第3章 数据分析工具Pandas

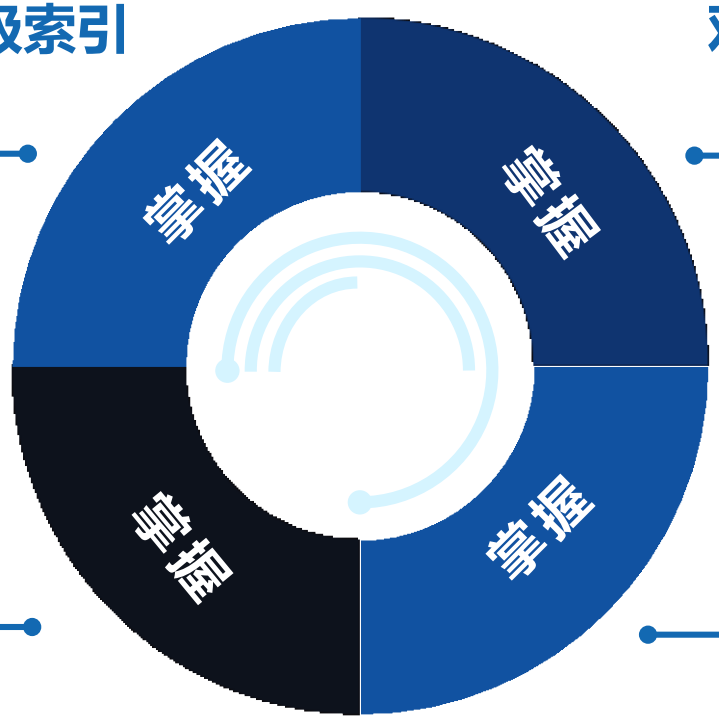
python™

- 数据结构分析
- 索引操作及高级索引
- 数据排序
- 统计计算与描述
- 层次化索引
- 读写数据操作

学习目标

1 掌握 数据结构分析，索引操作及高级索引

2 掌握 算术运算与数据对齐，数据排序



3 掌握 统计计算与描述，层次化索引

4 掌握 读写数据操作

目录页

过渡页

01 Pandas的数据结构分析

02 Pandas索引操作及高级索引

03 算术运算与数据对齐

04 数据排序

05 统计计算与描述

06 层次化索引

Series

Pandas中有两个主要的数据结构：Series和DataFrame。

一维的数据结构。	二维的、表格型的数据结构。
Series	DataFrame

Series

Series是一个类似一维数组的对象，它能够保存任何类型的数据，主要由一组数据和与之相关的索引两部分构成。

Series

index	element
0	1
1	2
2	3
3	4
4	5

- 注意：

Series的索引位于左边，数据位于右边。

Series

Pandas的Series类对象可以使用以下构造方法创建：

```
class pandas.Series（data = None， index = None， dtype = None，
name = None， copy = False， fastpath = False）
```

- data：表示传入的数据。
- index：表示索引，唯一且与数据长度相等，默认会自动创建一个从0~N的整数索引。

Series

通过传入一个列表来创建一个Series类对象：

```
# 创建Series类对象
ser_obj = pd.Series([1, 2, 3, 4, 5])
```

```
# 创建Series类对象，并指定索引
ser_obj = pd.Series([1, 2, 3, 4, 5],
                    index=['a', 'b', 'c', 'd', 'e'])
```

Series

除了使用列表构建Series类对象外，还可以使用dict进行构建。

```
year_data = {2001: 17.8, 2002: 20.1, 2003: 16.5}
ser_obj2 = pd.Series(year_data)
```

Series

为了能方便地操作Series对象中的索引和数据，所以该对象提供了两个属性index和values分别进行获取。

```
# 获取ser_obj的索引
ser_obj.index
# 获取ser_obj的数据
ser_obj.values
```

