是一个系列的值相对于另一个系列。当值无序时，这很有用。



【示例 5-23】散点图

```
import openpyxl
from openpyxl.chart import (
BarChart,
Reference,
ScatterChart
)
#绘制散点图
rows = [
    ['Size', 'Batch 1', 'Batch 2'],
    [2, 40, 30],
    [3, 40, 25],
    [4, 50, 30],
    [5, 30, 25],
    [6, 25, 35],
    [7, 20, 40],
]

ws = wb.create_sheet('Scatter Chart')
#添加数据
for row in rows:
    ws.append(row)
chart = ScatterChart()
chart.title = '散点图'
chart.style = 13
chart.x_axis.title = 'Size'
chart.y_axis.title = 'Percentage'
xvalues = Reference(ws, min_col=1, min_row=2, max_row=7)
for c in range(2,4):
    yvalues = Reference(ws,min_col=c,min_row=1,max_row=7)
    series = Series(values=yvalues,xvalues=xvalues,title_from_data=True)
    chart.series.append(series)
ws.add_chart(chart,'E1')
wb.save('char_excel_text.xlsx')
```

执行结果如图 5-16 所示：

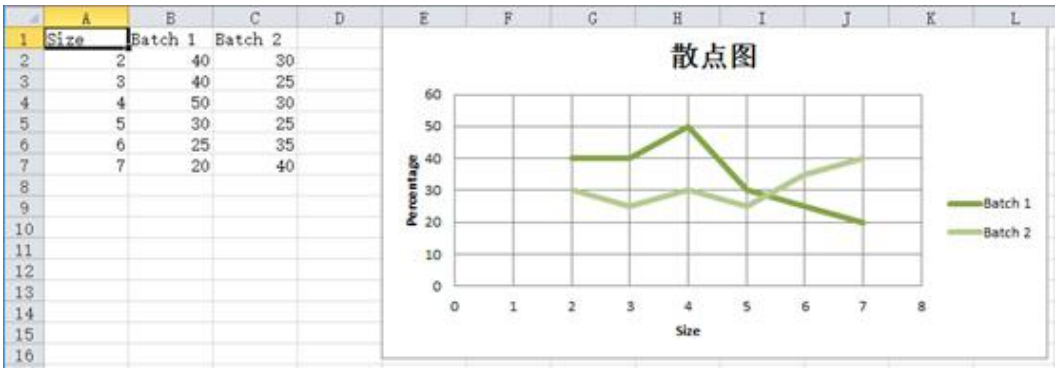
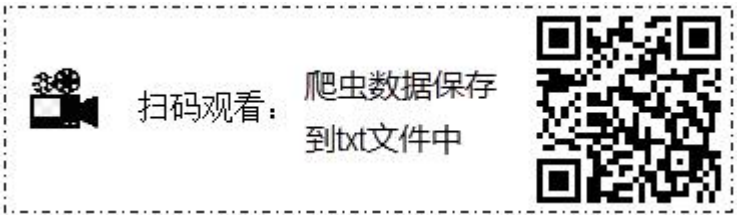


图 5-16 示例 5-23 运行效果图

5.3 数据存储

数据的存储是爬虫开发中一个很重要的环节，而存储的形式也分为很多种，大体来说分为两种。一种是将爬取到的数据储存到文件中，另一种就是将数据直接存储到数据库中。



本节讲解如何将爬取到的数据存储到 TXT，Excel 文件中。其实这些方法大同小异，原理都差不多，只不过是分别运用了不同的库函数，使用的不同的方法而已。

5.3.1 存储至 TXT

存到 TXT 文件是最简单也是最好理解的，主要是用到了 open 函数。我们爬取 http://seputu.com/，获取每个标题下的章节内容、链接。

首先使用 Requests 访问 http://seputu.com/，获取 HTML 文档内容。接着分析 http://seputu.com/首页的 HTML 结构，确定要抽取标记的位置，分析如下：

标题和章节都被包围在 标记下，标题位于其中的 下的<h2>中，章节位于其中的 下的<a>中，代码如下所示。

```
<div class="bg">
    <div class="mulu">
        <div class="mulu-title">
            <center>
                <h2>盗墓笔记 1 七星鲁王宫</h2>
            </center>
        </div>
        <div class="box">
            <ul>
                <li><a href="http://seputu.com/biji1/1.html" title="[2012-5-19 3:3:52] 七星鲁王 第一章 血尸">七星鲁王 第一章 血尸</a></li>
```

【示例 5-24】爬取数据存储至 txt

```
import requests
from bs4 import BeautifulSoup
headers = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.139 Safari/537.36'
}
url = 'http://seputu.com/'
r = requests.get(url,headers=headers)
soup = BeautifulSoup(r.text,'html.parser')
for mulu in soup.find_all(class_='mulu'):
    h2 = mulu.find('h2')
    if h2 != None:
        h2_title = h2.string   #获取标题
        #每个标题的章节内容和 url 保存到该标题的 txt 文件中
        #创建 txt 文件
        with open('e:/txt_data/'+h2_title+'.txt','a',encoding='utf8') as f:
            #获取所有的 a 标记中的 url 和章节内容
            for a in mulu.find(class_='box').find_all('a'):
                href = a.get('href')
                box_title = a.get('title')
                f.write(box_title+'\t'+href+'\n')
```

执行结果如图 5-17 所示：

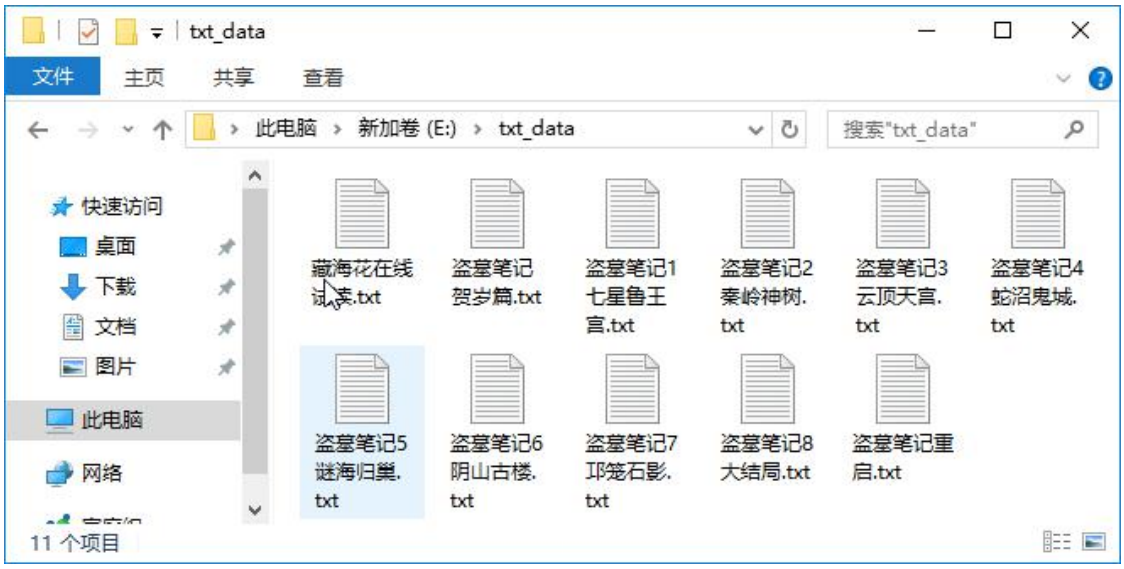
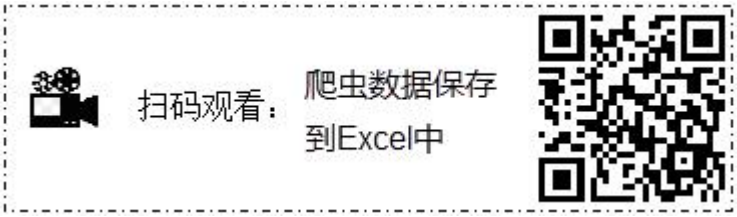


图 5-17 示例 5-24 运行效果图

5.3.2 存储至 excel



上述示例将获取到的章节内容和链接保存到 txt 中，下面我们将数据保存到 excel。

【示例 5-25】爬取数据保存到 excel

```
import requests
from bs4 import BeautifulSoup
import openpyxl
user_agent = 'Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.113 Mobile Safari/537.36'
headers = {'User-Agent':user_agent}
r = requests.get('http://seputu.com/',headers=headers)
soup = BeautifulSoup(r.text,'html.parser')
#创建工作簿
wb = openpyxl.Workbook()
for mulu in soup.find_all(class_='mulu'):
    h2 = mulu.find('h2')
    if h2!= None :
        h2_title = h2.string
        #每个标题创建一个 sheet
        #创建工作表 sheet
```

```
            ws = wb.create_sheet(h2_title)
            ws.append(['txt','href','title'])
            for a in mulu.find(class_='box').find_all('a'):
                href = a.get('href')
                title = a.get('title')
                txt = a.text
                ws.append([txt,href,title])
    wb.save('fiction.xlsx')
```

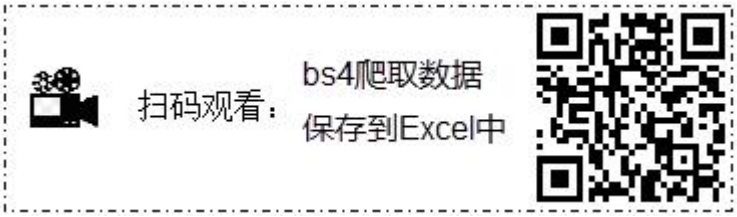
执行结果如图 5-18 所示：



	A	B	C
1	txt	href	title
2	七星鲁王	http://se	[2012-5-19 3:3:52] 七星鲁王 第一章 血尸
3	七星鲁王	http://se	[2012-5-19 3:5:22] 七星鲁王 第二章 五十年后
4	七星鲁王	http://se	[2012-5-19 3:6:15] 七星鲁王 第三章 瓜子庙
5	七星鲁王	http://se	[2012-5-19 3:30:57] 七星鲁王 第四章 尸洞
6	七星鲁王	http://se	[2012-5-19 4:52:12] 七星鲁王 第五章 水影
7	七星鲁王	http://se	[2012-5-19 4:53:14] 七星鲁王 第六章 积尸地
8	七星鲁王	http://se	[2012-5-22 14:42:52] 七星鲁王 第七章 一百多个人头
9	七星鲁王	http://se	[2012-5-22 14:43:24] 七星鲁王 第八章 山谷
10	七星鲁王	http://se	[2012-5-22 14:44:0] 七星鲁王 第九章 古墓
11	七星鲁王	http://se	[2012-5-22 14:44:35] 七星鲁王 第十章 影子
12	七星鲁王	http://se	[2012-5-22 14:45:8] 七星鲁王 第十一章 七星棺
13	七星鲁王	http://se	[2012-5-22 14:45:44] 七星鲁王 第十二章 门
14	七星鲁王	http://se	[2012-5-22 14:46:15] 七星鲁王 第十三章 02200059

