Python与数据科学导论-06

—— 网络编程简介，Pandas基础

胡俊峰 北京大学

2023/03/16

Python 网络编程简介

什么是计算机网络？

- 计算机网络：用通信设备将计算机连接起来，在计算机之间传输数据（信息）的系统。
- 连网的计算机根据其提供的功能将之区分为客户机或服务器（C/S）
- 通信协议：计算机之间以及计算机与设备之间进行数据交换而遵守的规则、标准或约定
  - 典型的协议：TCP/IP（在互联网上采用），IEEE802.3以太网协议（局域网），IEEE902.11（无线局域网，WIFI）

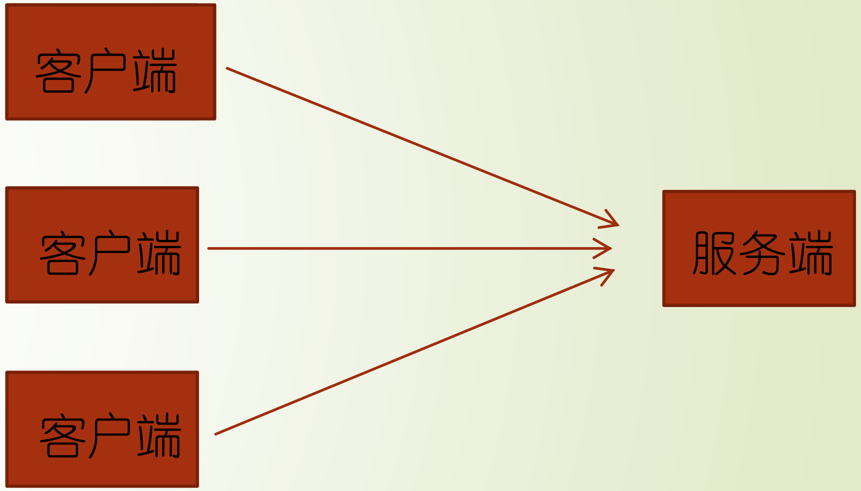
以买火车票为例：

客户端：

发出查询请求，如果有则购买一张票

服务端：

维护余票情况，如果有余票则卖票给客户端，余票数量减一；没有则返回购买失败



B/S模式：Browser/Server，对C/S模式的改进。一部分事务逻辑在前端实现，但是主要事务逻辑在服务器端实现，和数据库端形成三层结构。

建立在广域网之上，只要有网络、浏览器，可以随时随地进行业务处理。

以12306 APP买票是C/S服务模式，用12306网页购票是B/S模式。

Socket通信

套接字socket：网络中不同主机上的应用进程之间进行双向通信的端点。

- 每台主机有一个唯一的主机地址标识（IP），同时主机内还有标识服务的序号id，称作端口（port）。
- **socket绑定了相应的IP和port，可以用（IP：port）的形式表示一个socket地址。**
- 当客户端发起一个连接请求时，客户端socket地址中的端口由系统自动分配，服务器端套接字地址中的端口通常是某个和服务相对应的知名端口。（例如Web服务器常使用端口80，电子邮件服务器使用端口25）

一个连接由它两端的socket地址唯一确定：

(ClientIP：ClientPort, ServerIP：ServerPort)

信息：需要寄的快递

IP：小区地址

Port：门牌号,共有65536个端口

Socket：快递地址（小区+门牌号）

TCP，UCP等协议：快递公司

利用socket发送消息：把快递（消息）放到门口（socket），由快递公司（TCP等协议）负责送到对应的地址（对方socket）

传输层控制协议

- TCP：传输控制协议，面向连接、可靠。适用于要求可靠传输的应用。

  面向连接：发送数据之前必须在两端建立连接。

  仅支持单播传输：只能进行点对点数据传输。

  面向字节流：在不保留报文边界的情况下以字节流的方式进行传输。

  可靠：对每个包赋予序号，来判断是否出现丢包、误码。

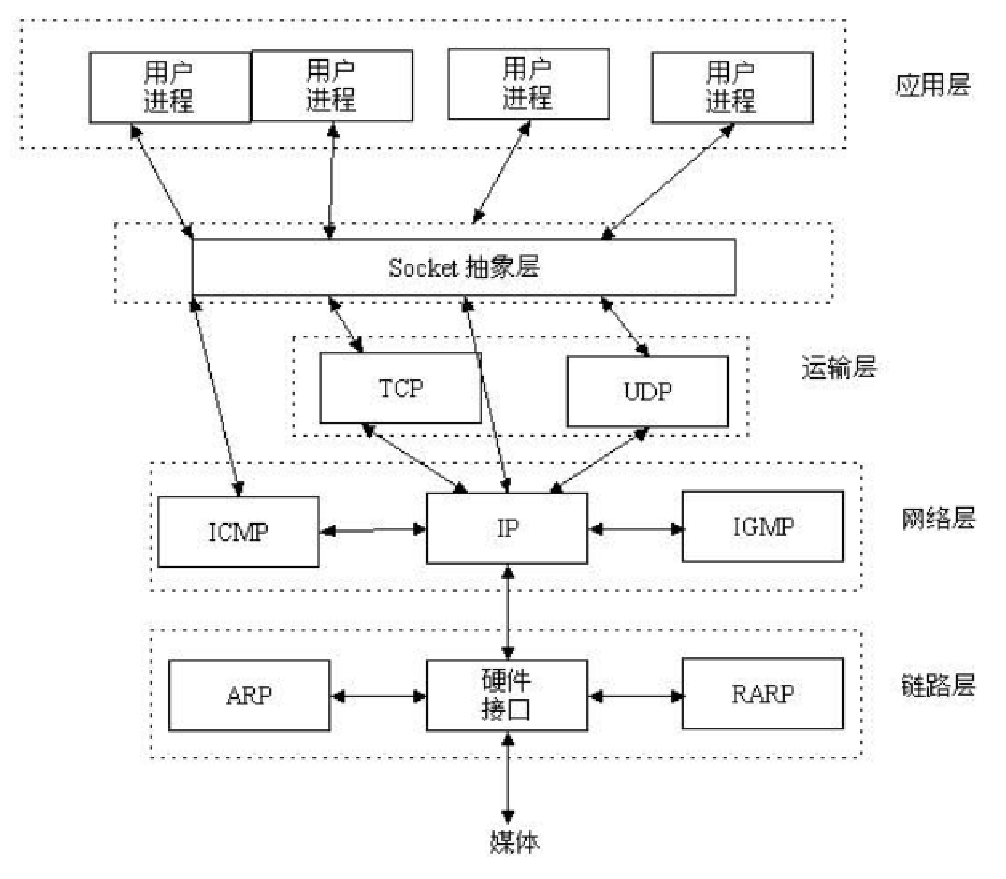
- UDP：用户数据报协议，面向非连接、不可靠。适用于实时应用。

  面向非连接：发送数据不需要建立连接。

  支持单播、多播、广播

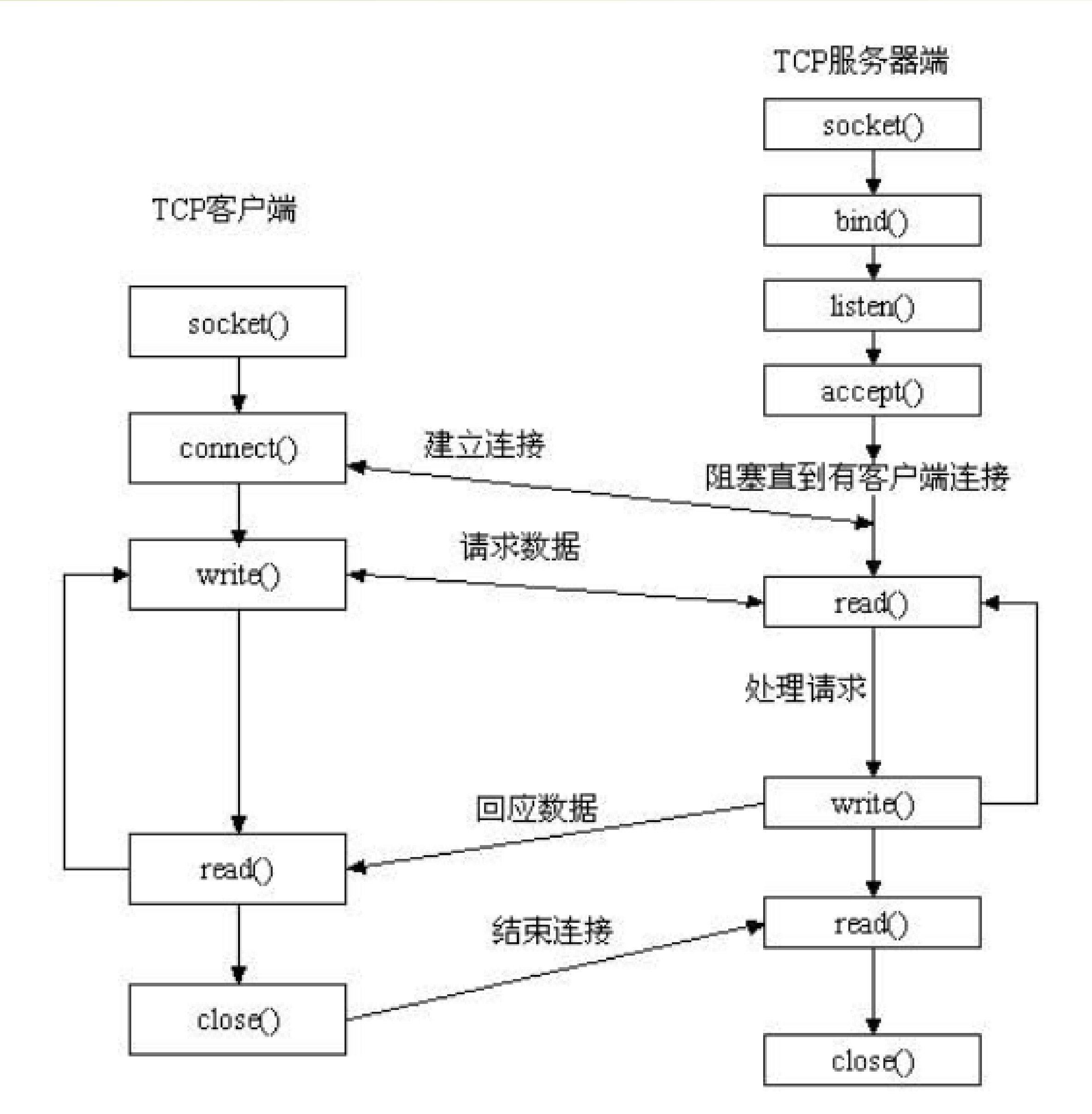
  面向报文：对应用层的报文添加首部后直接向下层交付。

  不可靠：没有拥塞控制，不会调整发送速率。



Socket是传输层和应用层之间的软件抽象层，是一组接口。

对于用户来说，socket把复杂的TCP/IP协议族隐藏在接口后，只需要遵循socket的规范，就能得到遵循TCP/UDP标准的程序。

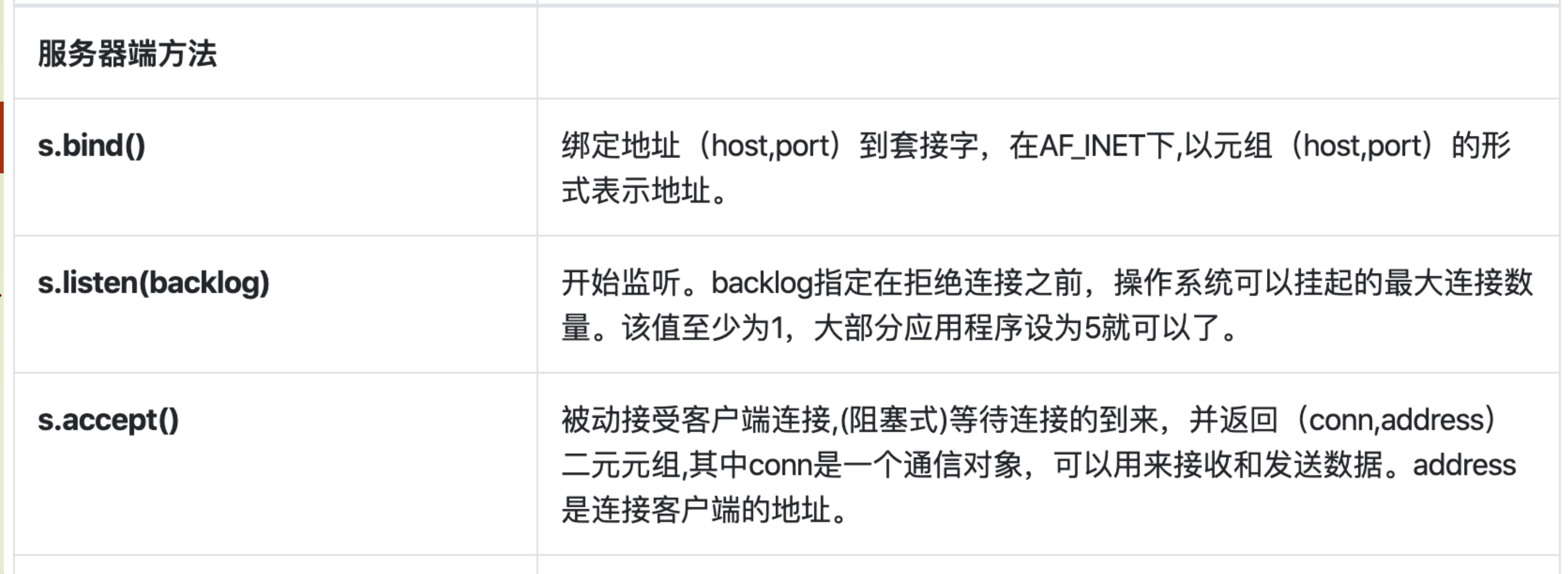


服务器端：
初始化socket，与IP端口绑定，对IP端口进行监听，调用accept()阻塞，等待客户端连接。

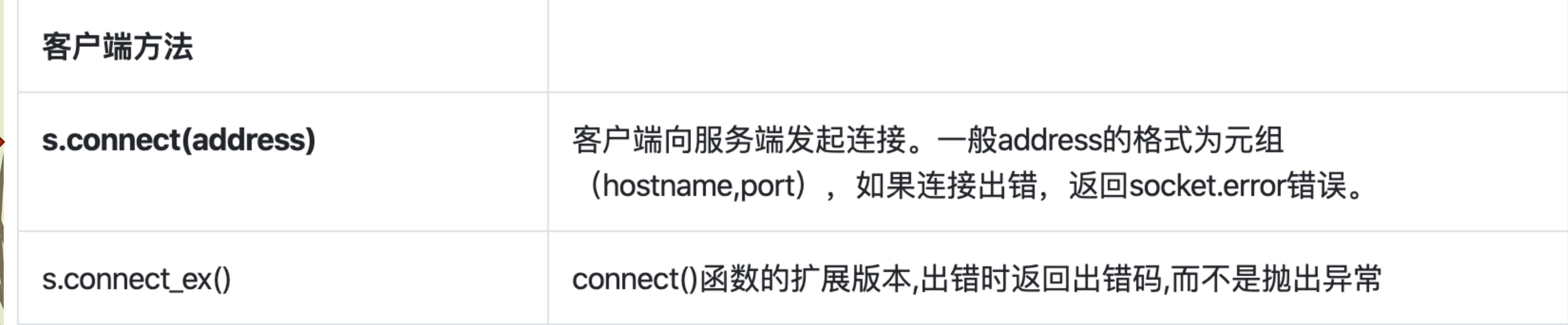
客户端：
初始化socket，连接服务器。

连接成功后客户端发送数据请求，服务器端接收并处理请求、回应数据，客户端读取数据。

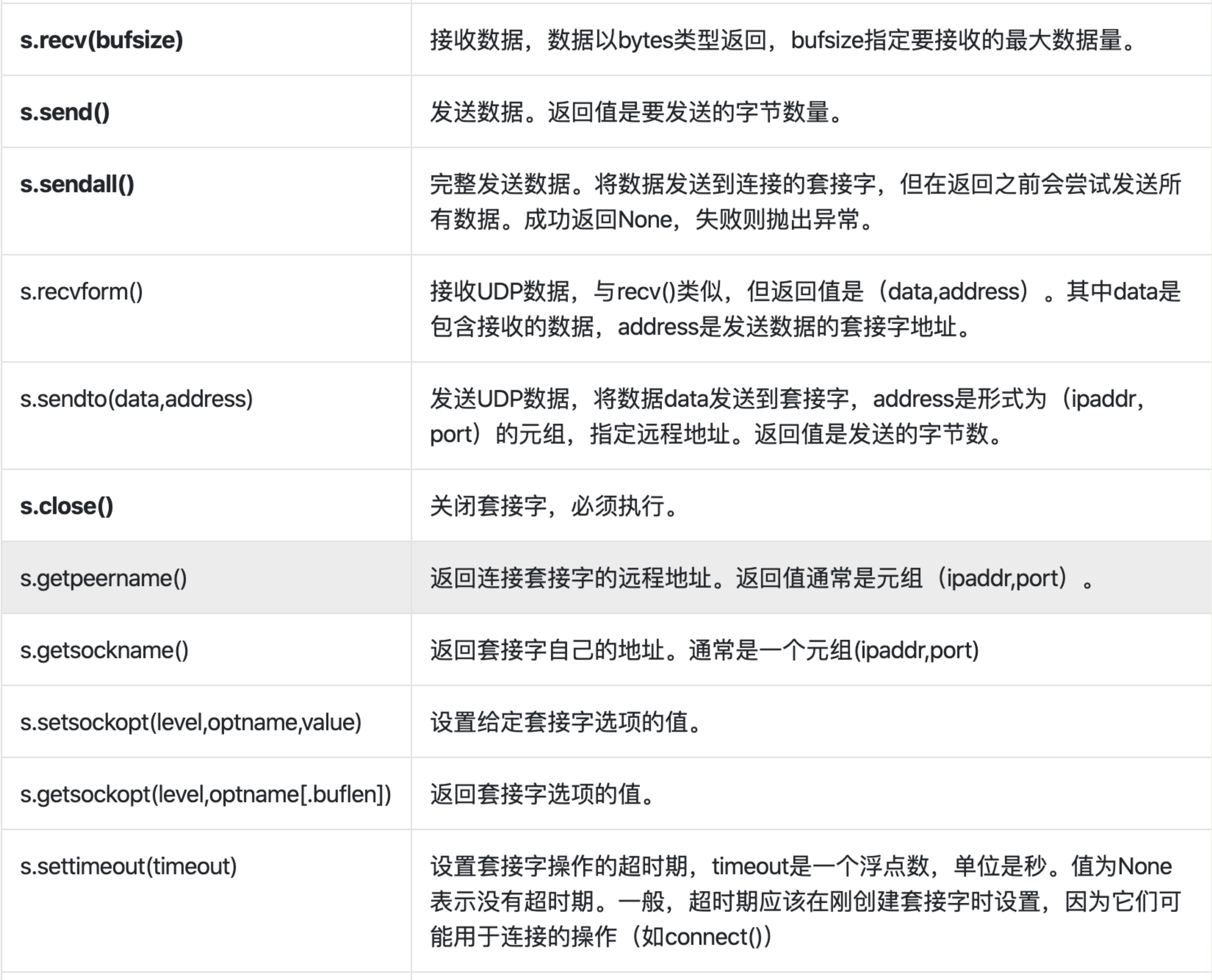
最后关闭连接，一次交互结束。



服务器端方法	
s.bind()	绑定地址（host,port）到套接字，在AF_INET下,以元组（host,port）的形式表示地址。
s.listen(backlog)	开始监听。backlog指定在拒绝连接之前，操作系统可以挂起的最大连接数量。该值至少为1，大部分应用程序设为5就可以了。
s.accept()	被动接受客户端连接,(阻塞式)等待连接的到来，并返回（conn,address）二元元组,其中conn是一个通信对象，可以用来接收和发送数据。address是连接客户端的地址。



客户端方法	
s.connect(address)	客户端向服务端发起连接。一般address的格式为元组（hostname,port），如果连接出错，返回socket.error错误。
s.connect_ex()	connect()函数的扩展版本,出错时返回出错码,而不是抛出异常



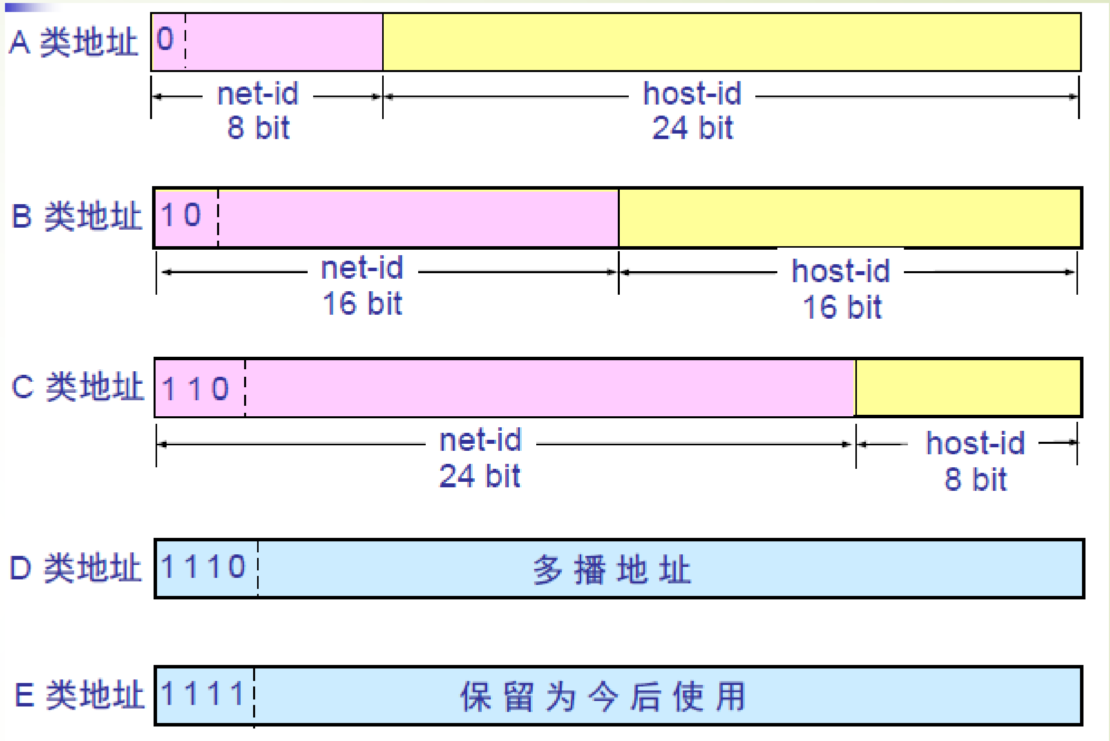
s.recv(bufsize)	接收数据，数据以bytes类型返回，bufsize指定要接收的最大数据量。
s.send()	发送数据。返回值是要发送的字节数量。
s.sendall()	完整发送数据。将数据发送到连接的套接字，但在返回之前会尝试发送所有数据。成功返回None，失败则抛出异常。
s.recvform()	接收UDP数据，与recv()类似，但返回值是（data,address）。其中data是包含接收的数据，address是发送数据的套接字地址。
s.sendto(data,address)	发送UDP数据，将数据data发送到套接字，address是形式为（ipaddr，port）的元组，指定远程地址。返回值是发送的字节数。
s.close()	关闭套接字，必须执行。
s.getpeername()	返回连接套接字的远程地址。返回值通常是元组（ipaddr,port）。
s.getsockname()	返回套接字自己的地址。通常是一个元组(ipaddr,port)
s.setsockopt(level,optname,value)	设置给定套接字选项的值。
s.getsockopt(level,optname[.buflen])	返回套接字选项的值。
s.settimeout(timeout)	设置套接字操作的超时期，timeout是一个浮点数，单位是秒。值为None表示没有超时期。一般，超时期应该在刚创建套接字时设置，因为它们可能用于连接的操作（如connect()）

- IP地址：IPv4 – 32位，IPv6 – 128位
- IP地址分类：每个地址由两个固定长度的字段组成，网络号net-id标志主机所连接到的网络，主机号host-id标志该主机。

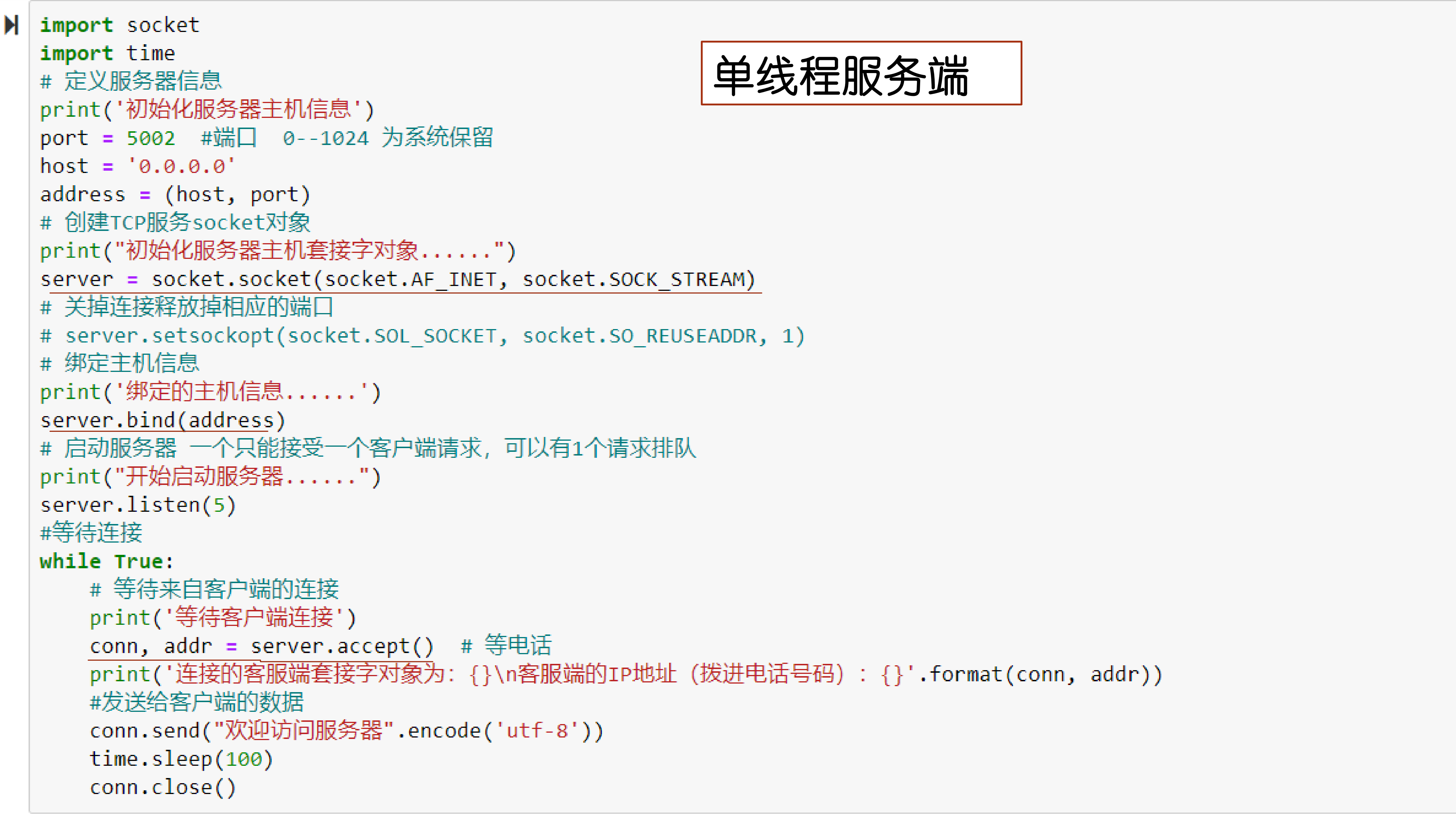
127.0.0.1和0.0.0.0的区别：

回环地址127.x.x.x：该范围内的任何地址都将环回到本地主机中，不会出现在任何网络中。主要用来做回环测试。

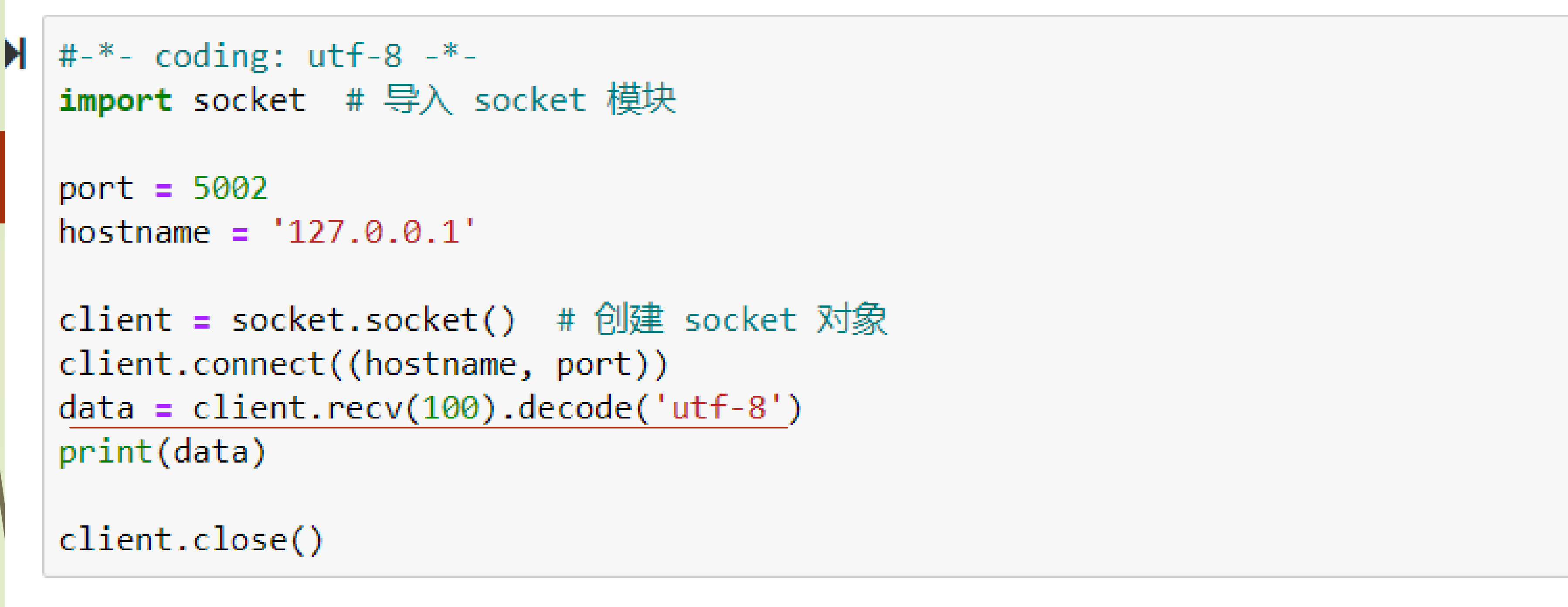
0.0.0.0：任何地址，包括了环回地址。不管主机有多少个网口，多少个IP，如果监听本机的0.0.0.0上的端口，就等于监听机器上的所有IP端口。数据报的目的地址只要是机器上的一个IP地址，就能被接受。



