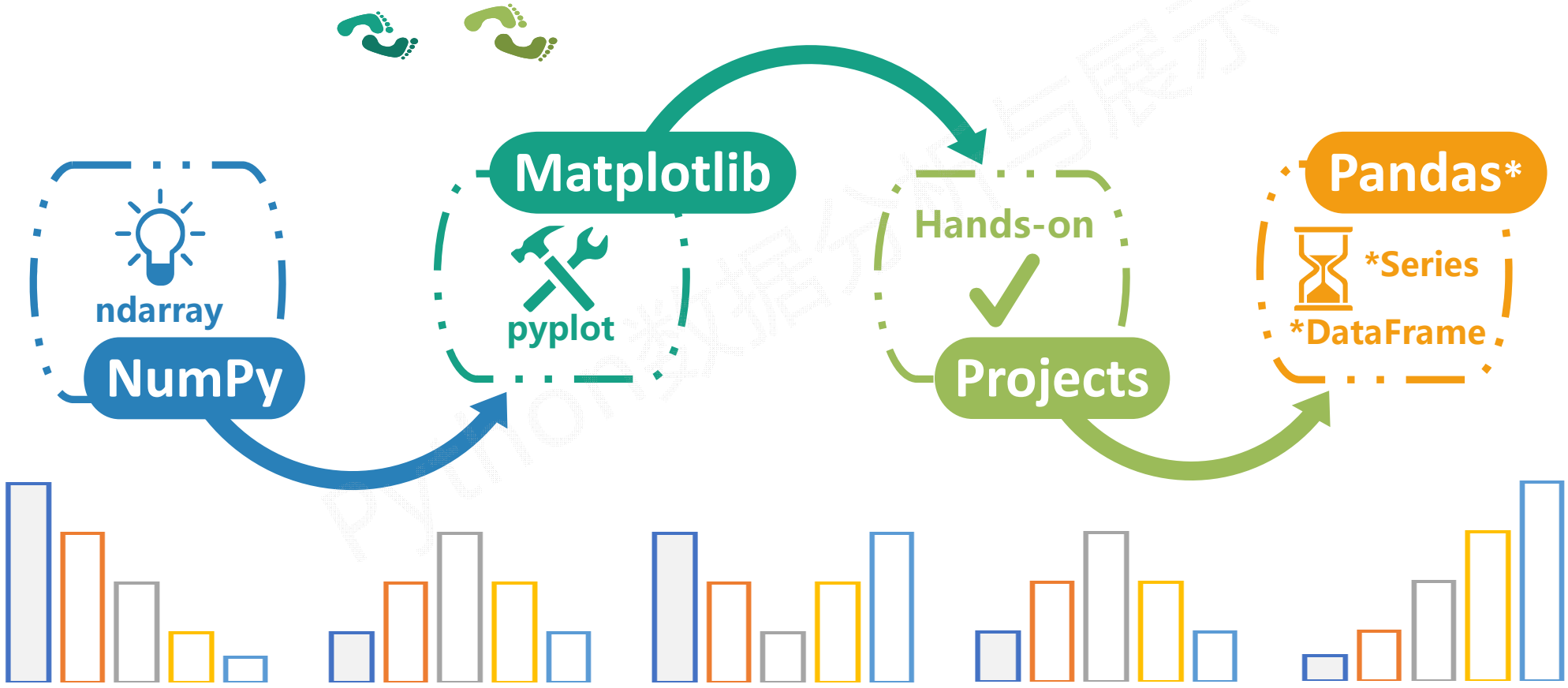
Pandas库入门

肖 毅

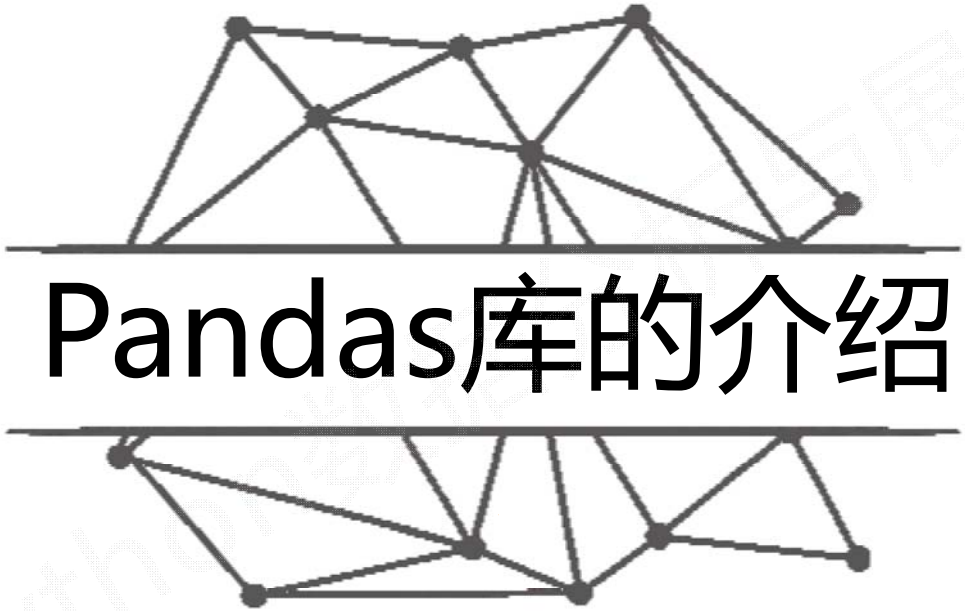
python
弹指之间 · 享受创新

Python数据分析与展示

掌握表示、清洗、统计和展示数据的能力



O4X -Tian



Pandas库的介绍

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

http://pandas.pydata.org

Python Data Analysis Library

pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

pandas is a NUMFocus sponsored project. This will help ensure the success of development of *pandas* as a world-class open-source project.

A Fiscally Sponsored Project of

NUMFOCUS

OPEN CODE = BETTER SCIENCE

0.19.2 Final (December 24, 2016)

This is a minor bug-fix release in the 0.19.x series and includes some small regression fixes, bug fixes and performance improvements.

Highlights include:

- Compatibility with Python 3.6
- Added a Pandas Cheat Sheet.