Series

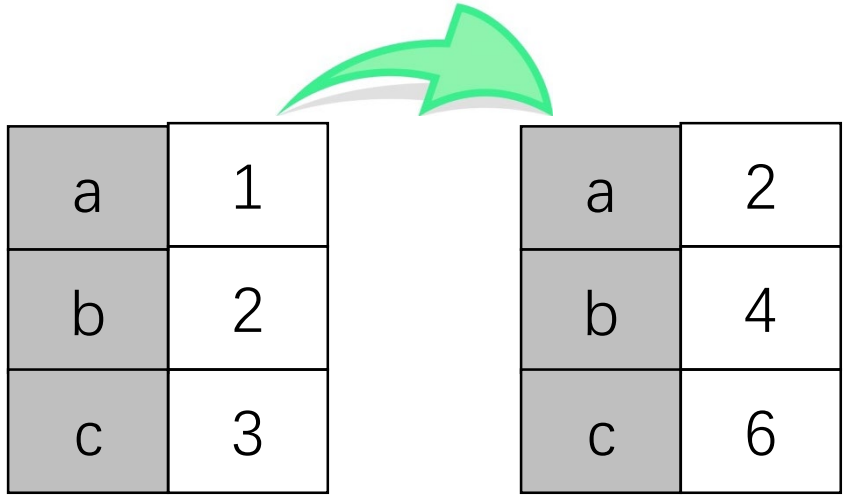
当然，我们也可以直接使用索引来获取数据。

```
# 获取位置索引3对应的数据
ser_obj[3]
```

Series

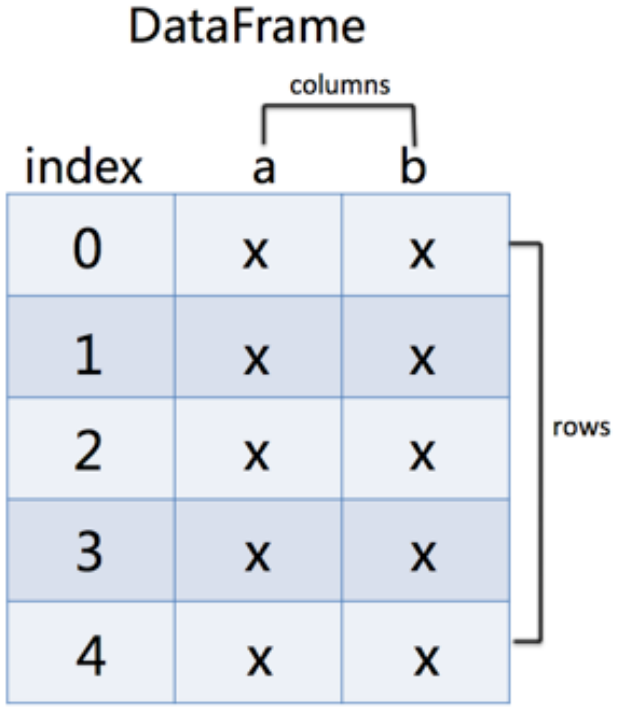
当某个索引对应的数据进行运算以后，其运算的结果会替换原数据，仍然与这个索引保持着对应的关系。

```
ser_obj * 2
```



DataFrame

DataFrame是一个类似于二维数组或表格（如excel）的对象，它每列的数据可以是不同的数据类型。



- 注意：

DataFrame的索引不仅有行索引，还有列索引，数据可以有多列。

DataFrame

Pandas的DataFrame类对象可以使用以下构造方法创建：

```
pandas.DataFrame（data = None， index = None， columns = None，
dtype = None， copy = False ）
```

- index：表示行标签。若不设置该参数，则默认会自动创建一个从0~N的整数索引。
- columns：列标签。

DataFrame

通过传入数组来创建DataFrame类对象：

```
# 创建数组
demo_arr = np.array([['a', 'b', 'c'],
                     ['d', 'e', 'f']])
# 基于数组创建DataFrame对象
df_obj = pd.DataFrame(demo_arr)
```

DataFrame

在创建DataFrame类对象时，如果为其指定了列索引，则DataFrame的列会按照指定索引的顺序进行排列。

```
df_obj = pd.DataFrame(demo_arr,
columns=['No1', 'No2', 'No3'])
```

