第3章

NumPy和pandas基础

【学习目标】

学完本章之后，读者将掌握以下内容。

- NumPy 库中提供的 ndarray 数组的创建方式、常用属性、数据类型以及算术操作。
- ndarray 数组的算术操作、索引和切片；NumPy 中轴的概念。
- pandas 库中常用的数据结构 Series 和 Dataframe。
- Series 和 Dataframe 的常用数据操作。

在做数据分析时，将会经常使用到 NumPy、pandas、Scipy、Matplotlib、Statsmodels、sklearn 等包和分析工具库。从某种程度上讲，利用 Python 进行数据分析的学习过程就是对库的学习过程。本章主要对在数据整理、描述与分析时常使用的 NumPy、pandas 两个库进行介绍。

3.1 NumPy 基础

NumPy 是 Numerical Python 的简称，是目前 Python 数值计算中最重要的基础包。在使用之前需要使用 import 语句导入(即 import numpy as np)。调用 NumPy 中的模块或函数时可以使用“np. 模块或函数名称”的方式。

NumPy 的核心特征之一是 N 维数组对象 ndarray。一个 ndarray 是一个通用的多维同类数据容器，也就是说，它包含的每一个元素均为相同类型。Python 的列表是异构的，因此列表的元素可以包含任何对象类型；而 NumPy 数组是同质的，只能存放同一种类型的对象。

NumPy 数组元素类型一致的好处是：因为数组元素的类型相同，所以能轻松确定存储数组所需空间的大小。同时，NumPy 数组能够运用向量化运算来处理整个数组，而完

成同样的任务,Python 的列表通常需要借助循环语句遍历列表并对逐个元素进行相应的处理。

例 3.1

```
 1:  import numpy as np
 2:  import time
 3:  np_0 = np.arange(1000000)
 4:  list_0 = list(range(1000000))
 5:  start_CPU_1 = time.perf_counter()
 6:  np_1 = np_0 * 2
 7:  end_CPU_1 = time.perf_counter()
 8:  print("Method 1: %f CPU seconds" % (end_CPU_1 - start_CPU_1))
 9:  start_CPU_2 = time.perf_counter()
10:  list_1 = list_0 * 2
11:  end_CPU_2 = time.perf_counter()
12:  print("Method 2: %f CPU seconds" % (end_CPU_2 - start_CPU_2))
```

【例题解析】

该例旨在对 NumPy 数组和 Python 列表的运算效率进行对比。

第 1 行和第 2 行分别表示引入 NumPy 模块和 time 模块。

第 3 行和第 4 行分别定义一个包含 100 万个整数的 NumPy 数组 np_0,以及一个等价的 Python 列表 list_0。

第 5～7 行获得了数组 np_0 中每一个元素乘以 2 的 CPU 运行时间;第 9～11 行获得了将列表 list_0 中每一个元素乘以 2 的 CPU 运行时间。

通过比较两个输出的 CPU 时间可以看出:完成同样的运算,NumPy 的方法比 Python 方法快 10～100 倍。

【运行结果】

第 8 行的输出结果:Method 1:0.000983 CPU seconds

第 12 行的输出结果:Method 2:0.007737 CPU seconds

本节将主要讲解 ndarray 多维数组的创建方法、数组的属性和数组的简单操作等。

3.1.1 ndarray 数组的创建

创建数组最简单的方式是使用 array 函数,其语法格式为:

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

常用的参数有 object 和 dtype。其中,object 表示公开数组接口的任何对象,或任何(嵌套)序列。dtype 表示可选数组所需的数据类型。利用该函数可直接将 Python 的基础数据类型(如列表、元组等)转换成一个数组。

例 3.2

```
1:  import numpy as np
2:  ndarray_1 = np.array([1, 5, 8, 11])
```

```
 3:  print(ndarray_1)
 4:  ndarray_2 = np.array((1, 5, 8, 11))
 5:  print(ndarray_2)
 6:  list = [[1,2,3,4],[5,6,7,8]]
 7:  ndarray_3 = np.array(list)
 8:  print(ndarray_3)
 9:  ndarray_4 = np.array(list, dtype = float)
10:  print(ndarray_4)
```

【例题解析】

该例题旨在演示如何利用列表、元组生成一维数组和多维数组。

第 2 行和第 4 行分别利用 array 函数对 Python 的基础数据类型列表和元组进行转换，生成了一维数组 ndarray_1 和一维数组 ndarray_2。

第 6 行定义了一个长度为 2 的嵌套列表类型序列 list。

第 7 行利用 array 函数将 list 转换成二维数组 ndarray_3。

第 9 行通过参数 dtype 指定数组中元素类型为 float 类型，利用 array 函数将 list 转换成浮点型的二维数组 ndarray_4。

【运行结果】

第 3 行的输出结果：[1 5 8 11]

第 5 行的输出结果：[1 5 8 11]

第 8 行的输出结果：[[1 2 3 4]
 [5 6 7 8]]

第 10 行的输出结果：[[1. 2. 3. 4.]
 [5. 6. 7. 8.]]

除了 np.array，还有很多其他的函数可以创建新数组。表 3-1 中列举了常用标准数组生成函数。

表 3-1 数组生成函数

函数名	描　　述
arange()	Python 内建函数 range()的数组版，返回一个数组
ones()	根据给定形状和数据类型生成全 1 数组
ones_like()	根据所给数组生成一个形状一样的全 1 数组
zeros()	根据所给形状和类型数据生成全 0 数组
zeros_like()	根据所给数组生成一个形状一样的全 0 数组
empty()	根据给定形状生成一个元素为随机数的数组
empty_like()	根据所给数组生成一个形状一样的空数组
full()	根据给定的形状和数据类型生成指定数值的数组
full_like()	根据所给的数组生成一个形状一样但内容是指定数值的数组
eye(),identity()	生成一个 $N\times N$ 特征矩阵(对角线位置都是 1，其余位置是 0)

例 3.3

```
1:  import numpy as np
2:  ndarray = np.arange(0, 10, 0.5)
3:  print(ndarray)
4:  print(ndarray * 10)
```

【例题解析】

该例演示了利用表 3-1 中的 arange()函数如何定义一个一维数组。

第 1 行表示引入 NumPy 库。

第 2 行定义了一个由 0～10 并且均匀间隔为 0.5 的数字组成的数组 ndarray。

第 4 行对数组中的每个元素进行了乘 10 的操作,并将结果输出。

【运行结果】

第 3 行的输出结果:[0. 0.5 1. 1.5 2. 2.5 3. 3.5 4. 4.5 5. 5.5 6. 6.5 7. 7.5 8. 8.5 9. 9.5]

第 4 行的输出结果:[0. 5. 10. 15. 20. 25. 30. 35. 40. 45. 50. 55. 60. 65. 70. 75. 80. 85. 90. 95.]