See the v0.19.2 Whatsnew page for an overview of all bugs that have been fixed in 0.19.2.

VERSIONS

Release
0.19.2 - December 2016
download // docs // pdf

Development
0.20.0 - 2017
github // docs

Previous Releases
0.19.1 - download // docs // pdf
0.19.0 - download // docs // pdf
0.18.1 - download // docs // pdf
0.18.0 - download // docs // pdf
0.17.1 - download // docs // pdf
0.17.0 - download // docs // pdf
0.16.2 - download // docs // pdf
0.16.1 - download // docs // pdf
0.16.0 - download // docs // pdf
0.15.2 - download // docs // pdf
0.15.1 - download // docs // pdf
0.15.0 - download // docs // pdf
0.14.1 - download // docs // pdf
0.14.0 - download // docs // pdf
0.13.1 - download // docs // pdf
0.13.0 - download // docs // pdf
0.12.0 - download // docs // pdf

Pandas库的引用

Pandas是Python第三方库，提供高性能易用数据类型和分析工具

```
import pandas as pd
```

Pandas基于NumPy实现，常与NumPy和Matplotlib一同使用

Python数据分析与应用

Pandas库小测

```
In [66]: import pandas as pd

In [67]: d = pd.Series(range(20))

In [68]: d
Out[68]:
0      0
1      1
2      2
3      3
4      4
5      5
6      6
7      7
8      8
9      9
10    10
11    11
12    12
13    13
14    14
15    15
16    16
17    17
18    18
19    19
dtype: int32
```

```
In [65]: d.cumsum()
Out[65]:
0       0
1       1
2       3
3       6
4      10
5      15
6      21
7      28
8      36
9      45
10     55
11     66
12     78
13     91
14    105
15    120
16    136
17    153
18    171
19    190
dtype: int32
```

计算前N项累加和

Python数据分析

Pandas库的理解

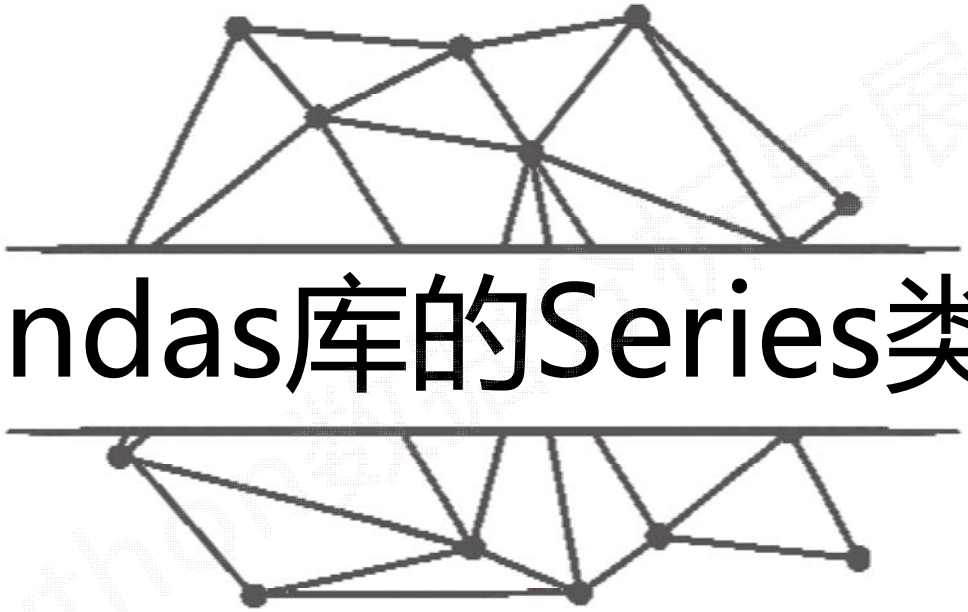
两个数据类型：Series, DataFrame

基于上述数据类型的各类操作

基本操作、运算操作、特征类操作、关联类操作

Pandas库的理解

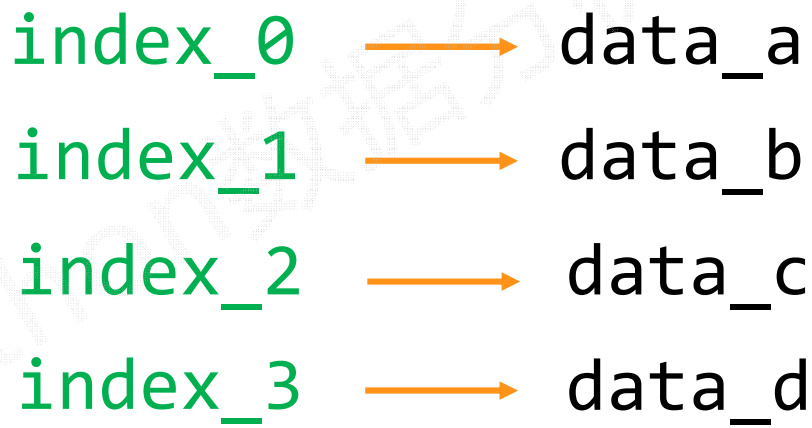
NumPy	Pandas
基础数据类型	扩展数据类型
关注数据的结构表达	关注数据的应用表达
维度：数据间关系	数据与索引间关系



Pandas库的Series类型

Series类型

Series类型由一组数据及与之相关的数据索引组成



Series类型

Series类型由一组数据及与之相关的数据索引组成

```
In [72]: import pandas as pd

In [73]: a = pd.Series([9, 8, 7, 6])

In [74]: a
Out[74]:
0    9
1    8
2    7
3    6
dtype: int64
```

自动索引

NumPy中数据类型

Series类型

Series类型由一组数据及与之相关的数据索引组成

```
In [76]: import pandas as pd

In [77]: b = pd.Series([9, 8, 7, 6], index=['a','b','c','d'])

In [78]: b
Out[78]:
a    9
b    8
c    7
d    6
dtype: int64
```