	No1	No2	No3
0	a	b	c
1	d	e	f

DataFrame

我们可以使用列索引的方式来获取一列数据，返回的结果是一个Series对象。

```
# 通过列索引的方式获取一列数据
element = df_obj['No2']
# 查看返回结果的类型
type(element)
```

pandas.core.series.Series

DataFrame

我们还可以使用访问属性的方式来获取一列数据，返回的结果是一个Series对象。

```
# 通过属性获取列数据
element = df_obj.No2
# 查看返回结果的类型
type(element)
```

pandas.core.series.Series

DataFrame

在获取DataFrame的一列数据时，推荐使用列索引的方式完成，主要是因为在实际使用中，列索引的名称中很有可能带有一些特殊字符（如空格），这时使用“点字符”进行访问就显得不太合适了。

DataFrame

要想为DataFrame增加一列数据，则可以通过给列索引或者列名称赋值的方式实现。

```
# 增加No4一列数据
df_obj['No4'] = ['g', 'h']
```

DataFrame

要想删除某一列数据，则可以使用del语句实现。

```
# 删除No3一列数据
del df_obj['No3']
```

过渡页

01 Pandas的数据结构分析

02 Pandas索引操作及高级索引

03 算术运算与数据对齐

04 数据排序

05 统计计算与描述

06 层次化索引

索引对象

Pandas中的索引都是Index类对象，又称为索引对象，该对象是不可以进行修改的，以保障数据的安全。

```
ser_index['2'] = 'cc'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-2-51beac9b8a3e> in <module>()
----> 1 ser_index['2'] = 'cc'

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in __s
etitem__(self, key, value)
   2048
   2049     def __setitem__(self, key, value):
-> 2050         raise TypeError("Index does not support mutable operations")
   2051
   2052     def __getitem__(self, key):

TypeError: Index does not support mutable operations
```

索引对象

Pandas还提供了很多Index的子类，常见的有如下几种：

（1）Int64Index：针对整数的特殊Index对象。

（2）MultiIndex：层次化索引，表示单个轴上的多层索引。

（3）DatetimeIndex：存储纳秒寄时间戳。

重置索引

Pandas中提供了一个重要的方法是reindex()，该方法的作用是对原索引和新索引进行匹配，也就是说，新索引含有原索引的数据，而原索引数据按照新索引排序。

如果新索引中没有原索引数据，那么程序不仅不会报错，而且会添加新的索引，并将值填充为NaN或者使用fill_vlues()填充其他值。

重置索引

reindex()方法的语法格式如下：

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

- index：用作索引的新序列。
- method：插值填充方式。
- fill_value：引入缺失值时使用的替代值。
- limit：前向或者后向填充时的最大填充量。

重置索引

如果不想填充为NaN，则可以使用fill_value参数来指定缺失值。

```
ser_obj.reindex(['a', 'b', 'c', 'd', 'e', 'f'],
                fill_value = 6)
```

重置索引

如果期望使用相邻的元素值进行填充，则可以使用method参数，该参数对应的值有多个。

参数	说明
ffill 或 pad	前向填充值
bfill 或 backfill	后向填充值
nearest	从最近的索引值填充

索引操作

Series有关索引的用法类似于NumPy数组的索引，只不过Series的索引值不只是整数。如果我们希望获取某个数据，既可以通过索引的位置来获取，也可以使用索引名称来获取。

```
ser_obj = pd.Series([1, 2, 3, 4, 5], index=['a', 'b', 'c', 'd', 'e'])
ser_obj[2]    # 使用索引位置获取数据
ser_obj['c']   # 使用索引名称获取数据
```

索引操作

如果使用的是位置索引进行切片，则切片结果是不包含结束位置；如果使用索引名称进行切片，则切片结果是包含结束位置的。

```
ser_obj[2: 4]          # 使用位置索引进行切片
ser_obj['c': 'e']      # 使用索引名称进行切片
```

索引操作

如果希望获取的是不连续的数据，则可以通过不连续索引来实现。

```
# 通过不连续位置索引获取数据集
ser_obj[[0, 2, 4]]
# 通过不连续索引名称获取数据集
ser_obj[['a', 'c', 'd']]
```