单线程服务端

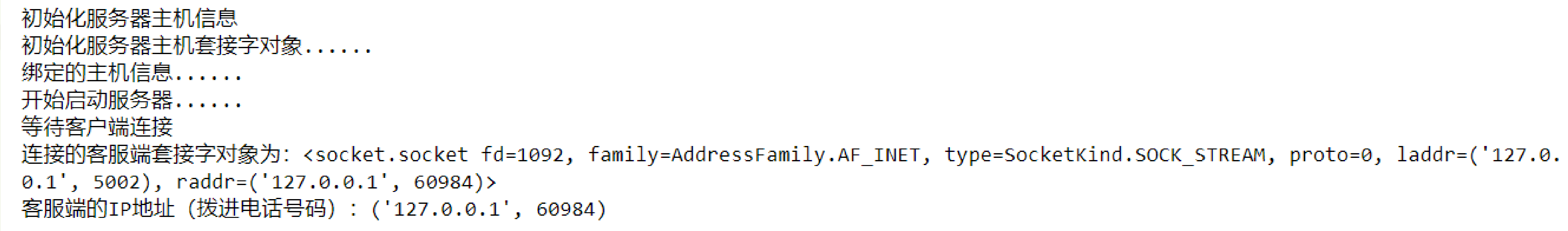


```
import socket
import time
# 定义服务器信息
print('初始化服务器主机信息')
port = 5002  #端口  0--1024 为系统保留
host = '0.0.0.0'
address = (host, port)
# 创建TCP服务socket对象
print("初始化服务器主机套接字对象......")
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# 关掉连接释放掉相应的端口
# server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
# 绑定主机信息
print('绑定的主机信息......')
server.bind(address)
# 启动服务器 一个只能接受一个客户端请求，可以有1个请求排队
print("开始启动服务器......")
server.listen(5)
#等待连接
while True:
    # 等待来自客户端的连接
    print('等待客户端连接')
    conn, addr = server.accept()  # 等电话
    print('连接的客服端套接字对象为：{}\n客服端的IP地址（拨进电话号码）：{}'.format(conn, addr))
    #发送给客户端的数据
    conn.send("欢迎访问服务器".encode('utf-8'))
    time.sleep(100)
    conn.close()
```



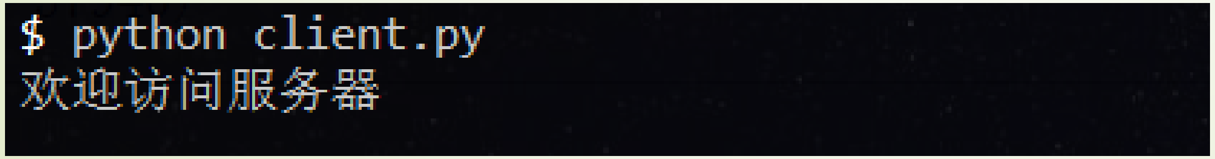
```python
#-*- coding: utf-8 -*-
import socket  # 导入 socket 模块

port = 5002
hostname = '127.0.0.1'

client = socket.socket()  # 创建 socket 对象
client.connect((hostname, port))
data = client.recv(100).decode('utf-8')
print(data)

client.close()
```

服务端输出：



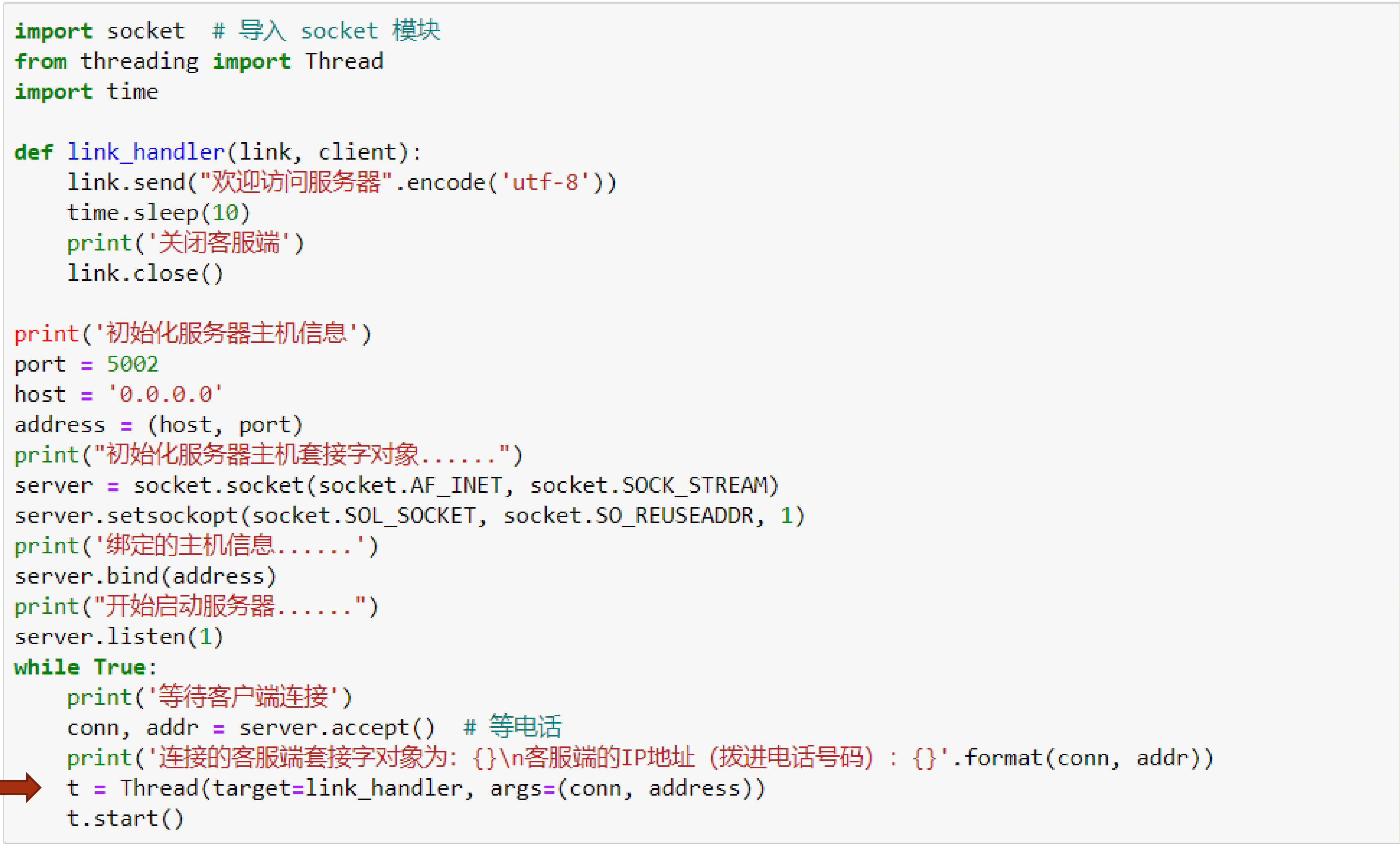
```
初始化服务器主机信息
初始化服务器主机套接字对象......
绑定的主机信息......
开始启动服务器......
等待客户端连接
连接的客服端套接字对象为：<socket.socket fd=1092, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('127.0.0.1', 5002), raddr=('127.0.0.1', 60984)>
客服端的IP地址（拨进电话号码）：('127.0.0.1', 60984)
```

客户端输出：



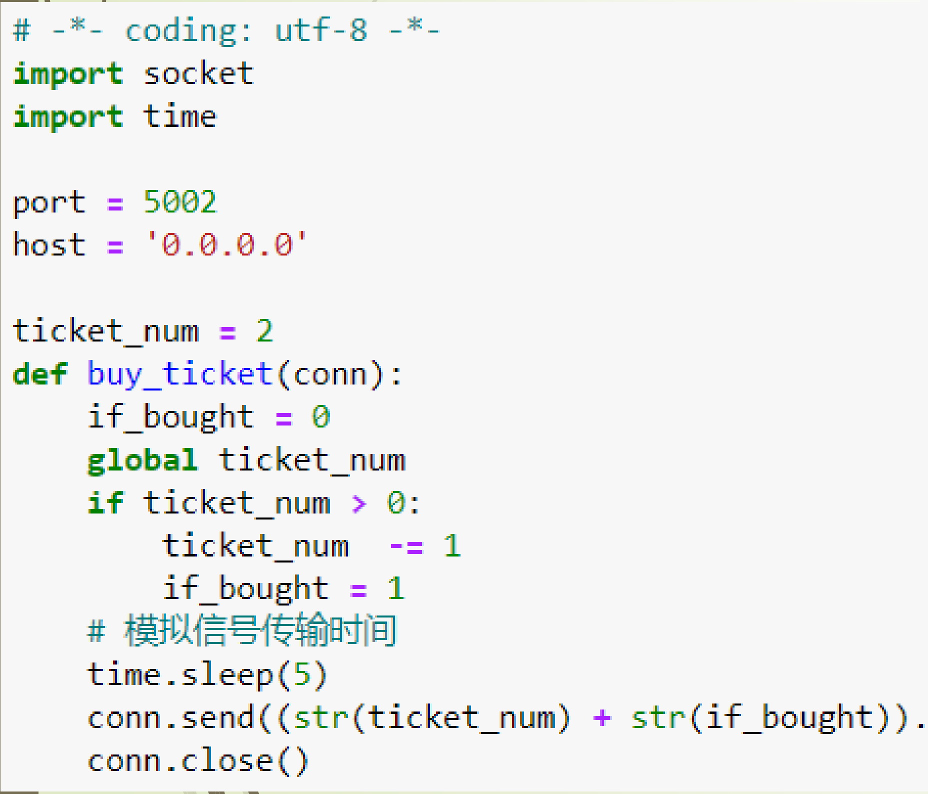
```
$ python client.py
欢迎访问服务器
```

多线程服务端

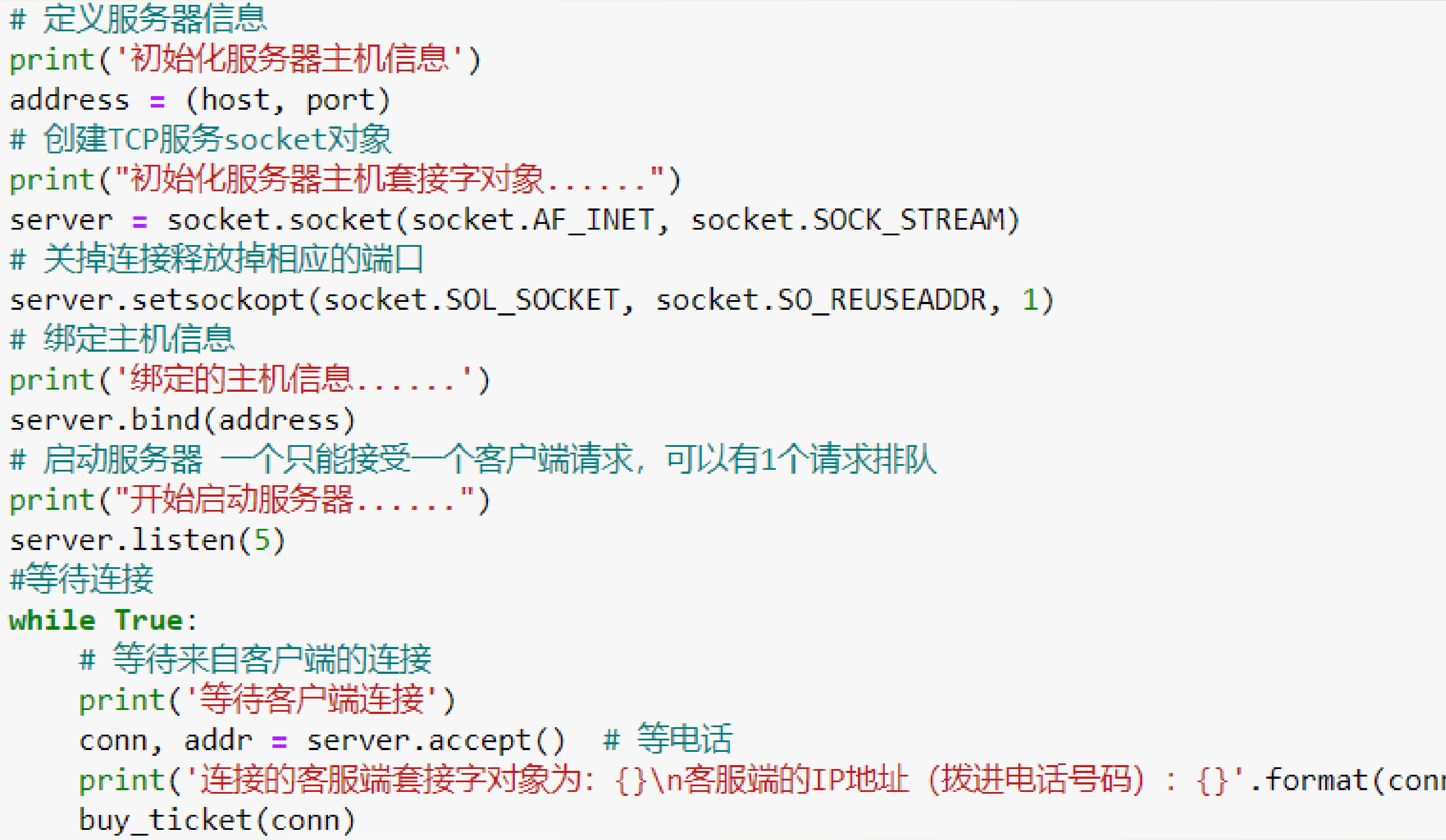


```python
import socket  # 导入 socket 模块
from threading import Thread
import time

def link_handler(link, client):
    link.send("欢迎访问服务器".encode('utf-8'))
    time.sleep(10)
    print('关闭客服端')
    link.close()

print('初始化服务器主机信息')
port = 5002
host = '0.0.0.0'
address = (host, port)
print("初始化服务器主机套接字对象......")
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
print('绑定的主机信息......')
server.bind(address)
print("开始启动服务器......")
server.listen(1)
while True:
    print('等待客户端连接')
    conn, addr = server.accept()  # 等电话
    print('连接的客服端套接字对象为：{}\n客服端的IP地址（拨进电话号码）：{}'.format(conn, addr))
    t = Thread(target=link_handler, args=(conn, address))
    t.start()
```

单进程服务端模拟购票



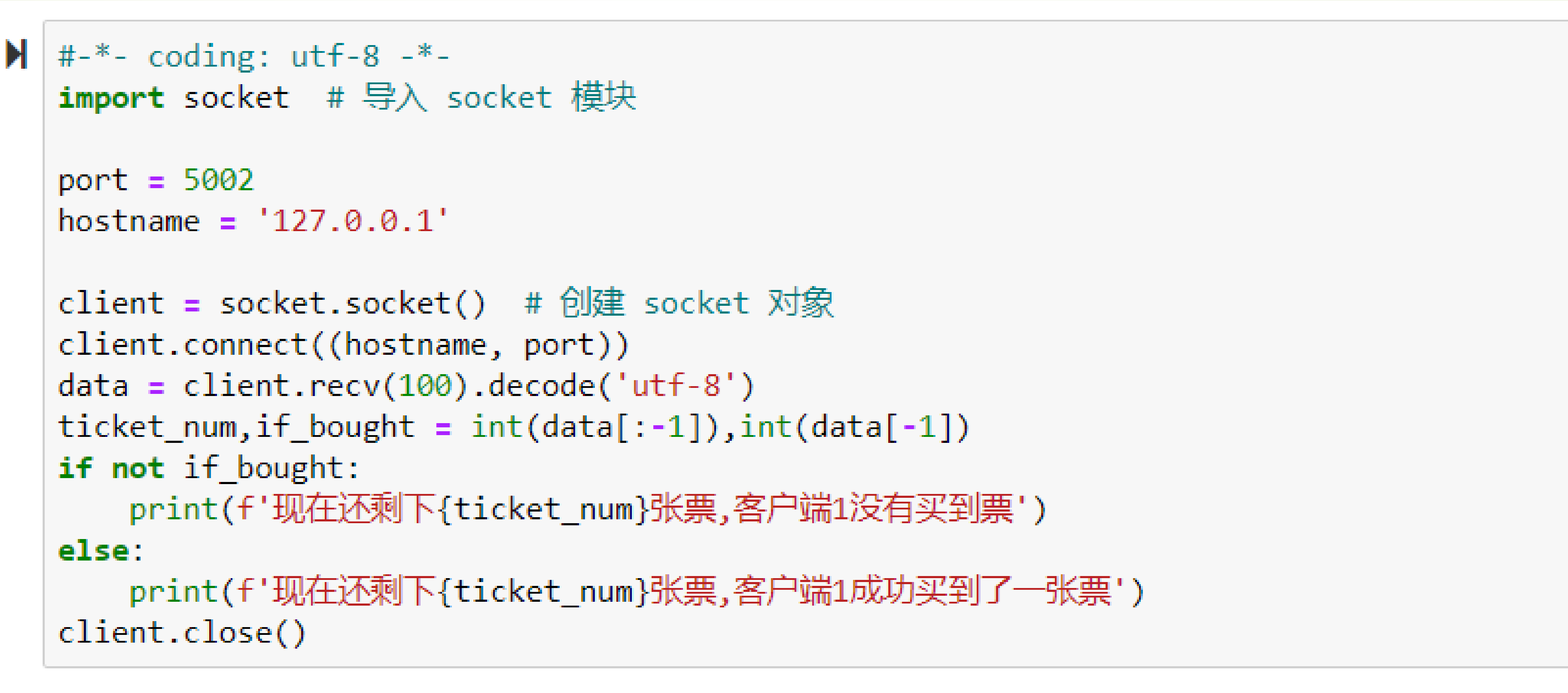
```python
# -*- coding: utf-8 -*-
import socket
import time

port = 5002
host = '0.0.0.0'

ticket_num = 2
def buy_ticket(conn):
    if_bought = 0
    global ticket_num
    if ticket_num > 0:
        ticket_num  -= 1
        if_bought = 1
    # 模拟信号传输时间
    time.sleep(5)
    conn.send((str(ticket_num) + str(if_bought)).encode('utf-8'))
    conn.close()
```



```python
# 定义服务器信息
print('初始化服务器主机信息')
address = (host, port)
# 创建TCP服务socket对象
print("初始化服务器主机套接字对象......")
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# 关掉连接释放掉相应的端口
server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
# 绑定主机信息
print('绑定的主机信息......')
server.bind(address)
# 启动服务器 一个只能接受一个客户端请求，可以有1个请求排队
print("开始启动服务器......")
server.listen(5)
#等待连接
while True:
    # 等待来自客户端的连接
    print('等待客户端连接')
    conn, addr = server.accept()  # 等电话
    print('连接的客服端套接字对象为：{}\n客服端的IP地址（拨进电话号码）：{}'.format(conn
    buy_ticket(conn)
```

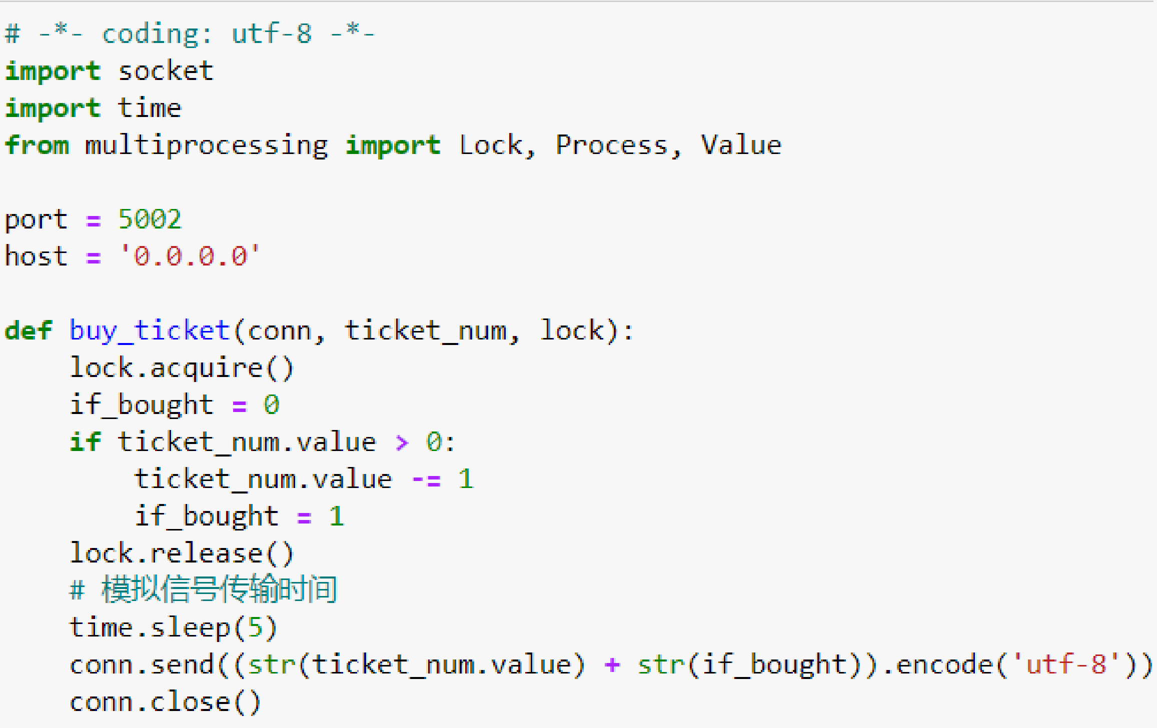
弊端：顺序，一个客户端堵塞会影响其余客户端

客户端

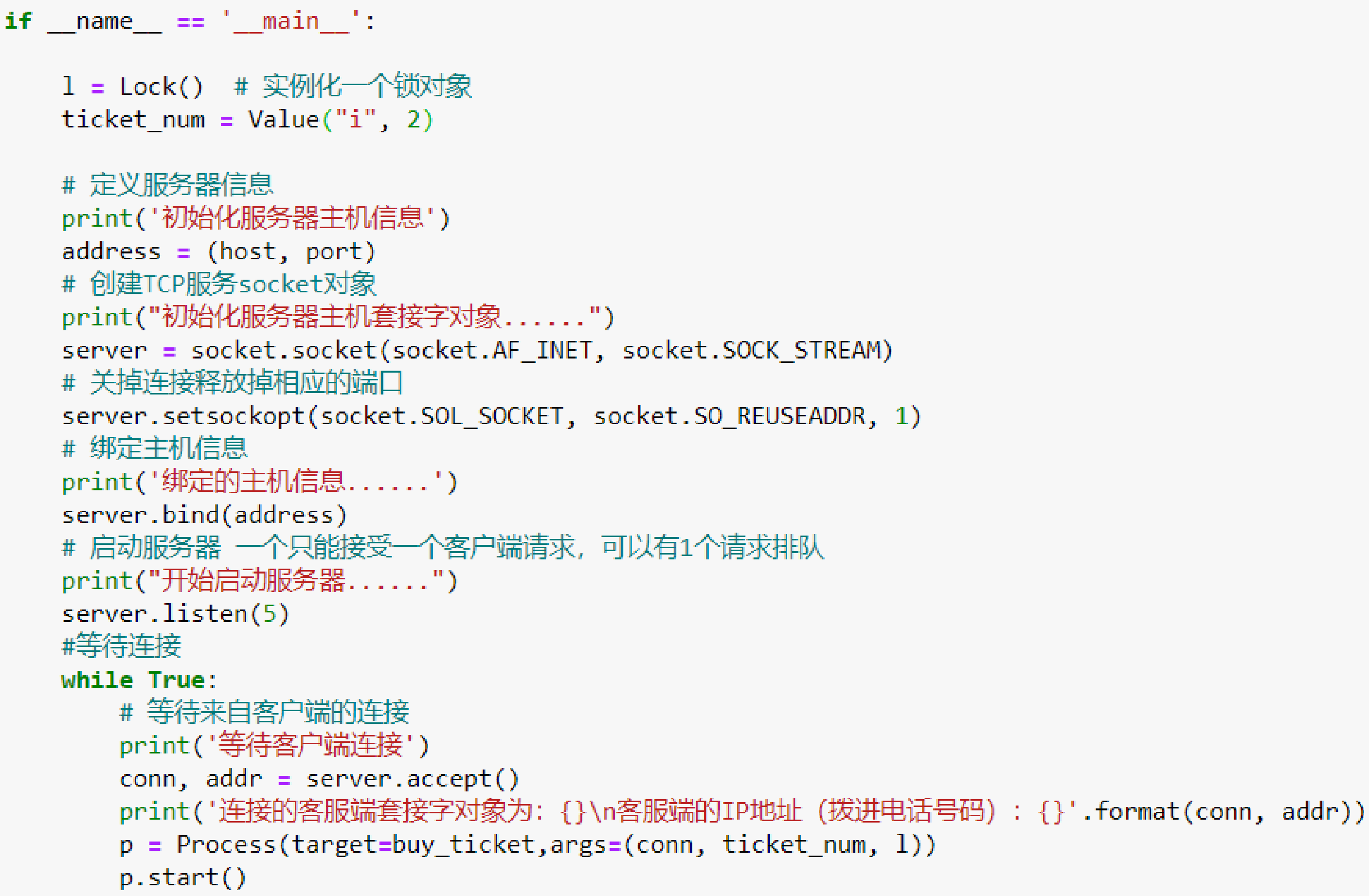


```
#-*- coding: utf-8 -*-
import socket  # 导入 socket 模块

port = 5002
hostname = '127.0.0.1'

client = socket.socket()  # 创建 socket 对象
client.connect((hostname, port))
data = client.recv(100).decode('utf-8')
ticket_num,if_bought = int(data[:-1]),int(data[-1])
if not if_bought:
    print(f'现在还剩下{ticket_num}张票,客户端1没有买到票')
else:
    print(f'现在还剩下{ticket_num}张票,客户端1成功买到了一张票')
client.close()
```

多进程服务端

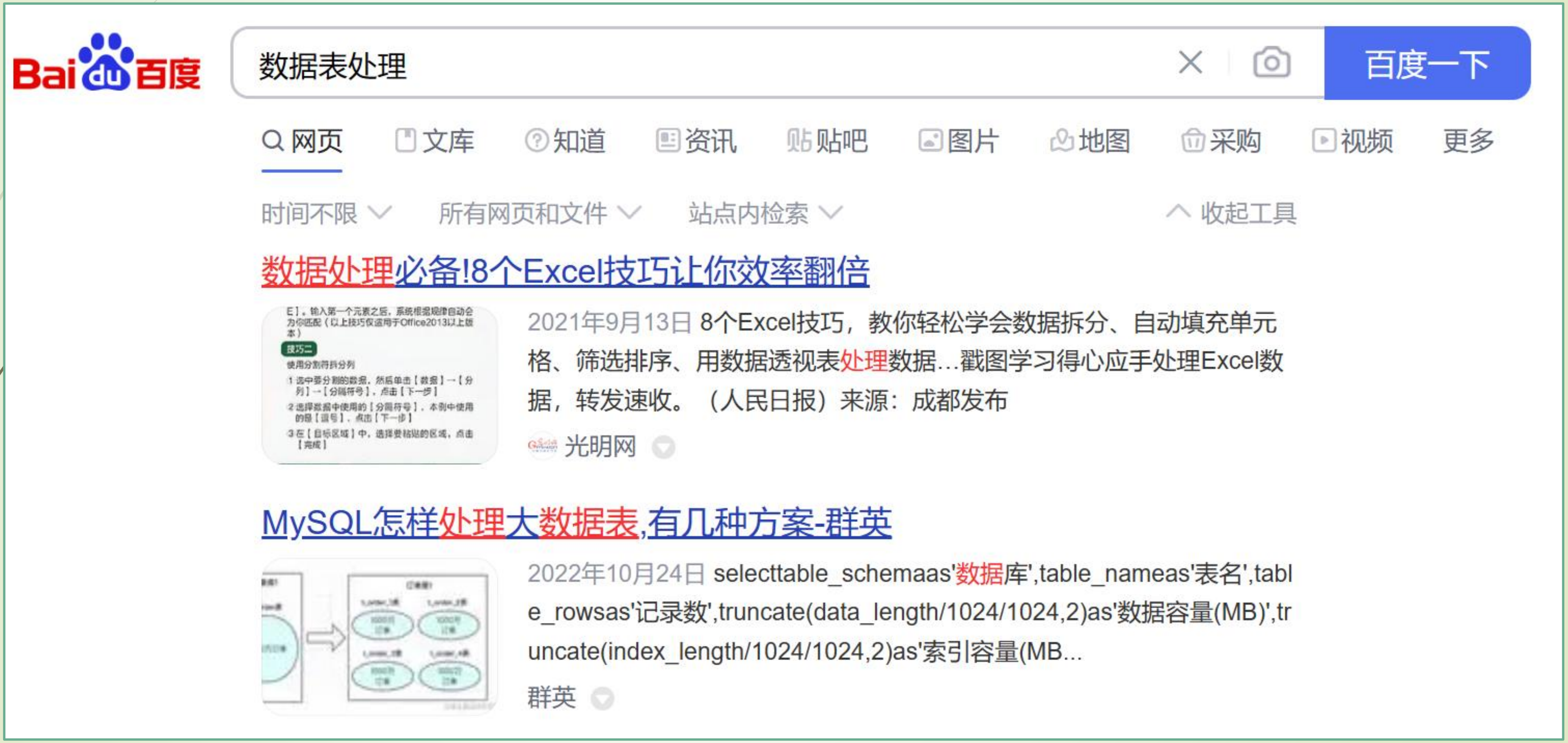


```python
# -*- coding: utf-8 -*-
import socket
import time
from multiprocessing import Lock, Process, Value

port = 5002
host = '0.0.0.0'

def buy_ticket(conn, ticket_num, lock):
    lock.acquire()
    if_bought = 0
    if ticket_num.value > 0:
        ticket_num.value -= 1
        if_bought = 1
    lock.release()
    # 模拟信号传输时间
    time.sleep(5)
    conn.send((str(ticket_num.value) + str(if_bought)).encode('utf-8'))
    conn.close()
```



```python
if __name__ == '__main__':

    l = Lock()  # 实例化一个锁对象
    ticket_num = Value("i", 2)

    # 定义服务器信息
    print('初始化服务器主机信息')
    address = (host, port)
    # 创建TCP服务socket对象
    print("初始化服务器主机套接字对象......")
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # 关掉连接释放掉相应的端口
    server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    # 绑定主机信息
    print('绑定的主机信息......')
    server.bind(address)
    # 启动服务器 一个只能接受一个客户端请求，可以有1个请求排队
    print("开始启动服务器......")
    server.listen(5)
    #等待连接
    while True:
        # 等待来自客户端的连接
        print('等待客户端连接')
        conn, addr = server.accept()
        print('连接的客服端套接字对象为：{}\n客服端的IP地址（拨进电话号码）：{}'.format(conn, addr))
        p = Process(target=buy_ticket,args=(conn, ticket_num, l))
        p.start()
```

Pandas：数据表处理 + 数据表关联处理



数据处理必备!8个Excel技巧让你效率翻倍

2021年9月13日 8个Excel技巧，教你轻松学会数据拆分、自动填充单元格、筛选排序、用数据透视表处理数据...戳图学习得心应手处理Excel数据，转发速收。（人民日报）来源：成都发布

光明网

MySQL怎样处理大数据表,有几种方案-群英

2022年10月24日 selecttable_schemaas'数据库',table_nameas'表名',table_rowsas'记录数',truncate(data_length/1024/1024,2)as'数据容量(MB)',truncate(index_length/1024/1024,2)as'索引容量(MB...