Sheet 盗墓笔记1七星鲁王宫 盗墓笔记2秦岭神树 盗墓笔记3云顶天宫 盗墓笔记4蛇沼鬼城 盗墓笔记5谜海归

图 5-18 示例 5-25 运行效果图



保存零点书屋热门小说精选中的书名和链接到 excel。

首先访问零点书屋网址 https://www.lingdianshuwu.com。接着分析 HTML 文档内容，确定要抽取标记的位置。热门小说精选的书名及链接都被包含在<li class=”new_2”>标记下的<a>中，如图 5-19 所示。



图 5-19 热门小说精选 HTML 内容

【示例 5-26】使用 bs4 爬取数据保存到 excel

```
    import requests
```

```
import openpyxl
from bs4 import BeautifulSoup

#1.获取请求
resp = requests.get('https://www.lingdianshuwu.com')
resp.encoding='gb2312'
#2.解析数据
bs = BeautifulSoup(resp.text,'html.parser')
#3.获取数据
li_tags = bs.find_all('li',class_='new_2')

#4.提取数据
li_a =[['小说名称','链接']]
for item in li_tags:
    a_name = item.text    #  名称
    a_url = 'https://www.lingdianshuwu.com/'+item.find('a')['href']
    li_a.append([a_name,a_url])

#将数据保存到 excel 中
wb = openpyxl.Workbook()
ws = wb.active
ws.title = '畅销小说'
for row in li_a:
    ws.append(row)
wb.save('story.xlsx')
```

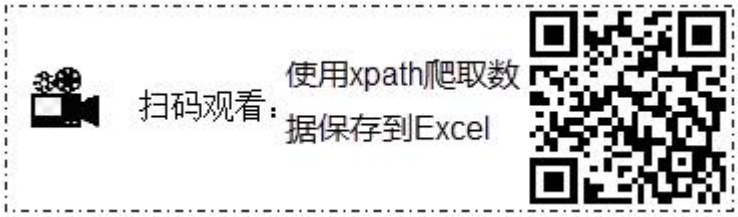
执行结果如图 5-20 所示：

	A	B	C	D	E	F	G
1	小说名称	链接					
2	天神诀	https://www.lingdianshuwu.com/html/books/3572517.html					
3	寒门崛起	https://www.lingdianshuwu.com/html/books/1763276.html					
4	赘婿	https://www.lingdianshuwu.com/html/books/202524.html					
5	天道图书馆	https://www.lingdianshuwu.com/html/books/3608347.html					
6	儒道至圣	https://www.lingdianshuwu.com/html/books/314344.html					
7	都市超级医	https://www.lingdianshuwu.com/html/books/3614360.html					
8	重生完美时	https://www.lingdianshuwu.com/html/books/3181958.html					
9	天醒之路	https://www.lingdianshuwu.com/html/books/292802.html					
10	超级怪兽工	https://www.lingdianshuwu.com/html/books/3335289.html					
11	放开那个女	https://www.lingdianshuwu.com/html/books/3364931.html					
12	我的1979	https://www.lingdianshuwu.com/html/books/3464155.html					
13	幻游猎人	https://www.lingdianshuwu.com/html/books/3534390.html					
14	独步天途	https://www.lingdianshuwu.com/html/books/3563273.html					

畅销小说

图 5-20 示例 5-26 运行效果图

保存起点月票榜的书名、作者到 excel 中。



首先访问起点月票网址 https://www.qidian.com/rank/yuepiao。接着分析 HTML 文档内容，确定要抽取标记的位置。起点月票榜的小说名称在 下的<h4>下的<a>标记中，作者在<p class='author'>下的第一个<a>中，如图 5-21 所示。

```
▼<div class="book-mid-info">
  ▼<h4>
     <a href="//book.qidian.com/info/1013562540" target="_blank" data-eid="qd_C40" data-bid="1013562540">第一序列</a>
   </h4>
  ▼<p class="author">
     <img src="//qidian.gtimg.com/qd/images/ico/user.f22d3.png">
     <a class="name" href="//me.qidian.com/authorIndex.aspx?id=5175790" target="_blank" data-eid="qd_C41">会说话的肘子</a>
     <em>|</em>
     <a href="//www.qidian.com/dushi" target="_blank" data-eid="qd_C42">都市</a>
     <em>|</em>
     <span>连载</span>
   </p>
  ▶<p class="intro">…</p>
  ▶<p class="update">…</p>
</div>
```

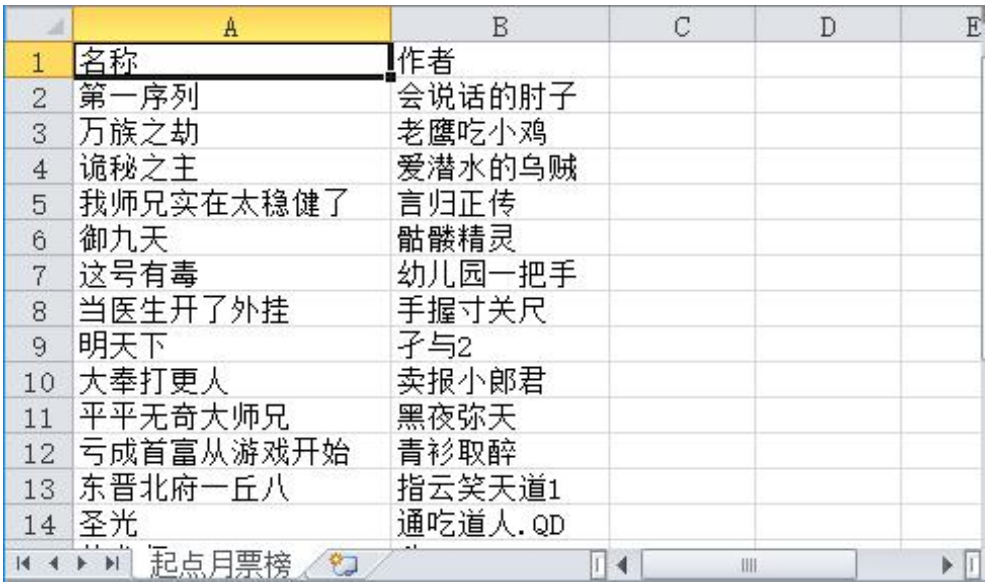
图 5-21 起点月票榜 HTML 内容

【示例 5-27】使用 XPath 爬取数据保存到 excel

```
import    requests
from lxml import etree
import openpyxl
#请求 url
url = 'https://www.qidian.com/rank/yuepiao'
```

```
    header = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.108 Safari/537.36'}
    resp = requests.get(url,headers=header)
    e = etree.HTML(resp.text)
    names = e.xpath('//div[@class="book-mid-info"]/h4/a/text()')
    authors = e.xpath('//p[@class="author"]/a[1]/text()')
    # 提取数据
    li_a =[['名称','作者']]
    for name,author in zip(names,authors):
        li_a.append([name,author])
    #将数据保存到 excel 中
    wb = openpyxl.Workbook()
    ws = wb.active
    ws.title = '起点月票榜'
    for row in li_a:
        ws.append(row)
    wb.save('qidian.xlsx')
```

执行结果如图 5-22 所示：



	A	B	C	D	E
1	名称	作者			
2	第一序列	会说话的肘子			
3	万族之劫	老鹰吃小鸡			
4	诡秘之主	爱潜水的乌贼			
5	我师兄实在太稳健了	言归正传			
6	御九天	骷髅精灵			
7	这号有毒	幼儿园一把手			
8	当医生开了外挂	手握寸关尺			
9	明天下	孑与2			
10	大奉打更人	卖报小郎君			
11	平平无奇大师兄	黑夜弥天			
12	亏成首富从游戏开始	青衫取醉			
13	东晋北府一丘八	指云笑天道1			
14	圣光	通吃道人.QD			

起点月票榜

图 5-22 示例 5-27 运行效果图

习题

一、 编码题

1. 纯文本文件 student.txt 为学生信息， 里面的内容（包括花括号）如下所示：

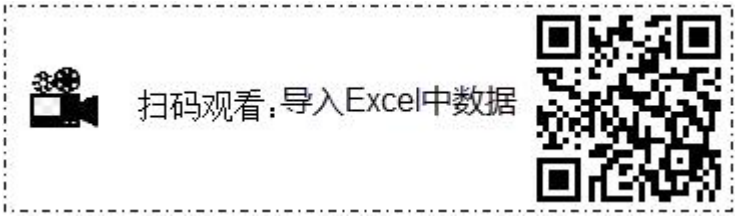
```
{ "1":["张三",150,120,100], "2":["李四",90,99,95], "3":["王五",60,66,68] }
```

请将上述内容写到 student.xls 文件中。

2. Python 操作 Excel，读取 student.xls 文件中的内容。
3. 使用 bs4 爬取零点书屋热门小说精选中的书名和链接保存到 bs4_data.xls 中。
4. Python 操作 Excel，读取 bs4_data.xls 文件中的内容。
5. 使用 XPath 爬取起点月票榜的书名、作者及链接到 xpath_data.xls 中。
6. Python 操作 Excel，读取 xpath_data.xls 文件中的内容。

第六章 Python 进行数据分析

6.1 导入外部数据



导入数据主要用到的是 Pandas 里的 read_x()方法，x 表示待导入文件的格式。

6.1.1 导入.xlsx 文件

Python 中导入.xlsx 文件的方法是 read_excel()。

1) 基本导入

使用 read_excel()方法导入文件，首先要指定文件的路径，也就是这个文件在电脑中的哪个文件夹下存着。

【示例 6-1】导入.xlsx 文件

```
import pandas as pd
df = pd.read_excel(r"E:\excelTest.xlsx")
df
```

执行结果如图 6-1 所示：

	姓名	年龄	工作	薪水	入职日期
0	张三	30	CEO	50000	2010/3/11
1	李四	24	主管	30000	2018/8/19
2	王五	25	经理	40000	2015/10/11

图 6-1 示例 6-1 运行效果图

电脑中的文件默认路径使用\，这个时候需要在路径前面加一个 r（转义符）避免路径里面的\被转义。也可以不加 r，但是需要把路径里面的所有\转换成/，这个规则在导入其他格式文件时也是一样的，一般在路径前面加 r。

