第 5 章

数据处理基础

在 Excel 中，无论是用函数，还是用 VBA 做数据处理，都需要用到运算符、分支判断、循环和数组等技术，以及一些常见的数据统计方法。本章将讲解这些技术，并且在 Pandas 中的表示方法更加简洁。

本章主要讲解 Pandas 中的常用运算符，使用这些运算符对单值、Series 和 DataFrame 数据做运算，条件分支判断，对 Series 和 DataFrame 这两种数据的遍历以及常用的数据统计函数。

5.1 数据运算处理

在 Excel 中写公式时，如果要用到数组运算，一般是单值、一维数组和二维数组这 3 种数据之间的运算。在 Pandas 中如果要进行数组运算，一般是单值、Series（一维数组）和 DataFrame（二维数组）之间的运算。

在 Pandas 中进行数据运算时，尽量做矩阵运算（数组运算），避免单值与单值之间进行运算，这样才能体现 Pandas 处理速度快的优势。

5.1.1 运算符与运算函数

想要学习 Pandas 中的数组运算，首先要学习一下常用的运算符，下面罗列了在 Pandas 中提供的常用运算符，见表 5-1。

表 5-1 Pandas 常用运算符

运算符类型	运 算 符 号	函 数	注 释
算术运算符	+	add	加
	−	sub	减
	*	mul	乘
	/	div	除
求模运算符	%	mod	求余数
求幂运算符	**	pow	幂次方
比较运算符	==		等于
	!=		不等于
	>		大于
	>=		大于或等于

续表

运算符类型	运 算 符 号	函　　数	注　　释
比较运算符	<		小于
	<=		小于或等于
连接运算符	+		连接
逻辑运算符	&	and	与
	|	or	或
	~	not	非

在5.1.2～5.1.6节中，以加运算为例，使用符号和函数两种方法演示运算符是如何在不同数据结构中进行运算的。

提示：读者在测试符号和函数这两种方法时，必须注释其中的一种方法，否则结果会有累加。

5.1.2　Series与单值的运算

如图5-1(a)所示为原表格，对语文列中的每个数字加1，分别用符号和函数两种方法完成，示例代码如下：

```
#chapter5\5-1\5-1-1.ipynb
import pandas as pd                              #导入 Pandas 库，并命名为 pd
df = pd.read_excel('5-1-1.xlsx','1月')           #读取 Excel 中的 1 月工作表数据
df['语文'] = df['语文'] + 1                      #用符号进行相加运算(运算符)
df['语文'] = df['语文'].add(1)                   #用函数进行相加运算(函数)
df                                               #输出 df 表数据
```

运行结果如图5-1(b)所示。

	姓名	语文	数学
0	张三	71	81
1	王二	70	100
2	麻子	81	98

(a) 原表格

	姓名	语文	数学
0	张三	72	81
1	王二	71	100
2	麻子	82	98

(b) 相加运算后的表格

图5-1　Series与单值运算

在上面的代码中，df['语文']是Series数据，与单个值1相加，其运算结果也是Series数据，然后赋值给语文列。

5.1.3　DataFrame与单值运算

如图5-2(a)所示为原表格，对语文、数学两列中的每个数字加1，分别用符号和函数两种方法完成，示例代码如下：

```
#chapter5\5-1\5-1-2.ipynb
import pandas as pd                              #导入 Pandas 库，并命名为 pd
df = pd.read_excel('5-1-2.xlsx','1月')           #读取 Excel 中的 1 月数据
```

```
df[['语文','数学']] = df[['语文','数学']] + 1                    #用符号做相加运算(运算符)
df[['语文','数学']] = df[['语文','数学']].add(1)                 #用函数做相加运算(函数)
df                                                              #输出 df 表数据
```

运行结果如图 5-2(b)所示。

	姓名	语文	数学
0	张三	71	81
1	王二	70	100
2	麻子	81	98

(a) 原表格

	姓名	语文	数学
0	张三	72	82
1	王二	71	101
2	麻子	82	99

(b) 相加运算后的表格

图 5-2　DataFrame 与单值运算

在上面的代码中，df[['语文','数学']]是 DataFrame 数据，与单个值 1 相加，其运算结果也是 DataFrame 数据，然后赋值给语文、数学两列。

5.1.4　Series 与 Series 运算

如图 5-3(a)所示为原表格，将两个表的语文列进行相加运算，分别用符号和函数两种方法完成，示例代码如下：

```
#chapter5\5-1\5-1-3.ipynb
import pandas as pd                                             #导入 Pandas 库,并命名为 pd
df1 = pd.read_excel('5-1-3.xlsx','1 月')                        #读取 Excel 中的 1 月数据
df2 = pd.read_excel('5-1-3.xlsx','2 月')                        #读取 Excel 中的 2 月数据
df1['语文'] = df1['语文'] + df2['语文']                         #用符号做相加运算(运算符)
df1['语文'] = df1['语文'].add(df2['语文'])                      #用函数做相加运算(函数)
df1                                                             #输出 df1 表数据
```

运行结果如图 5-3(b)所示。

	姓名	语文	数学
0	张三	71	81
1	王二	70	100
2	麻子	81	98

	姓名	语文	数学
0	张三	86	95
1	王二	73	77
2	麻子	100	85

(a) 原表格

	姓名	语文	数学
0	张三	157	81
1	王二	143	100
2	麻子	181	98

(b) 相加运算后的表格

图 5-3　两个 Series 相加

上面代码中，df1['语文']和 df2['语文']都是 Series 数据，两个 Series 相加的结果也是 Series 数据，然后根据要求赋值给 df1 的语文列或者 df2 的语文列。

5.1.5　DataFrame 与 DataFrame 运算

如图 5-4(a)所示为原表格，将原表格中的两个表进行相加运算，分别用符号和函数两种方法完成，示例代码如下：

```
#chapter5\5-1\5-1-4.ipynb
import pandas as pd                                             #导入 Pandas 库,并命名为 pd
```

```
df1 = pd.read_excel('5 - 1 - 4.xlsx','1 月')                    # 读取 Excel 中的 1 月数据
df2 = pd.read_excel('5 - 1 - 4.xlsx','2 月')                    # 读取 Excel 中的 2 月数据
df2 = df1 + df2                                                 # 用符号做相加运算(运算符)
df2 = df1.add(df2)                                              # 用函数做相加运算(函数)
df2                                                             # 输出 df2 表数据
```

运行结果如图 5-4(b)所示。

	姓名	语文	数学
0	张三	71	81
1	王二	70	100
2	麻子	81	98

	姓名	语文	数学
0	张三	86	95
1	王二	73	77
2	麻子	100	85

(a) 原表格

	姓名	语文	数学
0	张三张三	157	176
1	王二王二	143	177
2	麻子麻子	181	183

(b) 相加运算后的表格

图 5-4　两个 DataFrame 相加

上面代码中，df1 和 df2 都是 DataFrame 数据，两个 DataFrame 数据相加的结果也是 DataFrame 数据，然后赋值给 df1 或者 df2。观察相加运算之后的 DataFrame 表格可以发现，数字是对应相加的，而字符串是对应相连接的。

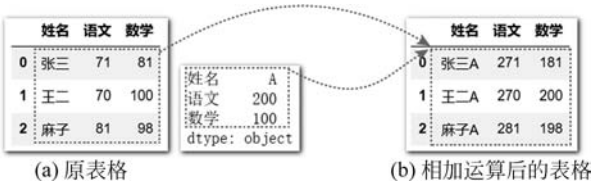
5.1.6　DataFrame 与 Series 运算

1. 列方向相加

如图 5-5(a)所示为原表格，将表格与 Series 数据进行相加运算，分别用符号和函数两种方法完成，示例代码如下：

```
# chapter5\5 - 1\5 - 1 - 5.ipynb
import pandas as pd                                             # 导入 Pandas 库,并命名为 pd
df = pd.read_excel('5 - 1 - 5.xlsx','1 月')                     # 读取 Excel 中的 1 月数据
df = df + pd.Series(['A',200,100],index = ['姓名','语文','数学'])   # 用符号做相加运算(运算符)
df = df.add(pd.Series(['A',200,100],index = ['姓名','语文','数学']))# 用函数做相加运算(函数)
df                                                              # 输出 df 表数据
```

运行结果如图 5-5(b)所示。



	姓名	语文	数学
0	张三	71	81
1	王二	70	100
2	麻子	81	98

姓名	A
语文	200
数学	100

dtype: object

	姓名	语文	数学
0	张三A	271	181
1	王二A	270	200
2	麻子A	281	198

(a) 原表格　　(b) 相加运算后的表格

图 5-5　DataFrame 与 Series 列方向相加

上面代码中，df 是 DataFrame 数据，与 Series 数据相加的结果也是 DataFrame 数据，但要记住 DataFrame 与 Series 做运算时，是 DataFrame 的每列数据与 Series 每个元素对应做运算，所以 Series 的元素个数必须与 DataFrame 的列数相同，否则不能正确运算。

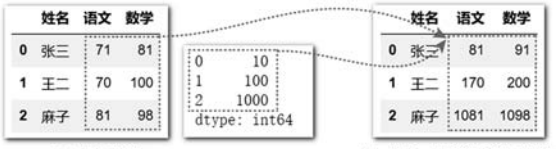
2. 行方向相加

如图 5-6(a)所示为原表格，将表格与 Series 数据做相加运算，只能用函数方法完成，示

例代码如下：

```
#chapter5\5-1\5-1-6.ipynb
import pandas as pd                                          #导入 Pandas 库,并命名为 pd
df = pd.read_excel('5-1-6.xlsx','1 月')                      #读取 Excel 中的 1 月数据
df[['语文','数学']] = df[['语文','数学']].add(pd.Series([10,100,1000],index = [0,1,2]),axis = 0)
                                                             #用函数做相加运算(函数)
df                                                           #输出 df 表数据
```