群英

数据表处理常用功能：

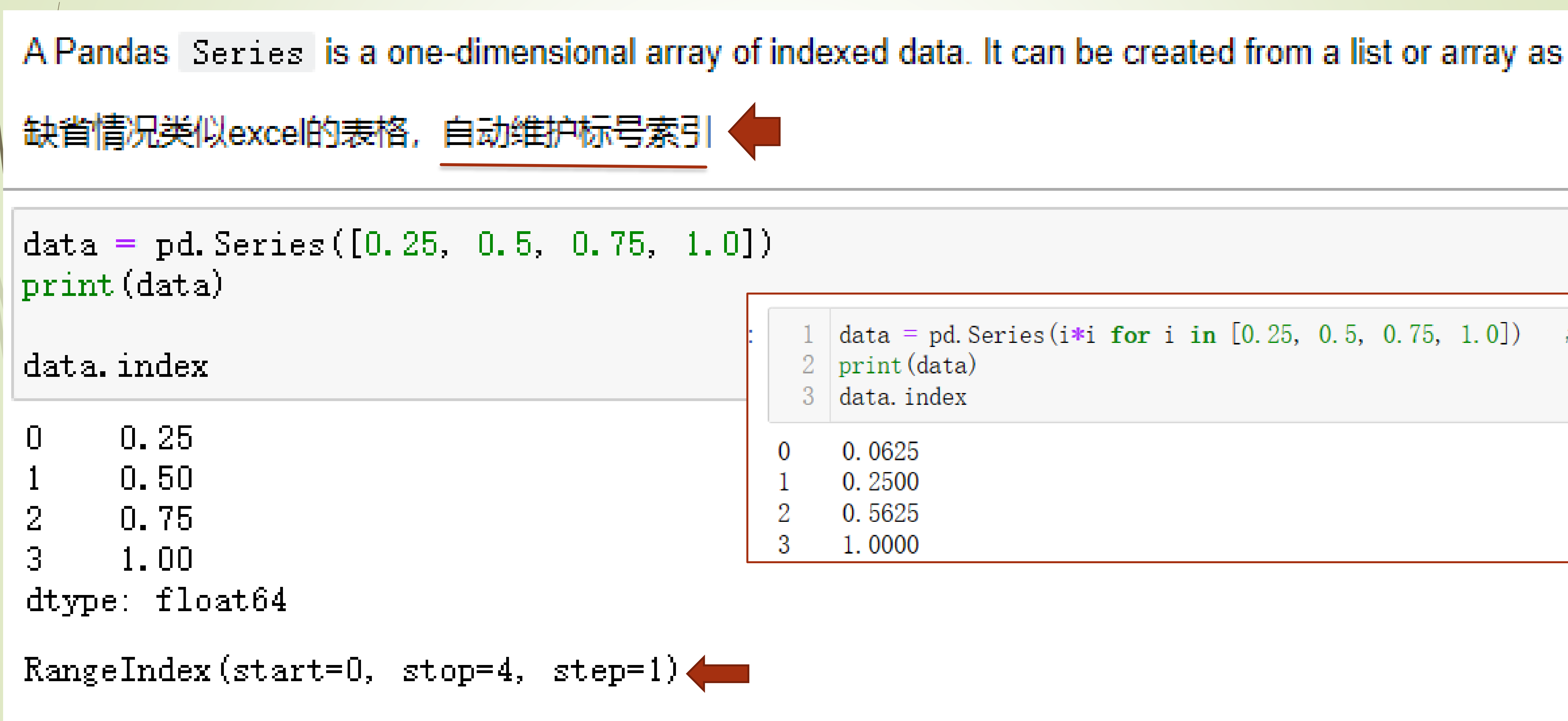
- 管理具有多个字段的记录列表
- 查找、筛选、处理元素单元
- 实现字段间（逐元素）的运算，生成新字段

- 数据表的拼接，堆叠
- 数据统计

Pandas —— Panel data analysis

- 序列：indexed list
- 多通道序列：record list
- 多字段二维表
- 表关联运算
- Pandas的统计功能与应用

The Pandas Series Object —— 序列



A Pandas `Series` is a one-dimensional array of indexed data. It can be created from a list or array as

缺省情况类似excel的表格，自动维护标号索引

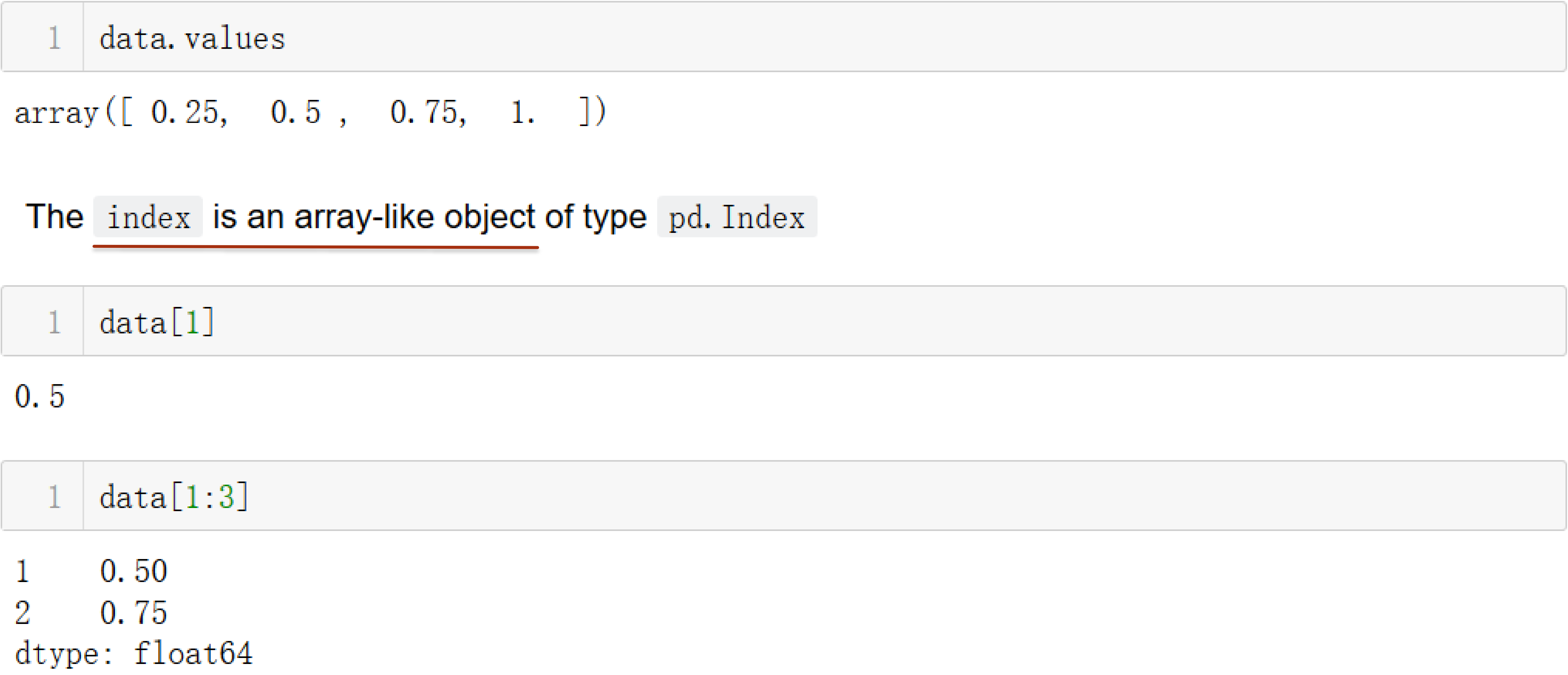
```
data = pd.Series([0.25, 0.5, 0.75, 1.0])
print(data)

data.index
```

```
0    0.25
1    0.50
2    0.75
3    1.00
dtype: float64

RangeIndex(start=0, stop=4, step=1)
```

```
data = pd.Series(i*i for i in [0.25, 0.5, 0.75, 1.0])
print(data)
data.index
```

```
0    0.0625
1    0.2500
2    0.5625
3    1.0000
```

与数组类似，支持下标切片访问操作



```
data.values
```

```
array([ 0.25,  0.5 ,  0.75,  1.  ])
```

The `index` is an array-like object of type `pd.Index`

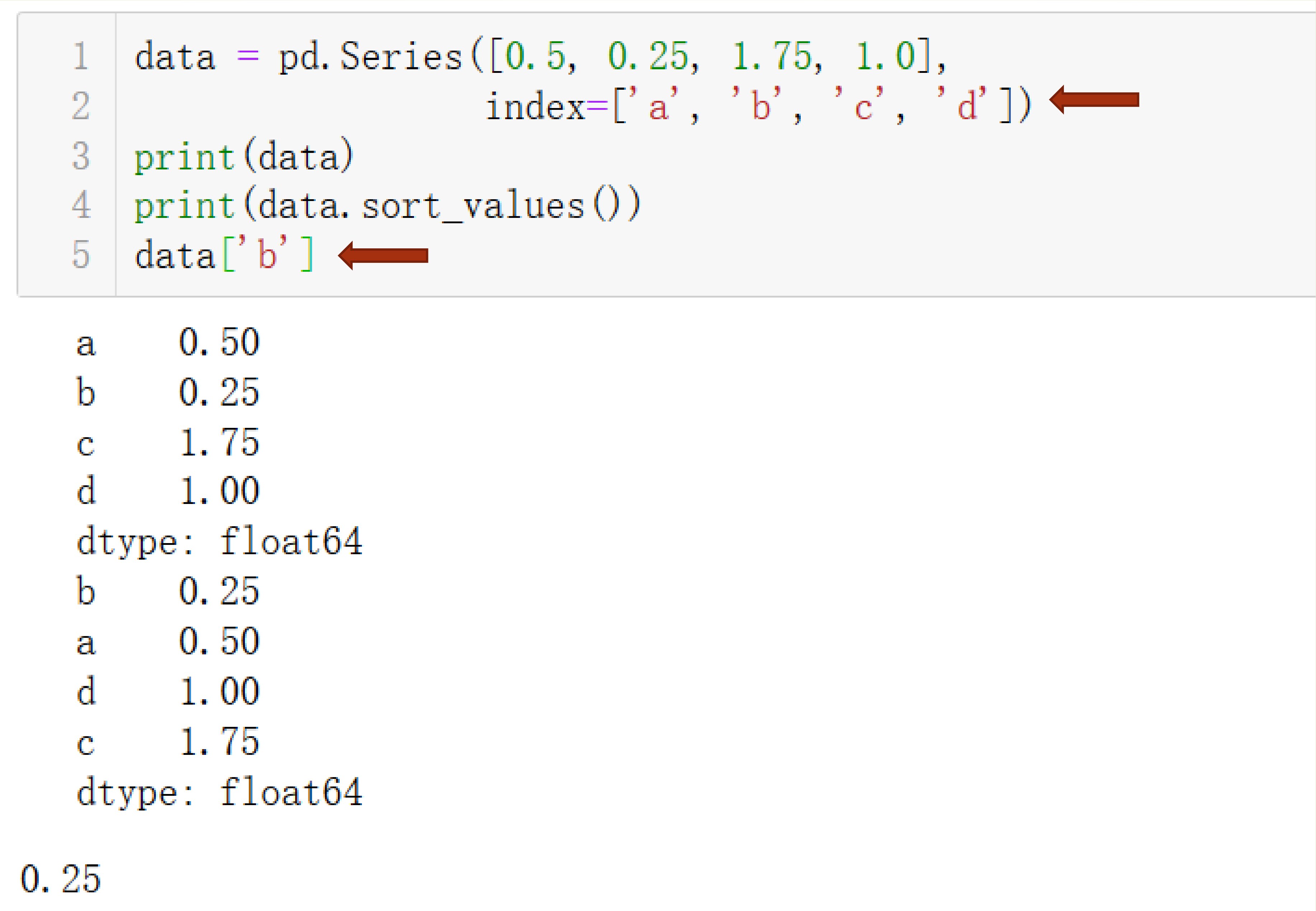
```
data[1]
```

```
0.5
```

```
data[1:3]
```

```
1    0.50
2    0.75
dtype: float64
```

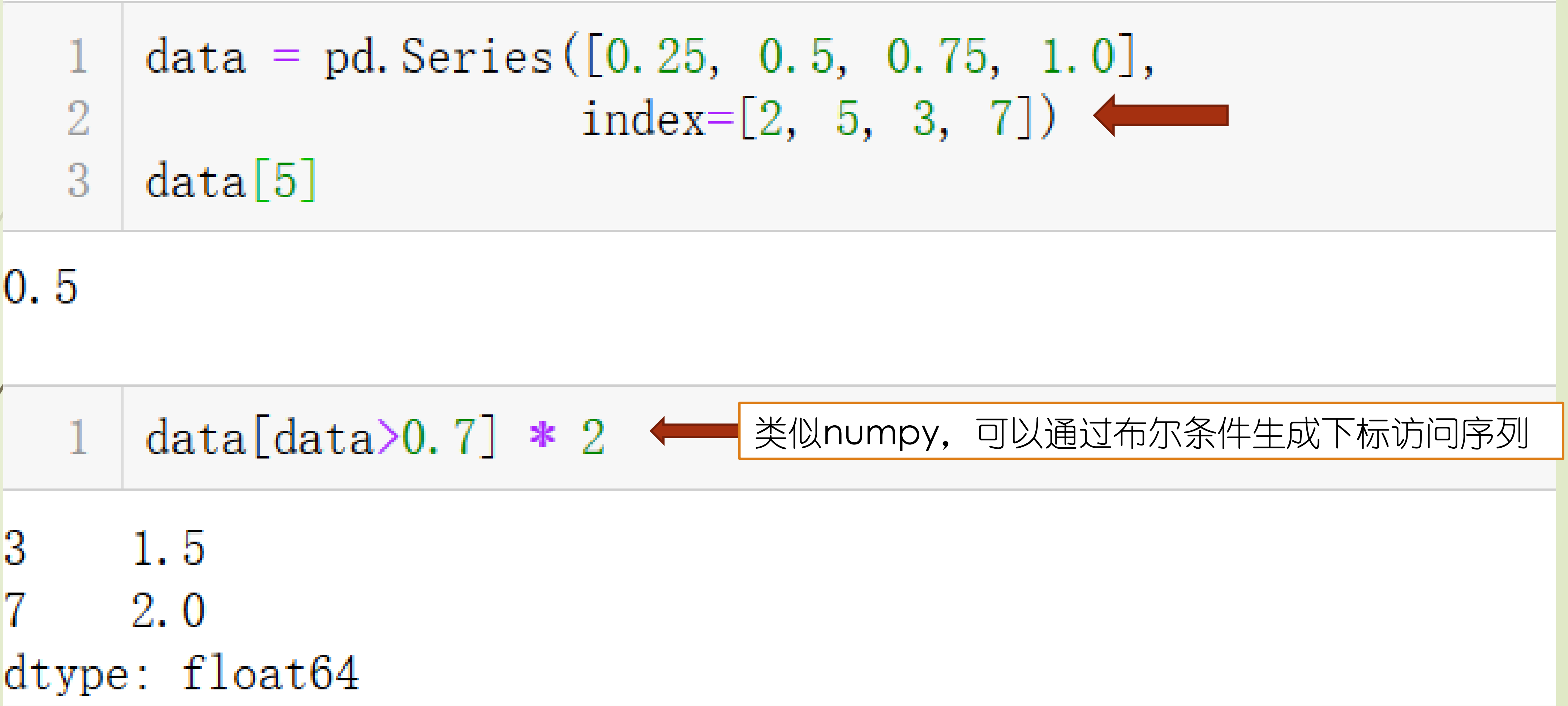
也可以指定可哈希的索引项，类似dict



```
data = pd.Series([0.5, 0.25, 1.75, 1.0],
                  index=['a', 'b', 'c', 'd'])
print(data)
print(data.sort_values())
data['b']
```

```
a    0.50
b    0.25
c    1.75
d    1.00
dtype: float64
b    0.25
a    0.50
d    1.00
c    1.75
dtype: float64

0.25
```

非连续的索引项也没可以，但一般建议避免非连续数字



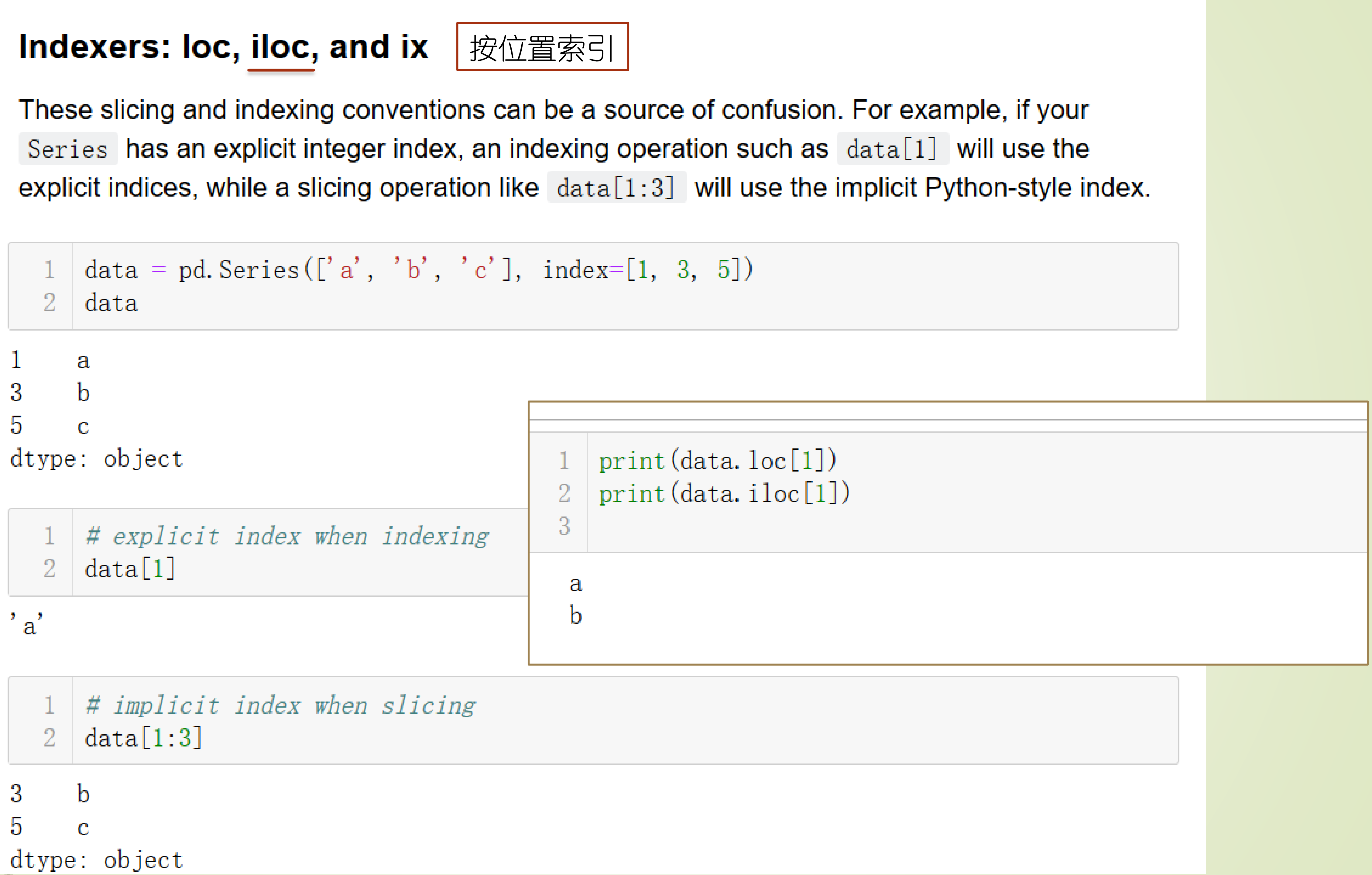
```
data = pd.Series([0.25, 0.5, 0.75, 1.0],
                 index=[2, 5, 3, 7])
data[5]
```

```
0.5
```

```
data[data>0.7] * 2
```

类似numpy，可以通过布尔条件生成下标访问序列

```
3    1.5
7    2.0
dtype: float64
```



Indexers: loc, iloc, and ix

按位置索引

These slicing and indexing conventions can be a source of confusion. For example, if your `Series` has an explicit integer index, an indexing operation such as `data[1]` will use the explicit indices, while a slicing operation like `data[1:3]` will use the implicit Python-style index.

```python
data = pd.Series(['a', 'b', 'c'], index=[1, 3, 5])
data
```

```
1    a
3    b
5    c
dtype: object
```

```python
# explicit index when indexing
data[1]
```

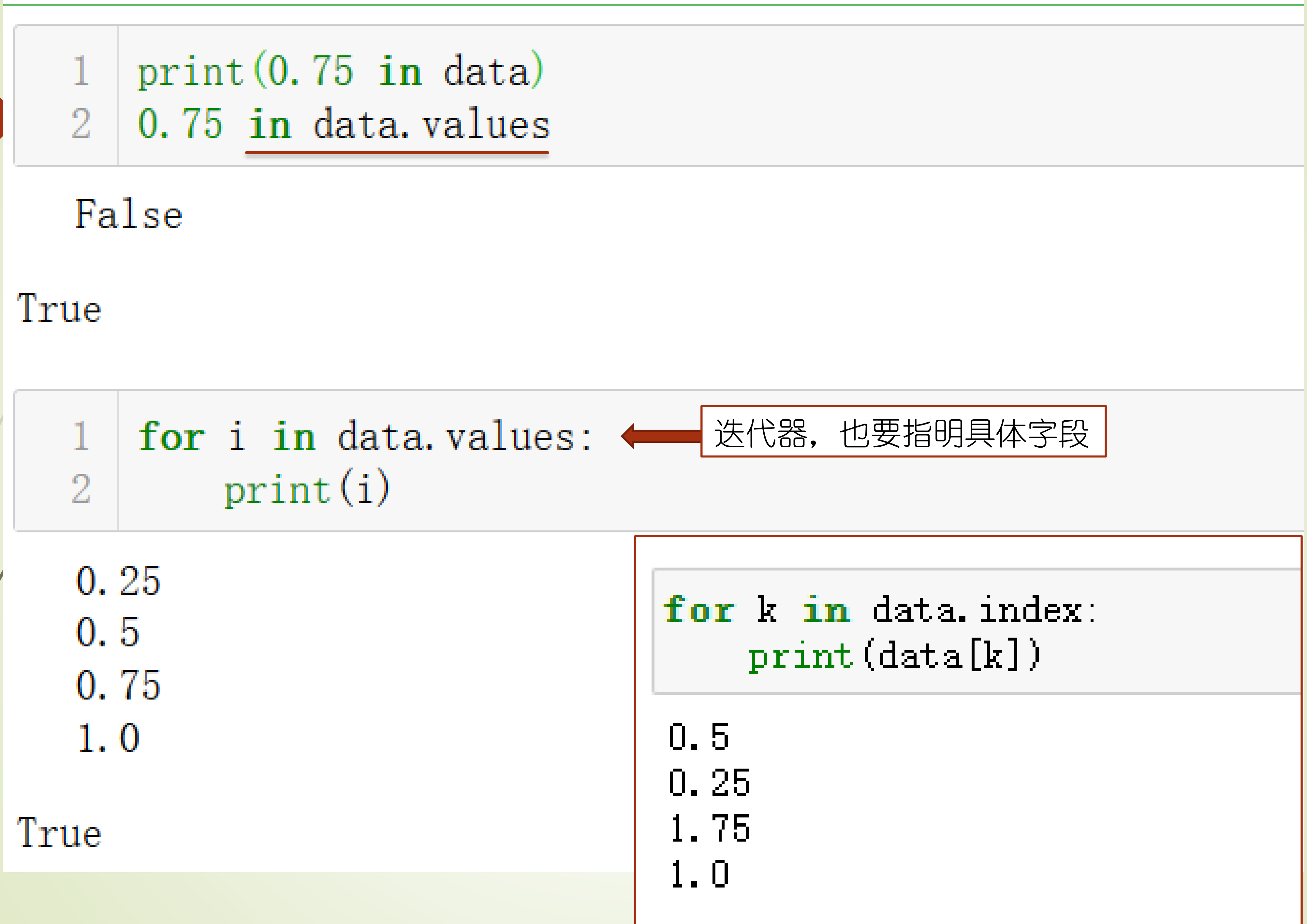
```
'a'
```

```python
print(data.loc[1])
print(data.iloc[1])

```

```
a
b
```

```python
# implicit index when slicing
data[1:3]
```

```
3    b
5    c
dtype: object
```



```python
print(0.75 in data)
0.75 in data.values
```

```
False

True
```