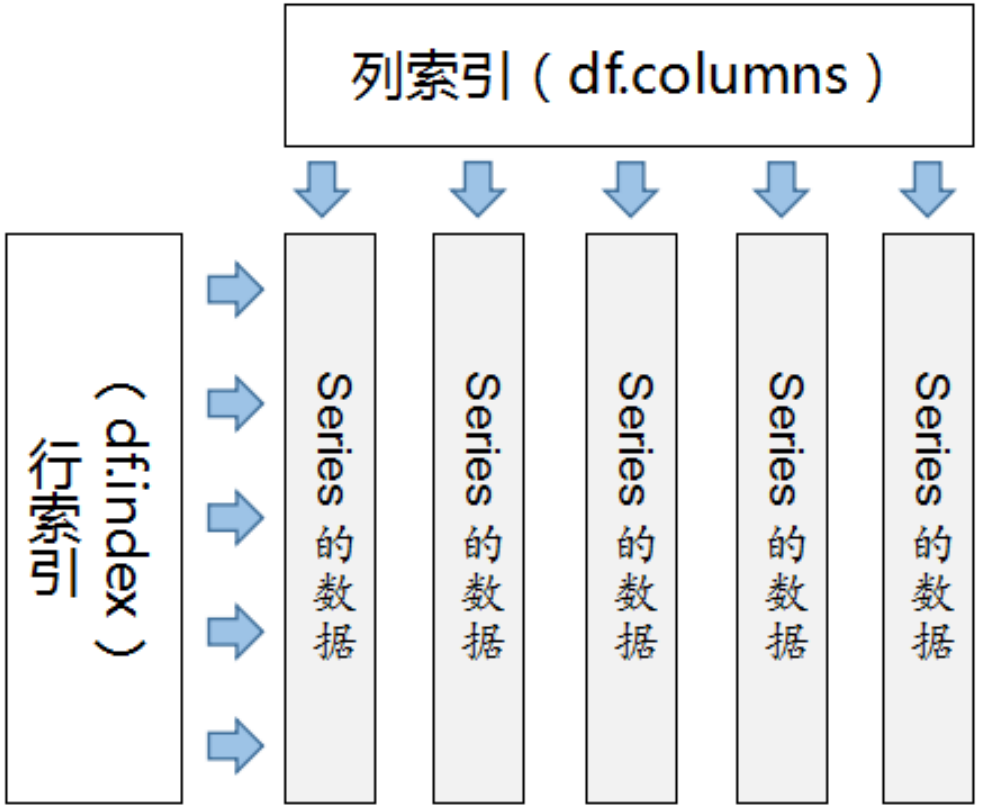
索引操作

布尔型索引同样适用于Pandas，具体的用法跟数组的用法一样，将布尔型的数组索引作为模板筛选数据，返回与模板中True位置对应的元素。

```
# 创建布尔型Series对象
ser_bool = ser_obj > 2
# 获取结果为True的数据
ser_obj[ser_bool]
```

索引操作

DataFrame结构既包含行索引，也包含列索引。其中，行索引是通过index属性进行获取的，列索引是通过columns属性进行获取的。



索引操作

虽然DataFrame操作索引能够满足基本数据查看请求，但是仍然不够灵活。为此，Pandas库中提供了操作索引的方法来访问数据，具体包括：

- loc：基于标签索引（索引名称），用于按标签选取数据。当执行切片操作时，既包含起始索引，也包含结束索引。
- iloc：基于位置索引（整数索引），用于按位置选取数据。当执行切片操作时，只包含起始索引，不包含结束索引。

过渡页

算术运算与数据对齐

Pandas执行算术运算时，会先按照索引进行对齐，对齐以后再进行相应的运算，没有对齐的位置会用NaN进行补齐。

0	10
1	11
2	12

0	20
1	21
2	22
3	23
4	24

0	30.0
1	32.0
2	34.0
3	NaN
4	NaN

算术运算与数据对齐

如果希望不使用NAN填充缺失数据，则可以在调用add方法时提供fill_value参数的值，fill_value将会使用对象中存在的数据进行补充。

```
# 执行加法运算，补充缺失值
obj_one.add(obj_two, fill_value = 0)
```