运行结果如图 5-6(b)所示。



(a) 原表格　　(b) 相加运算后的表格

图 5-6　DataFrame 与 Series 行方向相加

上面代码中，df 是 DataFrame 数据，与 Series 数据相加的结果也是 DataFrame 数据，但此时如果直接用符号(加号)做运算，不能正确运算，原因在于 DataFrame 与 Series 默认在列方向做运算，如果希望在行方向做运算，则只能使用对应的函数，add()函数的 axis 参数可以指定运算方向，默认值为 1，表示列方向，设置为 0 表示行方向。

5.1.7　数据运算时的对齐特性

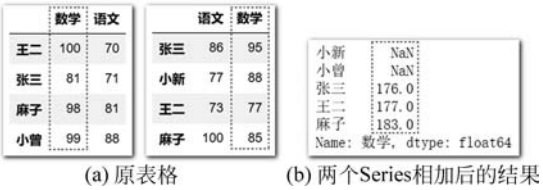
两个 Series 或者两个 DataFrame 在进行运算时，是具有对齐特性的，也就是它们的运算不是按位置对齐运算，而是按照索引对齐运算，下面以加运算为例做演示。

1. Series 的对齐特性

如图 5-7(a)所示为原表格，将两个表的数学列相加，分别用符号和函数两种方法完成，使用符号相加的代码为 df1['数学']+df2['数学']，示例代码如下：

```
#chapter5\5-1\5-1-7.ipynb
import pandas as pd                                          #导入 Pandas 库,并命名为 pd
df1 = pd.read_excel('5-1-7.xlsx','1 月',index_col = 0)       #读取 Excel 中的 1 月数据
df2 = pd.read_excel('5-1-7.xlsx','2 月',index_col = 0)       #读取 Excel 中的 2 月数据
df1['数学'] + df2['数学']                                    #用符号做相加运算(运算符)
```

运行结果如图 5-7(b)所示。



(a) 原表格　　(b) 两个Series相加后的结果

图 5-7　两个 Series 相加(1)

通过返回结果发现，索引标签相同的值会相加，例如'王二'分别处在不同位置，但一样能够正确相加。同时也发现一个新问题，'小曾'与'小新'两个标签并没有同时出现在两个Series中，返回的结果是缺失值(NaN)。如果希望即使另一个Series中是缺失值，也能相加出结果，则只能使用对应的add()函数，代码为df1['数学'].add(df2['数学'],fill_value=0)，其中fill_value=0表示如果有缺失值，则填充为0，示例代码如下：

```
#chapter5\5-1\5-1-8.ipynb
import pandas as pd                                          #导入Pandas库,并命名为pd
df1 = pd.read_excel('5-1-8.xlsx','1月',index_col = 0)        #读取Excel中的1月数据
df2 = pd.read_excel('5-1-8.xlsx','2月',index_col = 0)        #读取Excel中的2月数据
df1['数学'].add(df2['数学'],fill_value = 0)                  #用函数做相加运算(函数)
```

如图5-8(a)所示为原表格，运行结果如图5-8(b)所示。

	数学	语文
王二	100	70
张三	81	71
麻子	98	81
小曾	99	88

	语文	数学
张三	86	95
小新	77	88
王二	73	77
麻子	100	85

```
小新      88.0
小曾      99.0
张三     176.0
王二     177.0
麻子     183.0
Name: 数学, dtype: float64
```

(a) 原表格　　(b) 两个Series相加后的结果

图5-8　两个Series相加(2)

2. DataFrame的对齐特性

如图5-9(a)所示为原表格，将两个表相加，分别用符号和函数两种方法完成，使用符号相加的代码为df1+df2，示例代码如下：

```
#chapter5\5-1\5-1-9.ipynb
import pandas as pd                                          #导入Pandas库,并命名为pd
df1 = pd.read_excel('5-1-9.xlsx','1月',index_col = 0)        #读取Excel中的1月数据
df2 = pd.read_excel('5-1-9.xlsx','2月',index_col = 0)        #读取Excel中的2月数据
df1 + df2                                                    #用符号进行相加运算(运算符)
```

运行结果如图5-9(b)所示。

	数学	语文
王二	100	70
张三	81	71
麻子	98	81
小曾	99	88

	语文	数学
张三	86	95
小新	77	88
王二	73	77
麻子	100	85

	数学	语文
小新	NaN	NaN
小曾	NaN	NaN
张三	176.0	157.0
王二	177.0	143.0
麻子	183.0	181.0

(a) 原表格　　(b) 两个DataFrame相加后的结果

图5-9　两个DataFrame相加(1)

在原表格中，两个表格的行索引是姓名，列索引是科目，即使两个表的姓名和科目位置都不相同，也正确相加。同样，'小曾'和'小新'没有同时出现在两个表中，所以返回的结果是缺失值(NaN)，如果希望能相加出结果，则需使用对应的add()函数，代码为df1.add(df2,fill_value=0)，其中fill_value=0表示如果有缺失值，则填为0，示例代码如下：

```
#chapter5\5-1\5-1-10.ipynb
import pandas as pd                                    #导入 Pandas 库,并命名为 pd
df1 = pd.read_excel('5-1-10.xlsx','1月',index_col = 0)  #读取 Excel 中的 1 月数据
df2 = pd.read_excel('5-1-10.xlsx','2月',index_col = 0)  #读取 Excel 中的 2 月数据
df1.add(df2,fill_value = 0)                            #用函数做相加运算(函数)
```

如图 5-10(a)所示为原表格,运行结果如图 5-10(b)所示。

	数学	语文
王二	100	70
张三	81	71
麻子	98	81
小曾	99	88

	语文	数学
张三	86	95
小新	77	88
王二	73	77
麻子	100	85

(a) 原表格

	数学	语文
小新	88.0	77.0
小曾	99.0	88.0
张三	176.0	157.0
王二	177.0	143.0
麻子	183.0	181.0

(b) 两个DataFrame相加后的结果

图 5-10　两个 DataFrame 相加(2)

5.2　数据分支判断

在数据处理时,判断处理必不可少,可能需要对数字进行比较判断,对字符串进行匹配判断等。判断结果会产生布尔值 True 或者 False,我们需要根据判断的结果来决定不同的数据处理方式。在 Excel 中有 if()函数可以进行判断,而在 Pandas 中主要介绍 mask()函数、where()函数和 np.where()函数对 Series、DataFrame 两种数据的判断。

5.2.1　条件判断处理 1(mask()与 where())

mask()函数与 where()函数结构相同,含义相反。mask()函数在条件成立时做处理,where()函数在条件不成立时做处理,这两个函数均可以对 Series 和 DataFrame 进行判断处理,下面以 mask()函数为例进行演示说明。

1. Series 数据条件判断

例如对 pd.Series([99,84,100,79,91])中的元素做判断,如果元素大于 90,则对元素加 1,否则不进行任何处理,示例代码如下:

```
#chapter5\5-2\5-2-1.ipynb
import pandas as pd                          #导入 Pandas 库,并命名为 pd
s = pd.Series([99,84,100,79,91])             #Series 数据
s.mask(s > 90,s + 1)                         #Series 数据中的元素如果大于 90,则加 1
```

运行结果如下:

```
0    100
1     84
2    101
3     79
4     92
dtype: int64
```

下面给出一个实际的应用案例，如图 5-11(a)所示为原表格。如果民族不是'汉'，则总分加 10 分，示例代码如下：

```
#chapter5\5-2\5-2-2.ipynb
import pandas as pd                                    #导入 Pandas 库,并命名为 pd
df = pd.read_excel('5-2-2.xlsx','成绩表')              #读取 Excel 中的成绩表数据
df['总分'].mask(df['民族']!= '汉',df['总分'] + 10,inplace = True)
                                                       #民族不是汉族,总分加 10 分
df                                                     #输出 df 表数据
```

运行结果如图 5-11(b)所示。

	姓名	民族	总分
0	张三	藏	560
1	李中	汉	470
2	王二	回	500

(a) 原表格

	姓名	民族	总分
0	张三	藏	570
1	李中	汉	470
2	王二	回	510

(b) 处理后的表格

图 5-11　对 Series 进行判断处理

2. DataFrame 数据条件判断

例如对下面 df 表中的数字进行判断，如果数字小于 60，则返回 0，否则不进行任何处理，示例代码如下：

```
#chapter5\5-2\5-2-3.ipynb
import pandas as pd                          #导入 Pandas 库,并命名为 pd
df = pd.DataFrame({
'分数 1':[89,59,92,58],
'分数 2':[52,86,71,96]
})                                           #DataFrame 数据
df.mask(df < 60,0)                           #DataFrame 数据中的元素如果小于 60,则返回 0
```

运行结果如下：

	分数 1	分数 2
0	89	0
1	0	86
2	92	71
3	0	96

同样，给出一个实际的应用案例，如图 5-12(a)所示为原表格，如果各列的评价等级为差，则返回问号(?)，示例代码如下：

```
#chapter5\5-2\5-2-4.ipynb
import pandas as pd                                    #导入 Pandas 库,并命名为 pd
df = pd.read_excel('5-2-4.xlsx','评级表')              #读取 Excel 中的评级表数据
df.iloc[:,1:] = df.iloc[:,1:].mask(df.iloc[:,1:] == '差','?')
                                                       #如果评价等级为差,则返回问号(?)
df                                                     #输出 df 表数据
```

运行结果如图 5-12(b)所示。

	产品	评价1	评价2	评价3
0	A	中	中	差
1	B	差	差	优
2	C	良	中	优
3	D	差	优	差

(a) 原表格

	产品	评价1	评价2	评价3
0	A	中	中	?
1	B	?	?	优
2	C	良	中	优
3	D	?	优	?