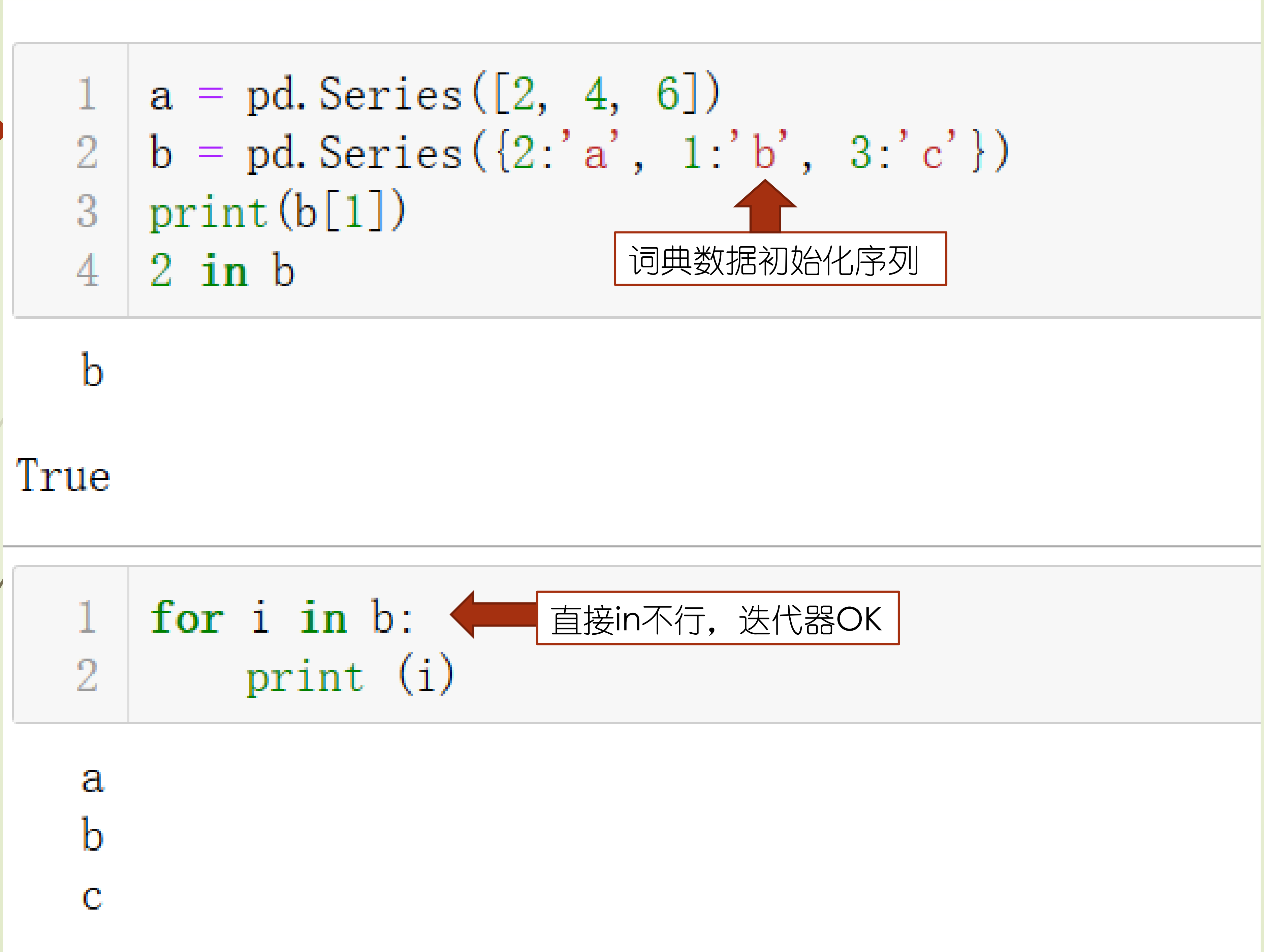
```python
for i in data.values:
    print(i)
```

迭代器，也要指明具体字段

```
0.25
0.5
0.75
1.0

True
```

```python
for k in data.index:
    print(data[k])
```

```
0.5
0.25
1.75
1.0
```



```
a = pd.Series([2, 4, 6])
b = pd.Series({2:’a’, 1:’b’, 3:’c’})
print(b[1])
2 in b
```

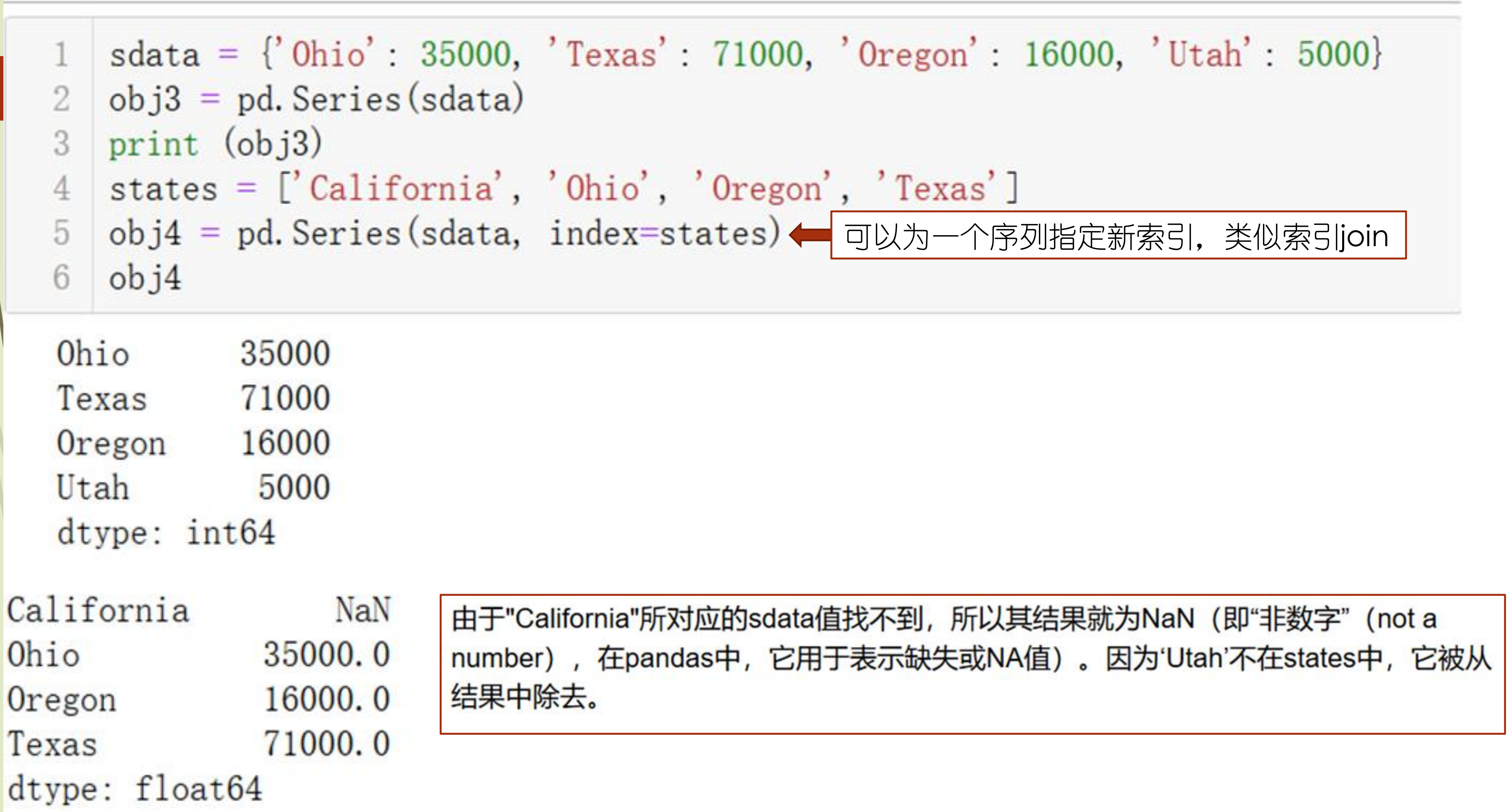
```
b

True
```

词典数据初始化序列

```
for i in b:
    print (i)
```

```
a
b
c
```

直接in不行，迭代器OK



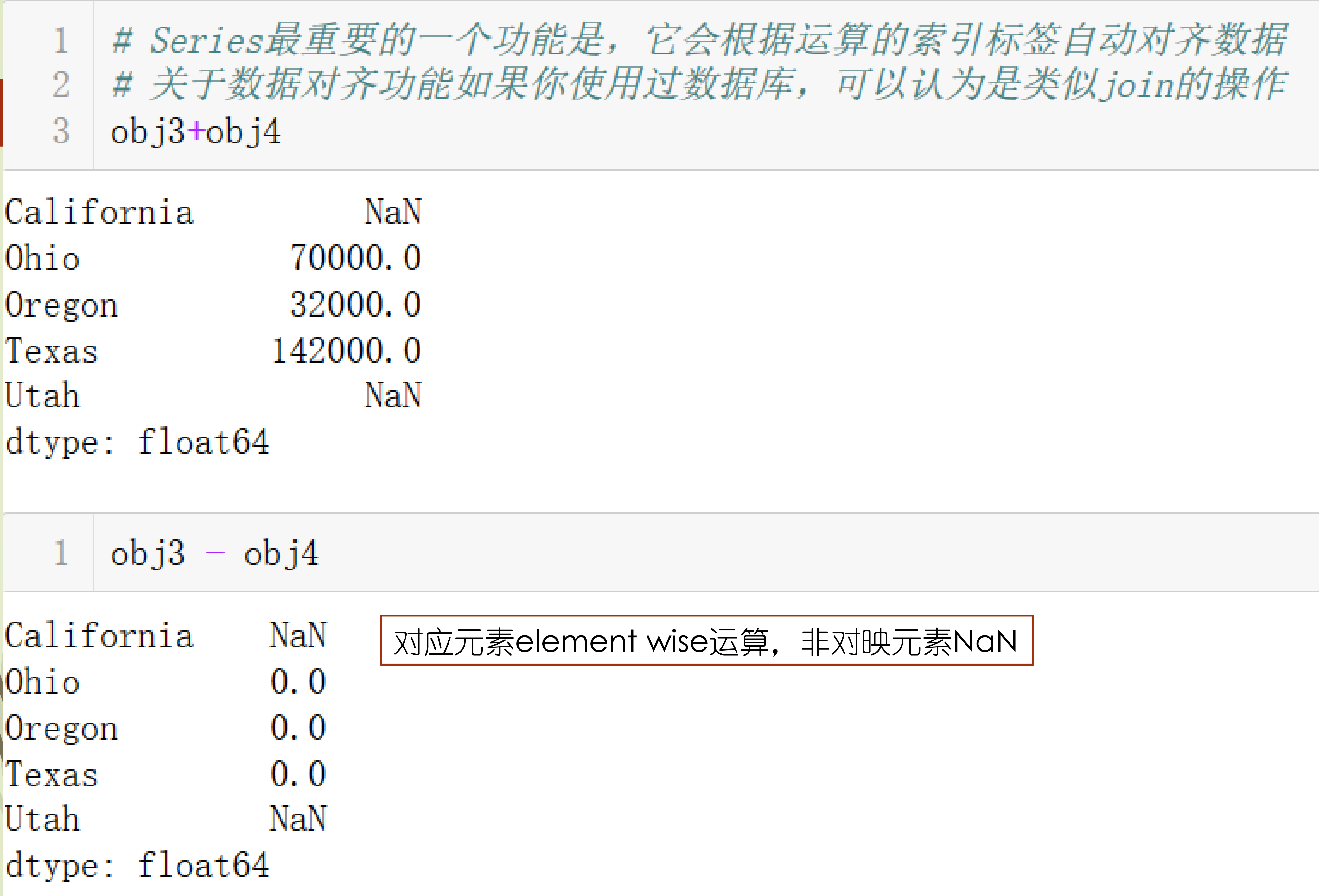
```python
sdata = {'Ohio': 35000, 'Texas': 71000, 'Oregon': 16000, 'Utah': 5000}
obj3 = pd.Series(sdata)
print (obj3)
states = ['California', 'Ohio', 'Oregon', 'Texas']
obj4 = pd.Series(sdata, index=states)
obj4
```

可以为一个序列指定新索引，类似索引join

```
Ohio      35000
Texas     71000
Oregon    16000
Utah       5000
dtype: int64
```

```
California        NaN
Ohio          35000.0
Oregon        16000.0
Texas         71000.0
dtype: float64
```

由于"California"所对应的sdata值找不到，所以其结果就为NaN（即“非数字”（not a number），在pandas中，它用于表示缺失或NA值）。因为‘Utah’不在states中，它被从结果中除去。



```
# Series最重要的一个功能是，它会根据运算的索引标签自动对齐数据
# 关于数据对齐功能如果你使用过数据库，可以认为是类似join的操作
obj3+obj4
```

```
California         NaN
Ohio           70000.0
Oregon         32000.0
Texas         142000.0
Utah               NaN
dtype: float64
```

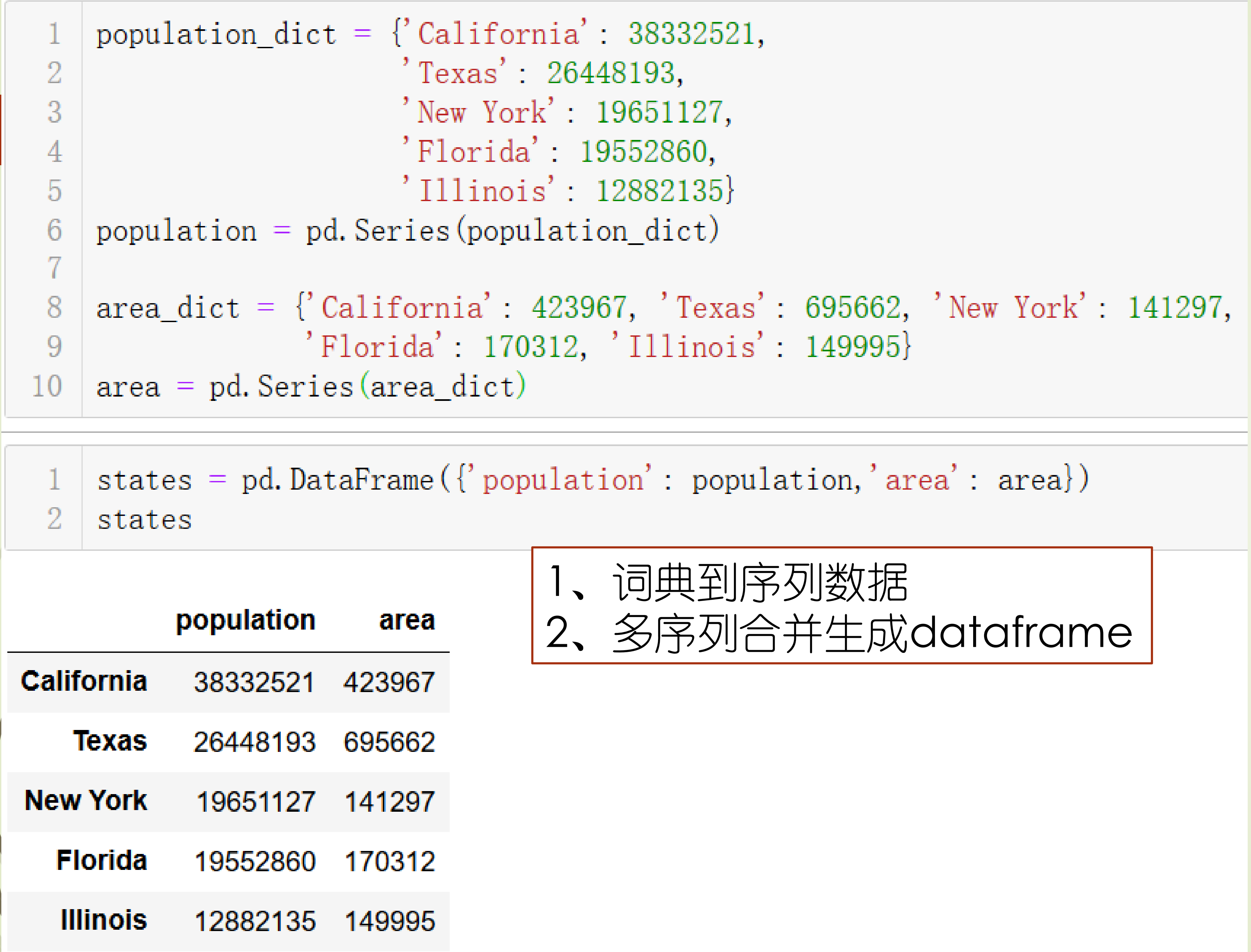
```
obj3 - obj4
```

```
California    NaN
Ohio          0.0
Oregon        0.0
Texas         0.0
Utah          NaN
dtype: float64
```

对应元素element wise运算，非对映元素NaN

The Pandas DataFrame Object

- 视角1：多个对齐的序列（series）的组合（record）
- 视角2：支持多层索引的二维数据表

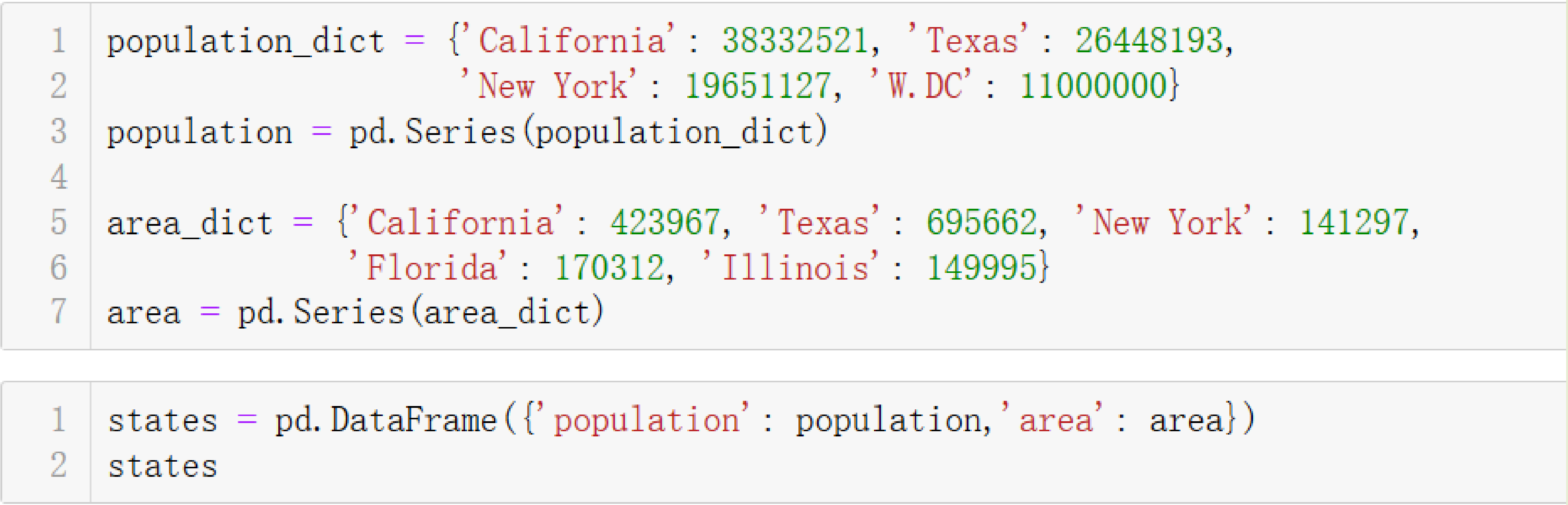


```python
population_dict = {'California': 38332521,
                   'Texas': 26448193,
                   'New York': 19651127,
                   'Florida': 19552860,
                   'Illinois': 12882135}
population = pd.Series(population_dict)

area_dict = {'California': 423967, 'Texas': 695662, 'New York': 141297,
             'Florida': 170312, 'Illinois': 149995}
area = pd.Series(area_dict)
```

```python
states = pd.DataFrame({'population': population,'area': area})
states
```

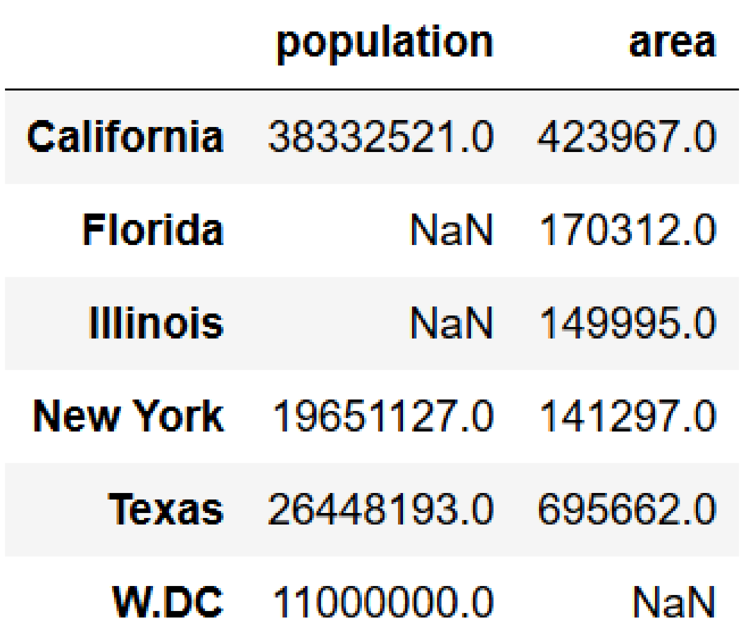
```
            population    area
California    38332521  423967
     Texas    26448193  695662
  New York    19651127  141297
   Florida    19552860  170312
  Illinois    12882135  149995
```

1、词典到序列数据
2、多序列合并生成dataframe



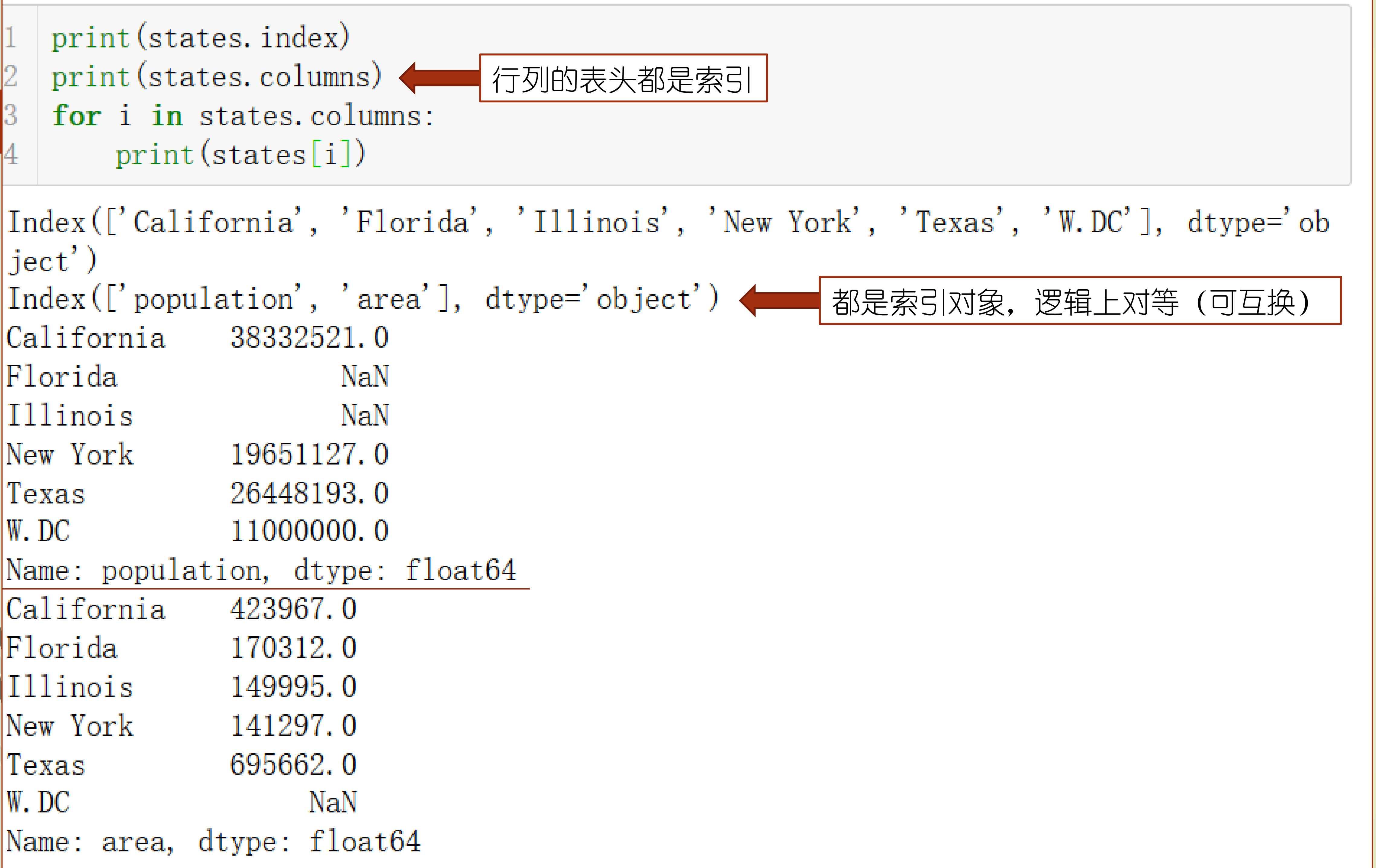
```python
population_dict = {'California': 38332521, 'Texas': 26448193,
                   'New York': 19651127, 'W.DC': 11000000}
population = pd.Series(population_dict)

area_dict = {'California': 423967, 'Texas': 695662, 'New York': 141297,
             'Florida': 170312, 'Illinois': 149995}
area = pd.Series(area_dict)
```

```python
states = pd.DataFrame({'population': population,'area': area})
states
```



	population	area
California	38332521.0	423967.0
Florida	NaN	170312.0
Illinois	NaN	149995.0
New York	19651127.0	141297.0
Texas	26448193.0	695662.0
W.DC	11000000.0	NaN

索引-数据 与 索引合并（非对齐情况）：
索引扩展，数据用NaN填充



```python
print(states.index)
print(states.columns)
for i in states.columns:
    print(states[i])
```

行列的表头都是索引

```
Index(['California', 'Florida', 'Illinois', 'New York', 'Texas', 'W.DC'], dtype='ob
ject')
Index(['population', 'area'], dtype='object')
California    38332521.0
Florida              NaN
Illinois             NaN
New York      19651127.0
Texas         26448193.0
W.DC          11000000.0
Name: population, dtype: float64
California    423967.0
Florida       170312.0
Illinois      149995.0
New York      141297.0
Texas         695662.0
W.DC               NaN
Name: area, dtype: float64
```