自定义索引

作为第二个参数，可以省略index=

Series类型

Series类型可以由如下类型创建：

- Python列表
- 标量值
- Python字典
- ndarray
- 其他函数

Python数据分析与展示

Series类型

从标量值创建

不能省略index

```
In [85]: import pandas as pd

In [86]: s = pd.Series(25, index=['a','b','c'])

In [87]: s
Out[87]:
a    25
b    25
c    25
dtype: int64
```

Series类型

从字典类型创建

index从字典中进行选择操作

```
In [88]: import pandas as pd

In [89]: d = pd.Series({'a':9, 'b':8, 'c':7})

In [90]: d
Out[90]:
a    9
b    8
c    7
dtype: int64
```

```
In [91]: e = pd.Series({'a':9, 'b':8, 'c':7}, index=['c', 'a', 'b', 'd'])

In [92]: e
Out[92]:
c    7.0
a    9.0
b    8.0
d    NaN
dtype: float64
```

Series类型

从ndarray类型创建

```
In [98]: import pandas as pd

In [99]: import numpy as np

In [100]: n = pd.Series(np.arange(5))

In [101]: n
Out[101]:
0    0
1    1
2    2
3    3
4    4
dtype: int32
```

```
In [102]: m = pd.Series(np.arange(5), index=np.arange(9,4,-1))

In [103]: m
Out[103]:
9    0
8    1
7    2
6    3
5    4
dtype: int32
```

Series类型

Series类型可以由如下类型创建：

- Python列表，index与列表元素个数一致
- 标量值，index表达Series类型的尺寸
- Python字典，键值对中的“键”是索引，index从字典中进行选择操作
- ndarray，索引和数据都可以通过ndarray类型创建
- 其他函数，range()函数等

Series类型的基本操作

Series类型包括index和values两部分

Series类型的操作类似ndarray类型

Series类型的操作类似Python字典类型

Series类型的基本操作

```
In [104]: import pandas as pd

In [105]: b = pd.Series([9, 8, 7, 6], ['a', 'b', 'c', 'd'])

In [106]: b
Out[106]:
a    9
b    8
c    7
d    6
dtype: int64

In [107]: b.index
Out[107]: Index(['a', 'b', 'c', 'd'], dtype='object')

In [108]: b.values
Out[108]: array([9, 8, 7, 6], dtype=int64)
```

.index　获得索引

.values　获得数据

Python数据分析与展示

Series类型的基本操作

```
In [109]: b['b']
Out[109]: 8

In [110]: b[1]
Out[110]: 8

In [111]: b[['c', 'd', 0]]
Out[111]:
c    7.0
d    6.0
0    NaN
dtype: float64

In [112]: b[['c', 'd', 'a']]
Out[112]:
c    7
d    6
a    9
dtype: int64
```

自动索引和自定义索引并存

两套索引并存，但不能混用

Series类型的基本操作

Series类型的操作类似ndarray类型：

- 索引方法相同，采用[]
- NumPy中运算和操作可用于Series类型
- 可以通过自定义索引的列表进行切片
- 可以通过自动索引进行切片，如果存在自定义索引，则一同被切片

Series类型的基本操作

```
In [113]: import pandas as pd

In [114]: b = pd.Series([9, 8, 7, 6], ['a', 'b', 'c', 'd'])

In [115]: b
Out[115]:
a    9
b    8
c    7
d    6
dtype: int64

In [116]: b[3]
Out[116]: 6

In [117]: b[:3]
Out[117]:
a    9
b    8
c    7
dtype: int64
```

```
In [118]: b[b > b.median()]
Out[118]:
a    9
b    8
dtype: int64

In [119]: np.exp(b)
Out[119]:
a    8103.083928
b    2980.957987
c    1096.633158
d     403.428793
dtype: float64
```

Python数据分析与展示

Series类型的基本操作

Series类型的操作类似Python字典类型：

- 通过自定义索引访问
- 保留字in操作
- 使用.get()方法

Python数据分析与展示

Series类型的基本操作

```
In [120]: import pandas as pd

In [121]: b = pd.Series([9, 8, 7, 6], ['a', 'b', 'c', 'd'])

In [122]: b['b']
Out[122]: 8

In [123]: 'c' in b
Out[123]: True

In [124]: 0 in b
Out[124]: False

In [125]: b.get('f', 100)
Out[125]: 100
```

Series类型对齐操作

Series + Series

```
In [134]: import pandas as pd

In [135]: a = pd.Series([1, 2, 3], ['c', 'd', 'e'])

In [136]: b = pd.Series([9, 8, 7, 6], ['a', 'b', 'c', 'd'])

In [137]: a + b
Out[137]:
a    NaN
b    NaN
c    8.0
d    8.0
e    NaN
dtype: float64
```

Series类型在运算中会自动对齐不同索引的数据

Series类型的name属性

Series对象和索引都可以有一个名字，存储在属性.name中

```
In [149]: import pandas as pd

In [150]: b = pd.Series([9, 8, 7, 6], ['a', 'b', 'c', 'd'])

In [151]: b.name

In [152]: b.name = 'Series对象'

In [153]: b.index.name = '索引列'

In [154]: b
Out[154]:
索引列
a    9
b    8
c    7
d    6
Name: Series对象, dtype: int64
```

Series类型的修改

Series对象可以随时修改并即刻生效

```
In [157]: import pandas as pd

In [158]: b = pd.Series([9, 8, 7, 6], ['a', 'b', 'c', 'd'])

In [159]: b['a'] = 15

In [160]: b.name = "Series"

In [161]: b
Out[161]:
a    15
b     8
c     7
d     6
Name: Series, dtype: int64
```