(b) 处理后的表格

图 5-12　对 DataFrame 进行判断处理(1)

5.2.2　条件判断处理 2(np.where())

通过 5.2.1 节的学习,我们知道 mask()函数只能在条件成立时做处理,where()函数只能在条件不成立时做处理。如果希望可以同时处理,则只能用 np.where()函数,它是 NumPy 库中的函数,所以返回值的数据类型也是数组。np.where()函数同样可以对 Series 和 DataFrame 进行判断处理。

1. Series 数据条件判断

例如对 pd.Series([80,79,97,86,93])中的数字做判断,数字大于或等于 90 时返回'优',否则返回'差',返回结果为一维数组,示例代码如下:

```
#chapter5\5-2\5-2-5.ipynb
import pandas as pd,numpy as np                    #导入 Pandas 库和 NumPy 库,并命名为 pd 和 np
s = pd.Series([80,79,97,86,93])                    #Series 数据
np.where(s>=90,'优','差')                          #值大于或等于 90 时返回优,否则返回差
```

运行结果如下:

```
array(['差', '差', '优', '差', '优'], dtype='<U1')
```

如图 5-13(a)所示为原表格,对分数进行判断。如果分数大于或等于 90,则返回'已达标',否则返回'未达标',最后将返回结果添加到新列,示例代码如下:

```
#chapter5\5-2\5-2-6.ipynb
import pandas as pd,numpy as np                    #导入 Pandas 库和 NumPy 库,并命名为 pd 和 np
df = pd.read_excel('5-2-6.xlsx','分数表')          #读取 Excel 中的分数表数据
df['是否达标'] = np.where(df['分数']>=90,'已达标','未达标')
                                                   #分数大于或等于 90 时返回'已达标',否则返回'未达标'
df                                                 #输出 df 表结果
```

运行结果如图 5-13(b)所示。

2. DataFrame 数据条件判断

例如对下面 df 表中的数字进行判断,如果数字大于或等于 80,则返回'优',否则返回'差',返回结果为二维数组,示例代码如下:

	姓名	分数
0	张三	80
1	李四	79
2	王二	97
3	麻子	86
4	小曾	93

(a) 原表格

	姓名	分数	是否达标
0	张三	80	未达标
1	李四	79	未达标
2	王二	97	已达标
3	麻子	86	未达标
4	小曾	93	已达标

(b) 处理后的表格

图 5-13　对 DataFrame 进行判断处理(2)

```
#chapter5\5-2\5-2-7.ipynb
import pandas as pd,numpy as np                    #导入 Pandas 库和 NumPy 库,并命名为 pd 和 np
df = pd.DataFrame({
'分数 1':[89,59,92,58],
'分数 2':[52,86,71,96]
})                                                 #DataFrame 数据
np.where(df >= 80,'优','差')                       #值大于或等于 90 时返回'优',否则返回'差'
```

运行结果如下：

```
array([['优', '差'],
       ['差', '优'],
       ['优', '差'],
       ['差', '优']], dtype = '<U1')
```

如图 5-14(a)所示为原表格,对业绩进行判断。如果大于或等于 40000,则返回'已达标',否则返回'未达标',示例代码如下：

```
#chapter5\5-2\5-2-8.ipynb
import pandas as pd,numpy as np                    #导入 Pandas 库和 NumPy 库,并命名为 pd 和 np
df = pd.read_excel('5-2-8.xlsx','业绩表')          #读取 Excel 中的业绩表数据
df.iloc[:,1:] = np.where(df.iloc[:,1:]>= 40000,'已达标','未达标')
                                  #业绩大于或等于 40000 时返回'已达标',否则返回'未达标'
df                                                 #输出 df 表结果
```

运行结果如图 5-14(b)所示。

	姓名	1月	2月	3月
0	张三	43000	30000	36000
1	李四	39000	26000	45000
2	王二	21000	29000	43000
3	麻子	34000	44000	20000
4	小曾	47000	30000	46000

(a) 原表格

	姓名	1月	2月	3月
0	张三	已达标	未达标	未达标
1	李四	未达标	未达标	已达标
2	王二	未达标	未达标	已达标
3	麻子	未达标	已达标	未达标
4	小曾	已达标	未达标	已达标

(b) 处理后的表格

图 5-14　对 DataFrame 进行判断处理(3)

5.3 数据遍历处理

遍历是指对一个序列中的所有元素都执行相同操作。遍历也可以通过循环语句来完成，但一般效率比较低。Excel 中的 VBA 编程就有 for 循环、do 循环等语句，当然在 Python 中也有对应的 for 循环、while 循环。在 Pandas 中，要遍历 Series 和 DataFrame 中的数据，也可以通过循环语句实现。不过，Pandas 提供了一些更高效的遍历函数。本节讲解 map()、apply()及 applymap()这 3 个最常用的遍历函数。

5.3.1 遍历 Series 元素(map())

map()函数对 Series 中的每个元素执行遍历处理，该函数的参数可以是字典，也可以是函数。

1. map()的参数为字典

Series 中有优、中、差 3 种值，如果希望'优'返回 10，'中'返回 5，'差'返回 1，则在 map()函数中写入{'优':10,'中':5,'差':1}，示例代码如下：

```
#chapter5\5-3\5-3-1.ipynb
import pandas as pd                                    #导入 Pandas 库,并命名为 pd
s = pd.Series(['优','中','优','中','差'])              #Series 数据
s.map({'优':10,'中':5,'差':1})                         #map()的参数为字典
```

运行结果如下：

```
0    10
1    5
2    10
3    5
4    1
dtype: int64
```

2. map()的参数为内置函数

如果 Series 中的每个元素都是数字，并且希望格式化每个数字，则在 map()函数中写入'{}分'.format，其中 format()函数是 Python 中的内置函数，示例代码如下：

```
#chapter5\5-3\5-3-2.ipynb
import pandas as pd                                    #导入 Pandas 库,并命名为 pd
s = pd.Series([80,79,97,86,93])                        #Series 数据
s.map('{}分'.format)                                   #map()的参数为内置函数
```

运行结果如下：

```
0    80 分
1    79 分
2    97 分
3    86 分
4    93 分
dtype: object
```

3. map()的参数为自定义函数

如果 Series 中的元素是数字，并且希望对每个数字除以 2，则需要自定义一个除以 2 的函数，然后将自定义的函数名称写入 map()函数中，示例代码如下：

```
#chapter5\5-3\5-3-3.ipynb
import pandas as pd                                    #导入 Pandas 库,并命名为 pd
s = pd.Series([80,79,97,86,93])                        #Series 数据
def fun(x):                                            #自定义 fun()函数
    return x/2
s.map(fun)                                             #map()的参数为自定义函数
```

运行结果如下：

```
0    40.0
1    39.5
2    48.5
3    43.0
4    46.5
dtype: float64
```

4. map()的参数为匿名函数

在用自定义函数作为 map()函数的参数时，如果自定义函数的处理比较简单，则可以使用匿名函数，将匿名函数写在 map()函数中即可。同样，要求将 Series 中的每个数字除以 2，示例代码如下：

```
#chapter5\5-3\5-3-4.ipynb
import pandas as pd                                    #导入 Pandas 库,并命名为 pd
s = pd.Series([80,79,97,86,93])                        #Series 数据
s.map(lambda x:x/2)                                    #map()的参数为匿名函数
```

运行结果如下：

```
0    40.0
1    39.5
2    48.5
3    43.0
4    46.5
dtype: float64
```

在 s.map(lambda x:x/2)中 x 代表获取 Series 中的每个元素，然后将其除以 2。返回的结果也是 Series 类型数据。

5.3.2　遍历 DataFrame 行和列(apply())

apply()函数可以像 map()函数一样遍历 Series 中的每个元素，但主要是使用 apply()函数来遍历 DatFrame 的行或列，遍历出来每行或每列均是 Series 数据。apply()函数也可以接收内置函数、自定义函数和匿名函数作为参数。

1. 遍历 DataFrame 的每列

如图 5-15 原表格所示，将表格中所有科目的分数按列求和，示例代码如下：

```
#chapter5\5-3\5-3-5.ipynb
import pandas as pd                                    #导入 Pandas 库,并命名为 pd
df = pd.read_excel('5-3-5.xlsx','成绩表')              #读取 Excel 中的成绩表数据
df.loc[:,'语文':].apply(sum,axis = 0)                  #按列遍历并求和
```

运行结果如下：

```
语文    480
数学    438
英语    422
dtype: int64
```

在代码 df.loc[:,'语文':].apply(sum,axis=0)中，sum 对遍历出来的每列数据(Series 数据)做求和计算，axis=1 表示按行方向遍历，axis=0 表示按列方向遍历，默认值为 0。因为求和的表格中只有 3 列数据，所以结果也是由 3 个数字组成的 Series 数据。