3.1.2 ndarray 的常用属性

表 3-2 列示了 ndarray 的一些常用属性。

表 3-2 ndarray 的常用属性

属性	描 述
ndarray.shape	数组的维度。这是一个整数的元组,表示每个维度上数组的大小
ndarray.dtype	ndarray 中元素的数据类型
ndarray.ndim	数组的维数,或数组轴的个数
ndarray.size	数组中元素的总个数
ndarray.itemsize	数组中的一个元素在内存中所占的字节数
ndarray.nbytes	整个数组所占的存储空间,其值为数组 itemsize 和 size 属性值的乘积
ndarray.T	数组的转置
ndarray.real	数组的实部,如果数组中仅含实数元素,则输出原数组
ndarray.imag	数组的虚部,如果数组中仅包含实数元素,则输出值均为 0
ndarray.flat	返回一个 NumPy.flatiter 对象。这个“扁平迭代器”可以实现像遍历一维数组一样遍历任意的多维数组

例 3.4

```
1:  import numpy as np
2:  ndarray = np.array([[0, 1, 2, 3, 4, 5, 6],[7, 8, 9, 10, 11, 12, 13]])
3:  print(ndarray.shape, ndarray.dtype)
4:  print(ndarray.ndim)
5:  print(ndarray.size)
```

```
 6:  print(ndarray.itemsize)
 7:  print(ndarray.nbytes)
 8:  print(ndarray.T)
 9:  print(ndarray.flat[1], ndarray.flat[4:9])
10:  ndarray.flat[[3,9]] = 8
11:  print(ndarray)
```

【例题解析】

该例意在说明表3-2中ndarray的常用属性。这里特别说明一下第9～11行，其他属性容易理解，不再赘述。

第9行利用flat属性遍历数组ndarray。其中，ndarray.flat[1]表示将数组ndarray降为一维的基础上索引位置1处的元素，即1；ndarray.flat[4：9]表示将数组ndarray降为一维的基础上索引位置4～9区间的元素（前面的索引取闭区间，后面的索引取开区间），即[4 5 6 7 8]。

第10行表示将数组ndarray降为一维的基础上把索引位置3和9处的元素值赋为“8”。

【运行结果】

第3行的输出结果：(2, 7) int32
第4行的输出结果：2
第5行的输出结果：14
第6行的输出结果：4
第7行的输出结果：56
第8行的输出结果：[[0 7]
 [1 8]
 [2 9]
 [3 10]
 [4 11]
 [5 12]
 [6 13]]
第9行的输出结果：1 [4 5 6 7 8]
第11行的输出结果：[[0 1 2 8 4 5 6]
 [7 8 8 10 11 12 13]]

3.1.3 ndarray的数据类型

Python支持的数据类型有整型、浮点型以及复数型，但这些类型不足以满足科学计算的需求。因此，NumPy添加了很多其他的数据类型。表3-3中列出了NumPy中支持的数据类型。在NumPy中，大部分数据类型名以数字结尾，这个数字表示其在内存中占用的位数。

表 3-3 NumPy 数据类型

类型	描述
bool	用一位存储的布尔类型(值为 TRUE 或 FALSE)
inti	由所在平台决定其精度的整数(一般为 int32 或 int64)
int8	整数,范围为-128～127
int16	整数,范围为-32 768～32 767
int32	整数,范围为-2^{31}～$2^{31}-1$
int64	整数,范围为-2^{63}～$2^{63}-1$
uint8	无符号整数,范围为 0～255
uint16	无符号整数,范围为 0～65 535
uint32	无符号整数,范围为 0～$2^{32}-1$
uint64	无符号整数,范围为 0～$2^{64}-1$
float16	半精度浮点数(16 位):其中,用 1 位表示正负号,5 位表示指数,10 位表示尾数
float32	单精度浮点数(32 位):其中,用 1 位表示正负号,8 位表示指数,23 位表示尾数
float64 或 float	双精度浮点数(64 位):其中,用 1 位表示正负号,11 位表示指数,52 位表示尾数
complex64	复数,分别用两个 32 位浮点数表示实部和虚部
complex128 或 complex	复数,分别用两个 64 位浮点数表示实部和虚部

如表 3-3 所示,属性 dtype 可以返回数组中元素的数据类型。另外,可以使用 astype()方法实现数据类型的显式转换。

例 3.5 利用 astype()方法实现整数到浮点数类型的转换。

```
1:  import numpy as np
2:  ndarray_int = np.array([1,2,3,4,5])
3:  print(ndarray_int.dtype)
4:  ndarray_float = ndarray_int.astype(np.float64)
5:  print(ndarray_float.dtype)
```

【例题解析】

第 2 行利用 array()函数定义了一个数组 ndarray_int;第 3 行利用 dtype 属性获取数组中元素的数据类型,即整数;第 4 行利用 astype()方法实现数组数据从整型向浮点型的转换。

【运行结果】

第 3 行的输出结果:int32

第 5 行的输出结果:float64

3.1.4 ndarray 的算术操作

数组之所以重要,是因为它可以进行批量操作而无需任何 for 循环,用户称这种特性为向量化。任何在两个等尺寸数组之间的算术操作都应用了逐元素操作的方式。接下来,主要从 5 方面介绍数组的批量算术操作。

1. 数组和标量间的运算

带有标量计算的算术操作，会把计算传递给数组的每一个元素。

例 3.6

```
1:  import numpy as np
2:  ndarray = np.array([[1.,2.,3.], [4.,5.,6.]])
3:  print(ndarray)
4:  print(1/ndarray)
5:  print(ndarray ** 0.5)
```

【例题解析】

第 2 行利用 array()函数定义了一个数组 ndarray；第 4、5 行分别表示求数组 ndarray 的倒数和开方。从这两行的输出结果可以看出，带有标量计算的数组算术操作，把计算传递给了数组 ndarray 中的每一个元素。

【运行结果】

```
第 3 行的输出结果：[[1. 2. 3.]
                    [4. 5. 6.]]
第 4 行的输出结果：[[1.        0.5         0.33333333]
                    [0.25      0.2         0.16666667]]
第 5 行的输出结果：[[1.        1.41421356  1.73205081]
                    [2.        2.23606798  2.44948974]]
```

2. 通用函数

通用函数(Universal Function，Ufunc)是一种对数组中的数据执行元素级运算的函数，用法也很简单。

1) 一元通用函数

一元通用函数是指导入一个数组作为对象，较为常用的函数及描述如表 3-4 所示。

表 3-4 一元 Ufunc 中较为常用的函数及描述

函　数	描　述
abs()	求绝对值
sqrt()	求平方根
square()	求平方
exp()	求以自然常数 e 为底的指数函数
log()	用于计算所有输入数组元素的自然对数

例 3.7

```
1:  import numpy as np
2:  ndarray = np.arange( - 1, 10, 2)
```

```
3:  print(ndarray)
4:  print(np.abs(ndarray))
5:  print(np.sqrt(ndarray))
6:  print(np.square(ndarray))
7:  print(np.exp(ndarray))
8:  print(np.log(ndarray))
```

【例题解析】

第2行利用np.arange()函数定义一个起始点为－1,终点为10,步长为2的数组ndarray。

第4行利用abs()函数求数组ndarray中每个元素的绝对值。

第5行利用sqrt()函数求数组ndarray中每个元素的平方根,由于负数没有平方根,所以该数组中元素－1的平方根显示为“nan”。

第6行利用square()函数求数组ndarray中每个元素的平方。

第7行利用exp()函数求以自然常数e为底,数组ndarray中的元素为指数的幂运算。

第8行利用log()函数求以数组ndarray中的元素为底的自然对数,由于负数不能求对数,所以该数组中元素－1的对数值显示为“nan”。

【运行结果】

第3行的输出结果:[－1 1 3 5 7 9]

第4行的输出结果:[1 1 3 5 7 9]

第5行的输出结果:[nan 1. 1.73205081 2.23606798 2.64575131 3.]