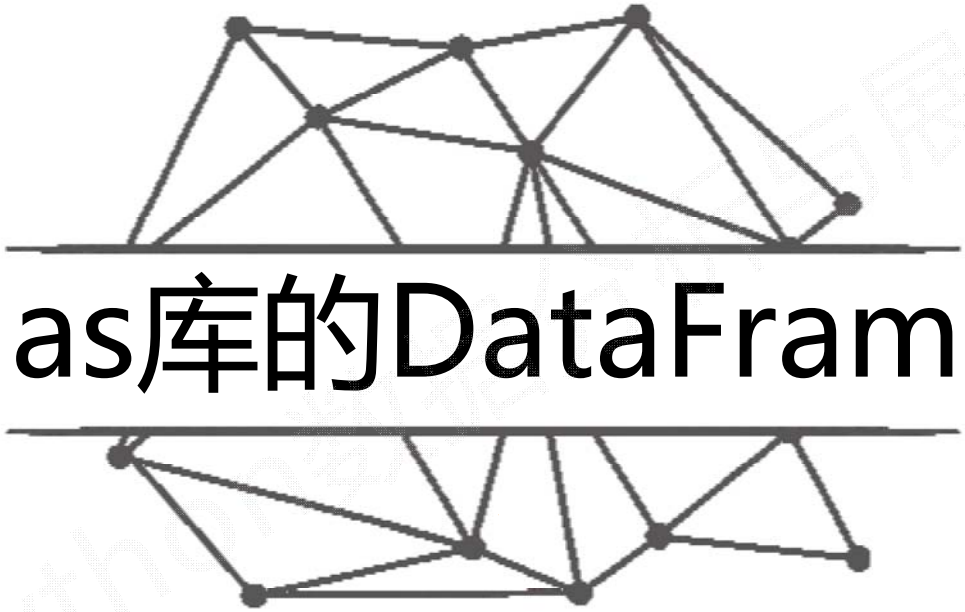
```
In [162]: b.name = "New Series"

In [163]: b['b', 'c'] = 20

In [164]: b
Out[164]:
a    15
b    20
c    20
d     6
Name: New Series, dtype: int64
```

Python数据分析与挖掘

Series类型

Series是一维带“标签”数组

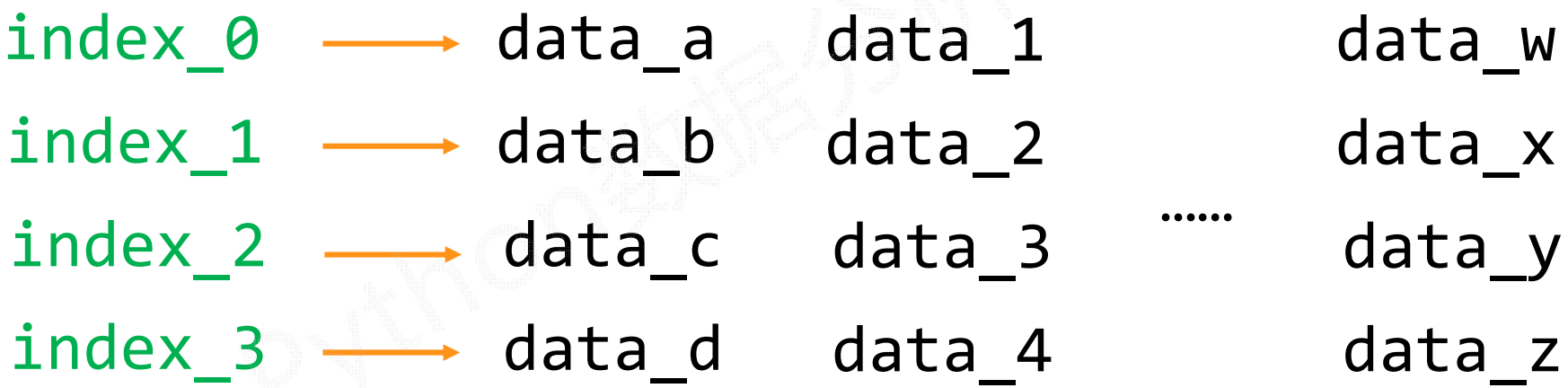
Series基本操作类似ndarray和字典，根据索引对齐



Pandas库的DataFrame类型

DataFrame类型

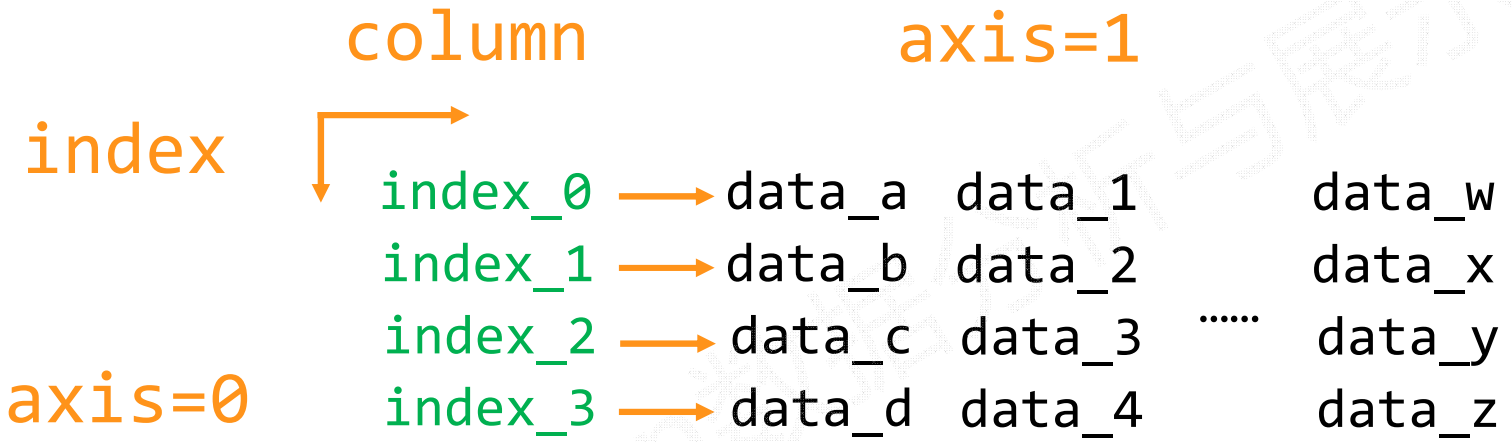
DataFrame类型由共用相同索引的一组列组成



索引

多列数据

DataFrame类型



DataFrame是一个表格型的数据类型，每列值类型可以不同

DataFrame既有行索引、也有列索引

DataFrame常用于表达二维数据，但可以表达多维数据

DataFrame类型

DataFrame类型可以由如下类型创建：

- 二维ndarray对象
- 由一维ndarray、列表、字典、元组或Series构成的字典
- Series类型
- 其他的DataFrame类型

DataFrame类型

从二维ndarray对象创建

```
In [165]: import pandas as pd

In [166]: import numpy as np

In [167]: d = pd.DataFrame(np.arange(10).reshape(2,5))

In [168]: d
Out[168]:
   0  1  2  3  4
0  0  1  2  3  4
1  5  6  7  8  9
```