2) 指定导入哪个 Sheet

.xlsx 格式的文件有多个 Sheet，可以通过设定 sheet_name 参数来指定要导入哪个 Sheet 的文件。

【示例 6-2】导入.xlsx 文件时，指定导入哪个 Sheet

```
import pandas as pd
df = pd.read_excel(r"E:\excelTest.xlsx",sheet_name='Sheet1')
df
```

执行结果如图 6-2 所示：

	姓名	年龄	工作	薪水	入职日期
0	张三	30	CEO	50000	2010/3/11
1	李四	24	主管	30000	2018/8/19
2	王五	25	经理	40000	2015/10/11

图 6-2 示例 6-2 运行效果图

除了可以指定具体 Sheet 的名字，还可以传入 Sheet 的顺序，从 0 开始计数。

【示例 6-3】导入.xlsx 文件时，传入 Sheet 的顺序

```
import pandas as pd
df = pd.read_excel(r"E:\excelTest.xlsx",sheet_name=0)
df
```

执行结果如图 6-3 所示：

	姓名	年龄	工作	薪水	入职日期
0	张三	30	CEO	50000	2010/3/11
1	李四	24	主管	30000	2018/8/19
2	王五	25	经理	40000	2015/10/11

图 6-3 示例 6-3 运行效果图

如果不指定 sheet_name 参数时，那么默认导入的都是第一个 Sheet 的文件。

3) 指定行索引

将本地文件导入 DataFrame 时，行索引使用的从 0 开始的默认索引，可以通过设置 index_col 参数来设置。

【示例 6-4】导入.xlsx 文件时，通过 index_col 指定行索引

```
import pandas as pd
df = pd.read_excel(r"E:\excelTest.xlsx",sheet_name=0,index_col=0)
df
```

执行结果如图 6-4 所示：

姓名	年龄	工作	薪水	入职日期
张三	30	CEO	50000	2010-03-11
李四	24	主管	30000	2018-08-19
王五	25	经理	40000	2015-10-11

图 6-4 示例 6-4 运行效果图

index_col 表示用.xlsx 文件中的第几列做行索引，从 0 开始计数。

4) 指定列索引

将本地文件导入 DataFrame 时，默认使用源数据表的第一行作为列索引，也可以通过设置 header 参数来设置列索引。header 参数值默认为 0，即用第一行作为列索引；也可以是其他行，只需要传入具体的那一行即可；

【示例 6-5】导入.xlsx 文件时，通过 header 指定行索引

```
import pandas as pd
df = pd.read_excel(r"E:\excelTest.xlsx",header=1)
df
```

执行结果如图 6-5 所示：

	张三	30	CEO	50000	2010-03-11 00:00:00
0	李四	24	主管	30000	2018-08-19
1	王五	25	经理	40000	2015-10-11

图 6-5 示例 6-5 运行效果图

【示例 6-6】导入.xlsx 文件时，通过 header 指定行索引

```
import pandas as pd
df = pd.read_excel(r"E:\excelTest.xlsx",header=None)
df
```

执行结果如图 6-6 所示：

	0	1	2	3	4
0	姓名	年龄	工作	薪水	入职日期
1	张三	30	CEO	50000	2010-03-11 00:00:00
2	李四	24	主管	30000	2018-08-19 00:00:00
3	王五	25	经理	40000	2015-10-11 00:00:00

图 6-6 示例 6-6 运行效果图

5) 指定导入列

有时候本地文件的列数太多，而我们又不需要那么多列时，我们就可以通过设置 usecols 参数来指定要导入的列。

【示例 6-7】导入.xlsx 文件时，通过 usecols 指定列索引

```
import pandas as pd
df = pd.read_excel(r"E:\excelTest.xlsx",usecols=1)
df
```

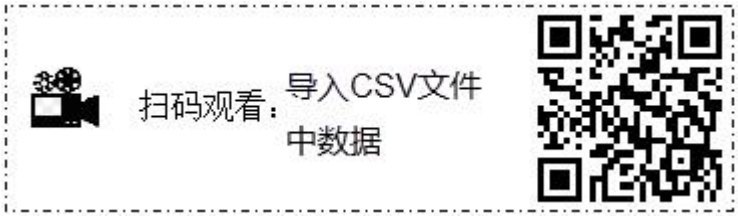
执行结果如图 6-7 所示：

	姓名	年龄
0	张三	30
1	李四	24
2	王五	25

图 6-7 示例 6-7 运行效果图

6.1.2 导入.csv 文件

1) 直接导入，指定编码格式



导入时除了指明文件路径，还需要设置编码格式。Python 中用得比较多的两种编码格式是 UTF-8 和 gbk，默认编码格式是 UTF-8。我们要根据导入文件本身的编码格式进行设置，通过设置参数 encoding 来设置导入的编码格式。

【示例 6-8】导入.csv 文件，文件编码格式是 gbk

```
import pandas as pd
```

```
df = pd.read_csv(r"E:\excelTest.csv",encoding='gbk')
df
```

执行结果如图 6-8 所示：

	姓名	年龄	工作	薪水	入职日期
0	张三	30	CEO	50000	2010/3/11
1	李四	24	主管	30000	2018/8/19
2	王五	25	经理	40000	2015/10/11

图 6-8 示例 6-8 运行效果图

2) 指明分隔符

在 Excel 和 DataFrame 中的数据都是很规整的排列的，这都是工具在后台根据某条规则进行切分的。read_csv()默认文件中的数据都是以逗号分开的，但是有的文件不是用逗号分开的，这时候就需要人为指定分隔符号，否则就会报错。

【示例 6-9】导入.csv 文件，指明分隔符

```
import pandas as pd
df = pd.read_csv(r"E:\excelTest.csv",encoding='gbk',sep=' ')
df
df = pd.read_csv(r"E:\excelTest.csv",encoding='gbk',sep=', ')
df
```

执行结果如图 6-9 所示：

```
import pandas as pd
df = pd.read_csv(r"E:\excelTest.csv",encoding='gbk',sep=' ')
df
```

	姓名,年龄,工作,薪水,入职日期
0	张三,30,CEO,50000,2010/3/11
1	李四,24,主管,30000,2018/8/19
2	王五,25,经理,40000,2015/10/11

```
df = pd.read_csv(r"E:\excelTest.csv",encoding='gbk',sep=',')
df
```

	姓名	年龄	工作	薪水	入职日期
0	张三	30	CEO	50000	2010/3/11
1	李四	24	主管	30000	2018/8/19
2	王五	25	经理	40000	2015/10/11

图 6-9 示例 6-9 运行效果图

从上面的示例可以看到，如果使用空格进行分隔，获取到的数据还是一个整体，并没有分开。用默认的逗号作为分隔符号进行分隔，数据被规整地分好了。常用的分隔符除了逗号、空格，还有制表符（\t）。

3) 指明读取行数

如果有一个几百兆的文件，想了解一席按这个文件里有哪些数据，那么这时候就没有必要把全部数据都导入，只要看到前面几行即可，可以通过设置 nrows 参数。

【示例 6-10】导入.csv 文件，nrows 参数的使用

```
import pandas as pd
pd.read_csv(r"E:\excelTest.csv",encoding='gbk',sep=',',nrows=1)
```

执行结果如图 6-10 所示：

	姓名	年龄	工作	薪水	入职日期
0	张三	30	CEO	50000	2010/3/11

图 6-10 示例 6-10 运行效果图

4) engine 指定

当文件路径或者文件名中包含中文时，如果还用上面的导入方式就会报错。因为当调用read_csv()方法时，默认使用 C 语言作为解析语言，可以通过设置 engine 参数，把默认值 C 改为 Python 就可以了，如果文件格式是 CSV UTF-8（逗号分隔）（*.csv），那么编码格式也需要跟着变为 utf-8-sig，如果文件格式是 CSV（逗号分隔）（*.csv）格式，对应的编码格式则为 gbk。

【示例 6-11】导入.csv 文件，engine 参数的使用

```
import pandas as pd
pd.read_csv(r"E:\文件\excelTest.csv",engine='python',encoding='gbk')
```

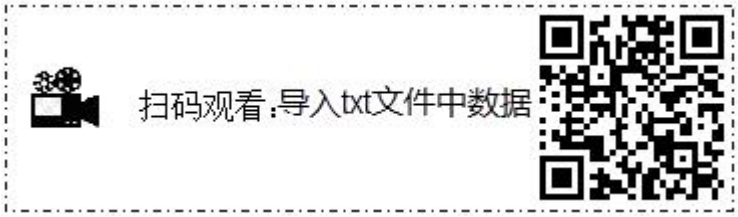
执行结果如图 6-11 所示：

	姓名	年龄	工作	薪水	入职日期
0	张三	30	CEO	50000	2010/3/11
1	李四	24	主管	30000	2018/8/19
2	王五	25	经理	40000	2015/10/11

图 6-11 示例 6-11 运行效果图

.csv 文件也涉及行、列索引设置及指定导入某列或某几列，设定方法与导入.xlsx 文件一致。

6.1.3 导入.txt 文件



导入.txt 文件用得方法时 read_table()，read_table()是将利用分隔符分开的文件导入 DataFrame 的通用函数。它不仅仅可以导入.txt 文件，还可以导入.csv 文件。

【示例 6-12】导入.txt 文件

```
import pandas as pd
pd.read_table(r"E:\testFile.txt",encoding='gbk',sep='\t')
```

执行结果如图 6-12 所示：

	姓名	年龄	工作	薪水	入职日期
0	张三	30	CEO	50000	2010/3/11
1	李四	24	主管	30000	2018/8/19
2	王五	25	经理	40000	2015/10/11

图 6-12 示例 6-12 运行效果图

【示例 6-13】使用 read_table()方法导入.csv 文件

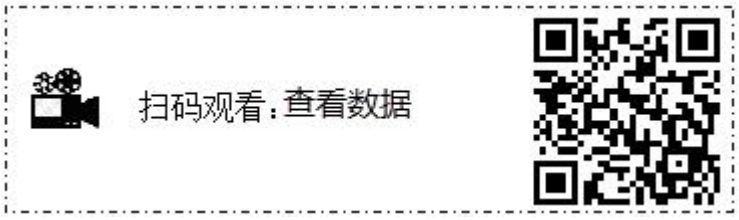
```
import pandas as pd
pd.read_table(r"E:\excelTest.csv",encoding='gbk',sep=',')
```