第6行的输出结果:[1 1 9 25 49 81]

第7行的输出结果:[3.67879441e－01 2.71828183e＋00 2.00855369e＋01
1.48413159e＋02 1.09663316e＋03 8.10308393e＋03]

第8行的输出结果:[nan 0. 1.09861229 1.60943791 1.94591015 2.19722458]

2) 二元通用函数

二元通用函数指导入两个数组(假定为x_1和x_2)作为对象,并返回一个数组。较为常用的函数及描述如表3-5所示。

表3-5 二元Ufunc中较为常用的函数及描述

函 数	描 述
add(x_1,x_2)	数组x_1和x_2中的元素对应相加
subtract(x_1,x_2)	数组x_1和x_2中的元素对应相减
multiply(x_1,x_2)	数组x_1和x_2中的元素对应相乘
divide(x_1,x_2)	数组x_1和x_2中的元素对应相除
power(x_1,x_2)	数组x_1和x_2中的元素对应做$x_1^{x_2}$运算

例 3.8

```
 1:  import numpy as np
 2:  ndarray_1 = np.array([1, 2, 3])
 3:  ndarray_2 = np.array([4, 5, 6])
 4:  print(ndarray_1)
 5:  print(ndarray_2)
 6:  print(np.add(ndarray_1, ndarray_2))
 7:  print(np.subtract(ndarray_1, ndarray_2))
 8:  print(np.multiply(ndarray_1, ndarray_2))
 9:  print(np.divide(ndarray_1, ndarray_2))
10:  print(np.power(ndarray_1, ndarray_2))
```

【例题解析】

第 6～9 行分别利用 add()函数、subtract()函数、multiply()函数和 divide()函数求数组 ndarray_1 与数组 ndarray_2 的和、差、积、商；第 10 行利用 power()函数，将数组 ndarray_1 中的元素作为底数，计算它与数组 ndarray_2 中相应元素的幂。

【运行结果】

第 4 行的输出结果：[1 2 3]
第 5 行的输出结果：[4 5 6]
第 6 行的输出结果：[5 7 9]
第 7 行的输出结果：[-3 -3 -3]
第 8 行的输出结果：[4 10 18]
第 9 行的输出结果：[0.25 0.4 0.5]
第 10 行的输出结果：[1 32 729]

3. 统计运算

NumPy 库支持对整个数组或按指定轴向的数据进行统计计算，较为常用的函数如表 3-6 所示，这些函数都可以传入 axis 参数，用于计算指定轴方向的统计值。

表 3-6 统计运算中常用的函数及描述

函 数	描 述
mean()	算术平均数
std()；var()	标准差和方差
min()；max()	最小值和最大值
argmin()；argmax()	最小值和最大值的索引
ptp()	沿轴的值的范围(最大值－最小值)
percentile()	一个多维数组的任意百分比分位数
median()	计算指定轴的中位数

例 3.9

```
 1:  import numpy as np
 2:  ndarray = np.array([[1,2,3],[3,4,5],[4,5,6]])
 3:  print(ndarray)
 4:  print(np.mean(ndarray))
 5:  print(np.std(ndarray), np.var(ndarray))
 6:  print(np.min(ndarray), np.max(ndarray))
 7:  print(np.argmin(ndarray), np.argmax(ndarray))
 8:  print(np.ptp(ndarray))
 9:  print(np.percentile(ndarray,90))
10:  print(np.median(ndarray))
```

【例题解析】

第 2 行利用 array()函数定义了一个数组 ndarray。

第 4 行利用 mean()函数求整个数组 ndarray 的算术平均数。

第 5 行利用 std()、var()函数分别求整个数组 ndarray 的标准差、方差。

第 6 行利用 min()、max()函数分别求整个数组 ndarray 的最小值、最大值。

第 7 行利用 argmin()、argmax()函数分别求整个数组 ndarray 的最小值的索引、最大值的索引。

第 8 行利用 ptp()函数求整个数组 ndarray 中最大值与最小值的差。

第 9 行利用 percentile()函数求多维数组 ndarray 的 90%分位数。

第 10 行利用 median()函数求整个数组 ndarray 的中位数。以上函数均可以用 axis=0 或者 axis=1 计算数组指定轴方向的各种统计值。

【运行结果】

第 3 行的输出结果：[[1 2 3]
 [3 4 5]
 [4 5 6]]

第 4 行的输出结果：3.66666666667

第 5 行的输出结果：1.490711985 2.22222222222

第 6 行的输出结果：1 6

第 7 行的输出结果：0 8

第 8 行的输出结果：5

第 9 行的输出结果：5.2

第 10 行的输出结果：4.0

4. 布尔型数组运算

对于布尔型数组，其布尔值会被强制转换为 1(True)和 0(False)。另外，还有两个方法 any()和 all()也可以用于布尔型数组运算。其中，any()方法用于检测数组中是否存在一个或多个 True；all()方法用于检测数组中的所有值是否为 True。

例 3.10

```
1:  import numpy as np
2:  ndarray_randn = np.random.randn(20)
3:  print(ndarray_randn)
4:  print((ndarray_randn > 0).sum())
5:  ndarray_bool = np.array([True, False, False, True])
6:  print(ndarray_bool.any())
7:  print(ndarray_bool.all())
```

【例题解析】

第 2 行从标准正态分布中返回秩为 1 的数组 ndarray_randn。

第 4 行是求 ndarray_randn 数组中大于 0 的值的个数。具体来讲，“ndarray_randn>0”是布尔判断，即当数组 ndarray_randn 中元素的值大于 0 时，其布尔值会被强制转换为 1，反之为 0；然后通过调用 sum()函数，求 ndarray_randn 数组中布尔值 0、1 的总和。

第 5 行利用 array()函数定义了一个布尔类型的数组 ndarray_bool；第 6 行利用 any()方法检测数组 ndarray_bool 中是否存在一个或多个 True，由于 ndarray_bool 中含有 True，返回值为 True；第 7 行利用 all()方法检测数组中的所有值是否为 True，由于 ndarray_bool 中包含 False，返回值为 False。

【运行结果】

第 3 行的输出结果(该结果不唯一)：

[2.10755404　0.46192799　0.32168445　−0.63041907　−0.47205041
 0.24468571
−0.51070383　0.96475473　−0.87134876　−0.66024855　0.34708465
−1.91519907
0.44135922　−0.31775647　−1.2535673　−1.12947125　0.2778703
−0.1532834　0.40627728　0.22485375]

第 4 行的输出结果：10(该结果不唯一)

第 6 行的输出结果：True

第 7 行的输出结果：False

5. 排序

NumPy 提供的较为常用的函数如表 3-7 所示。

表 3-7　数组排序常用的函数及描述

函　数	描　述
sort()	返回排序后的数组
argsort()	返回数组排序后的下标(下标对应的数是排序后的结果)
lexsort()	对数组按指定行或列的顺序排序；是间接排序，不修改原数组，返回索引

例 3.11

```
1:  import numpy as np
2:  ndarray = np.array([1, 2, 4, 3, 1, 2, 2, 4, 6, 7, 2, 4, 8, 4, 5])
3:  print(np.sort(ndarray))
4:  print(np.argsort(ndarray))
5:  ndarray_reshape = ndarray.reshape(3, 5)
6:  print(ndarray_reshape)
7:  print(np.sort(ndarray_reshape, axis = 1))
8:  print(np.sort(ndarray_reshape, axis = 0))
```

【例题解析】

第 2 行利用 array 函数定义了一个数组 ndarray；第 3 行利用 sort()函数将数组中的元素升序排序；第 4 行利用 argsort()函数返回数组排序后的下标(下标在 ndarray 中对应的数据是排序后的结果)。换句话说，排序后在第一个位置上的元素是 ndarray[0]，即 1，第二个位置上的元素是 ndarray[4]，即 1，以此类推。

第 5 行利用 reshape()函数将数组 ndarray 变成一个 3 行 5 列的二维数组 ndarray_reshape。第 7 行、第 8 行分别通过将 axis=1、axis=0 实现将数组 ndarray_reshape 按照第 1 维和第 0 维排序。

【运行结果】

```
第 3 行的输出结果：[1 1 2 2 2 2 3 4 4 4 4 5 6 7 8]
第 4 行的输出结果：[0 4 1 5 6 10 3 2 7 11 13 14 8 9 12]
第 6 行的输出结果：[[1 2 4 3 1]
                    [2 2 4 6 7]
                    [2 4 8 4 5]]
第 7 行的输出结果：[[1 1 2 3 4]
                    [2 2 4 6 7]
                    [2 4 4 5 8]]
第 8 行的输出结果：[[1 2 4 3 1]
                    [2 2 4 4 5]
                    [2 4 8 6 7]]
```

3.1.5 ndarray 的索引和切片

在数据分析中常需要选取符合条件的数据，数组的索引和切片方法就显得非常重要了。NumPy 中多维数组的索引与切片跟 Python 中的列表类似，但最大的区别是，数组切片是原始数组的视图，也就是说，对视图上的修改直接会影响到原始数组。因为 NumPy 主要处理大数据，如果每次切片都进行一次复制，对性能和内存是相当大的考验。