自动行索引

自动列索引

DataFrame类型

从一维ndarray对象字典创建

```
In [171]: import pandas as pd

In [172]: dt = {'one': pd.Series([1, 2, 3], index=['a', 'b', 'c']),
     ...:       'two': pd.Series([9, 8, 7, 6], index = ['a', 'b', 'c', 'd'])}

In [173]: d = pd.DataFrame(dt)

In [174]: d
Out[174]:
   one  two
a  1.0    9
b  2.0    8
c  3.0    7
d  NaN    6

In [175]: pd.DataFrame(dt, index=['b', 'c', 'd'], columns=['two', 'three'])
Out[175]:
   two three
b    8   NaN
c    7   NaN
d    6   NaN
```

自定义行索引

自定义列索引

数据根据行列索引自动补齐

DataFrame类型

从列表类型的字典创建

```
In [176]: import pandas as pd

In [177]: dl = {'one': [1, 2, 3, 4], 'two': [9, 8, 7, 6]}

In [178]: d = pd.DataFrame(dl, index = ['a', 'b', 'c', 'd'])

In [179]: d
Out[179]:
   one  two
a    1    9
b    2    8
c    3    7
d    4    6
```

2016年7月部分大中城市新建住宅价格指数

城市	环比	同比	定基
北京	101.5	120.7	121.4
上海	101.2	127.3	127.8
广州	101.3	119.4	120.0
深圳	102.0	140.9	145.5
沈阳	100.1	101.4	101.6

```
In [194]: import pandas as pd

In [195]: dl = {'城市': ['北京', '上海', '广州', '深圳', '沈阳'],
     ...:       '环比': [101.5 , 101.2 , 101.3, 102.0 , 100.1 ],
     ...:       '同比': [120.7 , 127.3 , 119.4, 140.9 , 101.4 ],
     ...:       '定基': [121.4 , 127.8 , 120.0, 145.5 , 101.6 ]}

In [196]: d = pd.DataFrame(dl, index=['c1', 'c2', 'c3', 'c4', 'c5'])

In [197]: d
Out[197]:
      同比  城市     定基     环比
c1  120.7  北京  121.4  101.5
c2  127.3  上海  127.8  101.2
c3  119.4  广州  120.0  101.3
c4  140.9  深圳  145.5  102.0
c5  101.4  沈阳  101.6  100.1
```

```
In [200]: d.index
Out[200]: Index(['c1', 'c2', 'c3', 'c4', 'c5'], dtype='object')

In [201]: d.columns
Out[201]: Index(['同比', '城市', '定基', '环比'], dtype='object')

In [202]: d.values
Out[202]:
array([[120.7, '北京', 121.4, 101.5],
       [127.3, '上海', 127.8, 101.2],
       [119.4, '广州', 120.0, 101.3],
       [140.9, '深圳', 145.5, 102.0],
       [101.4, '沈阳', 101.6, 100.1]], dtype=object)
```

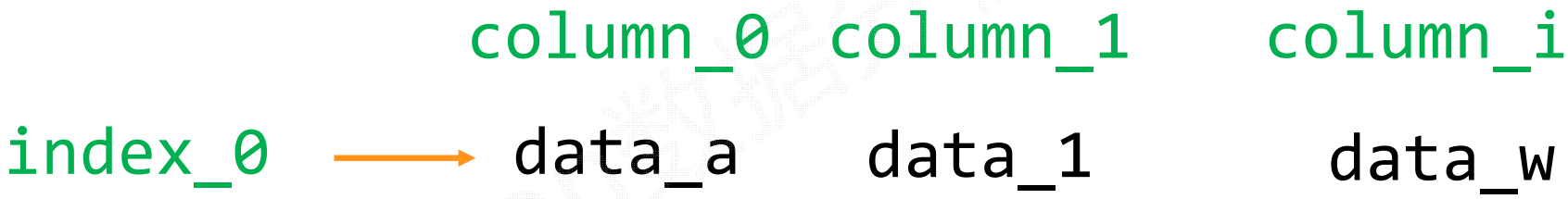
```
In [194]: import pandas as pd

In [195]: dl = {'城市': ['北京', '上海', '广州', '深圳', '沈阳'],
     ...:       '环比': [101.5 , 101.2 , 101.3, 102.0 , 100.1 ],
     ...:       '同比': [120.7 , 127.3 , 119.4, 140.9 , 101.4 ],
     ...:       '定基': [121.4 , 127.8 , 120.0, 145.5 , 101.6 ]}

In [196]: d = pd.DataFrame(dl, index=['c1', 'c2', 'c3', 'c4', 'c5'])

In [197]: d
Out[197]:
      同比  城市     定基     环比
c1  120.7  北京  121.4  101.5
c2  127.3  上海  127.8  101.2
c3  119.4  广州  120.0  101.3
c4  140.9  深圳  145.5  102.0
c5  101.4  沈阳  101.6  100.1
```

```
In [209]: d['同比']
Out[209]:
c1    120.7
c2    127.3
c3    119.4
c4    140.9
c5    101.4
Name: 同比, dtype: float64

In [210]: d.ix['c2']
Out[210]:
同比     127.3
城市        上海
定基     127.8
环比     101.2
Name: c2, dtype: object

In [211]: d['同比']['c2']
Out[211]: 127.3
```