执行结果如图 6-13 所示：

	姓名	年龄	工作	薪水	入职日期
0	张三	30	CEO	50000	2010/3/11
1	李四	24	主管	30000	2018/8/19
2	王五	25	经理	40000	2015/10/11

图 6-13 示例 6-13 运行效果图

6.2 查看数据



有了数据以后，先查看数据，只有对数据充分熟悉后，才能更好地进行分析。

6.2.1 预览前、后几行

当数据表中包含数据行数过多时，我们只想看一下每一列数据都是什么样，可以只把数据的前几行或者后几行进行查看。

导入一个文件后，可以用 head()方法来控制要显示的前几行，tail()方法来控制要显示的后几行。在 head()、tail()方法中直接设置行数，默认展示 5 行。

【示例 6-14】预览前、后几行

```
import pandas as pd
df = pd.read_excel(r"E:\excelTest.xlsx")
display(df,df.head(),df.tail(2))
```

执行结果如图 6-14 所示：

	姓名	年龄	工作	薪水	入职日期
0	A1	30	CEO	50000	2010-03-11
1	B1	24	主管	30000	2018-08-19
2	C1	25	经理	40000	2015-10-11
3	A2	23	员工	2000	2017-04-23
4	B2	28	主任	3200	2013-11-13
5	C2	25	员工	2200	2016-09-12
6	D2	26	员工	2400	2018-10-12

	姓名	年龄	工作	薪水	入职日期
0	A1	30	CEO	50000	2010-03-11
1	B1	24	主管	30000	2018-08-19
2	C1	25	经理	40000	2015-10-11
3	A2	23	员工	2000	2017-04-23
4	B2	28	主任	3200	2013-11-13

	姓名	年龄	工作	薪水	入职日期
5	C2	25	员工	2200	2016-09-12
6	D2	26	员工	2400	2018-10-12

图 6-14 示例 6-14 运行效果图

6.2.2 调用 shape 获取数据表的大小

调用 shape 获取数据表的行、列数。

【示例 6-15】调用 shape 获取数据表的行、列数

```
import pandas as pd
df = pd.read_excel(r"E:\excelTest.xlsx")
display(df,df.shape)
```

执行结果如图 6-15 所示：

	姓名	年龄	工作	薪水	入职日期
0	A1	30	CEO	50000	2010-03-11
1	B1	24	主管	30000	2018-08-19
2	C1	25	经理	40000	2015-10-11
3	A2	23	员工	2000	2017-04-23
4	B2	28	主任	3200	2013-11-13
5	C2	25	员工	2200	2016-09-12
6	D2	26	员工	2400	2018-10-12

(7, 5)

图 6-15 示例 6-15 运行效果图

shape 方法会以元组的形式返回行、列数，上面代码中的（7,5）表示 df 表有 7 行 5 列数据。这里需要注意的是，Python 中利用 shape 方法获取行数和列数时不会把行索引和列索引计算在内，而 Excel 中是把行索引和列索引计算在内的。

6.2.3 调用 info()获取数据类型

查看数据的第二点就是看一下数据类型，不同的数据类型的分析思路是不一样的，比如数值类型的数据可以求均值，但是字符串类型的数据就没法求均值了。

利用 info()查看数据表中的数据类型，而且不需要一列一列查看，在调用 info()方法以后就会输出整个表中所有列的数据类型。

【示例 6-16】调用 info()获取数据类型

```
import pandas as pd
df = pd.read_excel(r"E:\excelTest.xlsx")
df.info()
```

执行结果如图 6-16 所示：

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7 entries, 0 to 6
Data columns (total 5 columns):
姓名      7 non-null object
年龄      7 non-null int64
工作      7 non-null object
薪水      7 non-null int64
入职日期    7 non-null datetime64[ns]
dtypes: datetime64[ns](1), int64(2), object(2)
memory usage: 360.0+ bytes
```

图 6-16 示例 6-16 运行效果图

通过 info()方法可以看出表 df 的行索引 index 是 0-6，共 5columns，分别是姓名、年龄、工作、薪水及入职时间，且 5columns 中年龄和薪水是 int 类型，入职日期是 datetime 类型，其他 columns 都是 object 类型，共占用内存 360bytes。

6.2.4 调用 describe()获取数值分布情况

查看数据的第三点就是掌握数值的分布情况，即均值是多少，最大最小是多少，方差即分位数分别又是多少。调用 describe()方法就可以获取所有数值类型字段的分布值。

【示例 6-17】调用 describe()获取数值分布情况

```
import pandas as pd
df = pd.read_excel(r"E:\excelTest.xlsx")
df.describe()
```

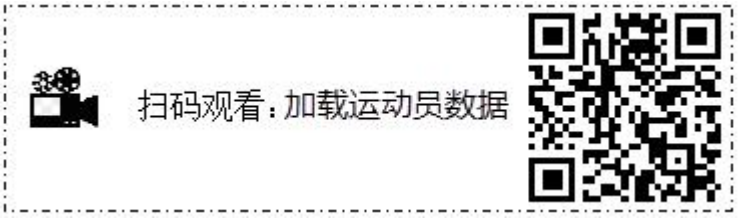
执行结果如图 6-17 所示：

	年龄	薪水
count	7.000000	7.000000
mean	25.857143	18542.857143
std	2.410295	20888.502192
min	23.000000	2000.000000
25%	24.500000	2300.000000
50%	25.000000	3200.000000
75%	27.000000	35000.000000
max	30.000000	50000.000000

图 6-17 示例 6-17 运行效果图

6.3 足球运动员数据分析

6.3.1 加载数据



【示例 6-18】加载足球运动员数据

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
#支持中文
#解决中文显示问题
plt.rcParams['font.sans-serif'] = ['KaiTi'] #  指定默认字体
plt.rcParams['axes.unicode_minus'] = False #  解决保存图像是负号'-'显示为方块的问题
player = pd.read_csv('./data/FullData.csv')
```

6.3.2 查看数据

【示例 6-19】查看数据前 5 行

```
player.head()
```

执行结果如图 6-18 所示：

	Name	Nationality	National_Position	National_Kit	Club	Club_Position	Club_Kit	Club_Joining	Contract_Expiry	Rating	...	Long_Shots
0	Cristiano Ronaldo	Portugal	LS	7.0	Real Madrid	LW	7.0	07/01/2009	2021.0	94	...	90
1	Lionel Messi	Argentina	RW	10.0	FC Barcelona	RW	10.0	07/01/2004	2018.0	93	...	88
2	Neymar	Brazil	LW	10.0	FC Barcelona	LW	11.0	07/01/2013	2021.0	92	...	77
3	Luis Suárez	Uruguay	LS	9.0	FC Barcelona	ST	9.0	07/11/2014	2021.0	92	...	86
4	Manuel Neuer	Germany	GK	1.0	FC Bayern	GK	1.0	07/01/2011	2021.0	92	...	16

5 rows × 53 columns

图 6-18 示例 6-19 运行效果图

【示例 6-20】查看数据的数据类型

```
player.info()
```

执行结果如下所示：

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 17588 entries, 0 to 17587
Data columns (total 53 columns):
Name                    17588 non-null object
Nationality             17588 non-null object
National_Position       1075 non-null object
National_Kit            1075 non-null float64
Club                    17588 non-null object
Club_Position           17587 non-null object
Club_Kit                17587 non-null float64
Club_Joining            17587 non-null object
Contract_Expiry         17587 non-null float64
Rating                  17588 non-null int64
Height                  17588 non-null object
Weight                  17588 non-null object
Preffered_Foot          17588 non-null object
Birth_Date              17588 non-null object
Age                     17588 non-null int64
Preffered_Position      17588 non-null object
Work_Rate               17588 non-null object
Weak_foot               17588 non-null int64
Skill_Moves             17588 non-null int64
Ball_Control            17588 non-null int64
Dribbling               17588 non-null int64
Marking                 17588 non-null int64
Sliding_Tackle          17588 non-null int64
Standing_Tackle         17588 non-null int64
Aggression              17588 non-null int64
Reactions               17588 non-null int64
Attacking_Position      17588 non-null int64
Interceptions           17588 non-null int64
Vision                  17588 non-null int64
Composure               17588 non-null int64
Crossing                17588 non-null int64
Short_Pass              17588 non-null int64
Long_Pass               17588 non-null int64
```

```
Acceleration          17588 non-null int64
Speed                 17588 non-null int64
Stamina               17588 non-null int64
Strength              17588 non-null int64
Balance               17588 non-null int64
Agility               17588 non-null int64
Jumping               17588 non-null int64
Heading               17588 non-null int64
Shot_Power            17588 non-null int64
Finishing             17588 non-null int64
Long_Shots            17588 non-null int64
Curve                 17588 non-null int64
Freekick_Accuracy     17588 non-null int64
Penalties             17588 non-null int64
Volleys               17588 non-null int64
GK_Positioning        17588 non-null int64
GK_Diving             17588 non-null int64
GK_Kicking            17588 non-null int64
GK_Handling           17588 non-null int64
GK_Reflexes           17588 non-null int64
dtypes: float64(3), int64(38), object(12)
memory usage: 7.1+ MB
```

6.3.3 缺值处理

从上述示例可以看到总共 17588 行，但 National_Position（国家队位置） 是 1075 行，Club_Position （俱乐部位置）17587 行。我们知道有的足球运动员是没有进入国家队的，所以 National_Position 缺值是正常情况。但 Club_Position 缺值需要处理。

【示例 6-21】删除 Club_Position 的缺失值

```
player = player[player['Club_Position'].notnull()]
player.info()
```

执行结果如下所示：

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 17587 entries, 0 to 17587
```

```
Data columns (total 53 columns):
Name                    17587 non-null object
Nationality             17587 non-null object
National_Position       1075 non-null object
National_Kit            1075 non-null float64
Club                    17587 non-null object
Club_Position           17587 non-null object
Club_Kit                17587 non-null float64
Club_Joining            17587 non-null object
Contract_Expiry         17587 non-null float64
Rating                  17587 non-null int64
Height                  17587 non-null object
Weight                  17587 non-null object
Preffered_Foot          17587 non-null object
Birth_Date              17587 non-null object
Age                     17587 non-null int64
Preffered_Position      17587 non-null object
Work_Rate               17587 non-null object
Weak_foot               17587 non-null int64
Skill_Moves             17587 non-null int64
Ball_Control            17587 non-null int64
Dribbling               17587 non-null int64
Marking                 17587 non-null int64
Sliding_Tackle          17587 non-null int64
Standing_Tackle         17587 non-null int64
Aggression              17587 non-null int64
Reactions               17587 non-null int64
Attacking_Position      17587 non-null int64
Interceptions           17587 non-null int64
Vision                  17587 non-null int64
Composure               17587 non-null int64
Crossing                17587 non-null int64
Short_Pass              17587 non-null int64
Long_Pass               17587 non-null int64
Acceleration            17587 non-null int64
```

```
Speed                17587 non-null int64
Stamina              17587 non-null int64
Strength             17587 non-null int64
Balance              17587 non-null int64
Agility              17587 non-null int64
Jumping              17587 non-null int64
Heading              17587 non-null int64
Shot_Power           17587 non-null int64
Finishing            17587 non-null int64
Long_Shots           17587 non-null int64
Curve                17587 non-null int64
Freekick_Accuracy    17587 non-null int64
Penalties            17587 non-null int64
Volleys              17587 non-null int64
GK_Positioning       17587 non-null int64
GK_Diving            17587 non-null int64
GK_Kicking           17587 non-null int64
GK_Handling          17587 non-null int64
GK_Reflexes          17587 non-null int64
dtypes: float64(3), int64(38), object(12)
memory usage: 7.2+ MB
```