1. 一维数组的索引和切片

一维数组的索引与 Python 的列表差不多，用“[]”选定下标来实现，也可采用“:”分

隔起止位置与间隔。

例 3.12

```
1:  import numpy as np
2:  ndarray = np.arange(1, 20, 2)
3:  print(ndarray)
4:  print(ndarray[3])
5:  print(ndarray[1:4])
6:  print(ndarray[:2])
7:  print(ndarray[ - 2])
```

【例题解析】

第 2 行利用 arange()函数定义了一个从 1 到 20,步长为 2 的数组 ndarray。

第 4 行取数组中下标位置为 3 的元素,即 7。

第 5 行取数组中下标位置为 1～3 的元素,即[3 5 7]。“:”前数字可以省略,表示从位置 0 开始选取元素。因此,第 6 行表示取数组中下标位置为 0、1 的元素,即[1 2]。如果下标位置为负数,则表示从数组的尾部取值,最后一个位置索引为－1。

第 7 行表示取数组尾部倒数第二个元素,即 17。

【运行结果】

第 3 行的输出结果:[1 3 5 7 9 11 13 15 17 19]

第 4 行的输出结果:7

第 5 行的输出结果:[3 5 7]

第 6 行的输出结果:[1 3]

第 7 行的输出结果:17

2. 多维数组的索引和切片

多维数组的索引是由外向内逐层选取;多维数组的切片就有所不同了,行是从上到下,列是从左到右进行切片的,用“,”隔开行、列的索引或切片。

例 3.13

```
1:  import numpy as np
2:  ndarray = np.array([[1,2,3], [4,5,6], [7,8,9]])
3:  print(ndarray[1][2])
4:  print(ndarray[:2, 1:])
```

【例题解析】

第 2 行利用 array()函数定义了一个 3 行 3 列的数组 ndarray。

第 3 行取数组行索引为 1、列索引为 2(索引从 0 开始)位置处的值,即元素 6。索引的写法也可以直接写成[1,2]。

第 4 行表示对多维数组 ndarray 进行切片。即“:2”表示从上到下取该数组第 0 行到第 1 行,“1:”表示从左到右取该数组的第 1 列到最后 1 列完成对该数组的切片。

【运行结果】

输出结果为：6

输出结果为：[[2　3]
[5 6]]

3. 布尔型索引

布尔型索引指的是一个布尔型 ndarray 数组(一般为一维)对应另一个 ndarray 数组的每行,布尔型数组的个数必须与另一个多维数组的行数一致。若布尔型数组内的某个元素为 True,则选取另一个多维数组的相应行,反之不选取。

例 3.14

```
1:  import numpy as np
2:  ndarray_randn = np.random.randn(5,4)
3:  print(ndarray_randn)
4:  ndarray_bool = np.array([True, False, False, False, True])
5:  print(ndarray_randn[ndarray_bool])
```

【例题解析】

第 1 行表示引入 NumPy 库。

第 2 行定义了一个 5 行 4 列的二维数组 ndarray_randn。

第 4 行定义了一个布尔型数组 ndarray_bool。

第 5 行表示利用布尔型数组 ndarray_bool 内 True 的位置选取数组 ndarray_randn 的行,即选取数组 ndarray_randn 中第一行和最后一行。

【运行结果】(该结果不唯一)

第 2 行的输出结果：[[0.93129863　−0.11879981　1.05443786　−0.20675621]
[−0.70109137 −0.89017318　1.15989982　−0.70514891]
[−1.59165051 1.00758237　− 0.36689707　0.92416095]
[0.45114108　0.29765091　0.14566542　0.98974861]
[−1.10993879 −0.27959499　−0.29546634 0.61503587]]

第 4 行的输出结果：[[0.93129863　−0.11879981　1.05443786　−0.20675621]
[−1.10993879 −0.27959499　−0.29546634　0.61503587]]

3.1.6　对轴的理解

就像坐标系一样,NumPy 阵列也有轴。如图 3-1 所示,在 NumPy 数组中,对于二维数组,axis 0 是第 1 根轴,表示沿行(Row)向下的轴；axis 1 是第 2 根轴,表示沿列(Columns)横穿的轴。接下来,主要介绍在进行数据分析时对于不同维度的数组中 axis 参数控制的内容。

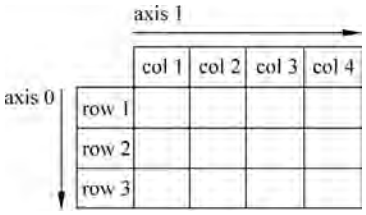

图 3-1　NumPy 阵列的轴向

在 NumPy 中，维数(Dimensions)通过轴(Axes)来扩展，轴的个数被称作 Rank。这里的 Rank 不是线性代数中的 Rank(秩)，它指代的是维数。也就是说，Axes、Dimensions、Rank 这几个概念是相通的。

例 3.15

```
1:  import numpy as np
2:  ndarray = np.array([[1,2,3],[2,3,4],[3,4,5]])
3:  print(ndarray)
4:  print(np.ndim(ndarray))
5:  print(np.shape(ndarray))
```

【例题解析】

第 2 行定义了一个数组 ndarray，该数组只有两个轴，每个轴的长度(即 Length)均为 3；第 4 行输出数组维度，即 2。第 5 行表示输出数组的形状，即 3 行 3 列。

【运行结果】

第 3 行的输出结果：[[1 2 3]
[2 3 4]
[3 4 5]]

第 4 行的输出结果：2

第 5 行的输出结果：(3, 3)

一维 NumPy 数组只有一个轴(即 axis=0)；二维数据有两个轴(即 axis=0 和 axis=1)。对于二维 NumPy 数组来说，当在带有 axis 参数的二维数组上使用聚合函数时，如 np.sum()，它会将二维数组折叠为一维数组进行计算，即减少维度。换句话说，将 NumPy 聚合函数与 axis 参数一起使用时，指定的轴是折叠的轴。

例 3.16

```
1:  import numpy as np
2:  ndarray = np.array([[0,1,2], [3,4,5]])
3:  print(ndarray)
4:  print(np.sum(ndarray, axis = 0))
5:  print(np.sum(ndarray, axis = 1))
```

【例题解析】

第 2 行定义了一个数组 ndarray。

第 4 行表示将数组 ndarray 沿行(axis=0)向下求和，即[0+3, 1+4, 2+5]。

第 5 行表示将数组 ndarray 沿列(axis=1)横穿求和，即[0+1+2, 3+4+5]。

【运行结果】

第 3 行的输出结果：[[0 1 2]
[3 4 5]]

第 4 行的输出结果：[3 5 7]

第5行的输出结果：[3 12]

综上，之所以要设置不同的轴，是因为在进行数据分析时，可以根据不同的需求进行不同维度的处理。

3.2 pandas基础

pandas最初由AQR Capital Management于2008年4月开发，并于2009年年底开源面市。pandas含有使数据清洗和分析工作变得更快、更简单的数据结构和操作工具，是Python的一个数据分析包，经常和其他工具一同使用。

pandas支持大部分NumPy语言风格的数组计算，尤其是数组函数以及没有for循环的各种数据处理。尽管pandas采用了大量的NumPy编码风格，但二者最大的不同是pandas是专门为处理表格和混杂数据设计的。而NumPy更适合处理统一的数值数组数据。

在Python中调用pandas往往使用import pandas as pd，在调用pandas中的模块或函数时，应该使用“pd.模块或函数名称”的方式。

3.2.1 pandas数据结构

使用pandas需要先熟悉它的两个主要数据结构，即Series和DataFrame。这两个主要的数据结构并不能解决所有问题，但它们为大多数应用提供了一种可靠的、易于使用的基础。接下来，主要介绍这两种数据结构的创建和基本使用。