都是索引对象，逻辑上对等（可互换）

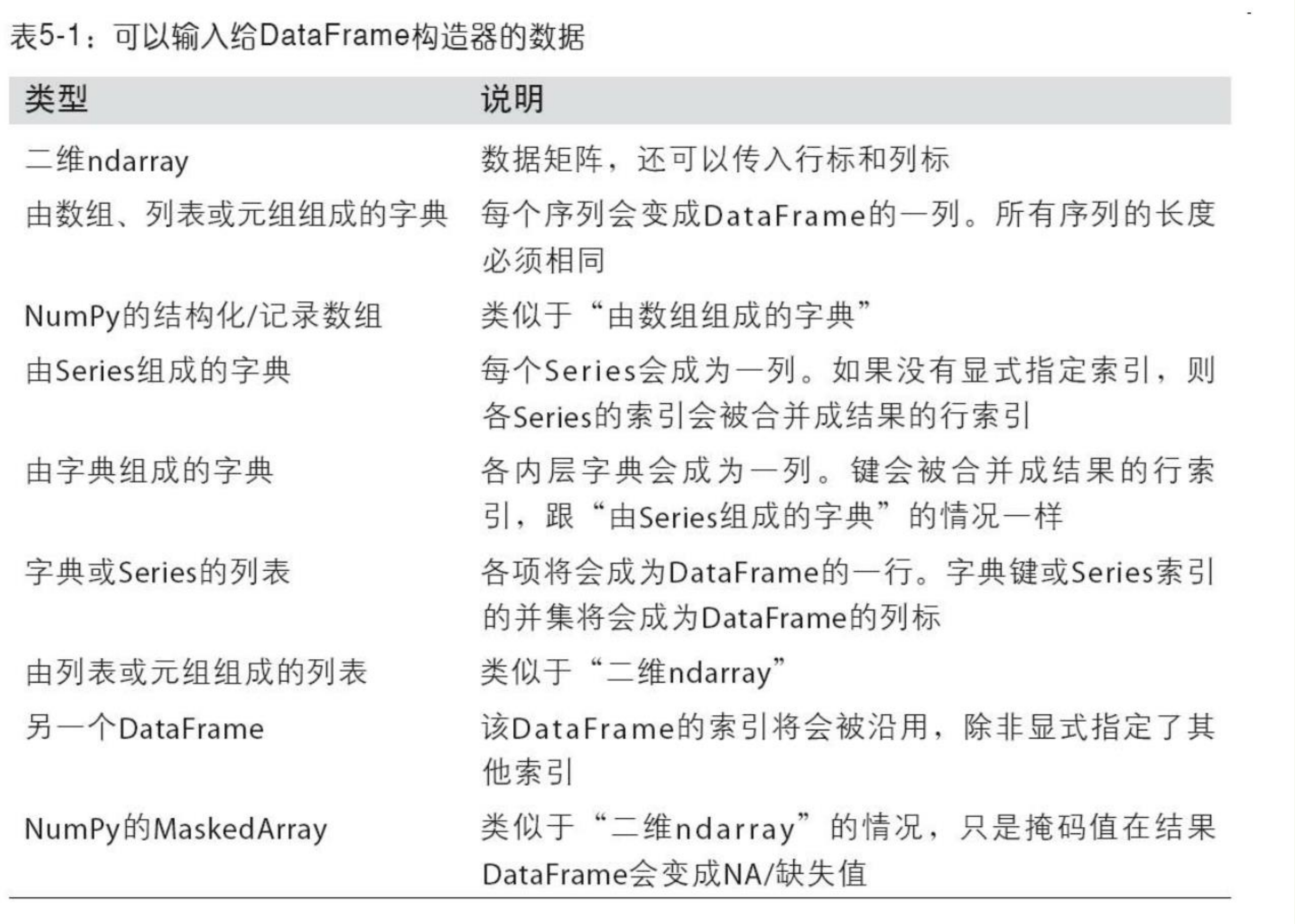
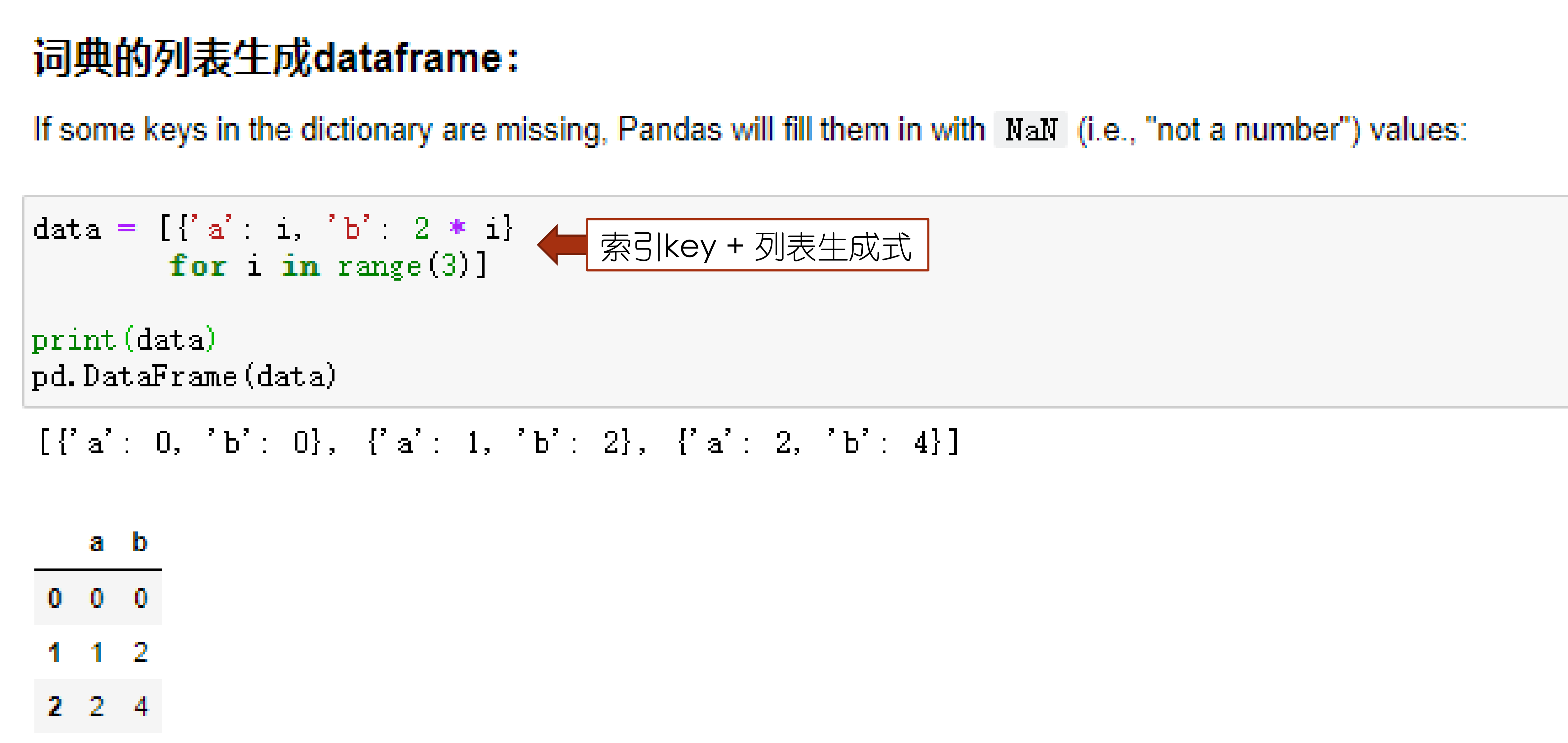


表5-1：可以输入给DataFrame构造器的数据

类型	说明
二维ndarray	数据矩阵，还可以传入行标和列标
由数组、列表或元组组成的字典	每个序列会变成DataFrame的一列。所有序列的长度必须相同
NumPy的结构化/记录数组	类似于“由数组组成的字典”
由Series组成的字典	每个Series会成为一列。如果没有显式指定索引，则各Series的索引会被合并成结果的行索引
由字典组成的字典	各内层字典会成为一列。键会被合并成结果的行索引，跟“由Series组成的字典”的情况一样
字典或Series的列表	各项将会成为DataFrame的一行。字典键或Series索引的并集将会成为DataFrame的列标
由列表或元组组成的列表	类似于“二维ndarray”
另一个DataFrame	该DataFrame的索引将会被沿用，除非显式指定了其他索引
NumPy的MaskedArray	类似于“二维ndarray”的情况，只是掩码值在结果DataFrame会变成NA/缺失值



词典的列表生成dataframe：

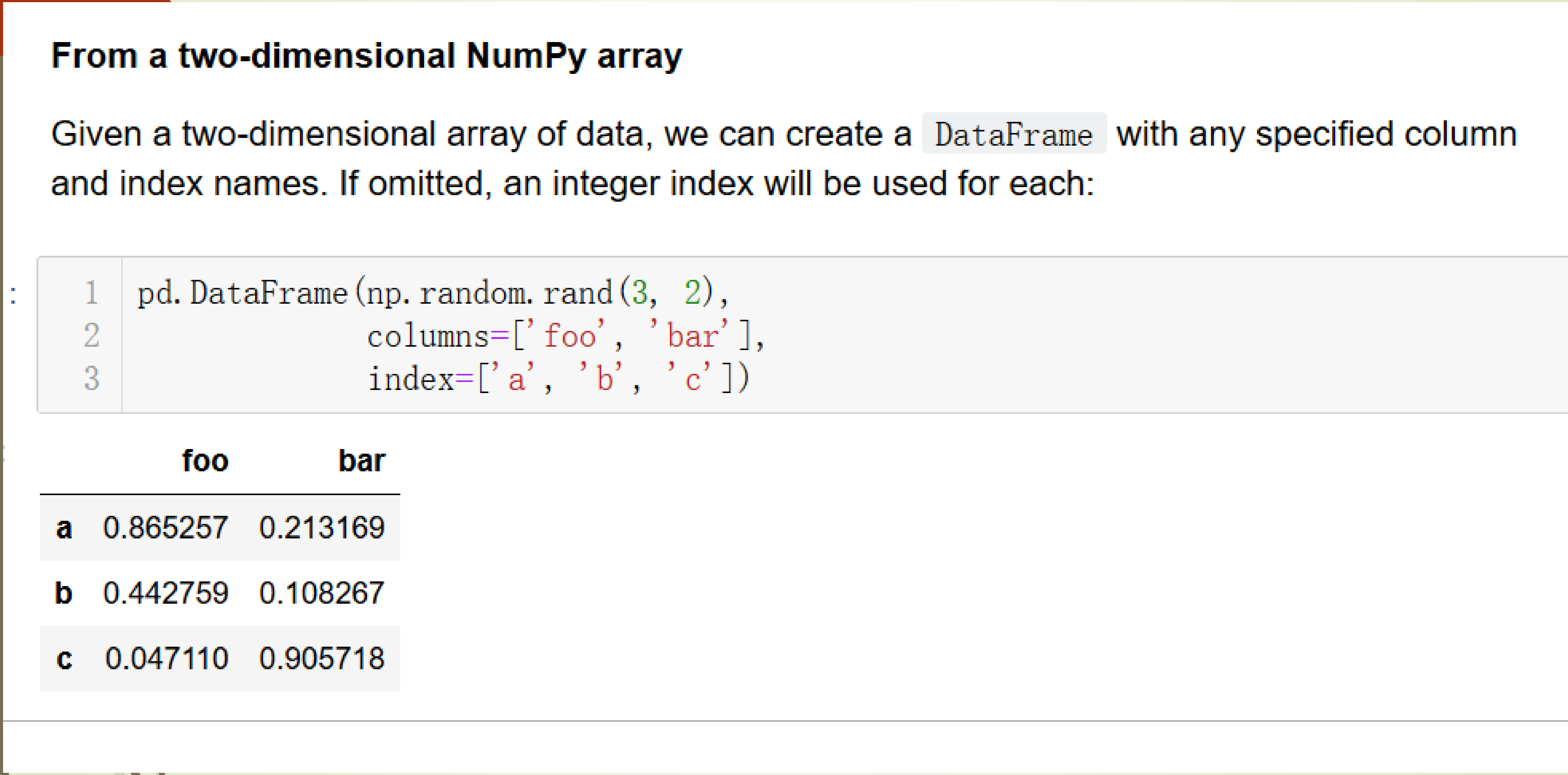
If some keys in the dictionary are missing, Pandas will fill them in with `NaN` (i.e., "not a number") values:

```
data = [{'a': i, 'b': 2 * i}
        for i in range(3)]

print(data)
pd.DataFrame(data)
```

索引key + 列表生成式

```
[{'a': 0, 'b': 0}, {'a': 1, 'b': 2}, {'a': 2, 'b': 4}]
```

	a	b
0	0	0
1	1	2
2	2	4



From a two-dimensional NumPy array

Given a two-dimensional array of data, we can create a `DataFrame` with any specified column and index names. If omitted, an integer index will be used for each:

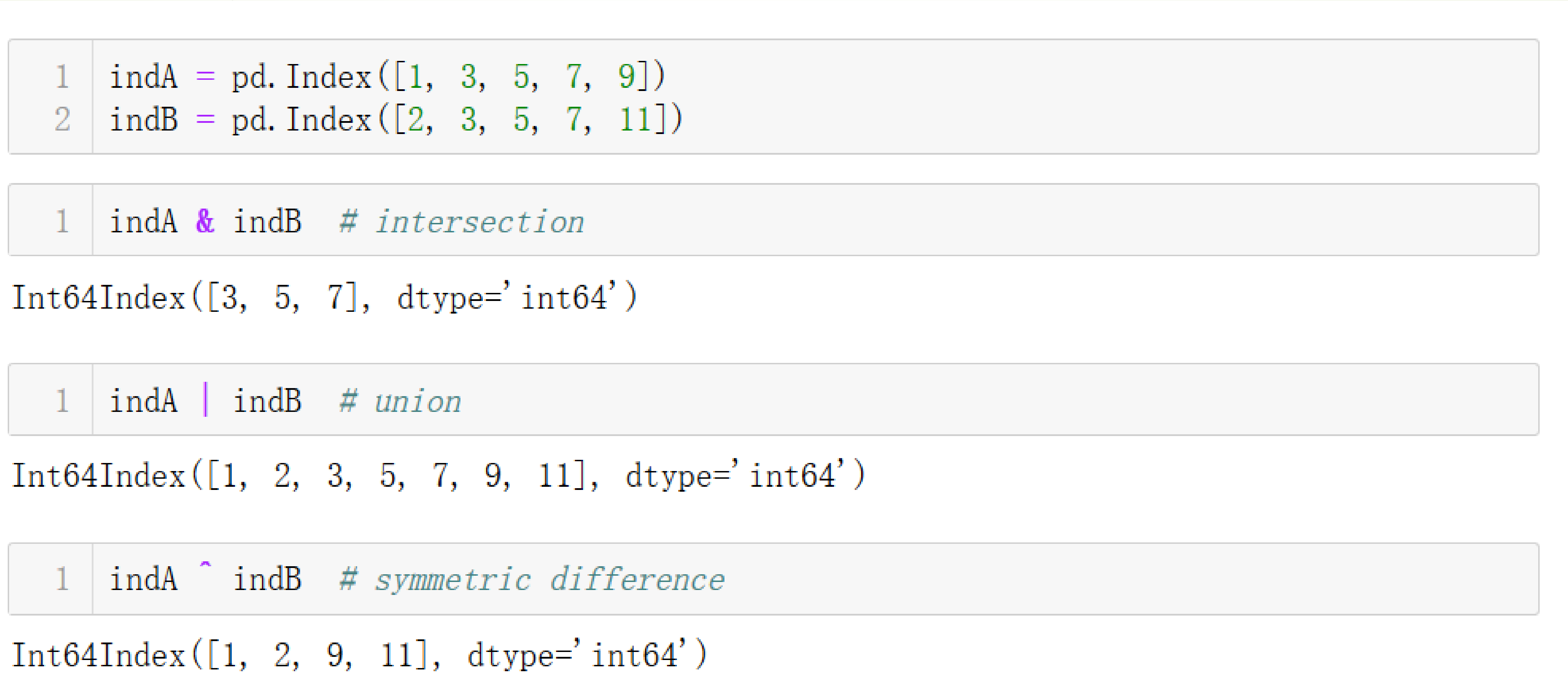
```
pd.DataFrame(np.random.rand(3, 2),
             columns=[’foo’, ’bar’],
             index=[’a’, ’b’, ’c’])
```

	foo	bar
a	0.865257	0.213169
b	0.442759	0.108267
c	0.047110	0.905718

Pandas Index Object 与 表间关联计算

- Index：不可修改的对象
- 有序
- 支持可重复 键值的index
- 表关联操作

Index计算 - 类ordered set（表关联计算的基础）



```
indA = pd.Index([1, 3, 5, 7, 9])
indB = pd.Index([2, 3, 5, 7, 11])
```

```
indA & indB  # intersection
```

```
Int64Index([3, 5, 7], dtype='int64')
```

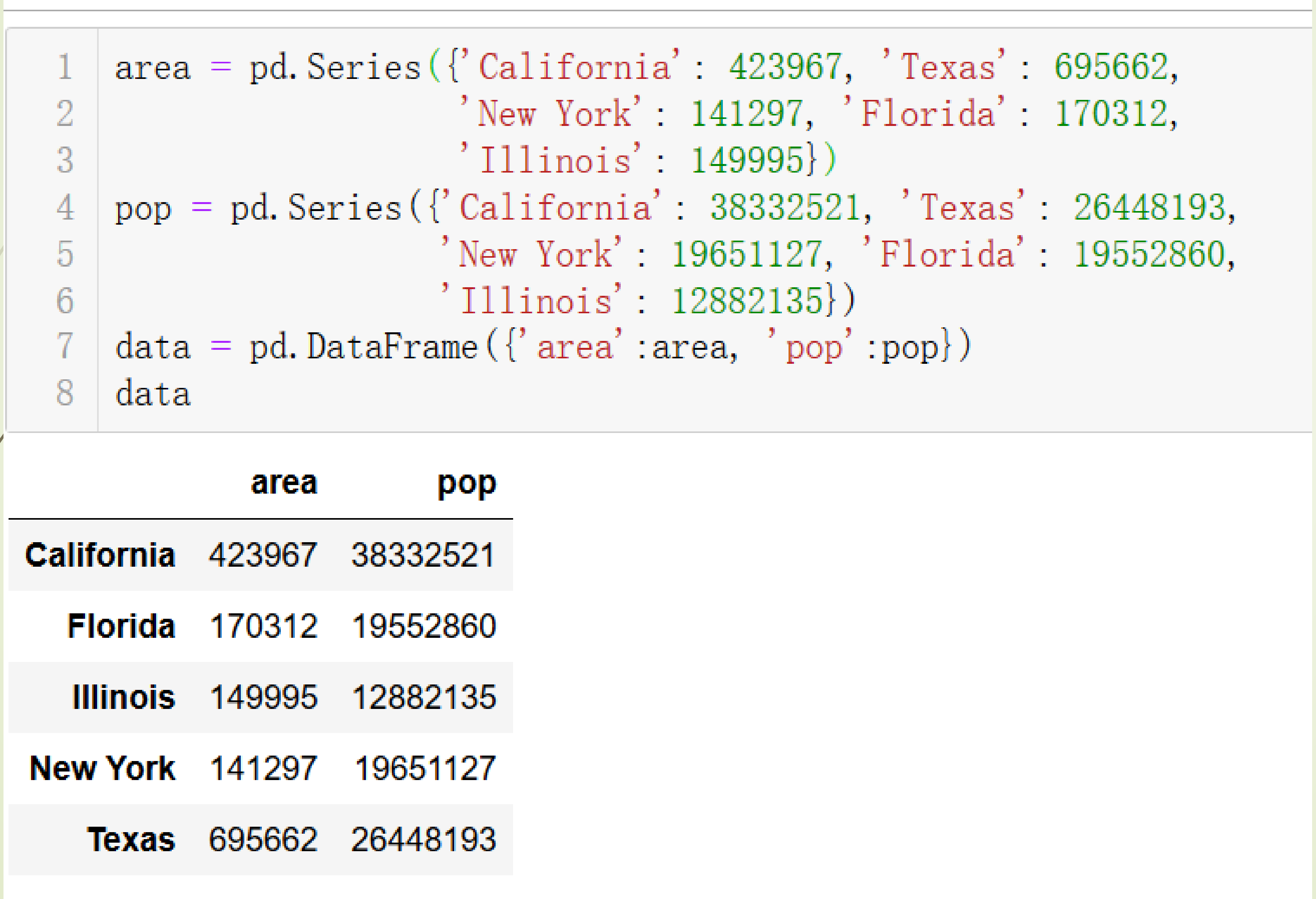
```
indA | indB  # union
```

```
Int64Index([1, 2, 3, 5, 7, 9, 11], dtype='int64')
```

```
indA ^ indB  # symmetric difference
```

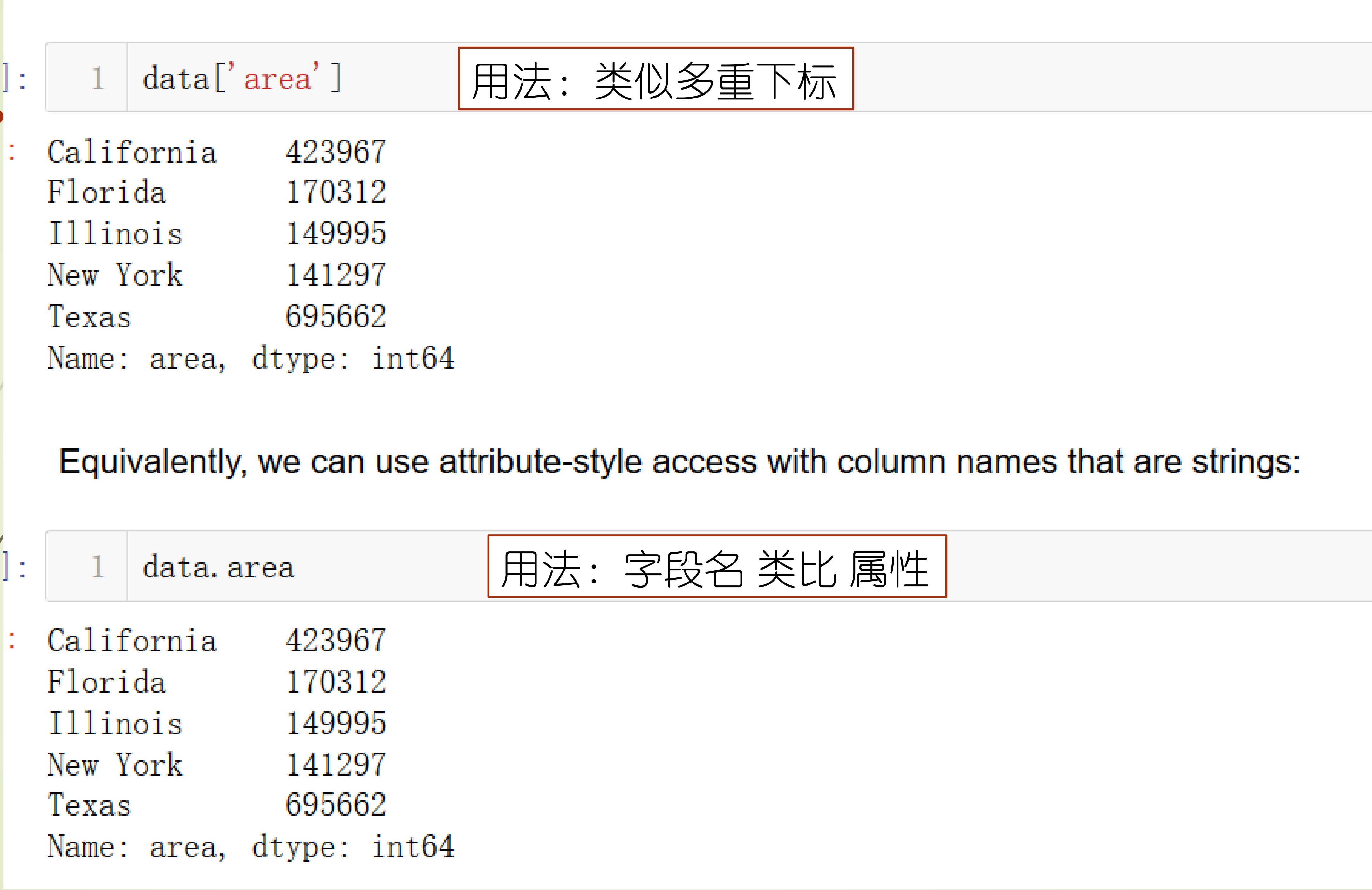
```
Int64Index([1, 2, 9, 11], dtype='int64')
```

Data Selection in DataFrame（按索引选择数据）



```
area = pd.Series({'California': 423967, 'Texas': 695662,
                  'New York': 141297, 'Florida': 170312,
                  'Illinois': 149995})
pop = pd.Series({'California': 38332521, 'Texas': 26448193,
                 'New York': 19651127, 'Florida': 19552860,
                 'Illinois': 12882135})
data = pd.DataFrame({'area':area, 'pop':pop})
data
```

```
              area       pop
California  423967  38332521
   Florida  170312  19552860
  Illinois  149995  12882135
  New York  141297  19651127
     Texas  695662  26448193
```



```
data[’area’]
```

用法：类似多重下标

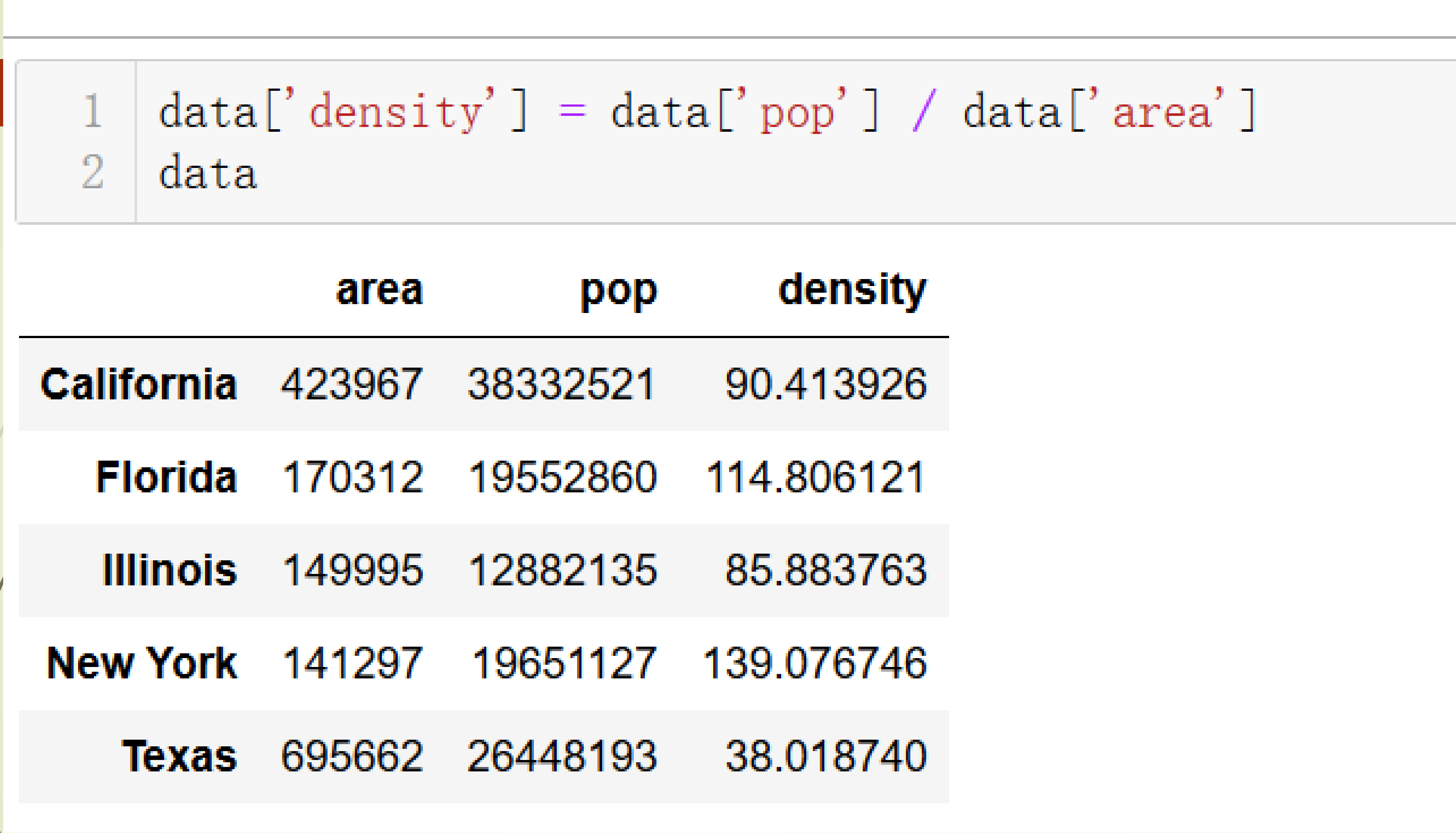
```
California    423967
Florida       170312
Illinois      149995
New York      141297
Texas         695662
Name: area, dtype: int64
```

Equivalently, we can use attribute-style access with column names that are strings:

```
data.area
```

用法：字段名 类比 属性

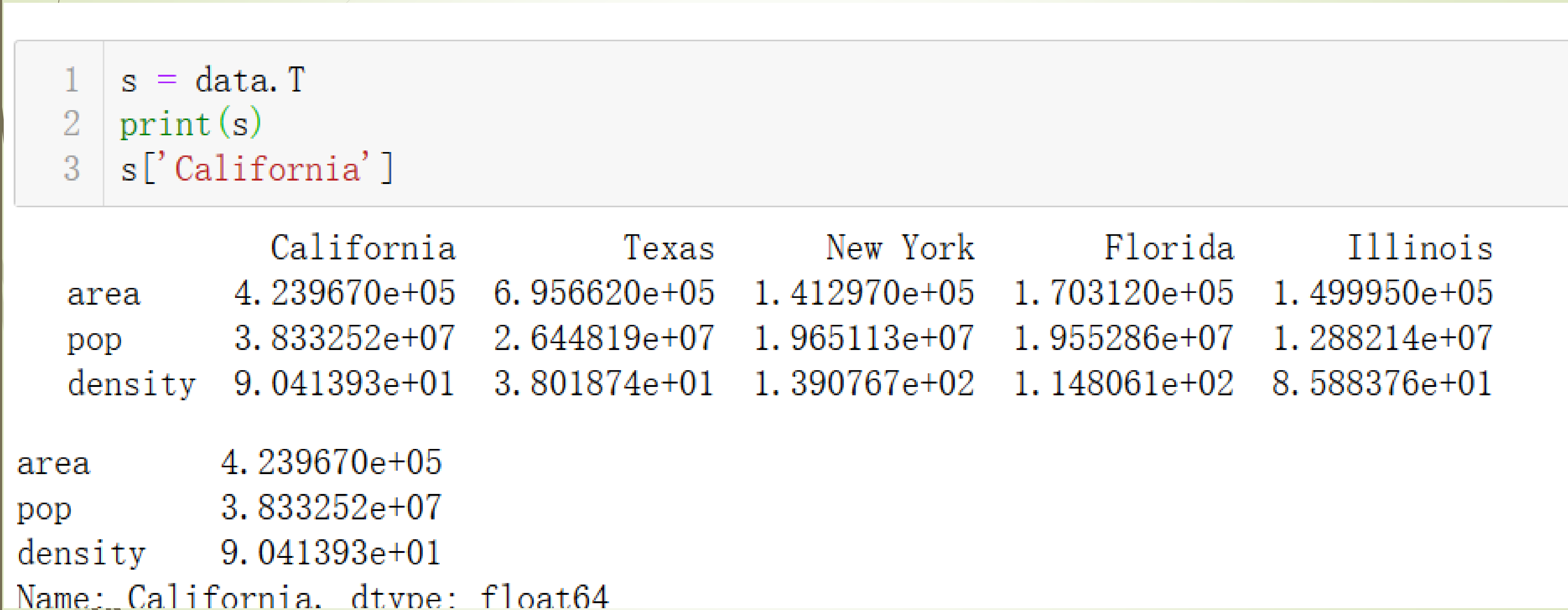
```
California    423967
Florida       170312
Illinois      149995
New York      141297
Texas         695662
Name: area, dtype: int64
```



```
data['density'] = data['pop'] / data['area']
data
```

	area	pop	density
California	423967	38332521	90.413926
Florida	170312	19552860	114.806121
Illinois	149995	12882135	85.883763
New York	141297	19651127	139.076746
Texas	695662	26448193	38.018740