如果希望将计算出的结果添加到原表格下面，示例代码如下：

```
#chapter5\5-3\5-3-6.ipynb
import pandas as pd                                    #导入 Pandas 库,并命名为 pd
df = pd.read_excel('5-3-6.xlsx','成绩表')              #读取 Excel 中的成绩表数据
df.loc[len(df)] = df.loc[:,'语文':].apply(sum,axis = 0)  #按列遍历并求和
df                                                     #输出 df 表数据
```

如图 5-15(a)所示为原表格，运行结果如图 5-15(b)所示。

	姓名	语文	数学	英语
0	张三	102	128	108
1	李四	128	112	109
2	王二	111	92	101
3	麻子	139	106	104

(a) 原表格

	姓名	语文	数学	英语
0	张三	102.0	128.0	108.0
1	李四	128.0	112.0	109.0
2	王二	111.0	92.0	101.0
3	麻子	139.0	106.0	104.0
4	NaN	480.0	438.0	422.0

(b) 处理后的表格

图 5-15　对 DataFrame 按列进行遍历处理(1)

2. 遍历 DataFrame 的每行

如图 5-16 所示为原表格，将表格中所有科目的分数按行求和，也就是对每个人的各科成绩求和，示例代码如下：

```
#chapter5\5-3\5-3-7.ipynb
import pandas as pd                                    #导入 Pandas 库,并命名为 pd
df = pd.read_excel('5-3-7.xlsx','成绩表')              #读取 Excel 中的成绩表数据
df.loc[:,'语文':].apply(sum,axis = 1)                  #按行遍历并求和
```

运行结果如下：

```
0    338
1    349
2    304
3    349
dtype: int64
```

与之前遍历列的代码基本相同，唯一要修改的是 axis＝1，因为现在执行按行遍历。如果希望将计算出的结果添加到原表格后面，示例代码如下：

```
#chapter5\5-3\5-3-8.ipynb
import pandas as pd                                          #导入 Pandas 库，并命名为 pd
df = pd.read_excel('5-3-8.xlsx','成绩表')                    #读取 Excel 中的成绩表数据
df['总分'] = df.loc[:,'语文':].apply(sum,axis = 1)           #按行遍历并求和
df                                                           #输出 df 表数据
```

运行结果如图 5-16(b)所示。

	姓名	语文	数学	英语
0	张三	102	128	108
1	李四	128	112	109
2	王二	111	92	101
3	麻子	139	106	104

(a) 原表格

	姓名	语文	数学	英语	总分
0	张三	102	128	108	338
1	李四	128	112	109	349
2	王二	111	92	101	304
3	麻子	139	106	104	349

(b) 处理后的表格

图 5-16　对 DataFrame 按行进行遍历处理(2)

5.3.3　遍历 DataFrame 元素(applymap())

applymap()函数的用法比较简单，是对 DataFrame 中的每个元素执行指定函数的处理，虽然用途不如 apply()广泛，但在某些场合下还是比较有用的，例如将数字保留到小数点后两位，示例代码如下：

```
#chapter5\5-3\5-3-9.ipynb
import pandas as pd                                          #导入 Pandas 库，并命名为 pd
df = pd.read_excel('5-3-9.xlsx','成绩表')                    #读取 Excel 中的成绩表数据
df.iloc[:,1:].applymap('{:.2f}'.format).astype(float)
                                                             #遍历处理 DataFrame 表格的每个元素
```

如图 5-17(a)所示为原表格，运行结果如图 5-17(b)所示。

	姓名	语文	数学	英语
0	张三	93.794100	89.094984	96.739332
1	李四	84.762550	97.381785	92.372683
2	王二	89.638000	98.129451	84.378914
3	麻子	89.892935	82.583513	82.307083

(a) 原表格

	语文	数学	英语
0	93.79	89.09	96.74
1	84.76	97.38	92.37
2	89.64	98.13	84.38
3	89.89	82.58	82.31

(b) 分数处理后的表格

图 5-17　对 DataFrame 的每个元素进行遍历处理

值得注意的是，使用 format()函数格式化数字后，是字符串类型，所以还要在后面加上 astype(float)，转换为小数类型。

如果要将修改后的表格区域写回原表格，示例代码如下：

```
#chapter5\5-3\5-3-10.ipynb
import pandas as pd                                          #导入 Pandas 库,并命名为 pd
df = pd.read_excel('5-3-10.xlsx','成绩表')                    #读取 Excel 中的成绩表数据
df.iloc[:,1:] = df.iloc[:,1:].applymap('{:.2f}'.format).astype(float)
                                                             #遍历处理 DataFrame 表格的每个元素
df                                                           #输出 df 表数据
```

如图 5-18(a)所示为原表格，运行结果如图 5-18(b)所示。

	姓名	语文	数学	英语
0	张三	93.794100	89.094984	96.739332
1	李四	84.762550	97.381785	92.372683
2	王二	89.638000	98.129451	84.378914
3	麻子	89.892935	82.583513	82.307083

(a) 原表格

	姓名	语文	数学	英语
0	张三	93.79	89.09	96.74
1	李四	84.76	97.38	92.37
2	王二	89.64	98.13	84.38
3	麻子	89.89	82.58	82.31

(b) 处理后的表格

图 5-18　对 DataFrame 的每个元素进行遍历处理

5.4　数据统计处理

在做数据处理时，数据统计是必不可少的，所以需要学习一些常用的统计函数。例如关于聚合的统计、多个布尔值的逻辑统计，以及极值统计、排名统计等。这些统计需求在日常工作中可能比较常见。绝大多数统计需求在 Excel 中有对应的函数，在 Pandas 中这些函数功能更强大。

5.4.1　聚合统计

聚合统计是最常见的统计方式，下面列出在 Python、Pandas 和 NumPy 中常用的聚合函数，见表 5-2。

表 5-2　Python、Pandas 与 NumPy 中常用的聚合函数

统 计 方 式	Python	Pandas	NumPy
求和	sum()	sum()	np.sum()
求最大值	max()	max()	np.max()
求最小值	min()	min()	np.min()
求平均值	无	mean()	np.mean()
求计数值	len()	count()	无

上面 3 种不同聚合函数的区别在于：Python 中的聚合函数可以对一组任意形式的序列值做统计；Pandas 中的聚合函数只能对 Series 和 DataFrame 做统计；NumPy 中的聚合函数可以对数组、Series 和 DataFrame 做统计，并且 Pandas 和 NumPy 中的聚合函数在对

二维数据统计时可以指定聚合方向。

1. Series 数据统计

现在分别使用 Python、Pandas 和 NumPy 的聚合函数对 Series 数据进行统计，示例代码如下：

```
#chapter5\5-4\5-4-1.ipynb
import pandas as pd,numpy as np                    #导入 Pandas 库和 NumPy 库,并命名为 pd 和 np
s = pd.Series([1,10,100])                          #Series 数据
print(sum(s),s.sum(),np.sum(s))                    #求和
print(max(s),s.max(),np.max(s))                    #求最大值
print(min(s),s.min(),np.min(s))                    #求最小值
print(s.mean(),np.mean(s))                         #求平均值
print(len(s),s.count())                            #求计数值
```

运行结果如下：

```
111    111    111
100    100    100
1      1      1
37.0   37.0
3      3
```

2. DataFrame 数据统计

在对 DataFrame 做数据统计时，聚合函数并不是将整个 DataFrame 表格的数据计算为一个值，而是对每行或每列执行计算，最后以 Series 数据结构返回聚合结果。接下来以 sum()函数为例，分别演示按行和按列的统计方法。

1) 按行统计

如图 5-19(a)所示为原表格，对表格中的分数按行执行求和统计，在 sum()函数的参数中设置 axis=1 即可，示例代码如下：

```
#chapter5\5-4\5-4-2.ipynb
import pandas as pd,numpy as np                    #导入 Pandas 库和 NumPy 库,并命名为 pd 和 np
df = pd.read_excel('5-4-2.xlsx','成绩表')          #读取 Excel 中的成绩表数据
df.iloc[:,1:4].sum(axis=1)                         #按行求和(Pandas 函数)
np.sum(df.iloc[:,1:4],axis=1)                      #按行求和(NumPy 函数)
```

运行结果如下：

```
0    338
1    349
2    304
3    349
dtype: int64
```

上面代码运算的结果是 Series 数据，存储的是每行求和的结果，接下来将该数据添加为新列，列名为总分，示例代码如下：

```
#chapter5\5-4\5-4-3.ipynb
import pandas as pd,numpy as np                    #导入 Pandas 库和 NumPy 库,并命名为 pd 和 np
df = pd.read_excel('5-4-3.xlsx','成绩表')          #读取 Excel 中的成绩表数据
df['总分'] = df.iloc[:,1:4].sum(axis = 1)          #按行求和(Pandas 函数)
df['总分'] = np.sum(df.iloc[:,1:4],axis = 1)       #按行求和(NumPy 函数)
df                                                 #输出 df 表数据
```

运行结果如图 5-19(b)所示。

	姓名	语文	数学	英语
0	张三	102	128	108
1	李四	128	112	109
2	王二	111	92	101
3	麻子	139	106	104

(a) 原表格

	姓名	语文	数学	英语	总分
0	张三	102	128	108	338
1	李四	128	112	109	349
2	王二	111	92	101	304
3	麻子	139	106	104	349