1. Series数据结构

Series数据结构类似于一维数组，由一组数据和对应的索引组成。Series的表现形式为索引在左边，值在右边。如果在创建Series对象时没有指定索引，会自动创建一个0～$N-1$（N为数据的长度）的整数型索引。Series的参数中可带入列表、元组、字典。其中，如果传入字典，字典的键key和值value将自动转换成Series对象的索引和元素。

例3.17

```
1:  import pandas as pd
2:  Series_1 =  pd.Series(['a',2,'螃蟹'],index=[1,2,3])
3:  print(Series_1)
4:  Series_2 = pd.Series((4, 5, 7, 1))
5:  print(Series_2)
6:  Series_3 = pd.Series({'a':1, 'b':2})
7:  print(Series_3)
```

【例题解析】

该例演示如何从列表、元组和字典创建Series对象。

第2行、第4行和第6行分别通过参数带入列表、元组和字典创建了Series对象。需要说明的是，当参数带入字典时，字典的键（即a和b）和值（即1和2）将分别自动转换成

Series 对象的索引和元素。

【运行结果】

```
第 4 行的输出结果：1       a
                   2       2
                   3       螃蟹
                   dtype：object
第 6 行的输出结果：0       4
                   1       5
                   2       7
                   3       1
                   dtype：int64
第 8 行的输出结果：a       1
                   b       2
                   dtype：int64
```

Series 中的单个或一组值可通过索引的方式选取。另外，使用 NumPy 函数或类似 NumPy 的运算（如根据条件进行过滤、标量乘法、应用数学函数等）都会保留索引值的链接。

例 3.18

```
1:  import pandas as pd
2:  series = pd.Series([4, 7, -5, 3], index = ['a', 'b', 'c', 'd'])
3:  print(series)
4:  print(series['a'])
5:  print(series[['a', 'b', 'c']])
6:  print(series[series > 0])
7:  print(series * 2)
```

【例题解析】

第 2 行创建了一个 Series 对象 series，其参数中代入的是列表且通过参数 index=['a', 'b', 'c', 'd']指定索引。

第 4 行通过指定索引的方式访问 Series 对象 series 中的单个值，即索引'a'处的值 4。

第 5 行通过索引的方式访问 Series 对象 series 中的多个值。

第 6 行表示对 Series 数据结构的 series 进行过滤运算，即保留了值大于 0 的内容。

第 7 行对 series 进行标量乘法运算，即将 series 中的每一个值都乘以 2。

【运行结果】

```
第 3 行的输出结果：a       4
                   b       7
                   c       -5
                   d       3
                   dtype：int64
```

```
第 4 行的输出结果：4
第 5 行的输出结果：a      4
                    b      7
                    c     -5
                    dtype：int64
第 6 行的输出结果：a      4
                    b      7
                    d      3
                    dtype：int64
第 7 行的输出结果：a      8
                    b     14
                    c    -10
                    d      6
                    dtype：int64
```

2. DataFrame 数据结构

DataFrame 是一个表格型的数据结构，含有一组有序的列，每列可以是不同的值类型（数值、字符串、布尔值等）。DataFrame 对象既有行索引也有列索引，其可以被视为一个共享相同索引的 Series 字典。构建 DataFrame 对象常用的方法是利用一个等长列表组成的字典或 NumPy 数组构建。

例 3.19

```
1:  import pandas as pd
2:  import numpy as np
3:  dt = {'name':['张三', '李四', '王五'],
          'sex':['male', 'female','male' ],
          'year':[2019, 2017, 2020],
          'city':['北京', '上海','深圳' ]}
4:  df_1 = pd.DataFrame(dt)
5:  print(df_1)
6:  df_2 = pd.DataFrame(np.random.randint(0,11,[2,3]),
                        index = np.arange(0,2),columns = ['A','B','C'])
7:  print(df_2)
8:  df_3 = DataFrame(dt,columns = ['name','sex','month','city'], index = ['a','b','c'])
9:  print(df_3)
```

【例题解析】

该例演示了利用一个等长列表组成的字典或 NumPy 数组构建 DataFrame 对象。

第 3 行定义了一个等长列表组成的字典 dt，字典中的所有关键字(key)将是未来定义的 DataFrame 对象的列索引，每一个索引下的列表值是 DataFrame 对象列索引下的具体值。

第 4 行创建了 DataFrame 对象 df_1。由第 5 行的输出结果可以看出，DataFrame 对

象有行索引和列索引，行索引类似于 Excel 表格中每行的编号（没有指定行索引的情况下），当没有指定行索引的情况下，会使用 0～$N-1$（N 为数据的长度）作为行索引；列索引类似于 Excel 表格的列名（通常也可称为字段）。

第 6 行利用一个 NumPy 数组（即 2 行 3 列，值为 0～10 的随机整数的数组）构建 DataFrame 对象 df_2。其中，index 指定了行索引的排列顺序，即 0、1，参数 columns 指定了列索引的排列顺序，即 A、B、C。通过输出结果可以看出，DataFrame 对象会自动加上索引，并且全部列按给定索引顺序有序排列。

与 Series 类似，如果传入的列在数据中找不到，就会在结果中产生缺失值。第 8 行利用等长的列表组成字典 dt（第 2 行定义）构建 DataFrame 对象 df_3。columns 参数指定的列名与数据字典的关键字进行匹配，没有匹配到的列以空值（NaN）填充。

【运行结果】

第 5 行的输出结果：

	name	sex	year	city
0	张三	male	2019	北京
1	李四	female	2017	上海
2	王五	male	2020	深圳

第 7 行的输出结果（结果不唯一）：

	A	B	C
0	7	1	6
1	3	4	6

第 9 行的输出结果：

	name	sex	month	city
a	张三	male	NaN	北京
b	李四	female	NaN	上海
c	王五	male	NaN	深圳

3.2.2 索引重命名与重新索引

本节主要介绍索引的重命名和重新索引操作。

rename()函数可以实现对 Series 或 DataFrame 对象索引名称的修改。rename()函数的语法格式为：

```
DataFrame.rename(mapper = None, index = None, columns = None, axis = None, copy = True,
inplace = False, level = None, errors = 'ignore')
```

该函数的功能是修改 Series 或 DataFrame 对象的索引名称，返回一个修改后的 Series 或者 DataFrame 对象。rename()函数的参数及描述如表 3-8 所示。其中，常用的参数为 index、columns 和 inplace。

表 3-8 rename()函数的参数描述

参数	描　述
mapper	映射结构，修改 columns 或 index 要传入一个映射体，可以是字典、函数。修改列标签跟 columns 参数一起；修改行标签跟 index 参数一起
index	行标签参数，mapper，axis=0 等价于 index=mapper

续表

参数	描　　述
columns	列标签参数,mapper, axis=1 等价于 columns=mapper
axis	轴标签格式,0 代表 index,1 代表 columns,默认为 index
copy	默认为 True,赋值轴标签后面的数据
inplace	默认为 False,不在原处修改数据,返回一个新的 DataFrame
level	默认为 None,处理单个轴标签(有的数据会有两个或多个 index 或 columns)
errors	默认 ignore,如果映射体里面包含 DataFrame 没有的轴标签,忽略不报错。

例 3.20

```
 1:  import pandas as pd
 2:  series = pd.Series([4.5, 7.2, -5.3, 3.6], index = ['d', 'b', 'a', 'c'])
 3:  print(series)
 4:  series_rename = series.rename(index = {'d':'m', 'b':'n', 'a':'o', 'c':'p'}, inplace =
False)
 5:  print(series_rename)
 6:  dt = {'name':['张三', '李四', '王五', '小红'],
           'sex':['male', 'female', 'male', 'male'],
           'year':[2019, 2017,2020,2021]}
 7:  df = pd.DataFrame(dt, index = ['a','b','c','d'])
 8:  print(df)
 9:  df_rename = df.rename(index = {'a':'m', 'b':'n', 'c':'o', 'd':'p'},
                           columns = {'year':'Year'}, inplace = False)
10:  print(df_rename)
```