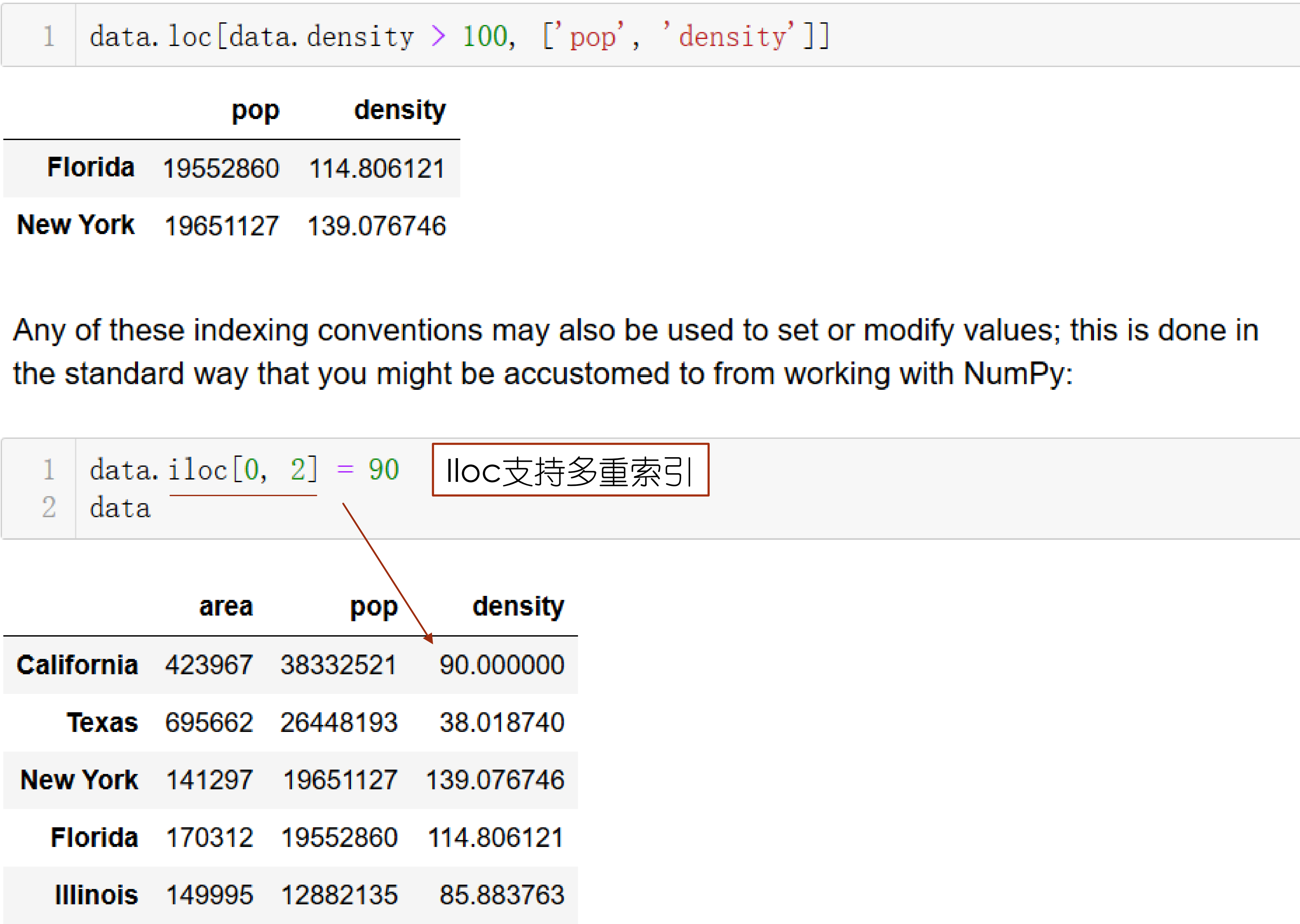
行列互换操作



```python
s = data.T
print(s)
s['California']
```

```
            California         Texas      New York       Florida      Illinois
area      4.239670e+05  6.956620e+05  1.412970e+05  1.703120e+05  1.499950e+05
pop       3.833252e+07  2.644819e+07  1.965113e+07  1.955286e+07  1.288214e+07
density   9.041393e+01  3.801874e+01  1.390767e+02  1.148061e+02  8.588376e+01

area       4.239670e+05
pop        3.833252e+07
density    9.041393e+01
Name: California, dtype: float64
```

筛选，赋值：



```
data.loc[data.density > 100, ['pop', 'density']]
```

	pop	density
Florida	19552860	114.806121
New York	19651127	139.076746

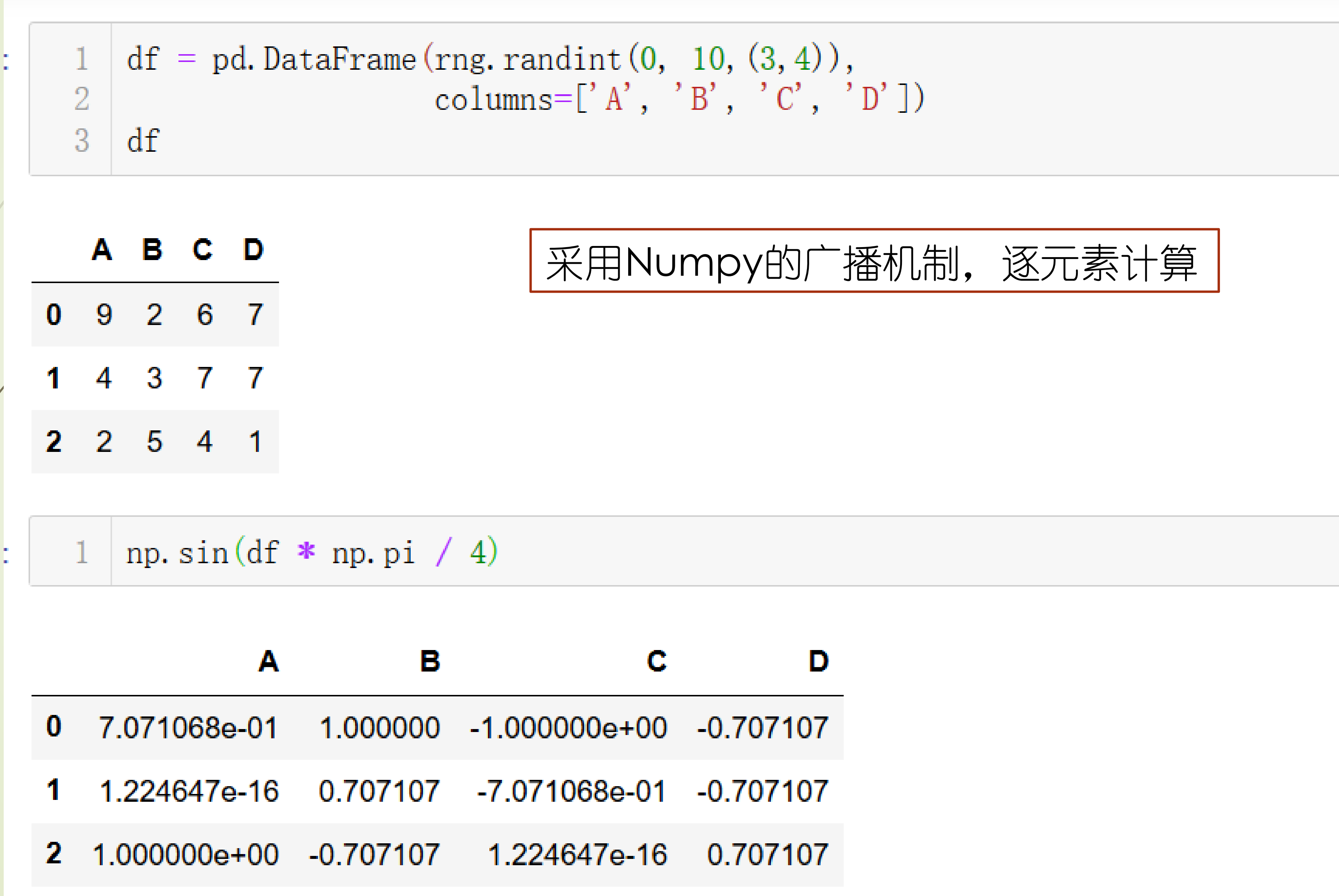
Any of these indexing conventions may also be used to set or modify values; this is done in the standard way that you might be accustomed to from working with NumPy:

```
data.iloc[0, 2] = 90
data
```

Iloc支持多重索引

	area	pop	density
California	423967	38332521	90.000000
Texas	695662	26448193	38.018740
New York	141297	19651127	139.076746
Florida	170312	19552860	114.806121
Illinois	149995	12882135	85.883763

Working with NumPy ufunc



```python
df = pd.DataFrame(rng.randint(0, 10, (3,4)),
                  columns=['A', 'B', 'C', 'D'])
df
```

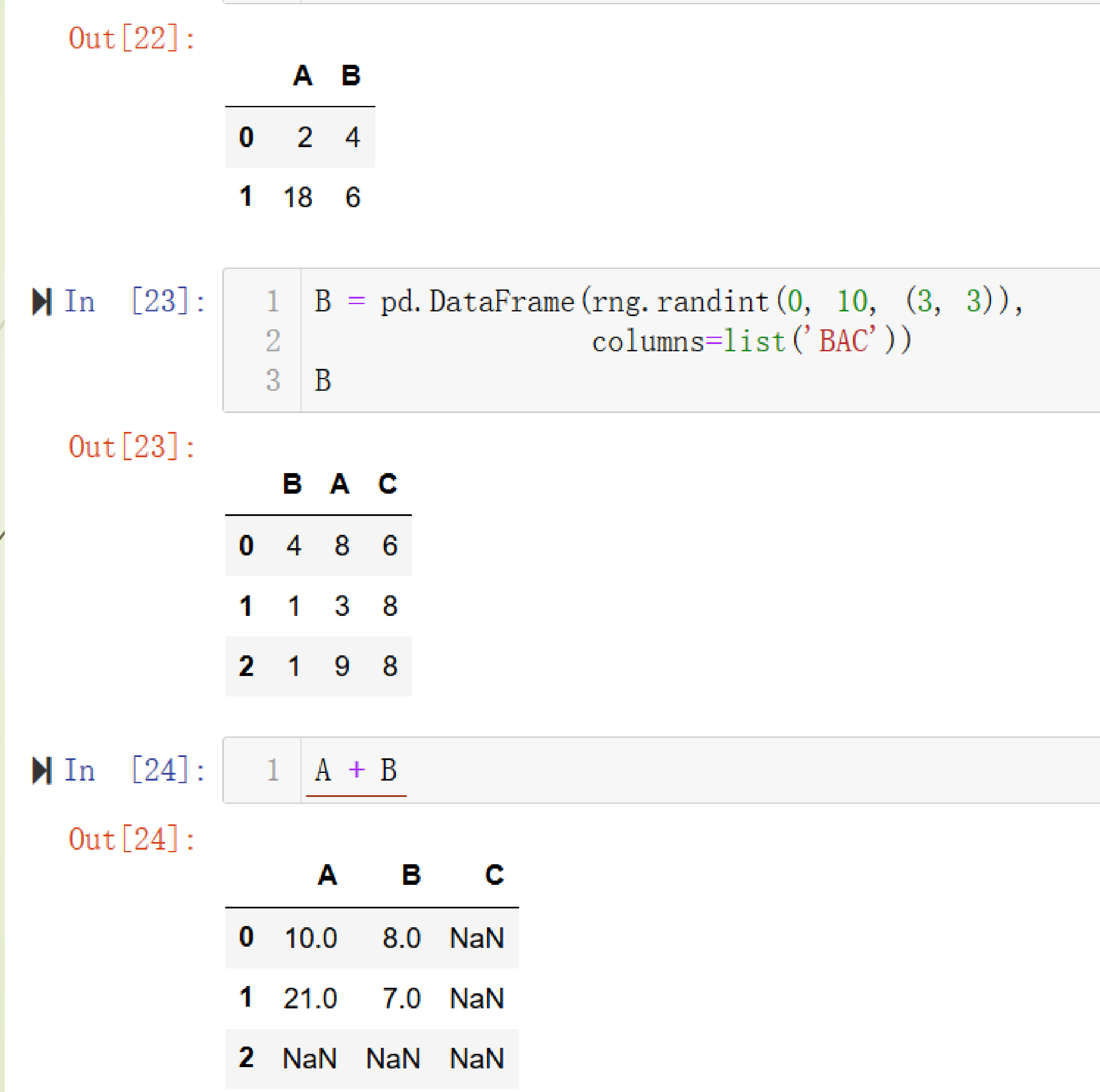
	A	B	C	D
0	9	2	6	7
1	4	3	7	7
2	2	5	4	1

采用Numpy的广播机制，逐元素计算

```python
np.sin(df * np.pi / 4)
```

	A	B	C	D
0	7.071068e-01	1.000000	-1.000000e+00	-0.707107
1	1.224647e-16	0.707107	-7.071068e-01	-0.707107
2	1.000000e+00	-0.707107	1.224647e-16	0.707107

Dataframe之间的运算自动进行索引键对齐/补缺（out join）



Out[22]:

```
    A  B
0   2  4
1  18  6
```

In [23]:

```python
B = pd.DataFrame(rng.randint(0, 10, (3, 3)),
                 columns=list(’BAC’))
B
```

Out[23]:

```
   B  A  C
0  4  8  6
1  1  3  8
2  1  9  8
```

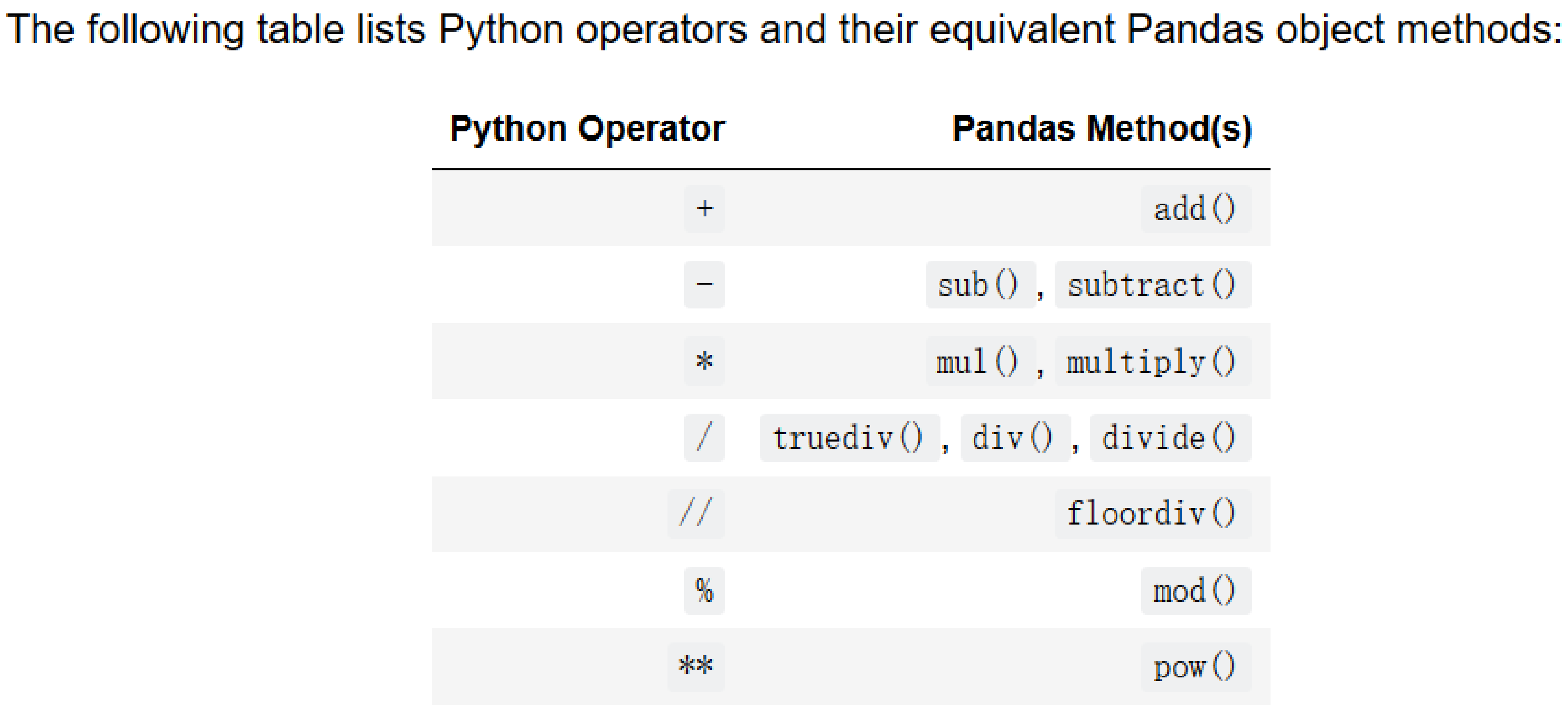
In [24]:

```python
A + B
```

Out[24]:

```
      A    B    C
0  10.0  8.0  NaN
1  21.0  7.0  NaN
2   NaN  NaN  NaN
```

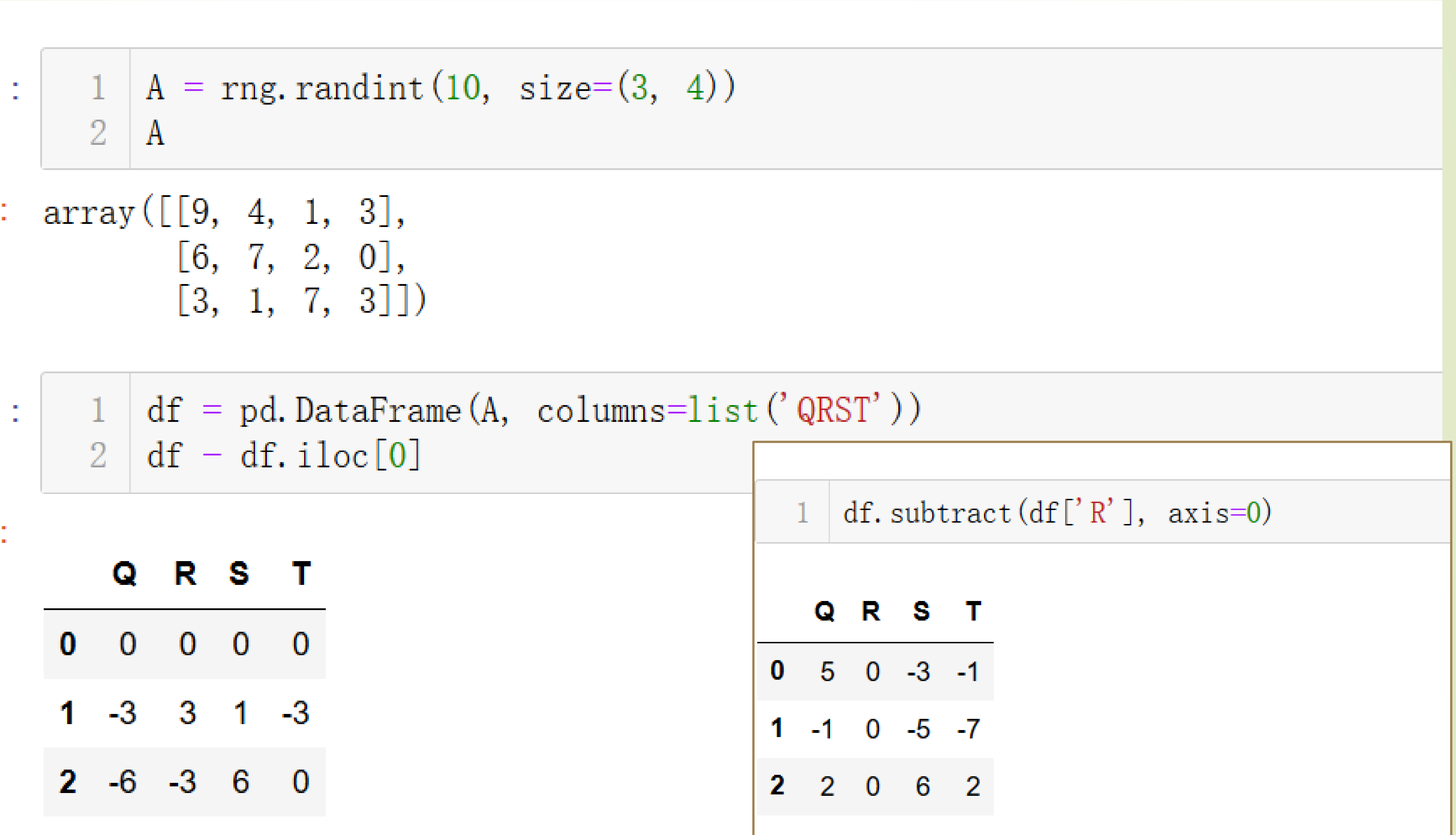
行列对象间支持的运算符：



The following table lists Python operators and their equivalent Pandas object methods:

Python Operator	Pandas Method(s)
`+`	`add()`
`-`	`sub()`, `subtract()`
`*`	`mul()`, `multiply()`
`/`	`truediv()`, `div()`, `divide()`
`//`	`floordiv()`
`%`	`mod()`
`**`	`pow()`

Frame 与 series 计算，按行broadcasting



```
A = rng.randint(10, size=(3, 4))
A
```

```
array([[9, 4, 1, 3],
       [6, 7, 2, 0],
       [3, 1, 7, 3]])
```

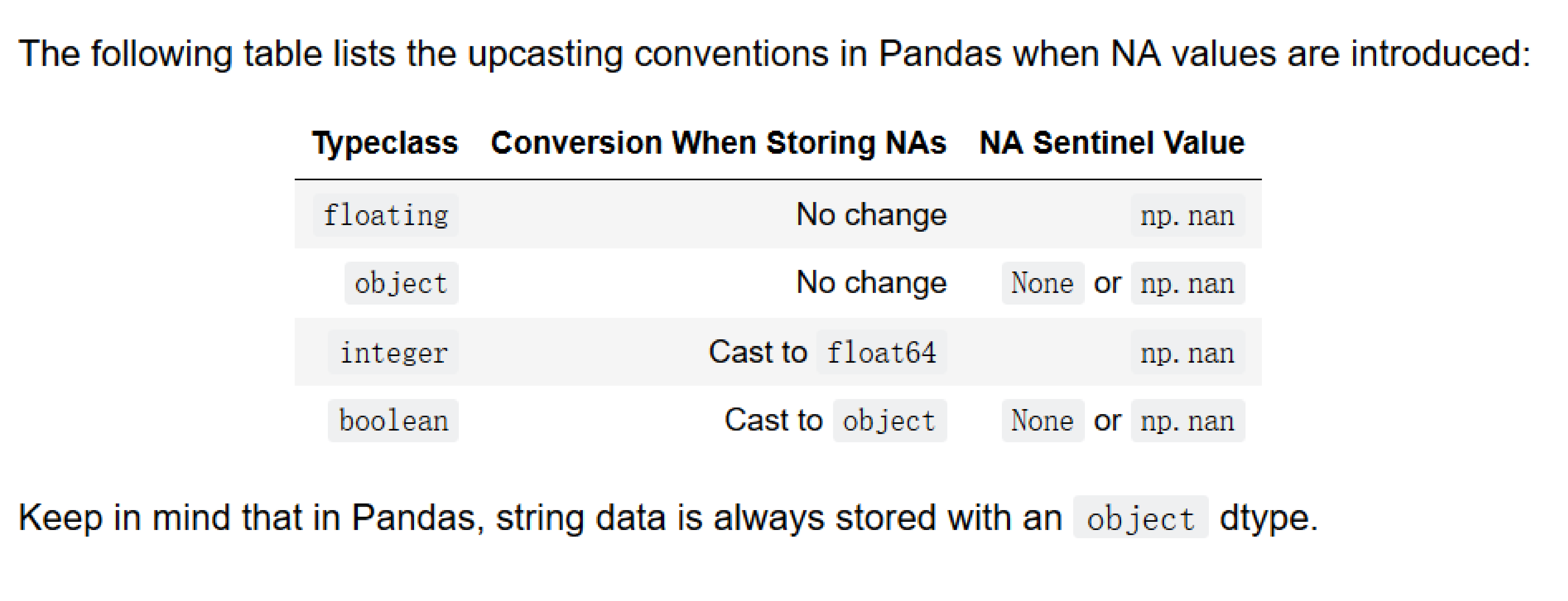
```
df = pd.DataFrame(A, columns=list(’QRST’))
df - df.iloc[0]
```

	Q	R	S	T
0	0	0	0	0
1	-3	3	1	-3
2	-6	-3	6	0

```
df.subtract(df[’R’], axis=0)
```

	Q	R	S	T
0	5	0	-3	-1
1	-1	0	-5	-7
2	2	0	6	2

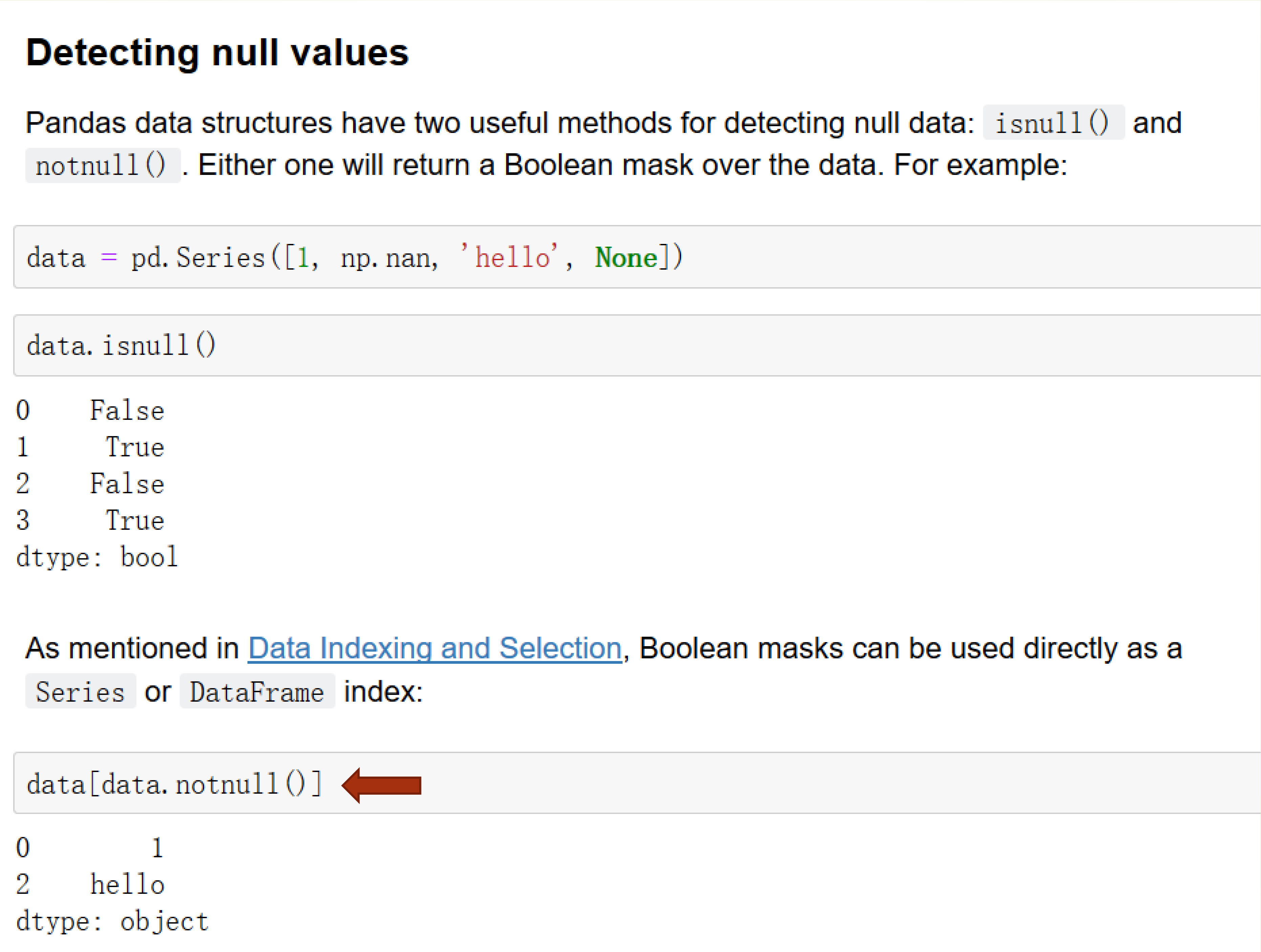
运算过程中类型自适应转换



The following table lists the upcasting conventions in Pandas when NA values are introduced:

Typeclass	Conversion When Storing NAs	NA Sentinel Value
`floating`	No change	`np.nan`
`object`	No change	`None` or `np.nan`
`integer`	Cast to `float64`	`np.nan`
`boolean`	Cast to `object`	`None` or `np.nan`

Keep in mind that in Pandas, string data is always stored with an `object` dtype.