过渡页

01 Pandas的数据结构分析

02 Pandas索引操作及高级索引

03 算术运算与数据对齐

04 数据排序

05 统计计算与描述

06 层次化索引

按索引排序

Pandas中按索引排序使用的是sort_index()方法，该方法可以用行索引或者列索引进行排序。

```
sort_index（axis = 0， level = None， ascending = True，
inplace = False， kind =' quicksort '， na_position ='last'，
sort_remaining = True ）
```

- axis：轴索引，0表示index（按行），1表示columns（按列）。
- level：若不为None，则对指定索引级别的值进行排序。
- ascending：是否升序排列，默认为True表示升序。

按索引排序

按索引对Series进行分别排序，示例如下。

```
ser_obj = pd.Series(range(10, 15), index=[5, 3, 1, 3, 2])
# 按索引进行升序排列
ser_obj.sort_index()
# 按索引进行降序排列
ser_obj.sort_index(ascending = False)
```

按索引排序

按索引对DataFrame进行分别排序，示例如下。

```
df_obj = pd.DataFrame(np.arange(9).reshape(3, 3),
                      index=[4, 3, 5])
# 按行索引升序排列
df_obj.sort_index()
# 按行索引降序排列
df_obj.sort_index(ascending=False)
```

按值排序

Pandas中用来按值排序的方法为sort_values()，该方法的语法格式如下。

```
sort_values(by,axis=0, ascending=True, inplace=False, kind='
quicksort',na_position='last')
```

by参数表示排序的列，na_position参数只有两个值：first和last，若设为first，则会将NaN值放在开头；若设为False，则会将NaN值放在最后。

按值排序

按值的大小对Series进行排序的示例如下：

```
ser_obj = pd.Series([4, np.nan, 6, np.nan, -3, 2])
# 按值升序排列
ser_obj.sort_values()
```

按值排序

在DataFrame中，sort_values()方法可以根据一个或多个列中的值进行排序，但是需要在排序时，将一个或多个列的索引传递给by参数才行。

```
df_obj = pd.DataFrame([[0.4, -0.1, -0.3, 0.0],
                       [0.2, 0.6, -0.1, -0.7],
                       [0.8, 0.6, -0.5, 0.1]])
# 对列索引值为2的数据进行排序
df_obj.sort_values(by=2)
```

过渡页

常用的统计计算

Pandas为我们提供了非常多的描述性统计分析的指标方法，比如总和、均值、最小值、最大值等。

函数名称	说明
sum	计算和
mean	计算平均值
median	获取中位数
max、min	获取最大值和最小值
idxmax、idxmin	获取最大和最小索引值
count	计算非 NaN 值的个数
head	获取前 N 个值
var	样本值的方差
std	样本值的标准差
skew	样本值的偏度（三阶矩）
kurt	样本值的峰度（四阶矩）
cumsum	样本值的累积和
cummin、cummax	样本值的累积最小值和累积最大值
cumprod	样本值的累计积
describe	对 Series 和 DataFrame 列计算汇总统计

统计描述

如果希望一次性输出多个统计指标，则我们可以调用describe()方法实现，语法格式如下。

```
describe(percentiles=None, include=None, exclude=None)
```

- percentiles：输出中包含的百分数，位于[0,1]之间。如果不设置该参数，则默认为[0.25,0.5,0.75]，返回25%，50%，75%分位数。

过渡页

01 Pandas的数据结构分析

02 Pandas索引操作及高级索引

03 算术运算与数据对齐

04 数据排序

05 统计计算与描述

06 层次化索引

认识层次化索引

思考：

什么是层次化索引？

认识层次化索引

结 论

前面所涉及的Pandas对象都只有一层索引结构，又称为单层索引，层次化索引可以理解为单层索引的延伸，即**在一个轴方向上具有多层索引**。

认识层次化索引

对于两层索引结构来说，它可以分为内层索引和外层索引。

单位：平方公里

河北省	石家庄市	15848
	唐山市	13472
	邯郸市	12073.8
	秦皇岛市	7813
河南省	郑州市	7446
	开封市	6444
	洛阳市	15230
	新乡市	8269