(b) 处理后的表格

图 5-19　对 DataFrame 按行求和

2) 按列统计

如图 5-20(a)所示为原表格,对表格中的分数按列执行求和统计,在 sum()函数的参数中设置 axis=0 即可,因为 0 是默认值,所以可以忽略不写,示例代码如下:

```
#chapter5\5-4\5-4-4.ipynb
import pandas as pd,numpy as np                    #导入 Pandas 库和 NumPy 库,并命名为 pd 和 np
df = pd.read_excel('5-4-4.xlsx','成绩表')          #读取 Excel 中的成绩表数据
df.iloc[:4,1:].sum(axis = 0)                       #按列求和(Pandas 函数)
np.sum(df.iloc[:4,1:],axis = 0)                    #按列求和(NumPy 函数)
```

运行结果如下:

```
语文     480
数学     438
英语     422
dtype: int64
```

上面代码运算的结果同样也是 Series 数据,接下来将该 Series 数据添加为新行,示例代码如下:

```
#chapter5\5-4\5-4-5.ipynb
import pandas as pd,numpy as np                    #导入 Pandas 库和 NumPy 库,命名为 pd 和 np
df = pd.read_excel('5-4-5.xlsx','成绩表')          #读取 Excel 中的成绩表数据
df.loc[4] = df.iloc[:4,1:].sum(axis = 0)           #按列求和(pandas 函数)
df.loc[4] = np.sum(df.iloc[:4,1:],axis = 0)        #按列求和(NumPy 函数)
df                                                 #输出 df 表数据
```

如图 5-20(a)所示为原表格,运行结果如图 5-20(b)所示。

	姓名	语文	数学	英语
0	张三	102	128	108
1	李四	128	112	109
2	王二	111	92	101
3	麻子	139	106	104

(a) 原表格

	姓名	语文	数学	英语
0	张三	102.0	128.0	108.0
1	李四	128.0	112.0	109.0
2	王二	111.0	92.0	101.0
3	麻子	139.0	106.0	104.0
4	NaN	480.0	438.0	422.0

(b) 处理后的表格

图 5-20　对 DataFrame 按列求和

5.4.2　逻辑统计

前面学习过逻辑运算符“&”和“|”，专门应用于多个逻辑布尔值之间的运算，但如果布尔值比较多，就会产生一些问题，先看一个简单的例子，判断 Series 中的所有元素是否大于 60，分别使用 4 种方法，示例代码如下：

```
#chapter5\5-4\5-4-6.ipynb
import pandas as pd,numpy as np                    #导入 Pandas 库和 NumPy 库,命名为 pd 和 np
s = pd.Series([79,80,91])                          #Series 数据
print((s[0]>60) & (s[1]>60) & (s[2]>60))           #所有元素是否大于 60(方法 1)
print(s[0]>60 and s[1]>60 and s[2]>60)             #所有元素是否大于 60(方法 2)
print((s>60).all())                                #所有元素是否大于 60(方法 3)
print(np.all(s>60))                                #所有元素是否大于 60(方法 4)
```

运行结果如下：

```
True
True
True
True
```

分析一下上面的代码，Series 中有 3 个数字，所以方法 1 和方法 2 分别做了 3 次判断，而方法 3 和方法 4 只做了 1 次判断。假设 Series 中有更多的数字，方法 1 和方法 2 就会做更多的判断，而方法 3 和方法 4 依然只判断 1 次，原因在于使用了 all()函数。

Pandas 和 NumPy 中均有对多个布尔值进行逻辑与、逻辑或运算的函数，对应的函数见表 5-3。

表 5-3　Pandas 与 NumPy 中的逻辑与、逻辑或运算

逻辑运算方式	Pandas	NumPy	注　　释
逻辑与	all	np.all	如果 Series 中的所有布尔值为 True，则返回值为 True，否则返回值为 False
逻辑或	any	np.any	如果 Series 中有 1 个及以上布尔值为 True，则返回值为 True；如果全部为 False，则返回值为 False

1. Series 逻辑统计

首先，判断 pd.Series([79,80,91])中的元素是否大于或等于 80，如果条件成立，则返回

值为 True；如果不成立，则返回值为 False，示例代码如下：

```
#chapter5\5-4\5-4-7.ipynb
import pandas as pd,numpy as np                    #导入 Pandas 库和 NumPy 库，并命名为 pd 和 np
s = pd.Series([79,80,91])                          #Series 数据
print(s>=80)                                       #打印 Series 中的布尔值元素
```

运行结果如下：

```
0    False
1    True
2    True
dtype: bool
```

再分别使用 Pandas 和 NumPy 中的 all()函数、any()函数进行逻辑统计，示例代码如下：

```
#chapter5\5-4\5-4-8.ipynb
import pandas as pd,numpy as np                    #导入 Pandas 库和 NumPy 库，并命名为 pd 和 np
s = pd.Series([79,80,91])                          #Series 数据
print((s>=80).any(),np.any(s>=80))                 #Series 元素是布尔值的或运算
print((s>=80).all(),np.all(s>=80))                 #Series 元素是布尔值的与运算
```

运行结果如下：

```
True True
False False
```

2. DataFrame 逻辑统计

如果布尔值在 DataFrame 表格中，则可使用 all()函数和 any()函数对整个表格进行逻辑统计也是一样的，但如果要对 DataFrame 表格中的布尔值按行或接列判断统计，则需要在函数中使用 axis 参数来确定方向。

1）按行统计

如图 5-21(a)所示为原表格，在每个人的 3 个科目中，如果至少有 1 个及以上科目的分数大于或等于 130，则返回'√'，否则返回'×'，示例代码如下：

```
#chapter5\5-4\5-4-9.ipynb
import pandas as pd,numpy as np                    #导入 Pandas 库和 NumPy 库，并命名为 pd 和 np
df = pd.read_excel('5-4-9.xlsx','成绩表')          #读取 Excel 中的成绩表数据
(df.iloc[:,1:]>=130).any(axis=1)                   #按行方向做 any 的逻辑统计(方法 1)
np.any(df.iloc[:,1:]>=130,axis=1)                  #按行方向做 np.any 的逻辑统计(方法 2)
```

运行结果如下：

```
0    True
1    False
2    True
```

```
3     True
dtype: bool
```

分析一下关键代码，df.iloc[:,1:]>=130 的运行结果是一个由布尔值组成的 DataFrame 表格，any()函数中的参数 axis=1 表示按行做逻辑统计。

将统计的布尔值结果再做判断，如果条件成立，则返回'√'，否则返回'×'，最后将判断的结果写入新列，示例代码如下：

```
#chapter5\5-4\5-4-10.ipynb
import pandas as pd,numpy as np                          #导入 Pandas 库和 NumPy 库，并命名为 pd 和 np
df=pd.read_excel('5-4-10.xlsx','成绩表')                  #读取 Excel 中的成绩表数据
arr=np.where((df.iloc[:,1:]>=130).any(axis=1),'√','×')    #按行方向判断(方法 1)
arr=np.where(np.any(df.iloc[:,1:]>=130,axis=1),'√','×')   #按行方向判断(方法 2)
df['是否达标']=arr                                        #将判断结果添加到新列
df                                                       #输出 df 表数据
```

运行结果如图 5-21(b)所示。

	姓名	语文	数学	英语
0	张三	130	128	102
1	李四	110	122	128
2	王二	101	132	111
3	麻子	104	146	139

(a) 原表格

	姓名	语文	数学	英语	是否达标
0	张三	130	128	102	√
1	李四	110	122	128	×
2	王二	101	132	111	√
3	麻子	104	146	139	√

(b) 处理后的表格

图 5-21　对 DataFrame 按行进行逻辑统计

2) 按列统计

如图 5-22(a)所示为原表格，如果每个科目列的元素全部大于或等于 120，则返回'√'，否则返回'×'，示例代码如下：

```
#chapter5\5-4\5-4-11.ipynb
import pandas as pd,numpy as np                          #导入 Pandas 库和 NumPy 库，并命名为 pd 和 np
df=pd.read_excel('5-4-11.xlsx','成绩表')                  #读取 Excel 中的成绩表数据
(df.iloc[:,1:]>=120).all(axis=0)                         #按列方向做 all 的逻辑统计(方法 1)
np.all(df.iloc[:,1:]>=120,axis=0)                        #按列方向做 np.all 的逻辑统计(方法 2)
```

运行结果如下：

```
语文     False
数学      True
英语     False
dtype: bool
```

分析一下关键代码，df.iloc[:,1:]>=120 的运行结果是一个由布尔值组成的 DataFrame 表格，all()函数中的参数 axis=0 表示按列做逻辑统计。

将统计的布尔值结果再做判断，如果条件成立，则返回'√'，否则返回'×'，最后将判断

的结果写入新行，示例代码如下：

```
#chapter5\5-4\5-4-12.ipynb
import pandas as pd,numpy as np                    #导入 Pandas 库和 NumPy 库,并命名为 pd 和 np
df = pd.read_excel('5-4-12.xlsx','成绩表')          #读取 Excel 中的成绩表数据
arr = np.concatenate([np.array(['是否达标']),np.where((df.iloc[:,1:]>=120).all(axis=0),
'√','×')])                                         #按列判断(方法 1)
arr = np.concatenate([np.array(['是否达标']),np.where(np.all(df.iloc[:,1:]>=120,axis=0),
'√','×')])                                         #按列判断(方法 2)
df.loc[len(df)] = arr                              #将判断结果添加到新行
df                                                 #输出 df 表数据
```