【示例 6-22】查看是否有重复数据

```
player.duplicated().any()
```

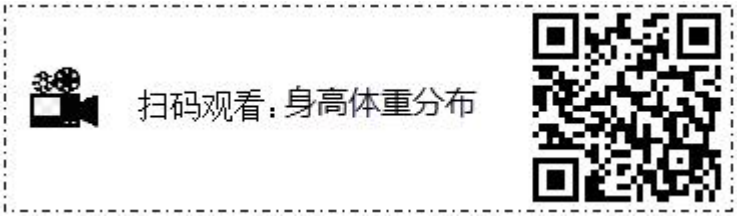
执行结果如图 6-19 所示：

```
#查看是否有重复数据
player.duplicated().any()

False
```

图 6-19 示例 6-22 运行效果图

6.3.4 运动员身高和体重分布

从查看数据结果可以看到运动员身高 Height、体重 Weight 的数据后都



添加了相应的单位。要分析运动员身高和体重的分布，首先需要将身高 Height 和 Weight 数据的单位去掉。

【示例 6-23】去掉 Height 和 Weight 数据的单位

```
player['Height'] = player['Height'].str.replace('cm','')
player['Weight'] = player['Weight'].str.replace('kg','')
player['Height'] = player['Height'].astype('int')
player['Weight'] = player['Weight'].astype('int')
display(player['Height'].head(),player['Weight'].head())
```

执行结果如图 6-20 所示：

```
display(player['Height'].head(),player['Weight'].head())
```

```
0    185
1    170
2    174
3    182
4    193
Name: Height, dtype: int32

0    80
1    72
2    68
3    85
4    92
Name: Weight, dtype: int32
```

图 6-20 示例 6-23 运行效果图

【示例 6-24】运动员 Height 数值分布

```
player['Height'].describe()
```

执行结果如图 6-21 所示，运动员 Height 数值分布的直方图 6-22 所示：

```
player['Height'].describe()
```

```
count    17587.000000
mean       181.105021
std          6.675084
min        155.000000
25%        176.000000
50%        181.000000
75%        186.000000
max        207.000000
Name: Height, dtype: float64
```

图 6-21 示例 6-24 运行效果图

```
player['Height'].plot(kind='hist')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x20bb9376c50>
```

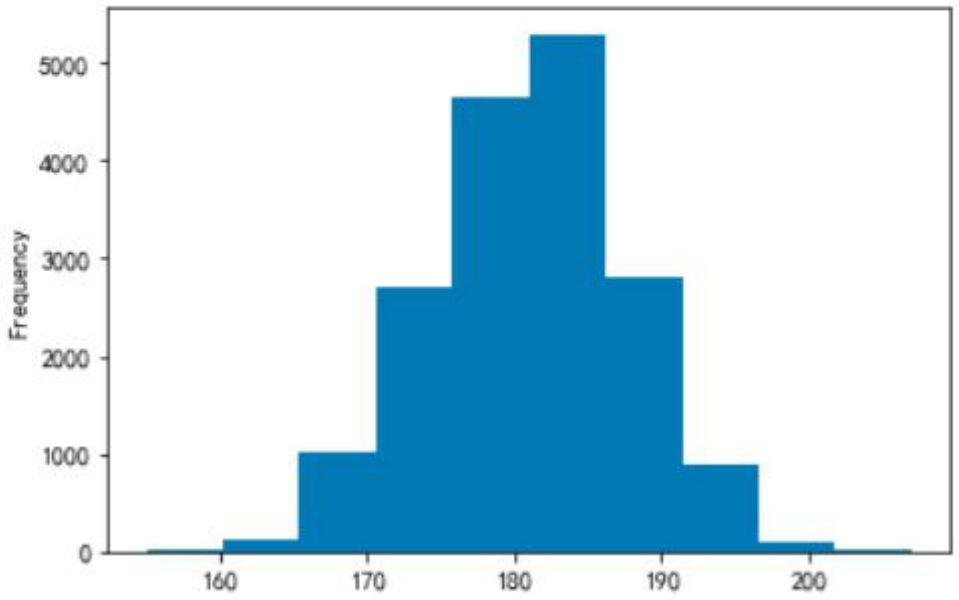


图 6-22 运动员 Height 直方图

【示例 6-25】运动员 Weight 数值分布

```
player['Weight'].describe()
```

执行结果如图 6-23 所示，运动员 Weight 数值分布的直方图 6-24 所示：

```
player['Weight'].describe()
```

```
count    17587.000000
mean        75.253085
std          6.898051
min         48.000000
25%         70.000000
50%         75.000000
75%         80.000000
max        110.000000
Name: Weight, dtype: float64
```

图 6-23 示例 6-25 运行效果图

```
player['Weight'].plot(kind='hist')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c05793dfd0>
```

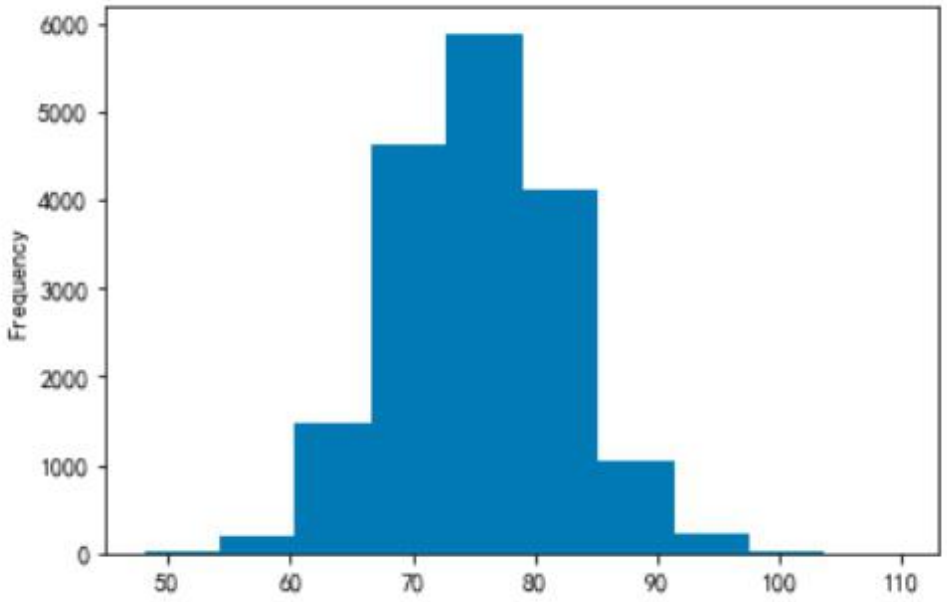
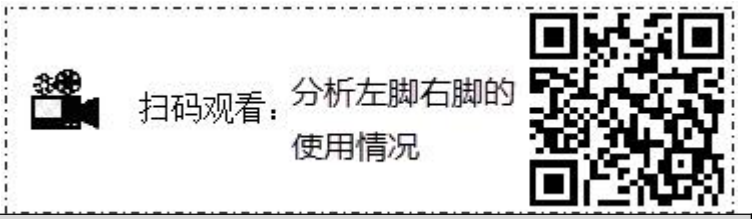


图 6-24 运动员 Weight 直方图

6.3.5 左脚右脚使用数量

【示例 6-26】查看足球运动员左脚右脚使用情况




```
player['Preffered_Foot'].head(10)
```

执行结果如图 6-25 所示：

```
player['Preffered_Foot'].head(10)
```

```
0    Right
1     Left
2    Right
3    Right
4    Right
5    Right
6    Right
7     Left
8    Right
9     Left
Name: Preffered_Foot, dtype: object
```

图 6-25 示例 6-26 运行效果图

从上述示例可以看到，足球运动员踢球有使用左脚也有使用右脚。要统计使用左脚和右脚的数量，需要按 Preffered_Foot 进行分组，计算其 count()值。我们可以使用饼状图来显示左脚右脚选手数量的差别。

【示例 6-27】使用饼状图来显示左脚右脚选手数量的差别

```
g = player.groupby('Preffered_Foot')
s = g['Preffered_Foot'].count()
s.plot(kind='pie',autopct='%.2f')
```

执行结果如图 6-26 所示：

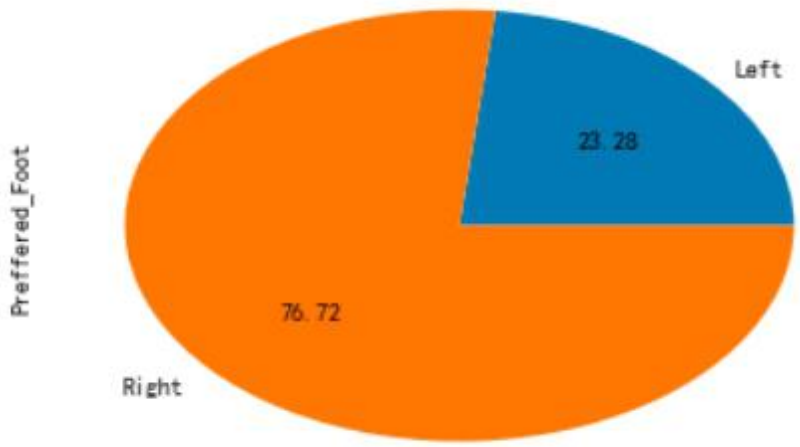


图 6-26 示例 6-27 运行效果图

按某列分组计算 count()值可以直接使用 value_counts()方法，示例如下：

【示例 6-28】value_counts()方法的使用

```
foot_counts = player['Preffered_Foot'].value_counts()
foot_counts.name=' '       #将左侧 Preffered_Foot 去掉
foot_counts.plot(kind='pie',autopct='%.2f')
```