【例题解析】

第 2 行定义了一个 Series 数据结构 series。第 4 行利用 rename()函数将标签参数“d,b,a,c”改为“m,n,o,p”,由于 inplace=False,series 数据没有被修改,而是返回一个新的 DataFrame 对象 series_rename。

第 6 行利用等长的列表定义了一个字典 dt。第 7 行利用字典 dt 构建 DataFrame 对象 df,并指明了行索引为“'a','b','c','d'”。第 9 行利用 rename()函数将 DataFrame 对象 df 行标签参数“a,b,c,d”改为“m,n,o,p”,列标签参数“year”改为“Year”,由于 inplace=False,df 数据没有被修改,而是返回一个新的 DataFrame 对象 df_rename。

【运行结果】

```
第 3 行的输出结果: d      4.5
                   b      7.2
                   a     -5.3
                   c      3.6
                   dtype: float64
第 5 行的输出结果: m      4.5
                   n      7.2
```

```
o    -5.3
p     3.6
dtype: float64
```

第 8 行的输出结果：

	name	sex	year
a	张三	male	2019
b	李四	female	2017
c	王五	male	2020
d	小红	male	2021

第 10 行的输出结果：

	name	sex	Year
m	张三	male	2019
n	李四	female	2017
o	王五	male	2020
p	小红	male	2021

重新索引并不是给索引重新命名，而是对索引进行排序，如果某个索引值不存在，将引入空值。reindex()是 pandas 对象的一个重要函数。reindex()函数的作用是对 Series 或 DataFrame 对象创建一个适应新索引的新对象。其语法格式为：

```
DataFrame.reindex(labels = None, index = None, columns = None, axis = None, method = None,
copy = True, level = None, fill_value = nan, limit = None, tolerance = None)
```

该方法的参数及描述如表 3-9 所示。

表 3-9 reindex()方法的参数及描述

参数	描 述
index	用作索引的新序列
method	差值(填充)方式；'ffill'为前向填充，'bfill'为后向填充
fill_value	在重新索引的过程中，需要引入缺失值时使用的替代值
limit	向前或向后填充时的最大填充量
tolerance	向前或向后填充时，填充不准确匹配项的最大间距(绝对值距离)
level	在 MultiIndex 的指定级别上匹配简单索引，否则选取其子集
copy	默认为 True，即使新索引等于旧索引，也总是复制底层数据；如果为 False，则新旧索引相同时就不复制。

续例 3.20(1)

```
11:  series_reindex = series.reindex(['a', 'b', 'c', 'd', 'e'])
12:  print(series_reindex)
```

【例题解析】

第 11 行利用 Series 对象的 reindex()函数，将 series 按照指定的顺序(即['a', 'b', 'c', 'd', 'e'])重新排序。由于 series 中并无索引“e”，即存在缺失值，用 NaN 填补。

【运行结果】

```
a    -5.3
b     7.2
c     3.6
d     4.5
e     NaN
dtype: float64
```

DataFrame 调用 reindex()函数时,可以对其行和列索引进行重新索引。其中,当只传递一个序列时,默认是对行重新索引,与指定 index 参数所起作用是一致的。如果要对列重新索引,需要设置 columns 参数。

续例 3.20(2)

```
13:  df_reindex_1 = df.reindex(['a', 'b', 'c', 'd', 'e'])
14:  print(df_reindex_1)
15:  list = ['name', 'year', 'id']
16:  df_reindex_2  = df.reindex(columns = list)
17:  print(df_reindex_2)
```

【例题解析】

第 13 行通过一个序列['a', 'b', 'c', 'd', 'e']实现行被重新索引,由于 df 中并无索引"e",即存在缺失值,用 NaN 填补得到 df_reindex_1。第 16 行通过使用 columns 关键字利用 list 实现了对 df 的列被重新索引。由于 df 中并无列索引"id",即存在缺失值,用 NaN 填补得到 df_reindex_2。

【运行结果】

第 14 行的输出结果:

	name	sex	year
a	张三	male	2019.0
b	李四	female	2017.0
c	王五	male	2020.0
d	小红	male	2021.0
e	NaN	NaN	NaN

第 17 行的输出结果:

	name	year	id
a	张三	2019	NaN
b	李四	2017	NaN
c	王五	2020	NaN
d	小红	2021	NaN

3.2.3 数据基本操作

在数据分析中,常用的数据基本操作为"增、删、改、查"。

1. 增加数据

增加包括向 DataFrame 对象中添加新的行和新的列两种不同的操作。

(1) 当向 DataFrame 对象中添加新的行时,可以使用 append()函数。其语法格式为:

```
DataFrame.append(other, ignore_index = False, verify_integrity = False, sort = None)
```

该函数的功能是为 DataFrame 对象的末尾添加新的行,返回一个新的 DataFrame 对象。append()函数的参数及描述如表 3-10 所示。

表 3-10 append()函数的参数描述

参 数	描 述
other	DataFrame、Series、dict、list 这样的数据结构
ignore_index	默认值为 False,如果为 True,则不使用 index 标签
verify_integrity	默认值为 False,如果为 True,当创建相同的 index 时会抛出 ValueError 的异常
sort	boolean,默认是 None

例 3.21

```
1:  import pandas as pd
2:  dt = {'name':['张三', '李四', '王五', '小红'], 'sex':['male', 'female', 'male', 'male'],
'year':[2019, 2017,2020,2021],'score':[90, 88,93,89]}
3:  df = pd.DataFrame(dt,index = ['a', 'b', 'c', 'd'])
4:  print('初始 DataFrame:\n',df)
5:  dict_add = {'name': '小张','sex': 'female', 'year':2021, 'score':95}
6:  df_add = df.append(dict_add, ignore_index = True)
7:  print('使用 append()函数添加一行数据后的 DataFrame:\n',df_add)
```

【例题解析】

第 3 行利用字典 dt 构建 DataFrame 对象 df,并指明了行索引。

第 5 行定义了一个字典 dict_add。

第 6 行利用 append()函数将字典 dict_add 作为新的行添加到 df 得到 DataFrame 对象 df_add。因为 ignore_index 为 True,即不使用 index 的标签。

【运行结果】

第 4 行的输出结果:初始 DataFrame:

	name	sex	year	score
a	张三	male	2019	90
b	李四	female	2017	88
c	王五	male	2020	93
d	小红	male	2021	89

第 7 行的输出结果:使用 append()函数添加一行数据后的 DataFrame:

	name	sex	year	score
0	张三	male	2019	90
1	李四	female	2017	88
2	王五	male	2020	93
3	小红	male	2021	89
4	小张	female	2021	95

(2) 当向 DataFrame 对象中添加新的列时,可以采用赋值语句或 insert 函数。在赋值语句中,如果赋值列在原数据框中不存在,则直接在数据框的最后增加该列。此外,也可以使用 insert()函数添加新的列,其语法格式为:

```
DataFrame.insert(loc, column, value, allow_duplicates = False)
```

该函数返回一个新的 DataFrame 对象。insert()函数的参数及描述如表 3-11 所示。

表 3-11 insert()函数的参数描述

参 数	描 述
loc	int 型,表示第几列;若在第一列插入数据,则 loc=0
column	给插入的列取名,如 column='新的一列'
value	数字,array,series 等都可以
allow_duplicates	是否允许列名重复,默认为 False,当为 True 时表示允许新的列名与已存在的列名重复

续例 3.21(1)

```
 8:  df_add['age'] = [23, 25, 22, 20,21]
 9:  print('使用赋值语句添加一列数据后的 DataFrame:\n',df_add)
10:  df_add.insert(loc = 1,column = 'id', value = [13, 15, 10, 8, 6], allow_duplicates =
False)
11:  print('使用 insert()函数添加一列数据后的 DataFrame:\n',df_add)
```