如图 5-22(a)所示为原表格，运行结果如图 5-22(b)所示。

	姓名	语文	数学	英语
0	张三	130	128	102
1	李四	110	122	128
2	王二	101	132	111
3	麻子	104	146	139

(a) 原表格

	姓名	语文	数学	英语
0	张三	130	128	102
1	李四	110	122	128
2	王二	101	132	111
3	麻子	104	146	139
4	是否达标	×	√	×

(b) 处理后的表格

图 5-22　对 DataFrame 按列进行逻辑统计

使用 np.where()函数进行判断之后，返回的是数组结果['×''√''×']，如果直接写入新行，将会出错。因为每行是 4 个值，而数组中只有 3 个元素，所以应先使用 np.concatenate()函数再添加'是否达标'，最后 arr 变量返回的值是['是否达标''×''√''×']，这样便可以成功将数据添加到新行。

注意：上面讲解的添加到新行的方法，从数据类型的角度讲是不科学的，在没有添加新行之前，每个科目下都是纯数字；添加新行后，每个科目下既有数字，又有字符串，不利于后续对每列数据进行统计。

5.4.3　极值统计

在数据统计中，我们经常需要求最大值或者最小值，但都只能取得一个值，如果希望获取前几个最大或最小值，则需要使用极值函数，极值分为极大值与极小值。在 Excel 中求极大值的函数是 LARGE()函数，求极小值的函数是 SMALL()函数；在 Pandas 中对应的函数是 nlargest()函数和 nsmallest()函数。极大值、极小值函数对应的参数见表 5-4。

表 5-4　Series 与 DataFrame 中的极值函数

极值	函数及参数
Series 极小值	s. nsmallest(n, keep='first')
Series 极大值	s. nlargest(n, keep='first')
DataFrame 极小值	df. nsmallest(n, columns,keep='first')
DataFrame 极大值	df. nlargest(n, columns,keep='first')

下面解释一下对应的几个参数的含义。

n：指定极值个数。

column：如果求极值的是 DataFrame 表格，则要指定求极值的列，可以指定多列。

keep：参数分别有 first、last 和 all 3 个值，含义分别如下。

- **first**：优先选取原始表格里排在前面的值，即默认值。
- **last**：优先选取原始表格里排在后面的值。
- **all**：选取所有的值，即使选取的个数超过了指定的极值个数。

1. Series 极值统计

对 Series 数据做极值处理之后，返回结果也是 Series 数据，下面以 pd.Series([57,91,87,46,87,99])为例，对求极大值和极小值分别做演示。

统计前 3 个最大的数字和前 3 个最小的数字，示例代码如下：

```
#chapter5\5-4\5-4-13.ipynb
import pandas as pd                                    #导入 Pandas 库,并命名为 pd
s = pd.Series([57,91,87,46,87,99])                     #Series 数据
print(s.nlargest(3,'first'))                           #统计前 3 个最大的数字
print(s.nsmallest(3,'first'))                          #统计前 3 个最小的数字
```

运行结果如下：

```
#统计前 3 个最大的数字,结果如下所示
5    99
1    91
2    87
dtype: int64

#统计前 3 个最小的数字,结果如下所示
3    46
0    57
2    87
dtype: int64
```

接下来还是统计前 3 个最大的数字和前 3 个最小的数字，但是如果有重复值，则全部获取，示例代码如下：

```
#chapter5\5-4\5-4-14.ipynb
import pandas as pd                                    #导入 Pandas 库,并命名为 pd
s = pd.Series([57,91,87,46,87,99])                     #Series 数据
print(s.nlargest(3,'all'))                             #统计前 3 个最大的数字
print(s.nsmallest(3,'all'))                            #统计前 3 个最小的数字
```

运行结果如下：

```
#统计前 3 个最大的数字,结果如下所示
5    99
1    91
```

```
2    87
4    87
dtype: int64

#统计前3个最小的数字,结果如下所示
3    46
0    57
2    87
4    87
dtype: int64
```

注意：上面的代码演示中，没有对keep参数为'last'做演示，原因在于对Series数据进行极值统计时，虽然结果都相同，但是取值位置不同，读者可以自行测试keep参数为'last'时，索引序号与参数为'first'时的不同。

2. DataFrame极值统计

对DataFrame表格做极值处理，返回结果不是极值，而是极值对应的整条记录，最终结果也是DataFrame表格，可以对DataFrame表格的单列或多列进行极值统计，下面对极大值和极小值分别进行演示。

1）统计单列极值

如图5-23(a)所示为原表格，在理论列的分数中，统计前3个最大数字对应的记录，示例代码如下：

```
#chapter5\5-4\5-4-15.ipynb
import pandas as pd                                    #导入Pandas库,并命名为pd
df = pd.read_excel('5-4-15.xlsx','成绩表')             #读取Excel中的成绩表数据
df.nlargest(3,'理论')                                  #统计理论列中前3个最大数字对应的记录
```

运行结果如图5-23(b)所示。

	姓名	理论	实操
0	张三	99	82
1	李四	96	80
2	王二	100	97
3	麻子	89	85
4	小曾	81	100

(a) 原表格

	姓名	理论	实操
2	王二	100	97
0	张三	99	82
1	李四	96	80

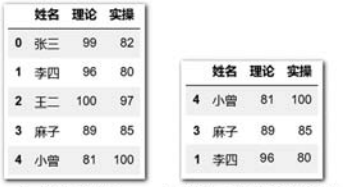
(b) 极大值统计结果

图5-23　统计理论分数最大的3条记录

如图5-24(a)所示为原表格，在理论列的分数中，统计前3个最小数字对应的记录，示例代码如下：

```
#chapter5\5-4\5-4-16.ipynb
import pandas as pd                                    #导入Pandas库,并命名为pd
df = pd.read_excel('5-4-16.xlsx','成绩表')             #读取Excel中的成绩表数据
df.nsmallest(3,'理论')                                 #统计理论列中前3个最小数字对应的记录
```

运行结果如图 5-24(b)所示。



	姓名	理论	实操
0	张三	99	82
1	李四	96	80
2	王二	100	97
3	麻子	89	85
4	小曾	81	100

(a) 原表格

	姓名	理论	实操
4	小曾	81	100
3	麻子	89	85
1	李四	96	80

(b) 极小值统计结果

图 5-24　统计理论分数最小的 3 条记录

2) 统计多列极值

对多列统计极值,在 columns 参数中以列表形式写入列名称,并且以列名称在列表中的先后顺序确定关键字顺序,例如 columns=['理论','实操'],表示以理论为第 1 关键字,以实操为第 2 关键字,也就是在理论列的数字相同的情况下,再对实操列的数字进行极值排列。

如图 5-25(a)所示为原表格,对理论和实操两列的分数进行极值统计,统计前 3 个最大数字对应的记录,示例代码如下:

```
#chapter5\5-4\5-4-17.ipynb
import pandas as pd                                  #导入 Pandas 库,并命名为 pd
df = pd.read_excel('5-4-17.xlsx','成绩表')           #读取 Excel 中的成绩表数据
df.nlargest(3,['理论','实操'])                       #统计理论、实操两列前 3 个最大值对应的记录
```

运行结果如图 5-25(b)所示。

	姓名	理论	实操
0	张三	99	100
1	李四	96	80
2	王二	100	97
3	麻子	99	85
4	小曾	81	100

(a) 原表格

	姓名	理论	实操
2	王二	100	97
0	张三	99	100
3	麻子	99	85

(b) 极大值统计结果

图 5-25　统计理论、实操分数最大的 3 条记录

如图 5-26(a)所示为原表格,对理论和实操两列的分数进行极值统计,统计前 3 个最小数字对应的记录,示例代码如下:

```
#chapter5\5-4\5-4-18.ipynb
import pandas as pd                                  #导入 Pandas 库,并命名为 pd
df = pd.read_excel('5-4-18.xlsx','成绩表')           #读取 Excel 中的成绩表数据
df.nsmallest(3,['理论','实操'])                      #统计理论、实操两列前 3 个最小值对应的记录
```

运行结果如图 5-26(b)所示。

	姓名	理论	实操
0	张三	99	100
1	李四	96	
2	王二	100	97
3	麻子	99	85
4	小曾	81	100

(a) 原表格

	姓名	理论	实操
4	小曾	81	100
1	李四	96	80
3	麻子	99	85

(b) 极小值统计结果

图 5-26　统计理论、实操分数最小的 3 条记录

5.4.4　排名统计

排名是同类事物客观实力的反映，相互之间带有比较性质，所以排名是很常见的统计方式。Excel 中的排名函数是 RANK()，Pandas 中的排名函数是 rank()，并且有更多的排名方式，先举一个简单的例子，示例代码如下：

```
#chapter5\5-4\5-4-19.ipynb
import pandas as pd                          #导入 Pandas 库，并命名为 pd
s = pd.Series([84,99,93,65])                 #Series 数据
s.rank()                                     #排名统计
```

运行结果如下：

```
0    2.0
1    4.0
2    3.0
3    1.0
dtype: float64
```

通过运行结果会发现，数字越大，排名越低，这是因为 rank()函数的 ascending 参数默认值为 True，也就是默认为升序排列。大多数排名统计是数字越大，排名越高，如果希望这样，则应将 ascending 设置为 False，示例代码如下：

```
#chapter5\5-4\5-4-20.ipynb
import pandas as pd                          #导入 Pandas 库，并命名为 pd
s = pd.Series([84,99,93,65])                 #Series 数据
s.rank(ascending = False)                    #排名统计
```