执行结果如图 6-27 所示：

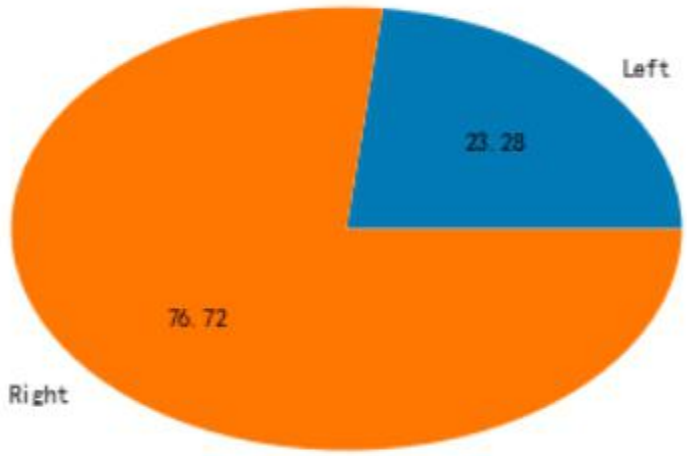


图 6-27 示例 6-28 运行效果图

6.3.6 相关性分析

通过散点图查看变量之间关系。



【示例 6-29】分析足球运动员的身高和体重是否相关

```
player.plot(kind='scatter',x='Height',y='Weight')
```

执行结果如图 6-28 所示：

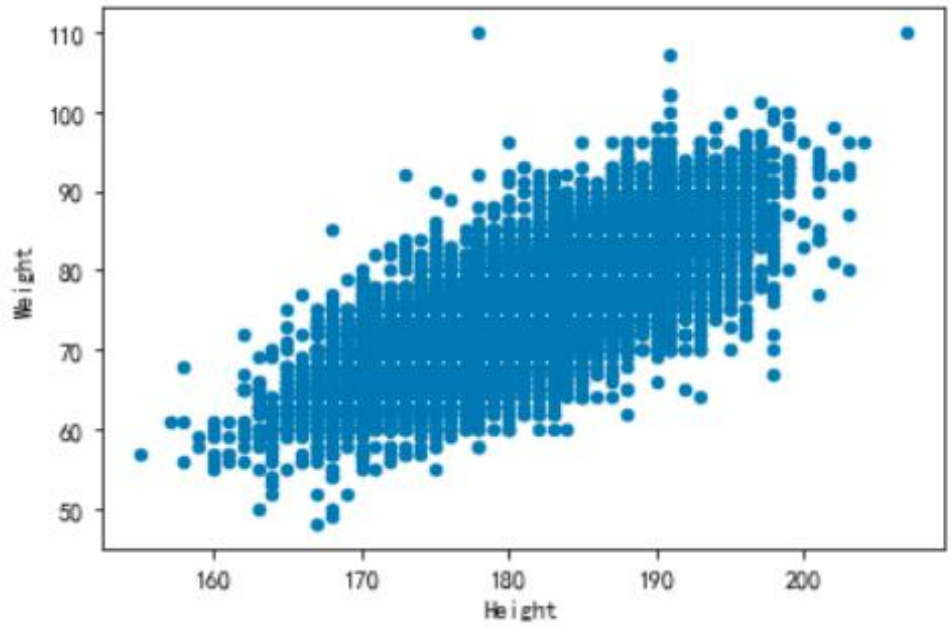


图 6-28 示例 6-29 运行效果图

从上面示例可以看到，身高和体重有一定的相关性，身高越高体重越重。

【示例 6-30】分析足球运动员的身高和评分是否相关

```
player.plot(kind='scatter',x='Height',y='Rating')
```

执行结果如图 6-29 所示：

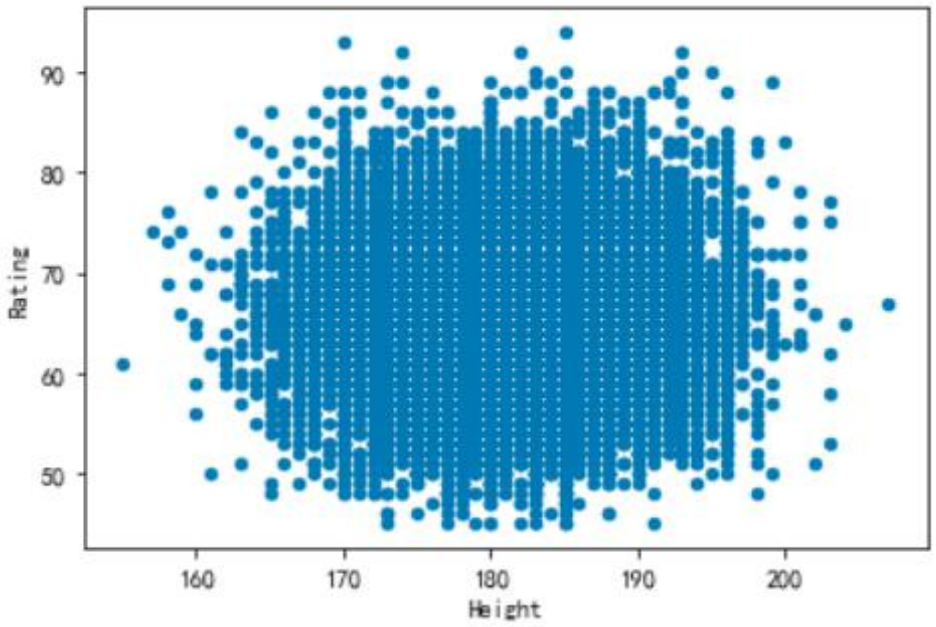


图 6-29 示例 6-30 运行效果图

从上面示例可以看到，身高和评分并没有一定的相关性。查看两个变量是否有相关性，可以通过 corr()方法，该方法返回两个变量的相关系数值，相关系数值越大表示这两个变量相关性越强。相关系数的最大值为 1，即获变量自己和自己的相关系数就是 1。

【示例 6-31】通过相关系数查看两个变量是否相关

```
print('身高与身高的相关系数：',player['Height'].corr(player['Height']))
print('身高与体重的相关系数：',player['Height'].corr(player['Weight']))
print('身高与评分的相关系数：',player['Height'].corr(player['Rating']))
```

执行结果如图 6-30 所示：

```
身高与身高的相关系数：  1.0
身高与体重的相关系数：  0.7582075774488368
身高与评分的相关系数：  0.046937095897011116
```

图 6-30 示例 6-31 运行效果图

【示例 6-32】分析对评分影响前 10 的变量

```
player.corr()['Rating'].sort_values(ascending=False).head(10)
```

执行结果如图 6-31 所示：

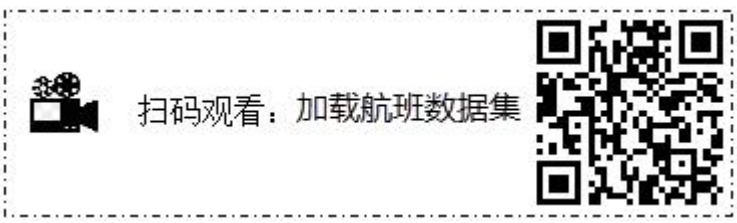
```
player.corr()['Rating'].sort_values(ascending=False).head(10)

Rating          1.000000
Reactions       0.828329
Composure       0.613612
Short_Pass      0.496239
Vision          0.489277
Long_Pass       0.483217
Ball_Control    0.463211
Age             0.458098
Shot_Power      0.441773
Curve           0.420796
Name: Rating, dtype: float64
```

图 6-31 示例 6-32 运行效果图

在上述示例中，首先获取影响评分的相关系数值，再对其进行排序。按照某列排序需要调用 sort_values()方法，在 sort_values()方法中通过 ascending 参数指定是升序还是降序。ascending 参数默认值为 True，表示升序排序，如果想降序需要将 ascending 参数设置为 False。

6.4 航班准点分析

数据集为美国各州机场的航班信息，包含出发地，目的地，是否出发延迟 15 分钟，是否到达延迟 15 分钟等。打开



https://www.transtats.bts.gov/Fields.asp?Table_ID=246 网址进行下载数据集。

6.4.1 数据探索和清洗

【示例 6-33】加载数据集

```
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
#支持中文显示
mpl.rcParams['font.family']='Kaiti'
# 使用非 unicode 的负号，当使用中文时候要设置
mpl.rcParams['axes.unicode_minus']=False
%matplotlib inline
data = pd.read_csv('data/airport-ontime.csv')
data.info()
```

执行结果如图 6-32 所示：

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 502617 entries, 0 to 502616
Data columns (total 17 columns):
FL_DATE                 502617 non-null object
UNIQUE_CARRIER          502617 non-null object
ORIGIN_AIRPORT_ID       502617 non-null int64
ORIGIN_AIRPORT_SEQ_ID   502617 non-null int64
ORIGIN_CITY_MARKET_ID   502617 non-null int64
ORIGIN_STATE_ABR        502617 non-null object
DEST_AIRPORT_ID         502617 non-null int64
DEST_AIRPORT_SEQ_ID     502617 non-null int64
DEST_CITY_MARKET_ID     502617 non-null int64
DEST_STATE_ABR          502617 non-null object
DEP_DELAY_NEW           492974 non-null float64
DEP_DEL15               492974 non-null float64
ARR_DELAY_NEW           490716 non-null float64
ARR_DEL15               490716 non-null float64
DISTANCE                502617 non-null float64
DISTANCE_GROUP          502617 non-null int64
Unnamed: 16             0 non-null float64
dtypes: float64(6), int64(7), object(4)
memory usage: 65.2+ MB
```

图 6-32 示例 6-33 运行效果图

从上述示例可以看到总共 502617 行、17 列。其中 DEP_DEL15(起飞是否延迟)非空值是 492974 行，ARR_DEL15（到达是否延迟）非空值是 490716 行，Unnamed 这列全部为空值。

【示例 6-34】查看及删除空值 Unnamed 列