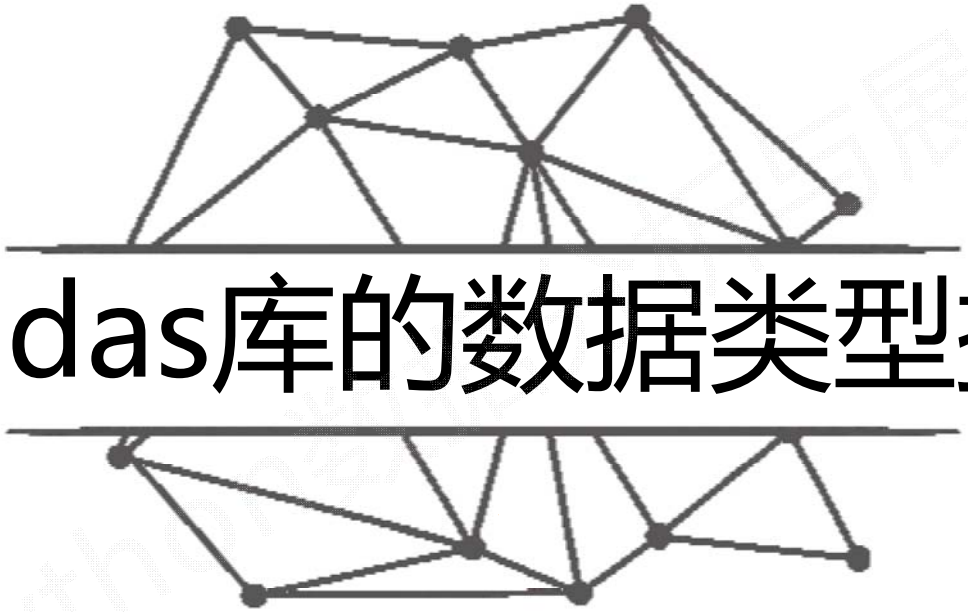
Python数据分析

DataFrame类型

DataFrame是二维带“标签”数组



DataFrame基本操作类似Series，依据行列索引



Pandas库的数据类型操作

数据类型操作

如何改变Series和DataFrame对象？

增加或重排：重新索引

删除：drop

Python数据分析与展示

重新索引

.reindex()能够改变或重排Series和DataFrame索引

```
In [194]: import pandas as pd

In [195]: dl = {'城市': ['北京', '上海', '广州', '深圳', '沈阳'],
     ...:       '环比': [101.5 , 101.2 , 101.3, 102.0 , 100.1 ],
     ...:       '同比': [120.7 , 127.3 , 119.4, 140.9 , 101.4 ],
     ...:       '定基': [121.4 , 127.8 , 120.0, 145.5 , 101.6 ]}

In [196]: d = pd.DataFrame(dl, index=['c1', 'c2', 'c3', 'c4', 'c5'])

In [197]: d
Out[197]:
     同比  城市     定基     环比
c1  120.7  北京  121.4  101.5
c2  127.3  上海  127.8  101.2
c3  119.4  广州  120.0  101.3
c4  140.9  深圳  145.5  102.0
c5  101.4  沈阳  101.6  100.1
```

```
In [218]: d = d.reindex(index=['c5','c4','c3','c2','c1'])

In [219]: d
Out[219]:
     同比  城市     定基     环比
c5  101.4  沈阳  101.6  100.1
c4  140.9  深圳  145.5  102.0
c3  119.4  广州  120.0  101.3
c2  127.3  上海  127.8  101.2
c1  120.7  北京  121.4  101.5

In [220]: d = d.reindex(columns=['城市','同比','环比','定基'])

In [221]: d
Out[221]:
    城市     同比     环比     定基
c5  沈阳  101.4  100.1  101.6
c4  深圳  140.9  102.0  145.5
c3  广州  119.4  101.3  120.0
c2  上海  127.3  101.2  127.8
c1  北京  120.7  101.5  121.4
```

重新索引

.reindex(index=None, columns=None, …)的参数

参数	说明
index, columns	新的行列自定义索引
fill_value	重新索引中，用于填充缺失位置的值
method	填充方法，ffill当前值向前填充，bfill向后填充
limit	最大填充量
copy	默认True，生成新的对象，False时，新旧相等不复制

重新索引

```
In [218]: d = d.reindex(index=['c5','c4','c3','c2','c1'])

In [219]: d
Out[219]:
       同比  城市     定基     环比
c5  101.4  沈阳  101.6  100.1
c4  140.9  深圳  145.5  102.0
c3  119.4  广州  120.0  101.3
c2  127.3  上海  127.8  101.2
c1  120.7  北京  121.4  101.5

In [220]: d = d.reindex(columns=['城市','同比','环比','定基'])

In [221]: d
Out[221]:
    城市     同比     环比     定基
c5  沈阳  101.4  100.1  101.6
c4  深圳  140.9  102.0  145.5
c3  广州  119.4  101.3  120.0
c2  上海  127.3  101.2  127.8
c1  北京  120.7  101.5  121.4
```

```
In [237]: newc = d.columns.insert(4,'新增')

In [238]: newd = d.reindex(columns=newc, fill_value=200)

In [239]: newd
Out[239]:
    城市     同比     环比     定基   新增
c5  沈阳  101.4  100.1  101.6  200
c4  深圳  140.9  102.0  145.5  200
c3  广州  119.4  101.3  120.0  200
c2  上海  127.3  101.2  127.8  200
c1  北京  120.7  101.5  121.4  200
```

索引类型

```
In [241]: d.index
Out[241]: Index(['c5', 'c4', 'c3', 'c2', 'c1'], dtype='object')

In [242]: d.columns
Out[242]: Index(['城市', '同比', '环比', '定基'], dtype='object')
```

Series和DataFrame的索引是Index类型

Index对象是不可修改类型

索引类型的常用方法

方法	说明
.append(idx)	连接另一个Index对象，产生新的Index对象
.diff(idx)	计算差集，产生新的Index对象
.intersection(idx)	计算交集
.union(idx)	计算并集
.delete(loc)	删除loc位置处的元素
.insert(loc,e)	在loc位置增加一个元素e

索引类型的使用

```
In [221]: d
Out[221]:
    城市     同比     环比     定基
c5  沈阳  101.4  100.1  101.6
c4  深圳  140.9  102.0  145.5
c3  广州  119.4  101.3  120.0
c2  上海  127.3  101.2  127.8
c1  北京  120.7  101.5  121.4
```

```
In [258]: nc = d.columns.delete(2)

In [259]: ni = d.index.insert(5, 'c0')

In [260]: nd = d.reindex(index=ni, columns=nc, method='ffill')

In [261]: nd
Out[261]:
    城市     同比     定基
c5  沈阳  101.4  101.6
c4  深圳  140.9  145.5
c3  广州  119.4  120.0
c2  上海  127.3  127.8
c1  北京  120.7  121.4
c0  北京  120.7  121.4
```