Detecting null values

Pandas data structures have two useful methods for detecting null data: `isnull()` and `notnull()`. Either one will return a Boolean mask over the data. For example:

```
data = pd.Series([1, np.nan, 'hello', None])
```

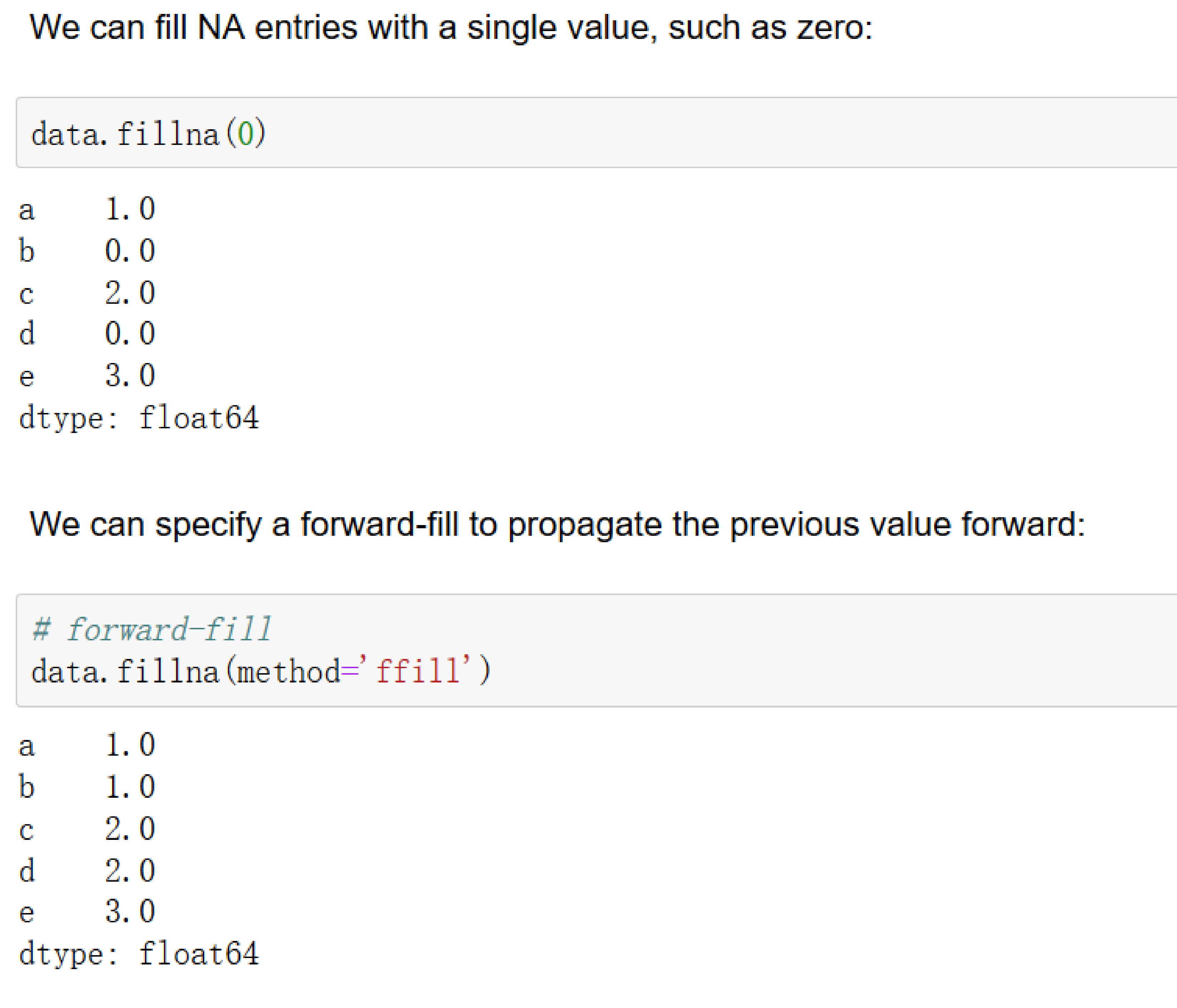
```
data.isnull()
```

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

As mentioned in Data Indexing and Selection, Boolean masks can be used directly as a `Series` or `DataFrame` index:

```
data[data.notnull()]
```

```
0        1
2    hello
dtype: object
```



We can fill NA entries with a single value, such as zero:

```
data.fillna(0)
```

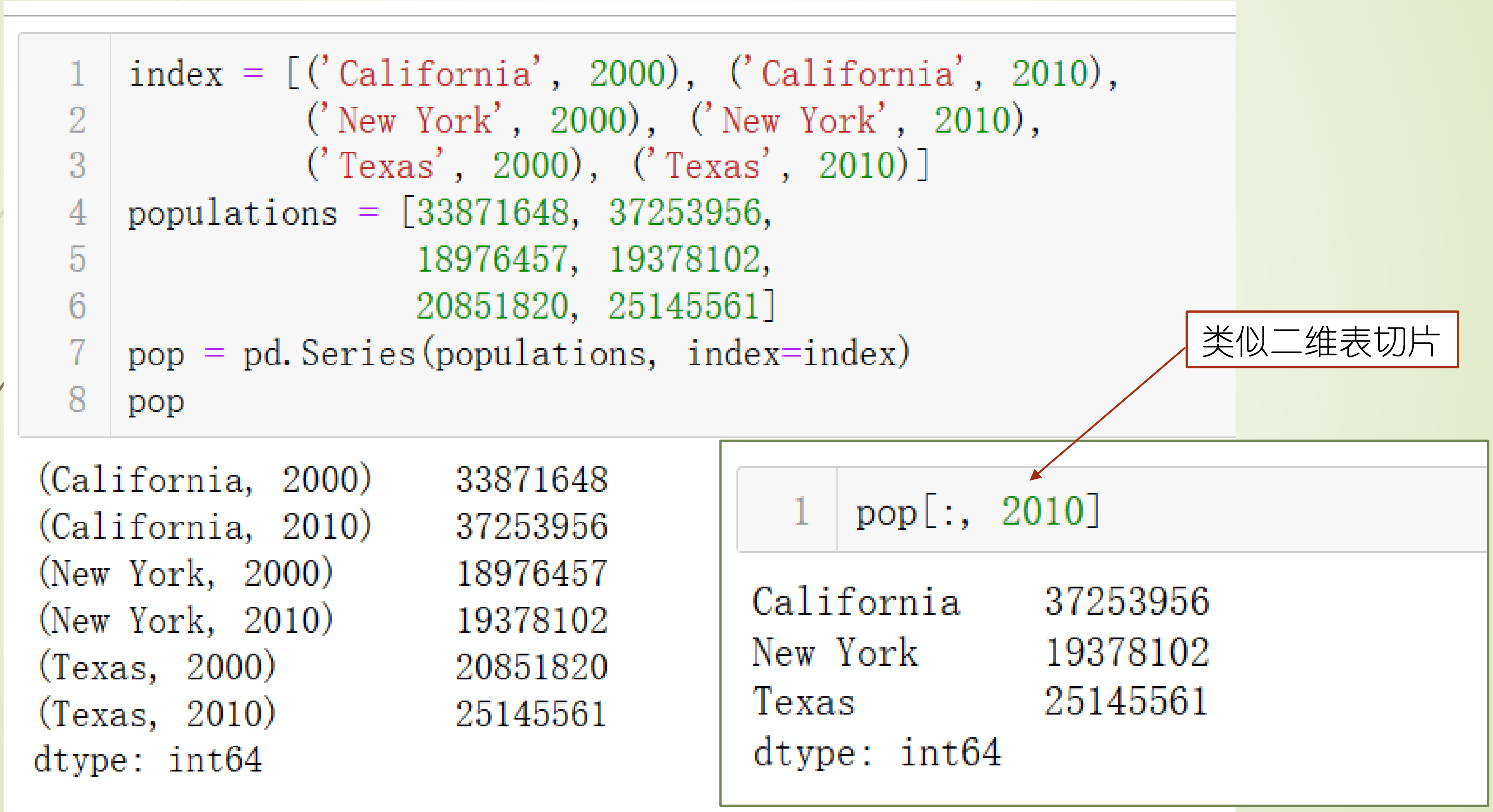
```
a     1.0
b     0.0
c     2.0
d     0.0
e     3.0
dtype: float64
```

We can specify a forward-fill to propagate the previous value forward:

```
# forward-fill
data.fillna(method=’ffill’)
```

```
a     1.0
b     1.0
c     2.0
d     2.0
e     3.0
dtype: float64
```

层次-组合 索引（ Hierarchical-Indexing ）



```
index = [('California', 2000), ('California', 2010),
         ('New York', 2000), ('New York', 2010),
         ('Texas', 2000), ('Texas', 2010)]
populations = [33871648, 37253956,
               18976457, 19378102,
               20851820, 25145561]
pop = pd.Series(populations, index=index)
pop
```

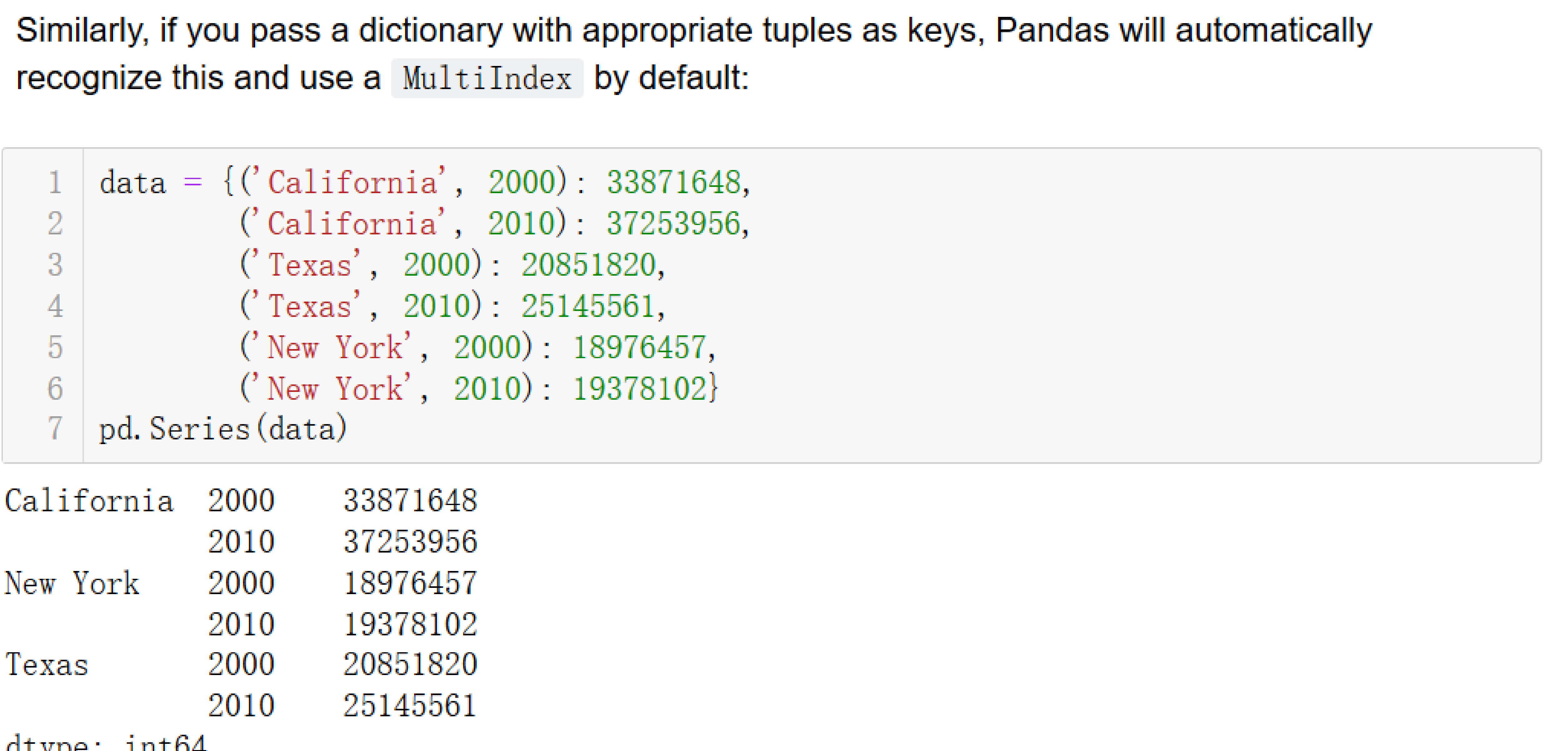
```
(California, 2000)    33871648
(California, 2010)    37253956
(New York, 2000)      18976457
(New York, 2010)      19378102
(Texas, 2000)         20851820
(Texas, 2010)         25145561
dtype: int64
```

类似二维表切片

```
pop[:, 2010]
```

```
California    37253956
New York      19378102
Texas         25145561
dtype: int64
```

多键值词典索引初始化：自动识别为多层索引

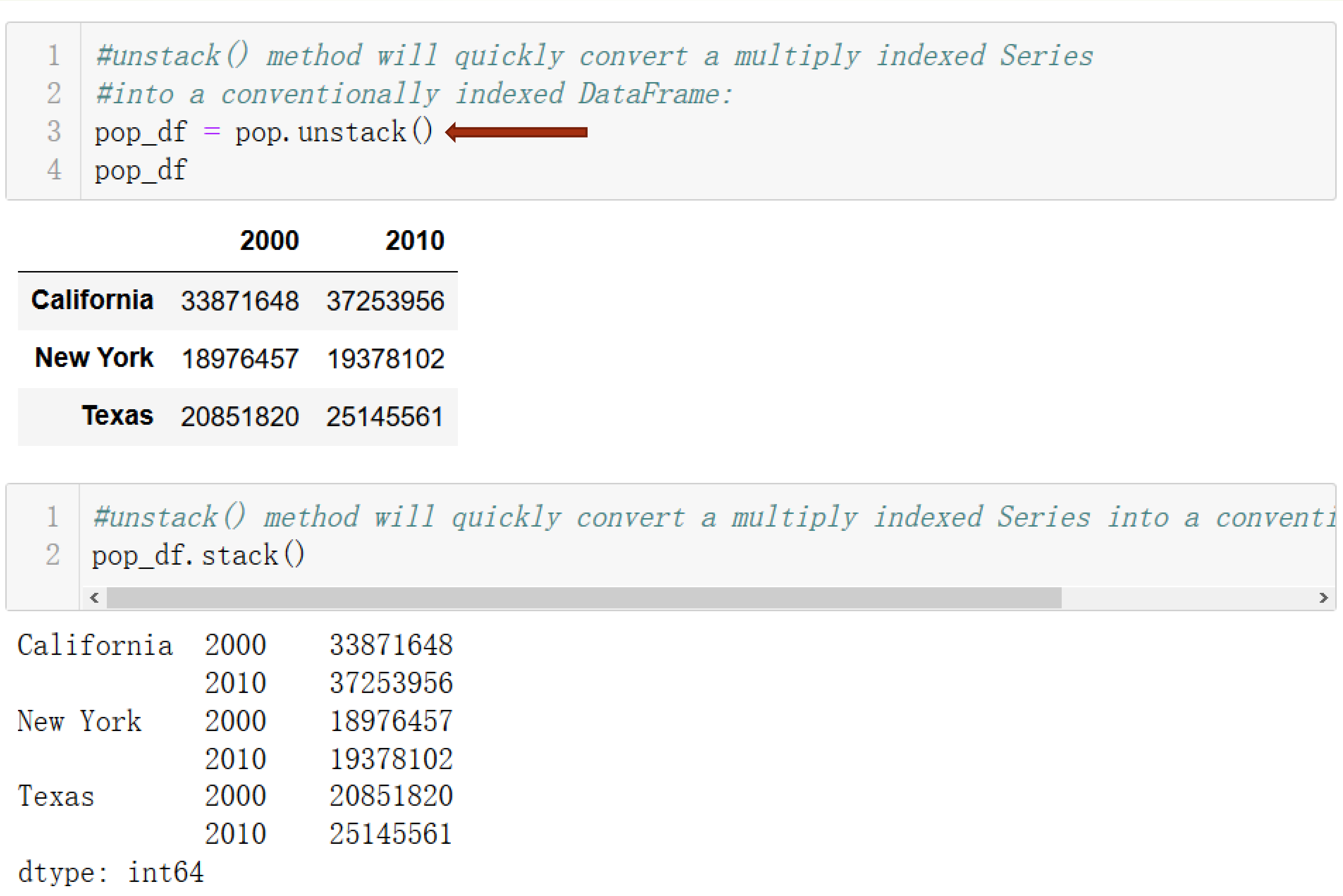


Similarly, if you pass a dictionary with appropriate tuples as keys, Pandas will automatically recognize this and use a `MultiIndex` by default:

```
data = {('California', 2000): 33871648,
        ('California', 2010): 37253956,
        ('Texas', 2000): 20851820,
        ('Texas', 2010): 25145561,
        ('New York', 2000): 18976457,
        ('New York', 2010): 19378102}
pd.Series(data)
```

```
California  2000    33871648
            2010    37253956
New York    2000    18976457
            2010    19378102
Texas       2000    20851820
            2010    25145561
dtype: int64
```

MultiIndex VS extra dimension

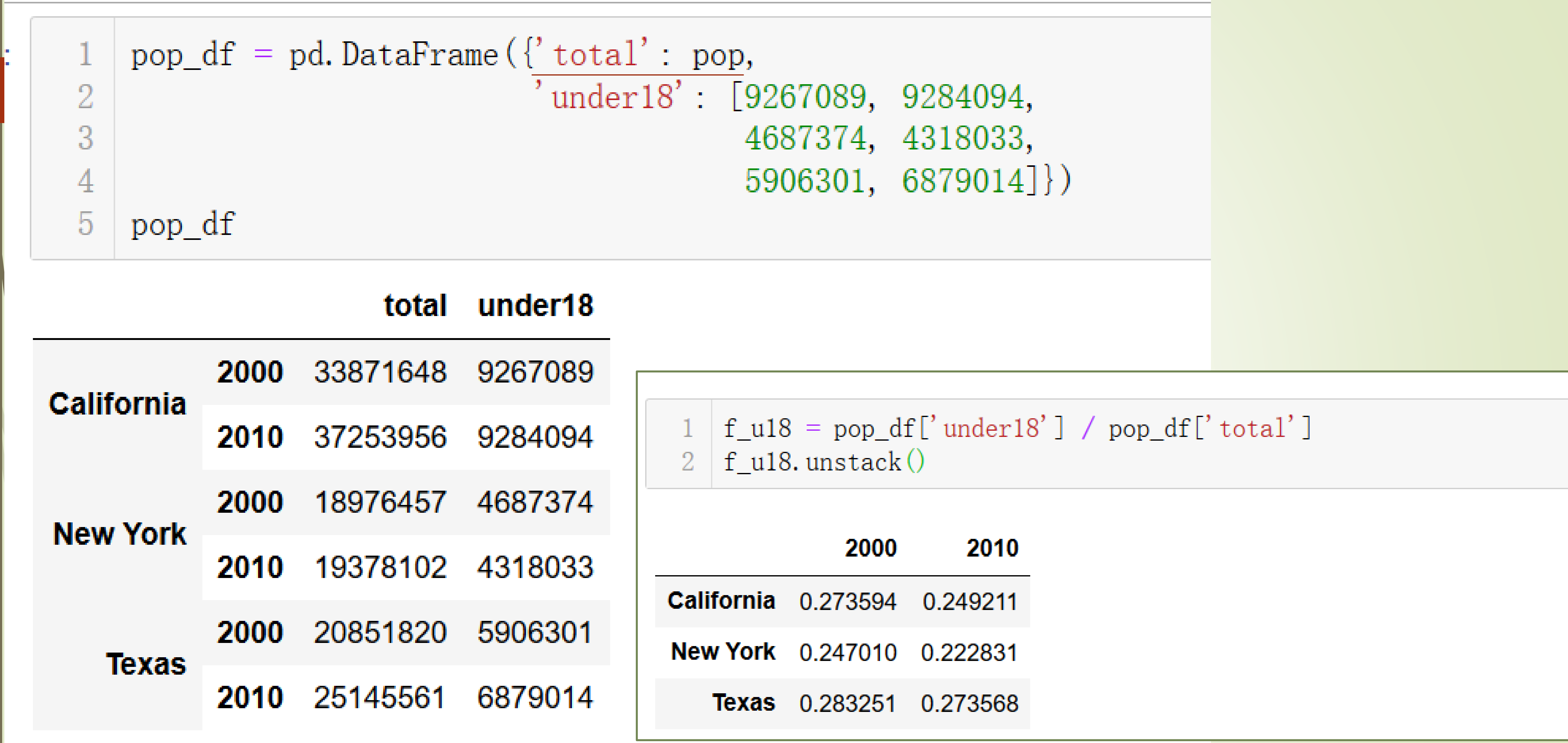


```python
#unstack() method will quickly convert a multiply indexed Series
#into a conventionally indexed DataFrame:
pop_df = pop.unstack()
pop_df
```

	2000	2010
California	33871648	37253956
New York	18976457	19378102
Texas	20851820	25145561

```python
#unstack() method will quickly convert a multiply indexed Series into a conventi
pop_df.stack()
```

```
California  2000    33871648
            2010    37253956
New York    2000    18976457
            2010    19378102
Texas       2000    20851820
            2010    25145561
dtype: int64
```



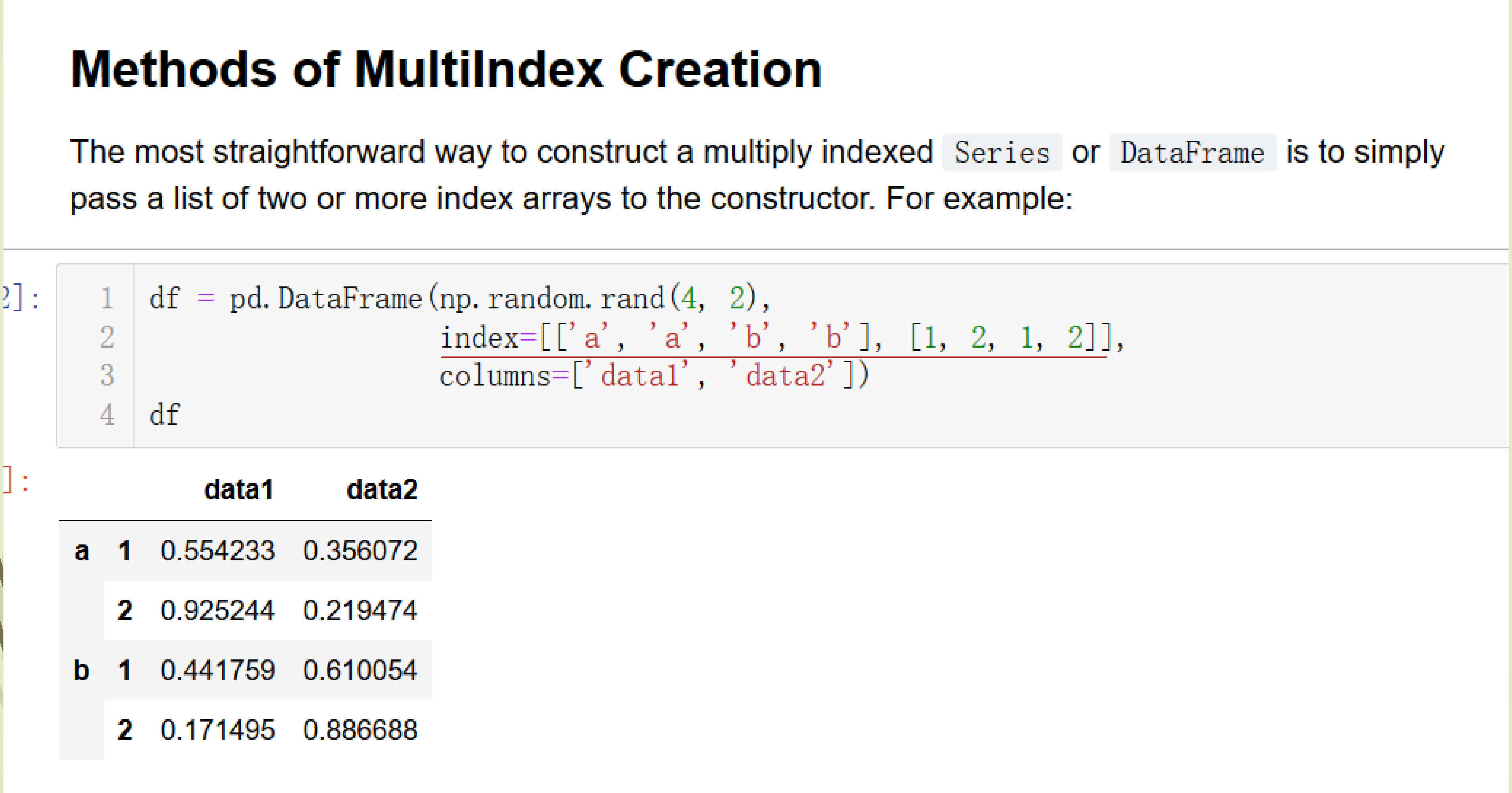
```
pop_df = pd.DataFrame({'total': pop,
                       'under18': [9267089, 9284094,
                                   4687374, 4318033,
                                   5906301, 6879014]})
pop_df
```

		total	under18
California	2000	33871648	9267089
	2010	37253956	9284094
New York	2000	18976457	4687374
	2010	19378102	4318033
Texas	2000	20851820	5906301
	2010	25145561	6879014

```
f_u18 = pop_df['under18'] / pop_df['total']
f_u18.unstack()
```

	2000	2010
California	0.273594	0.249211
New York	0.247010	0.222831
Texas	0.283251	0.273568

多个键值组合为多层索引：



Methods of MultiIndex Creation

The most straightforward way to construct a multiply indexed `Series` or `DataFrame` is to simply pass a list of two or more index arrays to the constructor. For example:

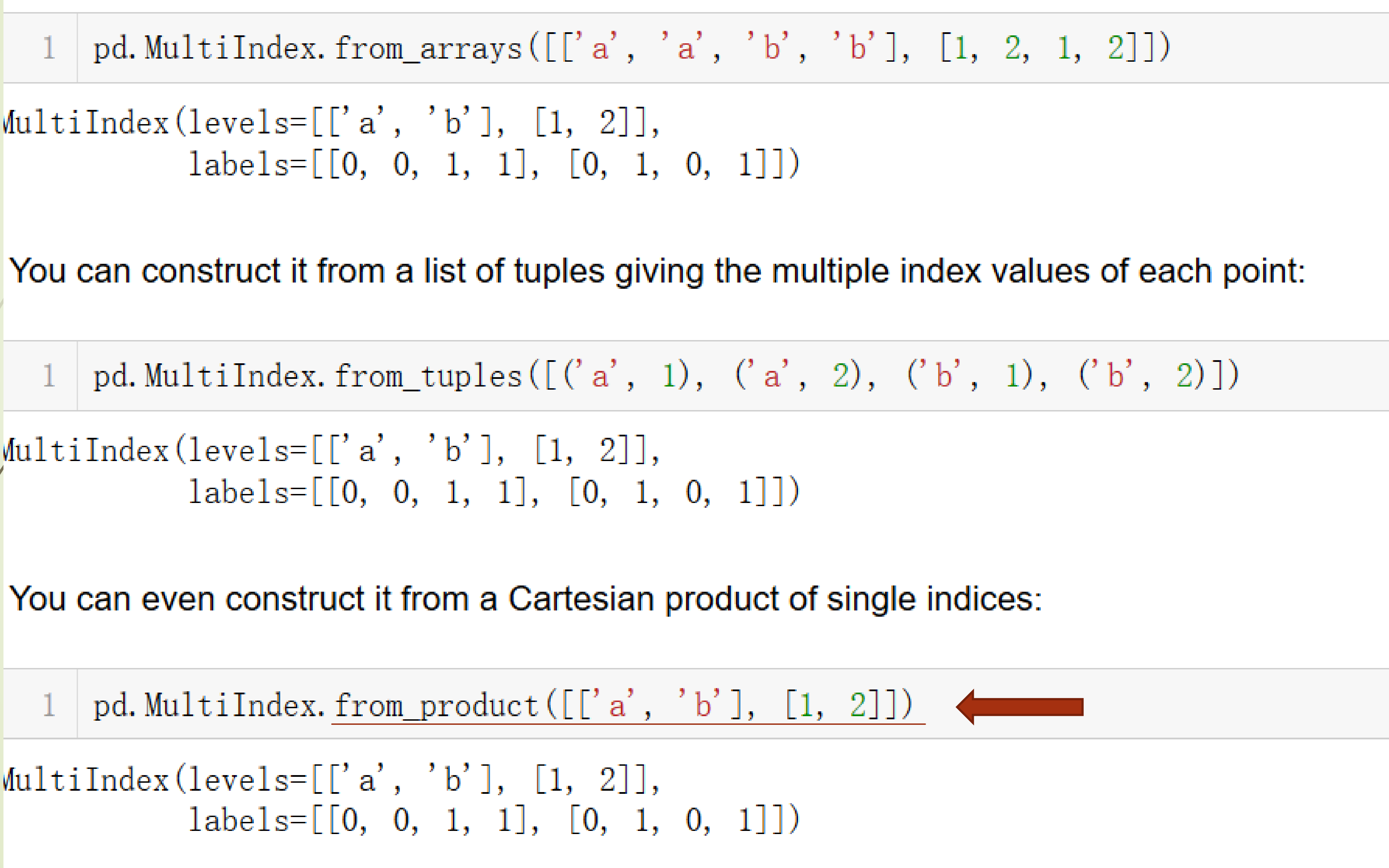
2]:

```
df = pd.DataFrame(np.random.rand(4, 2),
                  index=[['a', 'a', 'b', 'b'], [1, 2, 1, 2]],
                  columns=['data1', 'data2'])
df
```

]:

		data1	data2
a	1	0.554233	0.356072
	2	0.925244	0.219474
b	1	0.441759	0.610054
	2	0.171495	0.886688