【例题解析】

第 8 行采用赋值语句向数据框中添加了新列“age”,该列中的值为“23, 25, 22, 20, 21”。

第 10 行采用 insert()函数在 df_add 列索引为 1 处添加一列名为“id”的列,并且该列中的值为“13, 15, 10, 8, 6”。需要注意的是,该列的长度应该与其他列的长度一致。

【运行结果】

第 9 行的输出结果:使用赋值语句添加一列数据后的 DataFrame:

	name	sex	year	score	age
0	张三	male	2019	90	23
1	李四	female	2017	88	25
2	王五	male	2020	93	22
3	小红	male	2021	89	20

4	小张	female	2021	95	21

第 11 行的输出结果：使用 insert()函数添加一列数据后的 DataFrame：

	name	id	sex	year	score	age
0	张三	13	male	2019	90	23
1	李四	15	female	2017	88	25
2	王五	10	male	2020	93	22
3	小红	8	male	2021	89	20
4	小张	6	female	2021	95	21

2. 删除数据

如果想要删除 DataFrame 对象中的一行或一列数据，可以通过 drop()函数实现，其语法格式为：

```
DataFrame.drop(labels = None, axis = 0, index = None, columns = None, inplace = False)
```

该函数的功能是删除 DataFrame 对象中的指定行或者列，返回一个新的 DataFrame 对象。drop()函数的参数及描述如表 3-12 所示。

表 3-12 drop()函数的参数描述

参数	描 述
labels	要删除的索引或列标签
axis	从索引 0 或“index”还是从列 1 或“columns”中删除标签
index	直接指定要删除的行
columns	直接指定要删除的列
inplace	默认该删除操作不改变原数据，返回一个执行删除操作后的新 DataFrame

续例 3.21(2)

```
12:  df_delRow = df_add.drop(1)
13:  print('使用 drop()函数删除一行数据:\n',df_delRow)
14:  df_delCol = df_add.drop('age', axis = 1)
15:  print('使用 drop()函数删除一列数据:\n',df_delCol)
```

【例题解析】

第 12 行利用 drop()函数删除 df_delRow 中索引为 1 的行，即删除“李四”所在的行。

第 14 行通过设置 axis=1，删除 df_delCol 中的“age”列。

【运行结果】

第 13 行的输出结果：使用 drop()函数删除一行数据：

	name	id	sex	year	score	age
0	张三	13	male	2019	90	23
2	王五	10	male	2020	93	22
3	小红	8	male	2021	89	20

4	小张	6	female	2021	95	21

第 15 行的输出结果：使用 drop()函数删除一列数据：

	name	id	sex	year	score
0	张三	13	male	2019	90
1	李四	15	female	2017	88
2	王五	10	male	2020	93
3	小红	8	male	2021	89
4	小张	6	female	2021	95

3. 数据修改

如果想要修改 DataFrame 对象中的数据，可以通过 replace()函数实现，其语法格式为：

```
DataFrame. replace(old, new[, max])
```

其中，old 参数是将被替换的值；new 参数是新值，用于替换 old 值；max 为可选参数，表示替换不超过 max 次。该函数的功能是修改 DataFrame 对象中所有匹配的值，返回一个新的 DataFrame 对象。

修改 DataFrame 对象中的数据还可以使用 loc 函数和 iloc 函数实现。其中，loc[]函数用行列标签选择数据，选定的数据范围前闭后闭；iloc[]函数用于行列索引值选择数据，选定的数据范围前闭后开。

续例 3.21(3)

```
16:  df_update = df.replace(2020,2019)
17:  print('使用 replace()函数修改数据:\n',df_update)·
18:  df_update.loc['b','name'] = '小丽'
19:  print('使用 loc 函数修改一个数据:\n',df_update)
20:  df_update.loc['b'] = ['小丽', 'Female', 2021,88]
21:  print('使用 loc 函数修改一行数据:\n',df_update)
22:  df_update.loc['b',['name', 'year']] = ['小蒙', 2019]
23:  print('使用 loc 函数修改部分数据:\n',df_update)
24:  df_update.iloc[2,1] = 'female'
25:  print('使用 iloc 函数修改一个数据:\n',df_update)
26:  df_update.iloc[:,2] = [2018,2021,2019,2017]
27:  print('使用 iloc 函数修改一列数据:\n',df_update)
28:  df_update.iloc[0,:] = ['小娜', 'female', 2021,95]
29:  print('使用 iloc 函数修改一列数据:\n',df_update)
```

【例题解析】

修改前后 DataFrame 对象中元素的变化如图 3-2 所示。

该例旨在说明利用 replace 函数、loc 函数和 iloc 函数对 DataFrame 对象中的元素进行修改时的区别。

(1) replace()函数能修改 DataFrame 对象中所有匹配的值。第 16 行将 df 中所有的

图 3-2 修改前后 DataFrame 对象中元素的变化

“2020”修改为“2019”，并存到新的 DataFrame 对象 df_update 中。

(2) loc()函数中使用的参数是行列标签。

第 18 行利用 loc()函数将 DataFrame 对象 df_update 中行标签为'b'，列标签为'name'位置处的元素修改为“小丽”。

第 20 行利用 loc()函数修改行标签为 'b' 的行的所有元素，将“'小丽','female',2017,88”修改为“'小丽','Female',2021,88”。

第 22 行将 loc()函数的行标签设为'b'，列标签设为列表['name','year']，修改行列相交位置处的元素值“'小丽',2021”，修改为“'小蒙',2019”。

(3) iloc()函数中参数使用的是行列索引值。

第 24 行利用 iloc()函数将 DataFrame 对象 df_update 行索引为 2，列索引为 1 位置处的元素修改为“female”。

第 26 行利用 iloc()函数将 DataFrame 对象 df_update 中列索引为 2 所在列的所有元素修改为“2018,2021,2019,2017”。

第 28 行利用 iloc()函数将 DataFrame 对象 df_update 行索引为 0 所在行的所有元素修改为“'小娜','female', 2021,95”。

【运行结果】

第 17 行的输出结果：使用 replace()函数修改数据：

	name	sex	year	score
a	张三	male	2019	90
b	李四	female	2017	88
c	王五	male	2019	93
d	小红	male	2021	89

第 19 行的输出结果：使用 loc()函数修改一个数据：

	name	sex	year	score
a	张三	male	2019	90
b	小丽	female	2017	88
c	王五	male	2019	93
d	小红	male	2021	89

第 21 行的输出结果：使用 loc()函数修改一行数据：

	name	sex	year	score
a	张三	male	2019	90

b	小丽	female	2021	88
c	王五	male	2019	93
d	小红	male	2021	89

第 23 行的输出结果：使用 loc()函数修改部分数据：

	name	sex	year	score
a	张三	male	2019	90
b	小蒙	female	2019	88
c	王五	male	2019	93
d	小红	male	2021	89

第 25 行的输出结果：使用 iloc()函数修改一个数据：

	name	sex	year	score
a	张三	male	2019	90
b	小蒙	female	2019	88
c	王五	female	2019	93
d	小红	male	2021	89

第 27 行的输出结果：使用 iloc()函数修改一列数据：

	name	sex	year	score
a	张三	male	2018	90
b	小蒙	female	2021	88
c	王五	female	2019	93
d	小红	male	2017	89

第 29 行的输出结果：使用 iloc()函数修改一列数据：

	name	sex	year	score
a	小娜	female	2021	95
b	小蒙	female	2021	88
c	王五	female	2019	93
d	小红	male	2017	89

4. 数据查询

数据分析中，经常需要选取部分数据进行处理和分析。可以通过数据框对象的行列标签或者索引完成数据的提取工作。

1) 选取列

一般是通过列标签获取 DataFrame 对象的列数据，返回的数据为 Series 结构；通过列表可以获取多列的数据，返回的数据为 DataFrame 结构。

续例 3.21(4)

```
30:  print(df['name'])
31:  print(df[['name', 'sex']])
```

【例题解析】

第 30 行表示获取 DataFrame 对象 df 的列标签为‘name’的列数据。第 31 行表示获取 DataFrame 对象 df 的列标签为‘name’和‘sex’的列数据。