Python数据分析与展示

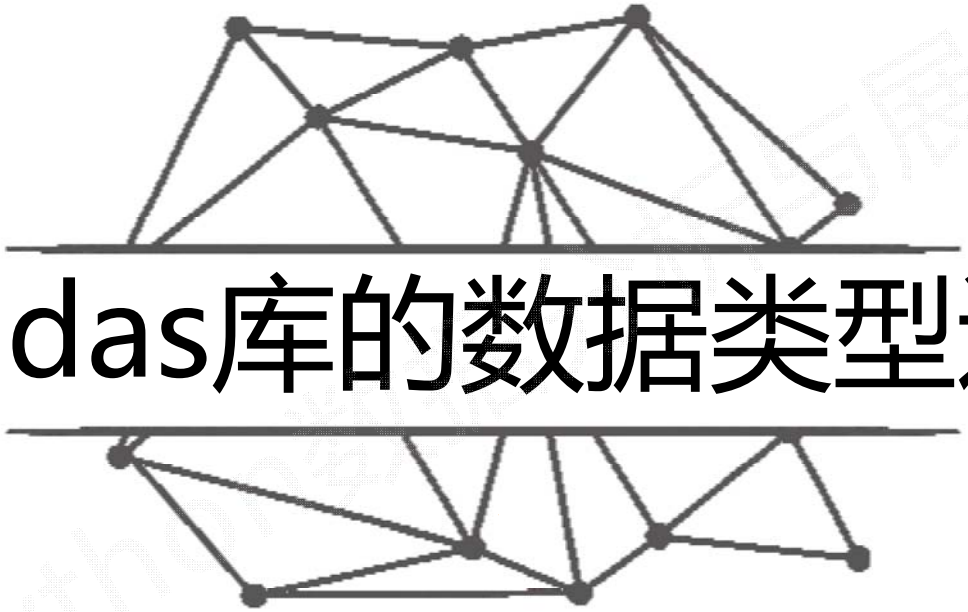
删除指定索引对象

.drop()能够删除Series和DataFrame指定行或列索引

```
In [276]: a = pd.Series([9, 8, 7, 6], index=['a','b','c','d'])

In [277]: a
Out[277]:
a    9
b    8
c    7
d    6
dtype: int64

In [278]: a.drop(['b', 'c'])
Out[278]:
a    9
d    6
dtype: int64
```

```
In [221]: d
Out[221]:
    城市     同比     环比     定基
c5  沈阳  101.4  100.1  101.6
c4  深圳  140.9  102.0  145.5
c3  广州  119.4  101.3  120.0
c2  上海  127.3  101.2  127.8
c1  北京  120.7  101.5  121.4
```

```
In [269]: d.drop('c5')
Out[269]:
    城市     同比     环比     定基
c4  深圳  140.9  102.0  145.5
c3  广州  119.4  101.3  120.0
c2  上海  127.3  101.2  127.8
c1  北京  120.7  101.5  121.4

In [270]: d.drop('同比', axis=1)
Out[270]:
    城市     环比     定基
c5  沈阳  100.1  101.6
c4  深圳  102.0  145.5
c3  广州  101.3  120.0
c2  上海  101.2  127.8
c1  北京  101.5  121.4
```



Pandas库的数据类型运算

算术运算法则

算术运算根据行列索引，补齐后运算，运算默认产生浮点数

补齐时缺项填充NaN（空值）

二维和一维、一维和零维间为广播运算

采用+ - * /符号进行的二元运算产生新的对象

数据类型的算术运算

```
In [298]: import pandas as pd

In [299]: import numpy as np

In [300]: a = pd.DataFrame(np.arange(12).reshape(3,4))

In [301]: a
Out[301]:
   0  1   2   3
0  0  1   2   3
1  4  5   6   7
2  8  9  10  11

In [302]: b = pd.DataFrame(np.arange(20).reshape(4,5))

In [303]: b
Out[303]:
    0   1   2   3   4
0   0   1   2   3   4
1   5   6   7   8   9
2  10  11  12  13  14
3  15  16  17  18  19
```

```
In [304]: a + b
Out[304]:
      0     1     2     3   4
0   0.0   2.0   4.0   6.0 NaN
1   9.0  11.0  13.0  15.0 NaN
2  18.0  20.0  22.0  24.0 NaN
3   NaN   NaN   NaN   NaN NaN

In [305]: a * b
Out[305]:
      0     1      2      3   4
0   0.0   1.0    4.0    9.0 NaN
1  20.0  30.0   42.0   56.0 NaN
2  80.0  99.0  120.0  143.0 NaN
3   NaN   NaN    NaN    NaN NaN
```

自动补齐，缺项补NaN

Python数据分析与展示

数据类型的算术运算

方法形式的运算

方法	说明
.add(d, **argws)	类型间加法运算，可选参数
.sub(d, **argws)	类型间减法运算，可选参数
.mul(d, **argws)	类型间乘法运算，可选参数
.div(d, **argws)	类型间除法运算，可选参数