认识层次化索引

Series和DataFrame均可以实现层次化索引，最常见的方式是在构造方法的index参数中传入一个嵌套列表。

```
mulitindex_series = pd.Series([15848,13472,12073.8,7813,
                               7446,6444,15230,8269],
                              index=[['河北省','河北省','河北省','河北省',
                                      '河南省','河南省','河南省','河南省'],
                                     ['石家庄市','唐山市','邯郸市','秦皇岛市',
                                      '郑州市','开封市','洛阳市','新乡市']])
```

认识层次化索引

在创建层次化索引对象时，嵌套函数中两个列表的长度必须是保持一致的，否则将会出现ValueError错误。

认识层次化索引

还可以通过MultiIndex类的方法构建一个层次化索引，该类提供了3种创建层次化索引的方法：

- MultiIndex.from_tuples()：将元组列表转换为MultiIndex。
- MultiIndex.from_arrays()：将数组列表转换为MultiIndex。
- MultiIndex.from_product()：从多个集合的笛卡尔乘积中创建一个MultiIndex。

认识层次化索引

from_tuples()方法可以将包含若干个元组的列表转换为MultiIndex对象，其中元组的第一个元素作为外层索引，元组的第二个元素作为内层索引。

```
list_tuples = [('A','A1'), ('A','A2'), ('B','B1'),('B','B2'), ('B','B3')]
# 根据元组列表创建一个MultiIndex对象
multi_index = MultiIndex.from_tuples(tuples=list_tuples,
                    names=[ '外层索引', '内层索引'])
```

认识层次化索引

from_arrays()方法是将数组列表转换为MultiIndex对象，其中嵌套的第一个列表将作为外层索引，嵌套的第二个列表将作为内层索引。

```
multi_array = MultiIndex.from_arrays(arrays =[['A', 'B', 'A', 'B', 'B'],
                                              ['A1', 'A2', 'B1', 'B2', 'B3']],
                                     names=['外层索引','内层索引'])
```

认识层次化索引

from_product()方法表示从多个集合的笛卡尔乘积中创建一个MultiIndex对象。

```
numbers = [0, 1, 2]
colors = ['green', 'purple']
multi_product = pd.MultiIndex.from_product(iterables=[numbers, colors],
                                           names=['number', 'color'])
```

认识层次化索引

在数学中，两个集合X和Y的笛卡尔积，又称直积，表示为X × Y，第一个对象是X的成员，而第二个对象是Y的所有可能有序对的其中一个成员 。

假设集合A={a，b}，集合B={0，1，2}，则两个集合的笛卡尔积为{(a，0)，(a，1)，(a，2)，(b，0)，(b，1)，(b，2)}。

层次化索引的操作

假设某商城在3月份统计了书籍的销售情况，并记录在下表中。

单位：本

小说	高山上的小邮局	50
	失踪的总统	60
	绿毛水怪	40
散文随笔	皮囊	94
	浮生六记	63
	自在独行	101
传记	梅西	200
	老舍自传	56
	库里传	45

从左边数第1列的数据表示书籍的类别，第2列的数据表示书籍的名称，第3列的数据表示书籍的销售数量。其中，**第1列作为外层索引使用，第2列作为内层索引使用**。

层次化索引的操作

根据书籍统计表，创建一个具有多层索引的Series对象，示例如下：

```
ser_obj = Series([50, 60, 40, 94, 63, 101, 200, 56, 45],
                 index=[['小说', '小说', '小说',
                         '散文随笔', '散文随笔', '散文随笔',
                         '传记', '传记', '传记'],
                        ['高山上的小邮局', '失踪的总统', '绿毛水怪',
                         '皮囊', '浮生六记', '自在独行',
                         '梅西', '老舍自传', '库里传']])
```

层次化索引的操作

如果商城管理员需要统计小说销售的情况，则可以从表中筛选出外层索引标签为小说的数据。

```
# 获取所有外层索引为“小说”的数据
ser_obj['小说']
```