运行结果如下：

```
0    3.0
1    1.0
2    2.0
3    4.0
dtype: float64
```

关于排名，还有一个重要的问题需要了解，就是相同值。rank()函数的 method 参数提

供了对相同值的 5 种处理方法，见表 5-5。

表 5-5　rank()函数的 method 参数说明

常　量	注　释
average	默认值，对相同值做平均排名
min	对相同值做最小排名
max	对相同值做最大排名
first	对相同值出现的顺序做排名
dense	与最小排名类似，但不同名次之间差值为 1

下面对每种常量做一个案例演示，以增强理解。假定 pd.Series([99,93,84,84,65])是一组考试分数，现在需要对其中的每个分数做排名。

当 method 参数为 average 时，示例代码如下：

```
#chapter5\5-4\5-4-21.ipynb
import pandas as pd                                    #导入 Pandas 库，并命名为 pd
s = pd.Series([99,93,84,84,65])                        #Series 数据
s.rank(method = 'average',ascending = False)           #method 参数为 average 的排名统计
```

运行结果如图 5-27 所示。

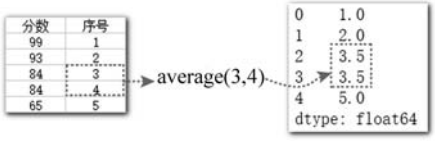


图 5-27　method 参数为 average

当 method 参数为 min 时，这种排名方式通常叫作美式排名，是应用得比较多的一种方式，示例代码如下：

```
#chapter5\5-4\5-4-22.ipynb
import pandas as pd                                    #导入 Pandas 库，并命名为 pd
s = pd.Series([99,93,84,84,65])                        #Series 数据
s.rank(method = 'min',ascending = False)               #method 参数为 min 的排名统计
```

运行结果如图 5-28 所示。

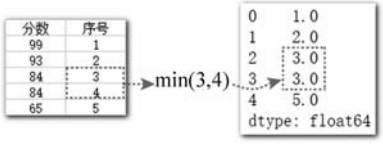


图 5-28　method 参数为 min

当 method 参数为 max 时，示例代码如下：

```
#chapter5\5-4\5-4-23.ipynb
import pandas as pd                                    #导入 Pandas 库,并命名为 pd
s = pd.Series([99,93,84,84,65])                        #Series 数据
s.rank(method = 'max',ascending = False)               #method 参数为 max 的排名统计
```

运行结果如图 5-29 所示。

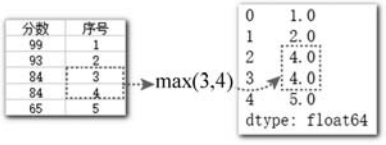


图 5-29　method 参数为 max

当 method 参数为 first 时,示例代码如下:

```
#chapter5\5-4\5-4-24.ipynb
import pandas as pd                                    #导入 Pandas 库,并命名为 pd
s = pd.Series([99,93,84,84,65])                        #Series 数据
s.rank(method = 'first',ascending = False)             #method 参数为 first 的排名统计
```

运行结果如图 5-30 所示。

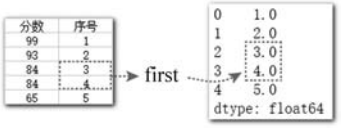


图 5-30　method 参数为 first

当 method 参数为 dense 时,这种排名方式通常叫作中式排名,也是应用得比较多的一种方式,示例代码如下:

```
#chapter5\5-4\5-4-25.ipynb
import pandas as pd                                    #导入 Pandas 库,并命名为 pd
s = pd.Series([99,93,84,84,65])                        #Series 数据
s.rank(method = 'dense',ascending = False)             #method 参数为 dense 的排名统计
```

运行结果如图 5-31 所示。

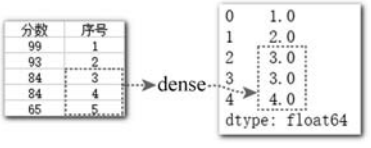


图 5-31　method 参数为 dense

1. Series 排名统计

直接获取表格的某列做排名统计,就相当于对指定的 Series 数据做排名,如图 5-32(a)

所示为原表格，对分数列分别做美式和中式排名，示例代码如下：

```
#chapter5\5-4\5-4-26.ipynb
import pandas as pd                                          #导入 Pandas 库,并命名为 pd
df = pd.read_excel('5-4-26.xlsx','分数表')                    #读取 Excel 中的分数表数据
df['美式排名'] = df['分数'].rank(method = 'min',ascending = False)      #美式排名
df['中式排名'] = df['分数'].rank(method = 'dense',ascending = False)    #中式排名
df                                                           #输出 df 表数据
```

如图 5-32(a)所示为原表格，运行结果如图 5-32(b)所示。

	姓名	分数
0	张三	90
1	李四	98
2	王二	90
3	麻子	95
4	小曾	86

(a) 原表格

	姓名	分数	美式排名	中式排名
0	张三	90	3.0	3.0
1	李四	98	1.0	1.0
2	王二	90	3.0	3.0
3	麻子	95	2.0	2.0
4	小曾	86	5.0	4.0

(b) 增加排名后的表格

图 5-32　单列的美式和中式排名

注意：rank()函数统计出的名次是小数类型，如果需要转换为整数类型，则在 rank()函数之后加 astype('int')即可。

2. DataFrame 排名统计

如果需要对 DataFrame 表格做排名，并且排名方式是各行、各列单独排名，用 rank()函数也可以实现，不过需要指定函数的 axis 参数，1 为按行排名，0 为按列排名。

1) 按行排名

如图 5-33(a)所示为原表格，需要对 1～6 月的销量数据按行排名，也就是对每个人 1～6 月的销量做排名，示例代码如下：

```
#chapter5\5-4\5-4-27.ipynb
import pandas as pd                                          #导入 Pandas 库,并命名为 pd
df = pd.read_excel('5-4-27.xlsx','销量表')                    #读取 Excel 中的销量表数据
df.iloc[:,1:] = df.iloc[:,1:].rank(method = 'min',ascending = False,axis = 1).astype('int')
                                                             #按行做美式排名
df                                                           #输出 df 表数据
```

运行结果如图 5-33(b)所示。

	姓名	1月	2月	3月	4月	5月	6月
0	张三	91	52	54	50	98	83
1	李四	104	66	70	87	76	80
2	王二	80	70	114	67	64	104
3	麻子	56	84	51	74	108	112

(a) 原表格

	姓名	1月	2月	3月	4月	5月	6月
0	张三	2	5	4	6	1	3
1	李四	1	6	5	2	4	3
2	王二	3	4	1	5	6	2
3	麻子	5	3	6	4	2	1

(b) 按行排名后的表格

图 5-33　按行进行美式排名

2）按列排名

如图 5-34(a)所示为原表格，如果需要按列排名，则可统计所有人在每个月的排名情况，示例代码如下：

```
#chapter5\5-4\5-4-28.ipynb
import pandas as pd                                          #导入 Pandas 库,并命名为 pd
df = pd.read_excel('5-4-28.xlsx','销量表')                    #读取 Excel 中的销量表数据
df.iloc[:,1:] = df.iloc[:,1:].rank(method = 'min',ascending = False,axis = 0).astype('int')
                                                             #按列进行美式排名
df                                                           #输出 df 表数据
```

运行结果如图 5-34(b)所示。

	姓名	1月	2月	3月	4月	5月	6月
0	张三	91	52	54	50	98	83
1	李四	104	66	70	87	76	80
2	王二	80	70	114	67	64	104
3	麻子	56	84	51	74	108	112

(a) 原表格

	姓名	1月	2月	3月	4月	5月	6月
0	张三	2	4	3	4	2	3
1	李四	1	3	2	1	3	4
2	王二	3	2	1	3	4	2
3	麻子	4	1	4	2	1	1

(b) 按列排名后的表格

图 5-34　按列进行美式排名

5.5　巩固案例

本章通过对运算、分支、遍历和统计知识点的学习，对 Pandas 的数据处理方式有了更深刻的理解，接下来本节通过实例应用对所学知识加以巩固，以达到学以致用的目的。

5.5.1　根据不同蔬菜的采购数量统计每天采购金额

如图 5-35(a)所示为原表格，分别有净白菜、小油菜和丝瓜这 3 种蔬菜的采购数量，现在给出这 3 种蔬菜每千克的价格分别为 1.3、2.4 和 5.6 元，各蔬菜的单价数据存储在 s=pd.Series([5.6,1.3,2.4],index=['丝瓜','净白菜','小油菜'])中，并且存储顺序并没有按原表格中的蔬菜顺序排列，现在需要统计出每天的采购金额，示例代码如下：

```
#chapter5\5-5\5-5-1.ipynb
import pandas as pd                                          #导入 Pandas 库,并命名为 pd
df = pd.read_excel('5-5-1.xlsx','采购表')                     #读取 Excel 中的采购表数据
s = pd.Series([5.6,1.3,2.4],index = ['丝瓜','净白菜','小油菜'])
                                                             #用 Series 数据结构存储蔬菜价格
df['金额'] = (df.iloc[:,1:] * s).sum(axis = 1)                #统计每天采购金额
df                                                           #输出 df 表数据
```