【运行结果】

```
第 27 行的输出结果：a       张三
                    b       李四
                    c       王五
                    d       小红
                    Name：name，dtype：object
第 28 行的输出结果：    name    sex
                    a   张三    male
                    b   李四    female
                    c   王五    male
                    d   小红    male
```

2）选取行

一般通过行标签或者行索引的方式获取 DataFrame 对象的行数据，返回的数据为 Series 结构；通过行标签或者行索引切片的方式可以获取多行的数据，返回的数据为 DataFrame 结构。

续例 3.21（5）

```
32:  print(df.loc['b'])
33:  print(df.iloc[1])
34:  print(df[0: 2])
35:  print(df['a': 'c'])
```

【例题解析】

第 32 行利用 loc()函数获取行标签为‘b’的单行数据，返回一个 Series 结构；第 33 行利用 iloc()函数单独获取行索引为 1 的行数据。比较二者的输出结果，无论是通过行标签还是行索引，在相同的位置上获取的行数据相同。

第 34 行利用行索引的切片形式获取行索引 0 至行索引 2 处的行数据（左闭右开）；第 35 行利用行标签的切片形式获取行标签为‘a’至行索引标签为‘c’处的行数据（左闭右闭）。

【运行结果】

```
第 29 行的输出结果：name      李四
                    sex       female
                    year      2017
                    score     88
                    Name：b，  dtype：object
第 30 行的输出结果：name      李四
                    sex       female
```

```
year        2017
score       88
Name: b, dtype: object
```

第 31 行的输出结果：

	name	sex	year	score
a	张三	male	2019	90
b	李四	female	2017	88

第 32 行的输出结果：

	name	sex	year	score
a	张三	male	2019	90
b	李四	female	2017	88
c	王五	male	2020	93

另外，如果仅仅想查询 DataFrame 对象中的前几行或者后几行的数据，可以通过 head()或者 tail()函数实现。其中，head()函数的语法格式为：DataFrame.head(n)，表示返回前 n 行的数据。例如，DataFrame.head(3)表示返回前 3 行数据。与之对应的是 tail()函数，语法格式与 head()函数类似，其含义为返回后 n 行的数据。

3) 选取行和列子集

在数据分析中，有时可能只是对某个位置的数据进行操作，这种操作与行列均有关系，可通过 at 和 iat 方法实现。其中，at 方法是按行列标签选取数据；iat 方法是按行列索引选取数据。

续例 3.21(6)

```
36:  print(df.at['a','name'])
37:  print(df.iat[0,0])
```

【例题解析】

第 36 行利用 at 方法获取行标签‘a’和列标签‘name’处的数据，即张三；第 37 行利用 iat 方法单独获取行索引 0 和列索引 0 处的数据，即张三。通过对二者的输出结果比较，在相同位置处通过行列标签和行列索引获取的数据相同。

【运行结果】

第 36 行的输出结果：张三

第 37 行的输出结果：张三

小结

本章重点介绍了 NumPy 和 pandas 基础。

NumPy 中主要介绍了其核心特征之一的 N 维数组对象 ndarray，包括 ndarray 数组的创建、常用属性、数据类型、算术操作、索引和切片以及理解 NumPy 中的轴。其中，创建数组最简单的方式就是使用 array 函数，该函数接收任意的序列型对象(也包括其他的数组)，生成一个新的包含传递数据的 NumPy 数组。ndarray 数组的常用属性包括 shape、dtype、ndim

等。常用的数据类型有浮点型(float)、整数(int)、布尔值(bool)等。常用算术操作主要包括数组和标量间的运算、通用函数、条件逻辑运算、统计运算、布尔型数组运算以及排序等。

pandas基础中主要介绍了常用的数据结构、索引操作以及数据基本操作。首先,两个主要的数据结构,即Series和DataFrame。Series数据结构类似于一维数组,但其是由一组数据(各种NumPy数据类型)和对应的索引组成。DataFrame是一个表格型的数据结构,其既有行索引也有列索引。然后,对Series和DataFrame从重新索引以及对数据行、列的操作两个主要方面进行了介绍。最后,介绍了DataFrame上执行基本的数据操作即"增、删、改、查"的常用方法。

习题

请从以下各题中选出正确答案(正确答案可能不止一个)。

1. 计算NumPy中元素个数的方法是(　　)。

   A. np.aqrt()　　B. np.size()　　C. np.identity()　　D. np.eye()

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

   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]])

   D. array([[4 6 8 10] [20 22 24 26] [36 38 40 42]])

3. 有数组n=np.arange(24).reshape(2,-1,2,2),n.shape返回的结果是(　　)。

   A. (2,3,2,2)　　B. (2,2,2,2)　　C. (2,4,2,2)　　D. (2,6,2,2)

4. NumPy中向量转成矩阵使用(　　)。

   A. reshape()　　B. resize()　　C. arange()　　D. random()

5. 以下(　　)可实现使用NumPy生成20个0～100随机数并创建DataFrame对象。

   A. temp = np.random.randint(1,100,20)
      DF = pd.DataFrame(temp)

   B. temp = np.random.randint(1,101,20)
      DF = pd.DataFrame(temp)

   C. temp = np.random.randint(0,101,20)
      DF = pd.DataFrame(temp)

   D. temp = np.random.randint(0,100,20)
      DF = pd.DataFrame(temp)

6. df.tail()这个函数是用来(　　)。

   A. 创建数据　　B. 查看数据　　C. 增加数据　　D. 修改数据

7. 最简单的series是由(　　)数据构成的。

   A. 一个数组　　B. 两个数组　　C. 三个数组　　D. 四个数组

8. 下列关于DataFrame说法正确的是(　　)。

   A. DataFrame是一个类似二维数组的对象

B. DataFrame 是由数据和索引组成

C. DataFrame 有行索引与列索引

D. 默认情况下 DataFrame 的行索引在最右侧

9.

```
1: import pandas as pd
2: df = pd.DataFrame({'a': list("opq"),'b': [3,2,1]}),index = ['e','f','g']
```

针对以上代码,以下说法正确的是(　　)

A. df[0:1]返回第 0 行的数据

B. df[0:1]返回第 0 列的数据

C. df[0]会报错

D. df['e']会报错

10. 以下代码的输出结果为(　　)。

```
1: import pandas as pd
2: obj = pd.Series([4.5, 7.2, -5.3, 3.6], index = ['d', 'b', 'a', 'c'])
3: obj2 = obj.reindex(['a', 'b', 'c', 'd', 'e'])
4: print(obj2)
5: obj3 = obj.reindex(['a', 'b', 'c', 'd', 'e'], fill_value = 0)
6: print(obj3)
```

A. 第 4 行的输出结果:
```
a  -5.3
b   7.2
c   3.6
d   4.5
e   4.5
dtype: float64
```
第 6 行的输出结果:
```
a  -5.3
b   7.2
c   3.6
d   4.5
e   0.0
dtype: float64
```

B. 第 4 行的输出结果:
```
a  -5.3
b   7.2
c   3.6
d   4.5
dtype: float64
```
第 6 行的输出结果:
```
a  -5.3
b   7.2
c   3.6
d   4.5
e   0.0
dtype: float64
```

C. 第 4 行的输出结果:
```
a  -5.3
b   7.2
c   3.6
d   4.5
e   NaN
dtype: float64
```
第 6 行的输出结果:
```
a  -5.3
b   7.2
c   3.6
d   4.5
dtype: float64
```

D. 第 4 行的输出结果：
```
a   -5.3
b    7.2
c    3.6
d    4.5
e    NaN
dtype: float64
```
第 6 行的输出结果：
```
a   -5.3
b    7.2
c    3.6
d    4.5
e    0.0
dtype: float64
```