```
#data.duplicated().any()
data.dropna(axis=1,how='all',inplace=True)
data.info()
```

执行结果如图 6-33 所示：

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 502617 entries, 0 to 502616
Data columns (total 16 columns):
FL_DATE                 502617 non-null object
UNIQUE_CARRIER          502617 non-null object
ORIGIN_AIRPORT_ID       502617 non-null int64
ORIGIN_AIRPORT_SEQ_ID   502617 non-null int64
ORIGIN_CITY_MARKET_ID   502617 non-null int64
ORIGIN_STATE_ABR        502617 non-null object
DEST_AIRPORT_ID         502617 non-null int64
DEST_AIRPORT_SEQ_ID     502617 non-null int64
DEST_CITY_MARKET_ID     502617 non-null int64
DEST_STATE_ABR          502617 non-null object
DEP_DELAY_NEW           492974 non-null float64
DEP_DEL15               492974 non-null float64
ARR_DELAY_NEW           490716 non-null float64
ARR_DEL15               490716 non-null float64
DISTANCE                502617 non-null float64
DISTANCE_GROUP          502617 non-null int64
dtypes: float64(5), int64(7), object(4)
memory usage: 61.4+ MB
```

图 6-33 示例 6-34 运行效果图

【示例 6-35】删除重复数据

```
data.drop_duplicates(inplace=True)
data.info()
```

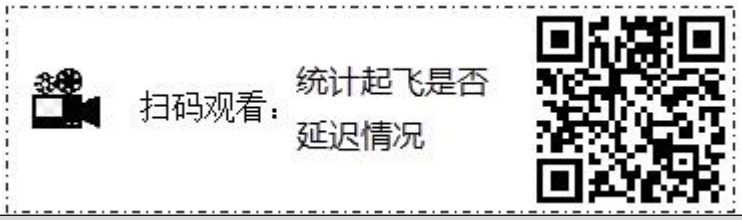
执行结果如图 6-34 所示：

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 394113 entries, 0 to 502616
Data columns (total 16 columns):
FL_DATE                 394113 non-null object
UNIQUE_CARRIER          394113 non-null object
ORIGIN_AIRPORT_ID       394113 non-null int64
ORIGIN_AIRPORT_SEQ_ID   394113 non-null int64
ORIGIN_CITY_MARKET_ID   394113 non-null int64
ORIGIN_STATE_ABR        394113 non-null object
DEST_AIRPORT_ID         394113 non-null int64
DEST_AIRPORT_SEQ_ID     394113 non-null int64
DEST_CITY_MARKET_ID     394113 non-null int64
DEST_STATE_ABR          394113 non-null object
DEP_DELAY_NEW           386058 non-null float64
DEP_DEL15               386058 non-null float64
ARR_DELAY_NEW           383812 non-null float64
ARR_DEL15               383812 non-null float64
DISTANCE                394113 non-null float64
DISTANCE_GROUP          394113 non-null int64
dtypes: float64(5), int64(7), object(4)
memory usage: 51.1+ MB
```

图 6-34 示例 6-35 运行效果图

6.4.2 统计起飞是否延迟情况

【示例 6-36】查询起飞是否延迟




```
data['DEP_DEL15'].head()
```

执行结果如图 6-35 所示：

```
data['DEP_DEL15'].head()

0    0.0
1    0.0
2    0.0
3    1.0
4    0.0
Name: DEP_DEL15, dtype: float64
```

图 6-35 示例 6-36 运行效果图

从上述示例执行结果可以看到，起飞是否延迟通过 0 和 1 来表示。其中 0 表示不延迟，

1 表示延迟。要统计起飞延迟与不延迟的数量，我们需要按 DEP_DEL15 列分组。

【示例 6-37】统计起飞延迟与不延迟，使用饼状图来显示

```
s = data['DEP_DEL15'].dropna()
delays = s.value_counts()
display(delays)
delays.name=''
delays.plot(kind='pie',labels=['起飞不延迟','起飞延迟'],autopct='%.2f',title='起飞延迟总体情况')
```

执行结果如图 6-36 所示：

```
0.0    260977
1.0    125081
Name: DEP_DEL15, dtype: int64

<matplotlib.axes._subplots.AxesSubplot at 0x17fd02d51d0>
```

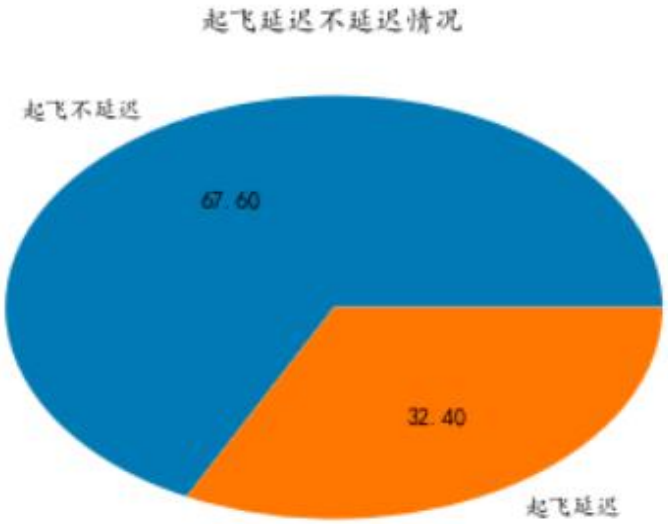
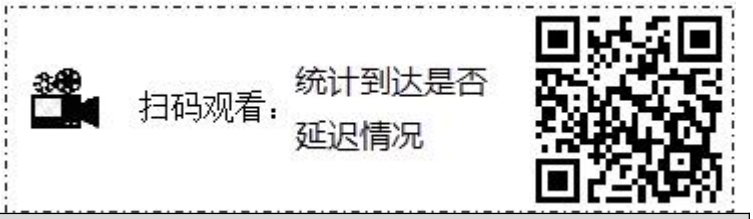


图 6-36 示例 6-37 运行效果图

6.4.3 统计到达是否延迟情况

【示例 6-38】统计到达延迟与不延迟，使用饼状图显示




```
s = data['ARR_DEL15'].dropna()
delays = s.value_counts()
```

```
    display(delays)
    delays.name=''
    delays.plot(kind='pie',labels=['到达不延迟','到达延迟'],autopct='%.2f',title='到达延迟总体情况')
```

执行结果如图 6-37 所示：

```
0.0    254200
1.0    129612
Name: ARR_DEL15, dtype: int64
<matplotlib.axes._subplots.AxesSubplot at 0x17fd28e85c0>
```

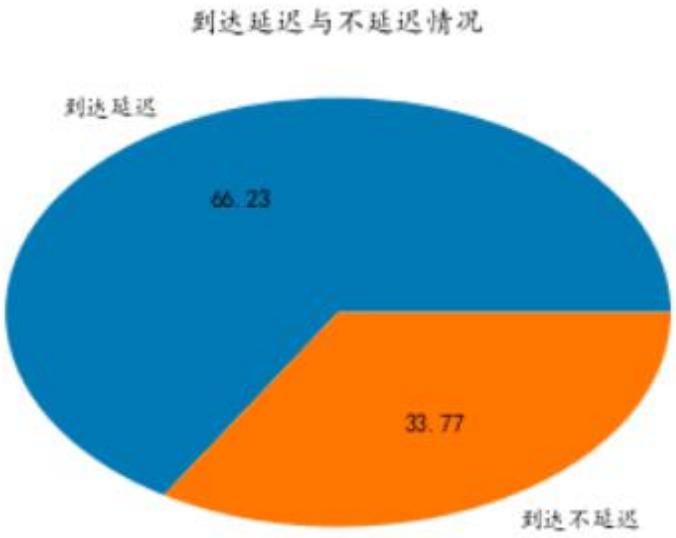
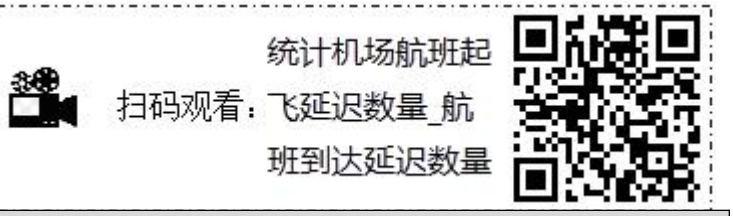


图 6-37 示例 6-38 运行效果图

6.4.4 统计机场航班起飞、到达延迟数量

【示例 6-39】统计机场航班起飞延迟数量，使用柱状图显示




```
    # 缺失值处理
    d = data[['ORIGIN_STATE_ABR','DEP_DEL15']].dropna()
    depart_delay_couots = d.groupby('ORIGIN_STATE_ABR')['DEP_DEL15'].sum()
    # 设置画布大小 figsize=(a,b) a 表示画布宽，b 表示画布高，单位英寸
    depart_delay_couots.sort_values(ascending=False).plot(kind='bar',figsize=(14,6))
```

执行结果如图 6-38 所示：

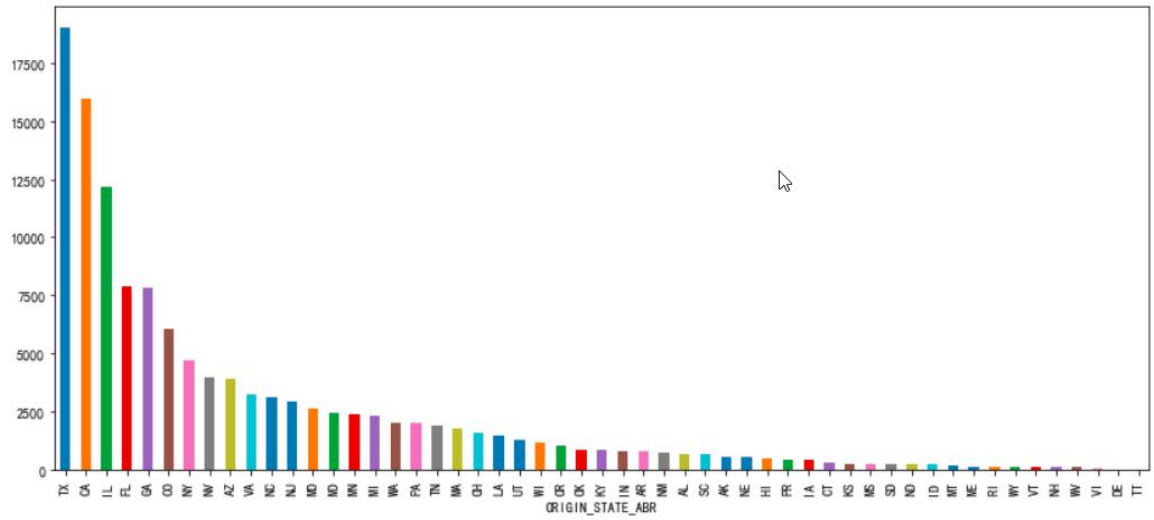


图 6-38 示例 6-39 运行效果图

【示例 6-40】统计机场航班到达延迟数量，使用柱状图显示