运行结果如图 5-35(b)所示。

分析一下关键代码，df['金额']=(df.iloc[:,1:] * s).sum(axis=1)，其中 df.iloc[:,1:]选择的是所有蔬菜的采购数量，返回的是 DataFrame 数据，然后与 s 相乘，s 是 Series 数据，本质就是 DataFrame 与 Series 的运算。可以参考 5.1.6 节 DataFrame 与 Series 运算的讲

	日期	净白菜	小油菜	丝瓜
0	2021-04-01	10	2	5
1	2021-04-02	8	1	4
2	2021-04-03	3	0	6
3	2021-04-04	7	3	15

(a) 原表格

	日期	净白菜	小油菜	丝瓜	金额
0	2021-04-01	10	2	5	45.8
1	2021-04-02	8	1	4	35.2
2	2021-04-03	3	0	6	37.5
3	2021-04-04	7	3	15	100.3

(b) 添加金额列后的表格

图 5-35　计算每天金额的前后效果对比

解，返回结果也是 DataFrame 数据，运算结果如下：

```
    丝瓜    净白菜    小油菜
0   28.0    13.0      4.8
1   22.4    10.4      2.4
2   33.6    3.9       0.0
3   84.0    9.1       7.2
```

之后 sum(axis=1)表示按行求和，运算结果如下：

```
0    45.8
1    35.2
2    37.5
3    100.3
dtype: float64
```

最后将计算结果写入原表格的新列中即可。

如果蔬菜的价格顺序与原表格蔬菜的顺序相同，也可以将价格存储在列表中，与蔬菜采购数量直接相乘即可，示例代码如下：

```
#chapter5\5-5\5-5-2.ipynb
import pandas as pd                                          #导入 Pandas 库，并命名为 pd
df = pd.read_excel('5-5-2.xlsx','采购表')                    #读取 Excel 中的采购表数据
df['金额'] = (df.iloc[:,1:] * [1.3,2.4,5.6]).sum(axis = 1)   #统计每天采购金额
df                                                           #输出 df 表数据
```

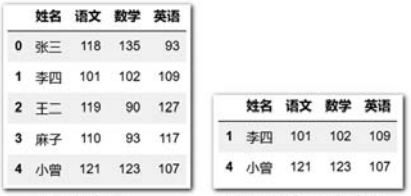
运行结果与图 5-35(b)添加金额列之后的表格相同。

5.5.2　筛选出成绩表中各科目均大于或等于 100 的记录

如图 5-36(a)所示为原表格，要求筛选出每个人 3 个科目的分数均大于或等于 100 的记录，示例代码如下：

```
#chapter5\5-5\5-5-3.ipynb
import pandas as pd,numpy as np                  #导入 Pandas 库和 NumPy 库，并命名为 pd 和 np
df = pd.read_excel('5-5-3.xlsx','成绩表')        #读取 Excel 中的成绩表数据
df[np.all(df.iloc[:,1:]>=100,axis=1)]            #输出筛选结果
```

运行结果如图 5-36(b)所示。



	姓名	语文	数学	英语
0	张三	118	135	93
1	李四	101	102	109
2	王二	119	90	127
3	麻子	110	93	117
4	小曾	121	123	107

(a) 原表格

	姓名	语文	数学	英语
1	李四	101	102	109
4	小曾	121	123	107

(b) 筛选后的表格

图 5-36　成绩筛选的前后效果对比(1)

分析一下关键代码，np.all(df.iloc[:,1:]>=100,axis=1)，其中 df.iloc[:,1:]>=100 表示判断所有科目的分数是否大于或等于 100，返回的结果是由布尔值构成的 DataFrame 数据，结果如下：

```
    语文    数学    英语
0   True    True    False
1   True    True    True
2   True    False   True
3   True    False   True
4   True    True    True
```

然后使用 an.all()函数对判断的结果按行做逻辑与运算，最后返回结果是由布尔值组成的 Series 数据，结果如下：

```
0    False
1    True
2    False
3    False
4    True
dtype: bool
```

观察发现，只有索引为 1 和 4 的返回结果为 True，所以最后筛选出行索引为 1 和 4 对应的两条记录。

5.5.3　筛选出成绩表中各科目的和大于或等于 300 的记录

如图 5-37(a)所示为原表格，对每个人的各科分数求和，然后筛选出总分大于或等于 300 的记录，示例代码如下：

```
#chapter5\5-5\5-5-4.ipynb
import pandas as pd                                    #导入 Pandas 库，并命名为 pd
df = pd.read_excel('5-5-4.xlsx','成绩表')              #读取 Excel 中的成绩表数据
df[df.iloc[:,1:].sum(axis=1)>=330]                     #输出筛选结果
```

运行结果如图 5-37(b)所示。

分析一下关键代码，df.iloc[:,1:].sum(axis=1)>=330，其中 df.iloc[:,1:]表示所有科目的分数，sum(axis=1)表示按行求和，求和结果是 Series 数据，结果如下：

	姓名	语文	数学	英语
0	张三	118	135	93
1	李四	101	102	109
2	王二	119	90	127
3	麻子	110	93	117
4	小曾	121	123	107

(a) 原表格

	姓名	语文	数学	英语
0	张三	118	135	93
2	王二	119	90	127
4	小曾	121	123	107

(b) 筛选后的表格

图 5-37　成绩筛选的前后效果对比(2)

```
0    346
1    312
2    336
3    320
4    351
dtype: int64
```

Series 数据存放的是每个人的总分,再判断是否大于或等于 330,结果如下:

```
0     True
1    False
2     True
3    False
4     True
dtype: bool
```

观察发现,索引为 0、2、4 的返回结果为 True,所以最后筛选出行索引为 0、2、4 对应的 3 条记录。

5.5.4　统计每个人各科目总分之和的排名

如图 5-38(a)所示为原表格,对每个人的各科分数求和,然后按总分的大小做中式排名,示例代码如下:

```
#chapter5\5-5\5-5-5.ipynb
import pandas as pd                                          #导入 Pandas 库,并命名为 pd
df = pd.read_excel('5-5-5.xlsx','成绩表')                    #读取 Excel 中的成绩表数据
df['排名'] = df.iloc[:,1:].sum(axis = 1).rank(method = 'dense',ascending = False).astype('int')
                                                             #中式排名统计
df                                                           #输出 df 表数据
```

运行结果如图 5-38(b)所示。

分析一下关键代码,df.iloc[:,1:].sum(axis=1).rank(method='dense',ascending=False).astype('int'),其中 df.iloc[:,1:].sum(axis=1)表示对所有科目的分数按行求和,求和结果是 Series 数据,存放的是每个人的总分,结果如下:

	姓名	语文	数学	英语
0	张三	118	135	93
1	李四	101	102	109
2	王二	119	90	127
3	麻子	110	93	117
4	小曾	121	123	107

(a) 原表格

	姓名	语文	数学	英语	排名
0	张三	118	135	93	2
1	李四	101	102	109	5
2	王二	119	90	127	3
3	麻子	110	93	117	4
4	小曾	121	123	107	1

(b) 添加列后的表格

图 5-38　排名设置的前后效果对比

```
0    346
1    312
2    336
3    320
4    351
dtype: int64
```

rank(method='dense',ascending=False)表示对求和结果做排名统计,astype('int')表示将名次转换为整数类型。

5.5.5　统计每个人所有考试科目的最优科目

如图 5-39(a)所示为原表格,统计每个人最高分对应的科目,示例代码如下:

```
#chapter5\5-5\5-5-6.ipynb
import pandas as pd                                          #导入 Pandas 库,命名为 pd
df = pd.read_excel('5-5-6.xlsx','成绩表')                    #读取 Excel 中的成绩表数据
df['最优科目'] = df.iloc[:,1:].apply(lambda s:s.nlargest(1,keep = 'all').index.to_list(),
axis = 1)                                                    #统计每个人最高分对应的科目
df                                                           #输出 df 表数据
```

运行结果如图 5-39(b)所示。

	姓名	语文	数学	英语
0	张三	118	135	93
1	李四	101	102	109
2	王二	119	90	127
3	麻子	110	93	117
4	小曾	121	123	107

(a) 原表格

	姓名	语文	数学	英语	最优科目
0	张三	118	135	93	[数学]
1	李四	101	102	109	[英语]
2	王二	119	90	127	[英语]
3	麻子	110	93	117	[英语]
4	小曾	121	123	107	[数学]

(b) 添加列后的表格

图 5-39　最优科目统计的前后效果对比

分析一下关键代码,df['最优科目']=df.iloc[:,1:].apply(lambda s:s.nlargest(1,keep='all').index.to_list(),axis=1),使用 apply()函数对所有科目的分数区域按行遍历,s 变量就是从表格每行遍历出来的 Series 数据,然后统计该 Series 数据第 1 个极大值,要知道统计出来的极大值不是一个单纯的值,而是存储在 Series 数据结构中的,然后在 nlargest()函数后写入 index 属性,表示获取每个极大值 Series 的索引,结果如下:

```
0    Index(['数学'], dtype = 'object')
1    Index(['英语'], dtype = 'object')
2    Index(['英语'], dtype = 'object')
3    Index(['英语'], dtype = 'object')
4    Index(['数学'], dtype = 'object')
dtype: object
```

最后，to_list()函数表示将极大值的索引标签转换为列表，也就是最高分对应的科目名称，结果如下：

```
0    [数学]
1    [英语]
2    [英语]
3    [英语]
4    [数学]
dtype: object
```

如果有多个科目获得了最高分，同样会统计出多个最高分对应的科目名称。