数据类型的算术运算

```
In [298]: import pandas as pd

In [299]: import numpy as np

In [300]: a = pd.DataFrame(np.arange(12).reshape(3,4))

In [301]: a
Out[301]:
   0  1   2   3
0  0  1   2   3
1  4  5   6   7
2  8  9  10  11

In [302]: b = pd.DataFrame(np.arange(20).reshape(4,5))

In [303]: b
Out[303]:
    0   1   2   3   4
0   0   1   2   3   4
1   5   6   7   8   9
2  10  11  12  13  14
3  15  16  17  18  19
```

```
In [315]: b.add(a, fill_value = 100)
Out[315]:
       0      1      2      3      4
0    0.0    2.0    4.0    6.0  104.0
1    9.0   11.0   13.0   15.0  109.0
2   18.0   20.0   22.0   24.0  114.0
3  115.0  116.0  117.0  118.0  119.0

In [316]: a.mul(b, fill_value = 0)
Out[316]:
      0     1      2      3    4
0   0.0   1.0    4.0    9.0  0.0
1  20.0  30.0   42.0   56.0  0.0
2  80.0  99.0  120.0  143.0  0.0
3   0.0   0.0    0.0    0.0  0.0
```

fill_value参数替代NaN，替代后参与运算

数据类型的算术运算

```
In [306]: import pandas as pd

In [307]: import numpy as np

In [308]: b = pd.DataFrame(np.arange(20).reshape(4,5))

In [309]: b
Out[309]:
    0   1   2   3   4
0   0   1   2   3   4
1   5   6   7   8   9
2  10  11  12  13  14
3  15  16  17  18  19

In [310]: c = pd.Series(np.arange(4))

In [311]: c
Out[311]:
0    0
1    1
2    2
3    3
dtype: int32
```

```
In [321]: c - 10
Out[321]:
0   -10
1    -9
2    -8
3    -7
dtype: int32

In [322]: b - c
Out[322]:
      0     1     2     3   4
0   0.0   0.0   0.0   0.0 NaN
1   5.0   5.0   5.0   5.0 NaN
2  10.0  10.0  10.0  10.0 NaN
3  15.0  15.0  15.0  15.0 NaN
```

不同维度间为广播运算，一维Series默认在轴1参与运算

数据类型的算术运算

```
In [306]: import pandas as pd

In [307]: import numpy as np

In [308]: b = pd.DataFrame(np.arange(20).reshape(4,5))

In [309]: b
Out[309]:
    0   1   2   3   4
0   0   1   2   3   4
1   5   6   7   8   9
2  10  11  12  13  14
3  15  16  17  18  19

In [310]: c = pd.Series(np.arange(4))

In [311]: c
Out[311]:
0    0
1    1
2    2
3    3
dtype: int32
```

```
In [323]: b.sub(c, axis=0)
Out[323]:
    0   1   2   3   4
0   0   1   2   3   4
1   4   5   6   7   8
2   8   9  10  11  12
3  12  13  14  15  16
```

使用运算方法可以令一维Series参与轴0运算

比较运算法则

比较运算只能比较相同索引的元素，不进行补齐

二维和一维、一维和零维间为广播运算

采用> < >= <= == !=等符号进行的二元运算产生布尔对象

数据类型的比较运算

```
In [335]: import pandas as pd

In [336]: import numpy as np

In [337]: a = pd.DataFrame(np.arange(12).reshape(3,4))

In [338]: a
Out[338]:
   0  1   2   3
0  0  1   2   3
1  4  5   6   7
2  8  9  10  11

In [339]: d = pd.DataFrame(np.arange(12, 0, -1).reshape(3,4))

In [340]: d
Out[340]:
    0   1   2  3
0  12  11  10  9
1   8   7   6  5
2   4   3   2  1
```

```
In [342]: a > d
Out[342]:
       0      1      2      3
0  False  False  False  False
1  False  False  False   True
2   True   True   True   True

In [343]: a == d
Out[343]:
       0      1      2      3
0  False  False  False  False
1  False  False   True  False
2  False  False  False  False
```

同维度运算，尺寸一致

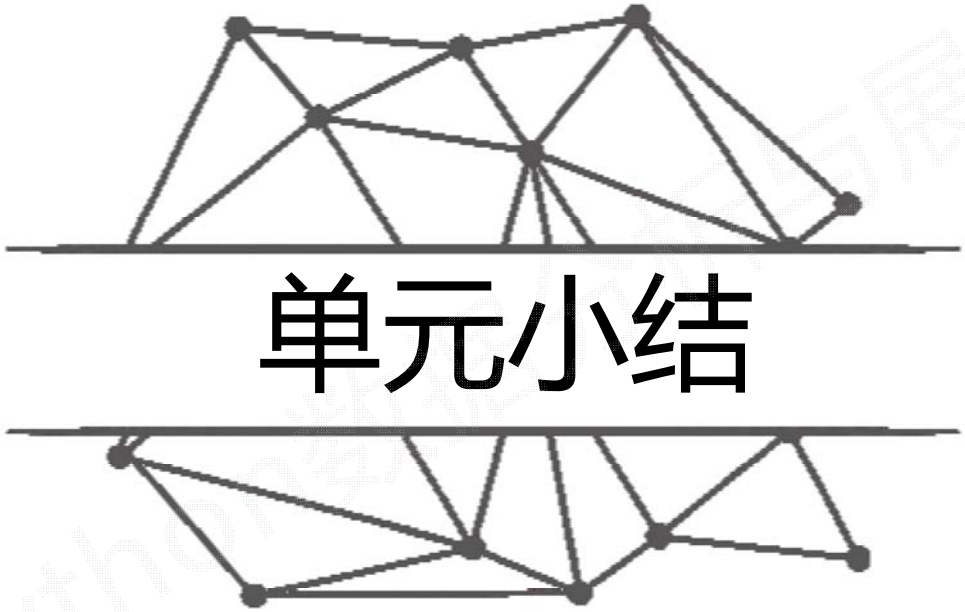
Python数据分析与展示

数据类型的比较运算

```
In [344]: import pandas as pd

In [345]: import numpy as np

In [346]: a = pd.DataFrame(np.arange(12).reshape(3,4))

In [347]: a
Out[347]:
   0  1   2   3
0  0  1   2   3
1  4  5   6   7
2  8  9  10  11

In [348]: c = pd.Series(np.arange(4))

In [349]: c
Out[349]:
0    0
1    1
2    2
3    3
dtype: int32
```

```
In [350]: a > c
Out[350]:
       0      1      2      3
0  False  False  False  False
1   True   True   True   True
2   True   True   True   True

In [351]: c > 0
Out[351]:
0    False
1     True
2     True
3     True
dtype: bool
```

不同维度，广播运算，默认在1轴



单元小结

Pandas库入门

Series = 索引 + 一维数据

DataFrame = 行列索引 + 二维数据

理解数据类型与索引的关系，操作索引即操作数据

重新索引、数据删除、算术运算、比较运算

像对待单一数据一样对待Series和DataFrame对象