高山上的小邮局 50
失踪的总统 60
绿毛水怪 40

层次化索引的操作

假设当前只知道书名为“自在独行”，但所属的类别和销售数量并不清楚，则需要操作内层索引获取该书籍的类别与销售数量。

```
# 获取内层索引对应的数据
ser_obj[:,'自在独行']
```

散文随笔　101

层次化索引的操作

交换分层顺序是指交换外层索引和内层索引的位置。

单位：本

小说	高山上的小邮局	50
	失踪的总统	60
	绿毛水怪	40
散文随笔	皮囊	94
	浮生六记	63
	自在独行	101
传记	梅西	200
	老舍自传	56
	库里传	45

交换索引前

单位：本

高山上的小邮局	小说	50
失踪的总统		60
绿毛水怪		40
皮囊	散文随笔	94
浮生六记		63
自在独行		101
梅西	传记	200
老舍自传		56
库里传		45

交换索引后

层次化索引的操作

在Pandas中，交换分层顺序的操作可以使用swaplevel()方法来完成。

```
# 交换外层索引与内层索引位置
ser_obj.swaplevel()
```

层次化索引的操作

要想按照分层索引对数据排序，则可以通过sort_index()方法实现。

```
sort_index（axis = 0， level = None， ascending = True，
inplace = False， kind =' quicksort '， na_position ='last'，
sort_remaining = True， by = None ）
```

- by：表示按指定的值排序。
- ascending：布尔值，表示是否升序排列，默认为True。

层次化索引的操作

在使用sort_index()方法排序时，会优先选择按外层索引进行排序，然后再按照内层索引进行排序。

		word	num
A	1	a	1
	3	b	2
	2	d	4
C	3	e	5
	1	f	3
	2	k	2
B	4	d	6
	5	s	2
	8	l	3

排序前

		word	num
A	1	a	1
	2	b	4
	3	d	2
B	4	d	6
	5	s	2
	8	l	3
C	1	f	3
	2	k	2
	3	5	2

排序后

过渡页

07 读写数据操作

读写文本文件

在进行数据分析时，通常不会将需要分析的数据直接写入到程序中，这样不仅造成程序代码臃肿，而且可用率很低。常用的解决方法是将待分析的数据存储到本地中，之后再对存储文件进行读取。

文本文件	Excel文件	HTML文件	数据库

读写文本文件

CSV文件是一种纯文本文件，可以使用任何文本编辑器进行编辑，它支持追加模式，节省内存开销。

读写文本文件

to_csv()方法的功能是将数据写入到CSV文件中。

```
to_csv(path_or_buf=None,sep=',',na_rep='',float_format=None,colum
ns=None,header=True, index=True, index_label=None, mode='w', ...)
```

- path_or_buf：文件路径。
- index：默认为True，若设为False，则将不会显示索引。
- sep：分隔符，默认用“，”隔开。

读写文本文件

read_csv()函数的作用是将CSV文件的数据读取出来，转换成DataFrame对象展示。

```
read_csv(filepath_or_buffer,sep=',', delimiter=None, header='infer',
names=None, index_col=None, usecols=None, prefix=None, ...)
```

- sep：指定使用的分隔符，默认用“，”分隔。
- header：指定行数用来作为列名。
- names：用于结果的列名列表。如果文件不包含标题行，则应该将该参数设置为None。

读写文本文件

Text格式的文件也是比较常见的存储数据的方式，后缀名为".txt"，它与上面提到的CSV文件都属于文本文件。

- 如果希望读取Text文件，既可以用前面提到的read_csv()函数，也可以使用read_table()函数。

读写文本文件

read_csv()与read_table()函数的区别在于使用的分隔符不同，前者使用“，”作为分隔符，而后者使用“\t”作为分隔符。

读写Excel文件

Excel文件也是比较常见的存储数据的文件，它里面均是以二维表格的形式显示的，可以对数据进行统计、分析等操作。Excel的文件扩展名有.xls和.xlsx两种。

.xls

.xlsx

读写Excel文件

to_excel()方法的功能是将DataFrame对象写入到Excel工作表中。

```
to_excel(excel_writer,sheet_name='Sheet1',na_rep='',
float_format=None, columns=None, header=True, index=True, ...)
```