```
# 缺失值处理
d = data[['DEST_STATE_ABR','ARR_DEL15']].dropna()
arrive_delay_couots = d.groupby('DEST_STATE_ABR')['ARR_DEL15'].sum()
# 设置画布大小 figsize=(a,b) a 表示画布宽，b 表示画布高，单位英寸
arrive_delay_couots.sort_values(ascending=False).plot(kind='bar',figsize=(14,6))
```

执行结果如图 6-39 所示：

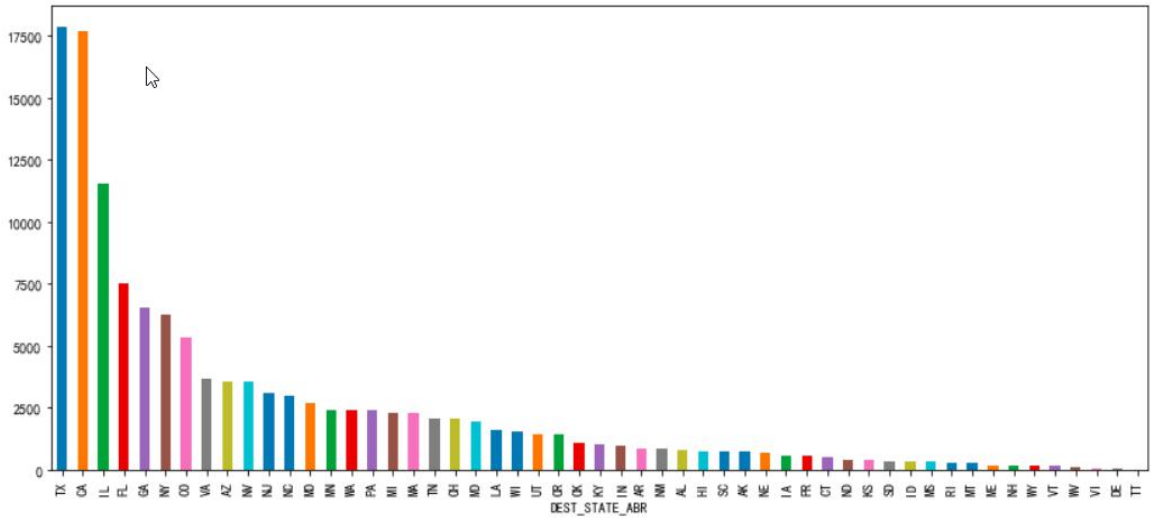


图 6-39 示例 6-40 运行效果图

【示例 6-41】合并机场航班起飞和到达延迟

```
delay_df=pd.DataFrame([depart_delay_couots,arrive_delay_couots]).T
delay_df.columns=['起飞延迟','到达延迟']
```

```
delay_df.sort_values('起飞延迟',ascending=False).plot(kind='bar',figsize=(14,6),title='机场起飞到达延迟状况')
```

执行结果如图 6-40 所示：

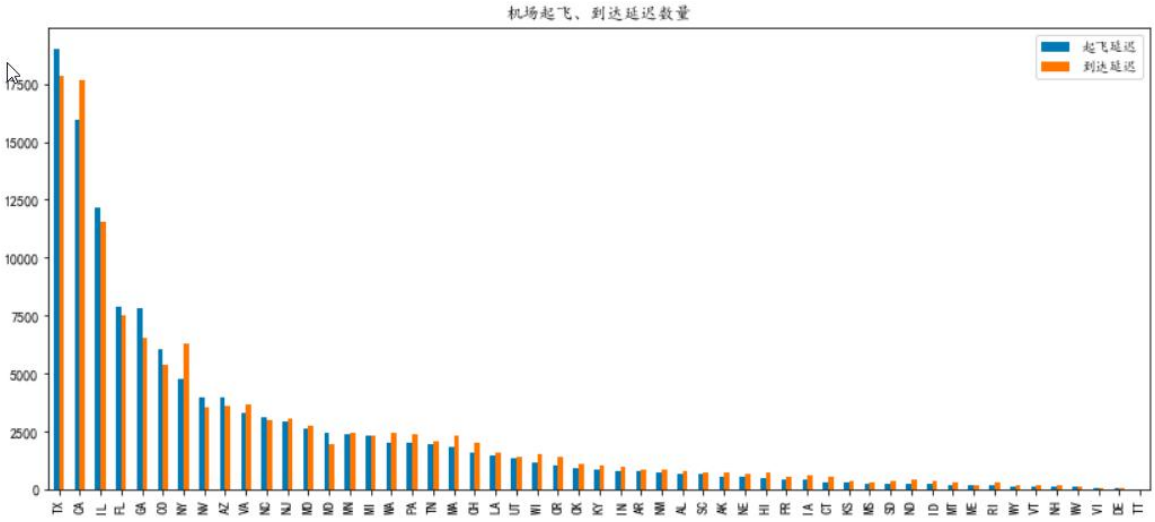
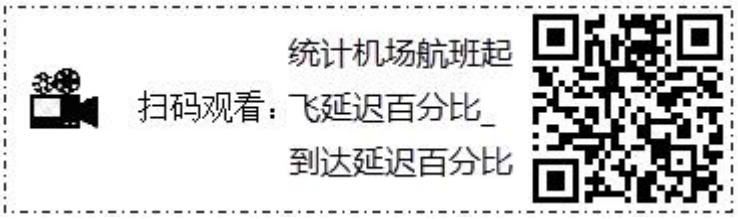


图 6-40 示例 6-41 运行效果图

6.4.5 统计机场航班起飞、到达延迟的百分比

1) 机场航班起飞延迟的百分比

机场航班起飞延迟百分比=机场起飞延迟的航班数/机场航班总起飞数。机场起飞延迟的航班数前面已经获取到变量 depart_delay_couots 中，还需要获取机场总航班，示例如下：



【示例 6-42】机场起飞总航班数

```
d = data[['ORIGIN_STATE_ABR','DEP_DEL15']].dropna()
departs = d['ORIGIN_STATE_ABR'].value_counts()
```

【示例 6-43】机场航班起飞延迟的百分比

```
pct_departure_delays = depart_delay_couots/departs
```

2) 机场航班到达延迟的百分比

同理，机场航班到达延迟百分比=机场到达延迟的航班数/机场航班达数总数。机场到达延迟航班数前面已经获取到变量 arrive_delay_couots 中，还需要获取机场航班到达总数，示例如下：

【示例 6-44】机场航班到达延迟的百分比

```
d = data[['DEST_STATE_ABR','ARR_DEL15']].dropna()
# 计算到达航班的数量
arrives = d['DEST_STATE_ABR'].value_counts()
# arrive_delay_couots 机场到达延迟航班数
pct_arrive_delays = arrive_delay_couots/arrives
```

【示例 6-45】组合起飞延迟和到达延迟百分比，柱状图描述

```
# 将起飞延迟和到达延迟组合成 DataFrame，柱状图描述
pct_delay_df=pd.DataFrame([pct_departure_delays,pct_arrive_delays]).T
pct_delay_df.columns=['起飞延迟比例','到达延迟比例']
# display(pct_departure_delays,pct_arrive_delays)
pct_delay_df.sort_values('起飞延迟比例',ascending=False).plot(kind='bar',title='机场起飞到达延迟百分比',figsize=(14,6))
```

执行结果如图 6-41 所示：

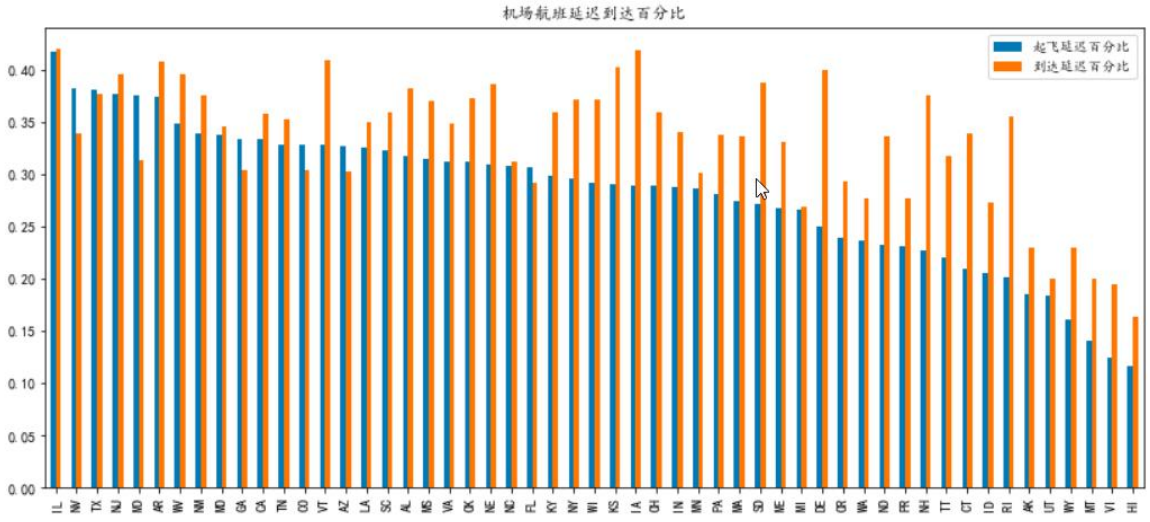


图 6-41 示例 6-45 运行效果图

习题

一、编码题

1. **Python 操作 Excel，完成对 Excel 的读和写。**
2. **导入.xlsx 文件，并完成如下操作：**
   a. 指定导入哪个 Sheet。
   b. 指定行索引。
   c. 指定列索引。
   d. 指定导入列。
3. **设有如下数据集，完成统计分析**

```
import pandas as pd
df = pd.DataFrame({'applied_from': [1,1,1,1,1,2,2,2,2,2,2],
                   'applied_type': [0,0,0,1,1,0,0,0,1,1,1],
                   'refused_id':     [100,100,100,102,102,102,102,102,102,102,100],
                                                                    'name':
['mark','python','C','HAHA','XX','XIXI','ROSE','RICHAR','BINGO','BUFF','YEVON_OU'],
                   'age':            [18,19,22,12,22,24,33,44,55,66,77],
                                                                    'country':
['china','china','USA','japan','japan','USA','uk','uk','uk','uk','USA']})
df
```

   a. 获取 applied_from 字段。
   b. country 中的类别个数计算。
   c. 对 country 分组再求和。
   d. 对 countrys 中 age 按照降序排列。
   e. 求 age 最大的索引值。
   f. 查看 age 年龄最小的这一行数据。
   g. 计算年龄标准差。
   h. df 中描述性统计信息查看。