- excel_writer：表示读取的文件路径。
- sheet_name：表示工作表的名称，默认为“Sheet1”。
- na_rep：表示缺失数据。
- index：表示是否写行索引，默认为True。

读写Excel文件

read_excel()函数的作用是将Excel中的数据读取出来，转换成DataFrame展示。

```
pandas.read_excel(io,sheet_name=0,header=0,names=None
,index_col=None, **kwds)
```

- io：表示路径对象。
- sheet_name：指定要读取的工作表，默认为0。
- header：用于解析DataFrame的列标签。
- names：要使用的列名称。

读取HTML表格数据

在浏览网页时，有些数据会在HTML网页中以表格的形式进行展示。

专业名称	专业代码	专业大类	专业小类	操作
哲学类	0101	哲学	哲学类	开设院校 加入对比
哲学	010101	哲学	哲学类	开设院校 加入对比
逻辑学	010102	哲学	哲学类	开设院校 加入对比
宗教学	010103	哲学	哲学类	开设院校 加入对比
伦理学	010104	哲学	哲学类	开设院校 加入对比
经济学类	0201	经济学	经济学类	开设院校 加入对比

读取HTML表格数据

对于网页中的表格，可以使用read_html()函数进行读取，并返回一个包含多个DataFrame对象的列表。

```
pandas.read_html(io, match='.+', flavor=None,header=None,
index_col=None,skiprows=None, attrs=None)
```

- io：表示路径对象。
- header：表示指定列标题所在的行。
- index_col：表示指定行标题对应的列。
- attrs：默认为None，用于表示表格的属性值。

读写数据库

大多数情况下，海量的数据是使用数据库进行存储的，这主要是依赖于数据库的数据结构化、数据共享性、独立性等特点。Pandas 支持Mysql、Oracle、SQLite等主流数据库的读写操作。

Mysql

Oracle

SQLite

读写数据库

为了高效地读取数据库中的数据，这里需要引入SQLAlchemy。

- SQLAlchemy是使用Python编写的一款开源软件，它提供的SQL工具包和对象映射工具能够高效地访问数据库。在使用SQLAlchemy时需要使用相应的连接工具包。

读写数据库

Pandas的io.sql模块中提供了常用的读写数据库函数。

函数名称	说明
read_sql_table()	将读取的整张数据表中的数据转换成DataFrame对象
read_sql_query()	将sql语句读取的结果转换成DataFrame对象
read_sql()	上述两个函数的结合，既可以读数据表也可以读SQL语句
to_sql()	将数据写入到SQL数据库中。

读写数据库

在连接mysql数据库时，这里使用的是mysqlconnector驱动，如果当前的Python环境中没有改模块，则需要使用pip install mysqlconnector命令安装该模块。

读写数据库

read_sql()函数既可以读取整张数据表，又可以执行SQL语句。

```
pandas.read_sql(sql,con,index_col=None,coerce_float=True,params=
None,parse_dates=None, columns=None, chunksize=None)
```

- sql：表示被执行的SQL语句。
- con：接收数据库连接，表示数据库的连接信息。
- columns：从SQL表中选择列名列表。

读写数据库

通过create_engine()函数创建连接时，需要指定格式如下：'数据库类型+数据库驱动名称://用户名:密码@机器地址:端口号/数据库名'。

读写数据库

to_sql()方法的功能是将Series或DataFrame对象以数据表的形式写入到数据库中。

```
to_sql（name，con，schema = None，if_exists ='fail'，index =
True，index_label = None，chunksize = None，dtype = None ）
```

- name：表示数据库表的名称。
- con：表示数据库的连接信息。
- if_exists：可以取值为fail、replace或append，默认为’fail’。

本章小结

- 本章主要针对Pandas库的基础内容进行了介绍，包括常用的数据结构、索引操作、算术运算、数据排序、统计计算与描述、层次化索引和读写数据操作等，并结合北京高考分数线的分析案例，讲解如何使用Pandas操作数据。

- 通过对本章的学习，希望大家可以用Pandas实现简单地操作，为后续深入地学习打好扎实的基础。