多层索引的生成方案：



```
pd.MultiIndex.from_arrays([['a', 'a', 'b', 'b'], [1, 2, 1, 2]])
```

```
MultiIndex(levels=[['a', 'b'], [1, 2]],
           labels=[[0, 0, 1, 1], [0, 1, 0, 1]])
```

You can construct it from a list of tuples giving the multiple index values of each point:

```
pd.MultiIndex.from_tuples([('a', 1), ('a', 2), ('b', 1), ('b', 2)])
```

```
MultiIndex(levels=[['a', 'b'], [1, 2]],
           labels=[[0, 0, 1, 1], [0, 1, 0, 1]])
```

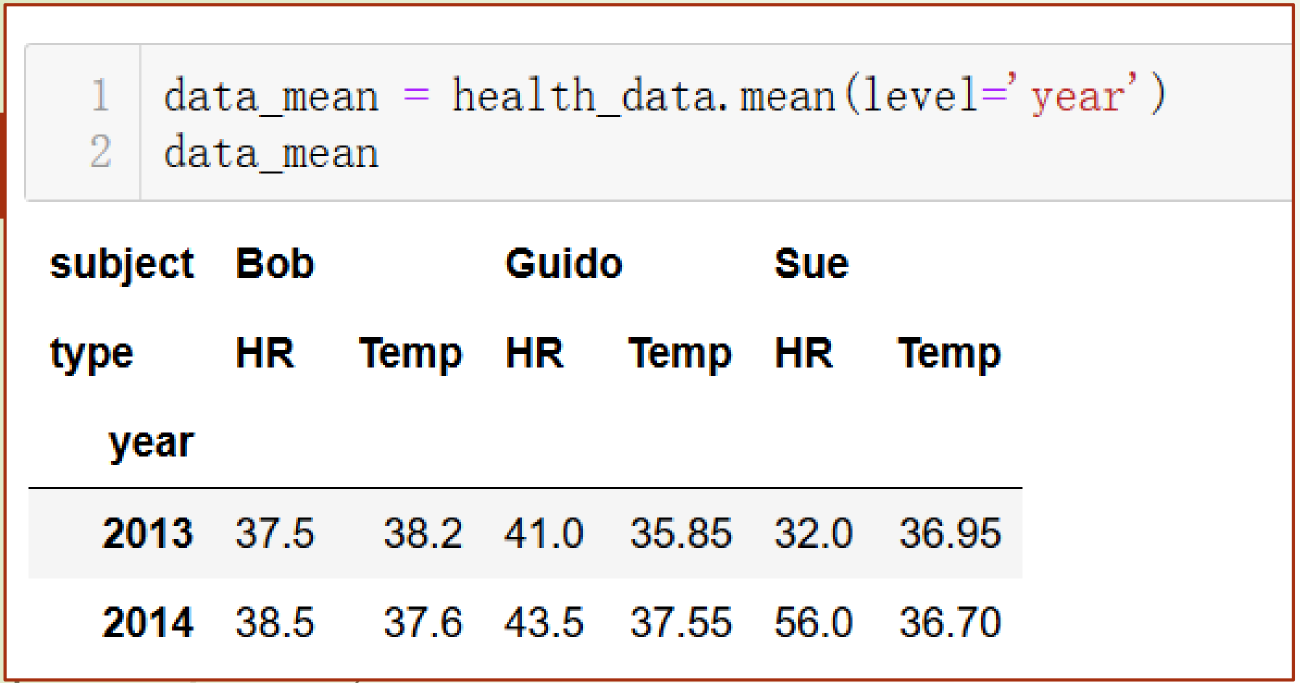
You can even construct it from a Cartesian product of single indices:

```
pd.MultiIndex.from_product([['a', 'b'], [1, 2]])
```

```
MultiIndex(levels=[['a', 'b'], [1, 2]],
           labels=[[0, 0, 1, 1], [0, 1, 0, 1]])
```

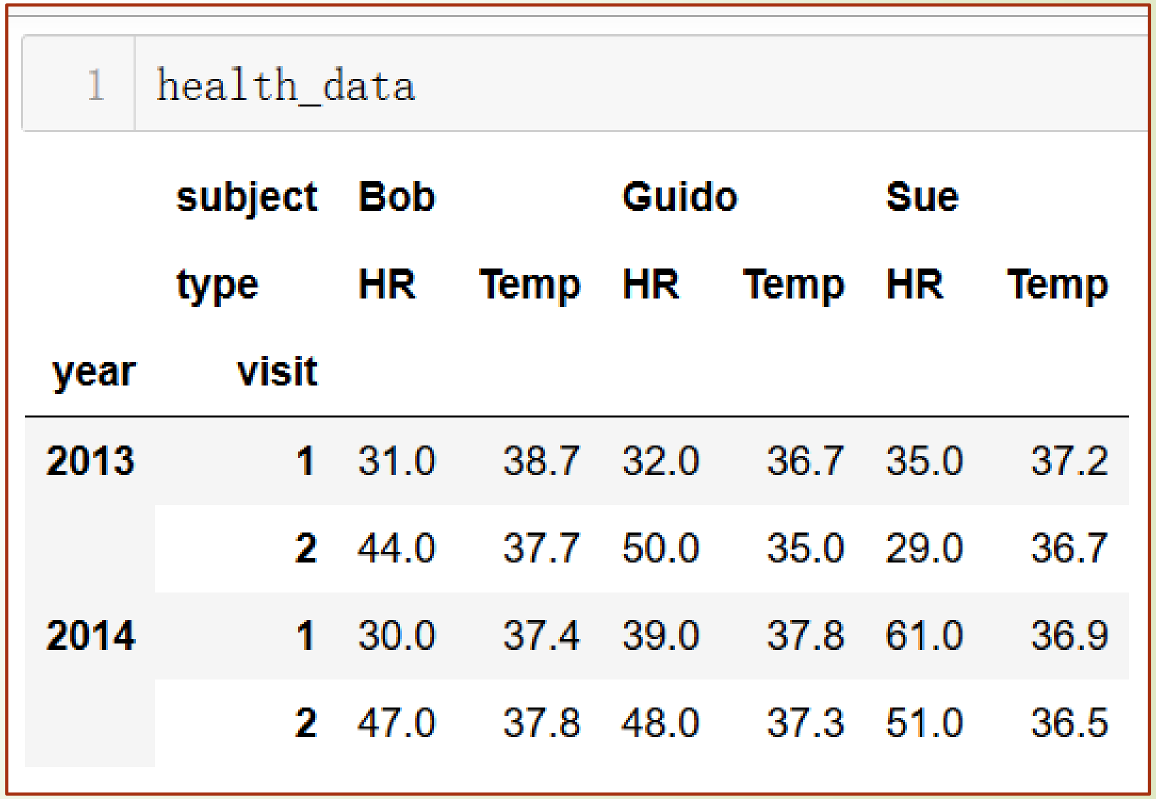
Data Aggregations on Multi-Indices

- Pandas has built-in data aggregation methods,
- such as mean(), sum(), and max().
- For hierarchically indexed data, these can be passed a level parameter that controls which **subset of the data the aggregate is computed on**.
- Group by certain Key



```python
data_mean = health_data.mean(level=’year’)
data_mean
```

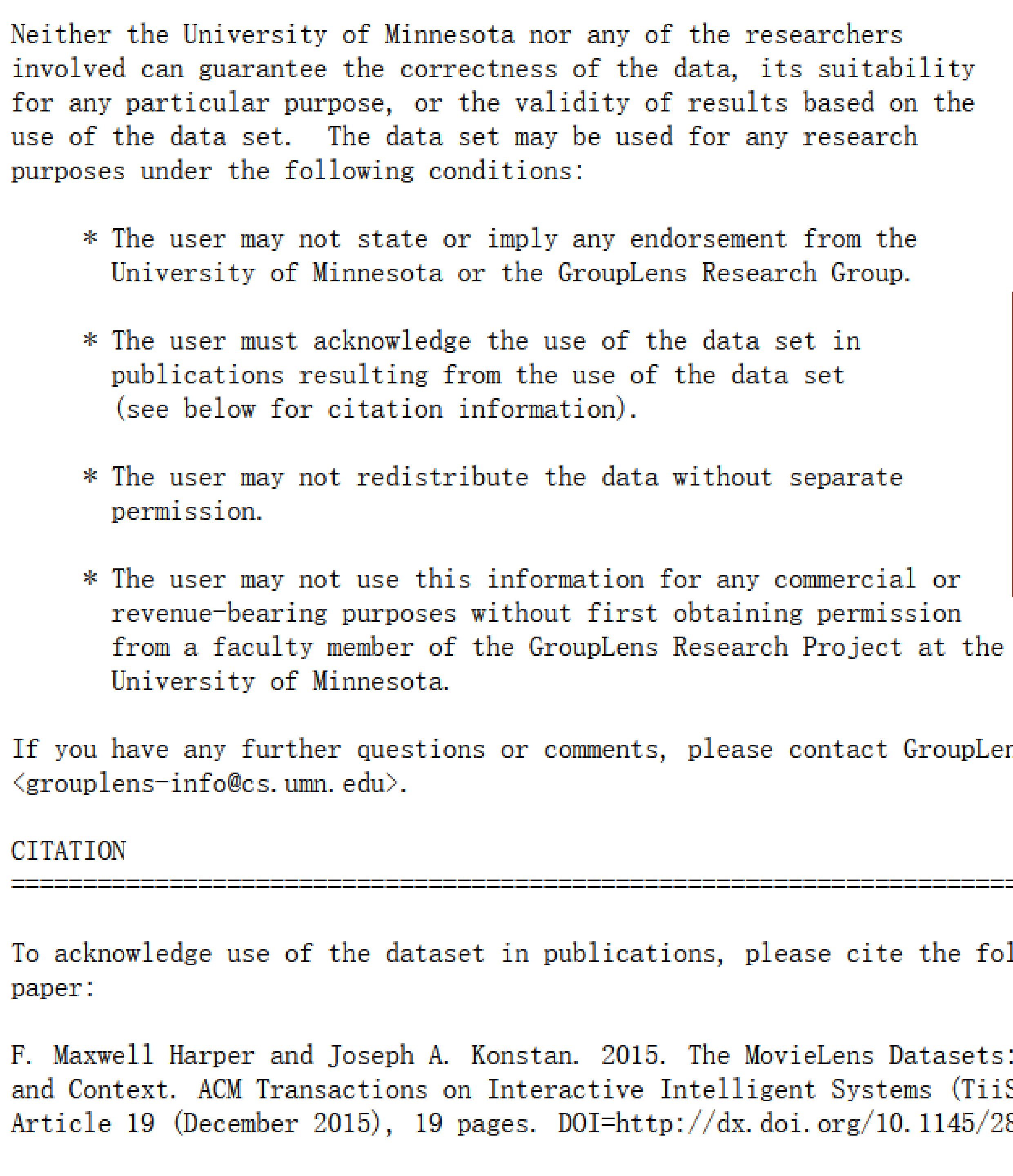
subject	Bob		Guido		Sue	
type	HR	Temp	HR	Temp	HR	Temp
year						
2013	37.5	38.2	41.0	35.85	32.0	36.95
2014	38.5	37.6	43.5	37.55	56.0	36.70



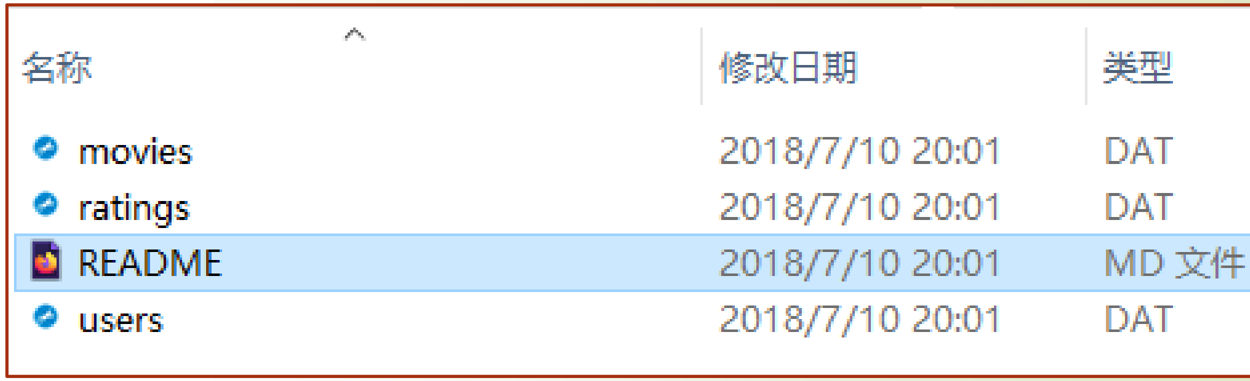
```python
health_data
```

	subject	Bob		Guido		Sue	
	type	HR	Temp	HR	Temp	HR	Temp
year	visit						
2013	1	31.0	38.7	32.0	36.7	35.0	37.2
	2	44.0	37.7	50.0	35.0	29.0	36.7
2014	1	30.0	37.4	39.0	37.8	61.0	36.9
	2	47.0	37.8	48.0	37.3	51.0	36.5

数据处理实例：The MovieLens Dataset
电影评分数据集

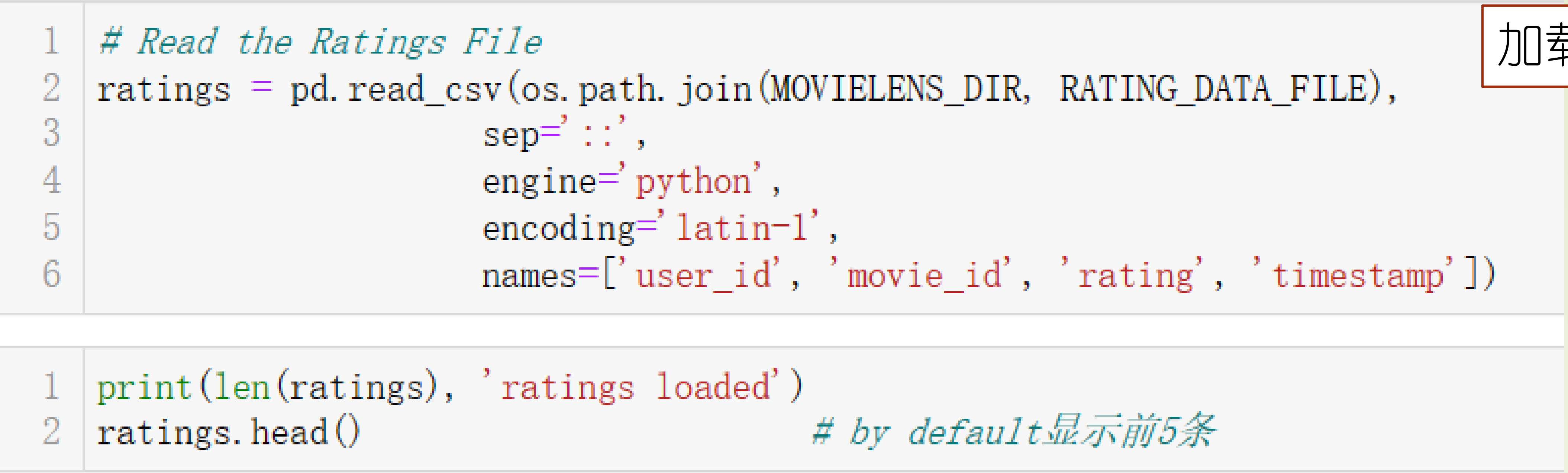


```
Neither the University of Minnesota nor any of the researchers
involved can guarantee the correctness of the data, its suitability
for any particular purpose, or the validity of results based on the
use of the data set.  The data set may be used for any research
purposes under the following conditions:

     * The user may not state or imply any endorsement from the
       University of Minnesota or the GroupLens Research Group.

     * The user must acknowledge the use of the data set in
       publications resulting from the use of the data set
       (see below for citation information).

     * The user may not redistribute the data without separate
       permission.

     * The user may not use this information for any commercial or
       revenue-bearing purposes without first obtaining permission
       from a faculty member of the GroupLens Research Project at the
       University of Minnesota.

If you have any further questions or comments, please contact GroupLens
<grouplens-info@cs.umn.edu>.

CITATION
================================================================================

To acknowledge use of the dataset in publications, please cite the following
paper:

F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History
and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4,
Article 19 (December 2015), 19 pages. DOI=http://dx.doi.org/10.1145/2827872
```



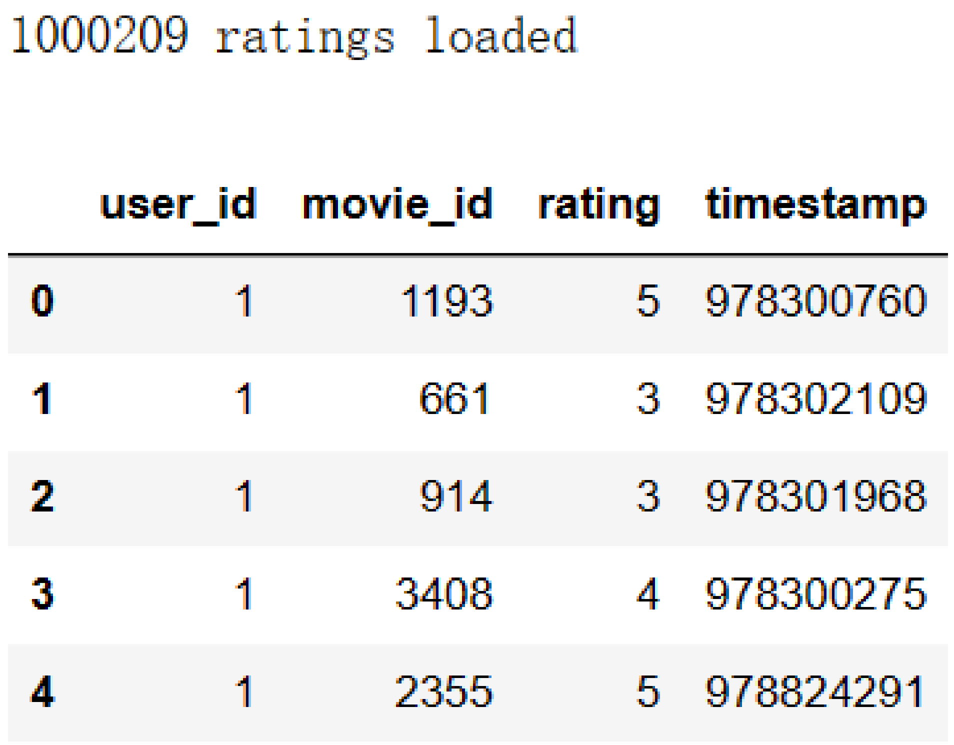
名称	修改日期	类型
movies	2018/7/10 20:01	DAT
ratings	2018/7/10 20:01	DAT
README	2018/7/10 20:01	MD 文件
users	2018/7/10 20:01	DAT

加载用户评分数据



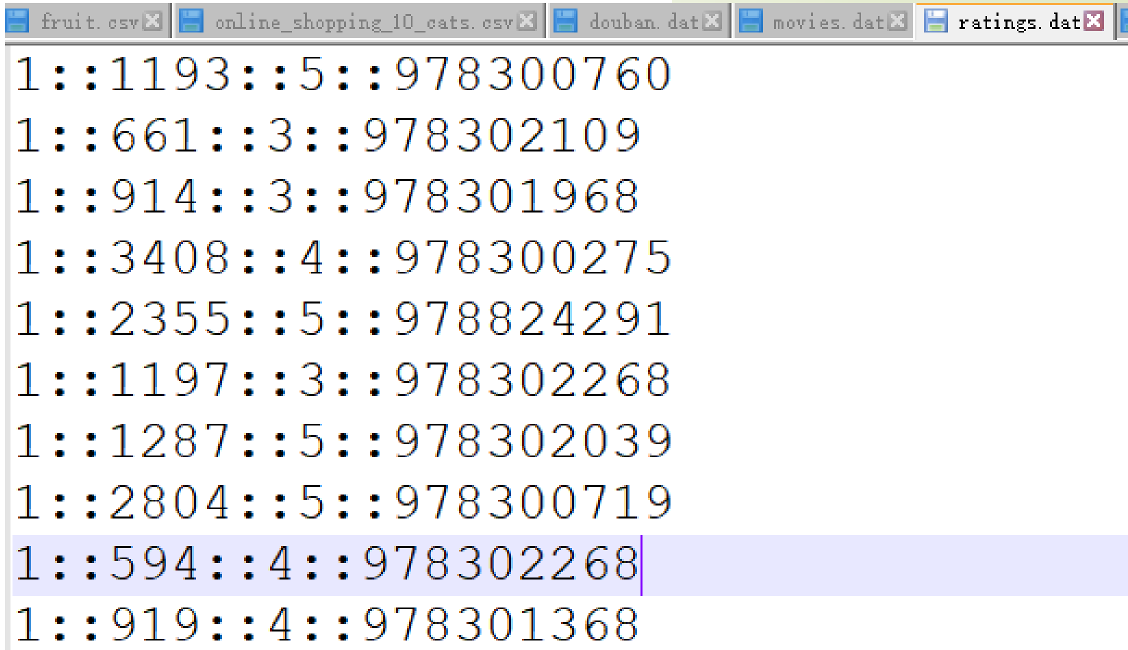
```python
# Read the Ratings File
ratings = pd.read_csv(os.path.join(MOVIELENS_DIR, RATING_DATA_FILE),
                    sep='::',
                    engine='python',
                    encoding='latin-1',
                    names=['user_id', 'movie_id', 'rating', 'timestamp'])
```

```python
print(len(ratings), 'ratings loaded')
ratings.head()                    # by default显示前5条
```

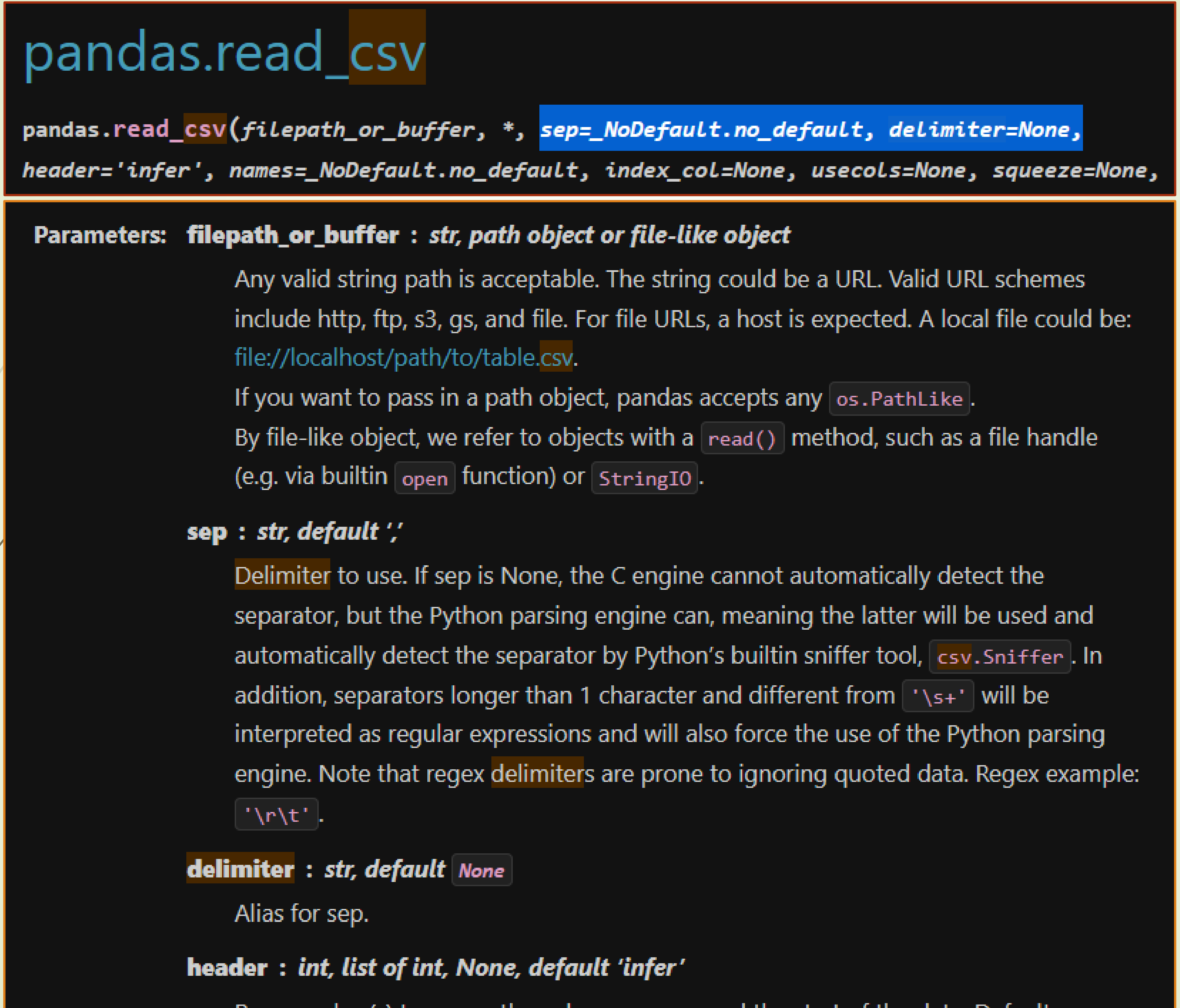


```
1000209 ratings loaded
```

	user_id	movie_id	rating	timestamp
0	1	1193	5	978300760
1	1	661	3	978302109
2	1	914	3	978301968
3	1	3408	4	978300275
4	1	2355	5	978824291



```
1::1193::5::978300760
1::661::3::978302109
1::914::3::978301968
1::3408::4::978300275
1::2355::5::978824291
1::1197::3::978302268
1::1287::5::978302039
1::2804::5::978300719
1::594::4::978302268
1::919::4::978301368
```



pandas.read_csv

```
pandas.read_csv(filepath_or_buffer, *, sep=_NoDefault.no_default, delimiter=None,
header='infer', names=_NoDefault.no_default, index_col=None, usecols=None, squeeze=None,
```

Parameters: **filepath_or_buffer** : *str, path object or file-like object*

Any valid string path is acceptable. The string could be a URL. Valid URL schemes include http, ftp, s3, gs, and file. For file URLs, a host is expected. A local file could be: file://localhost/path/to/table.csv.

If you want to pass in a path object, pandas accepts any `os.PathLike`.

By file-like object, we refer to objects with a `read()` method, such as a file handle (e.g. via builtin `open` function) or `StringIO`.

sep : *str, default ','*

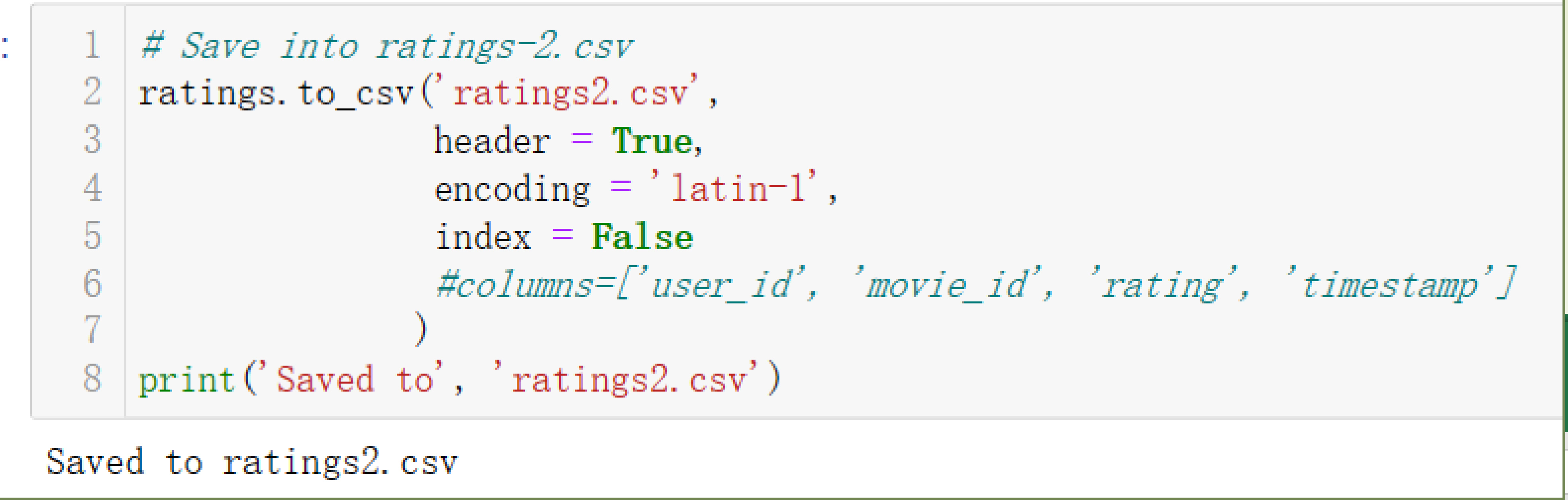
Delimiter to use. If sep is None, the C engine cannot automatically detect the separator, but the Python parsing engine can, meaning the latter will be used and automatically detect the separator by Python's builtin sniffer tool, `csv.Sniffer`. In addition, separators longer than 1 character and different from `'\s+'` will be interpreted as regular expressions and will also force the use of the Python parsing engine. Note that regex delimiters are prone to ignoring quoted data. Regex example: `'\r\t'`.

delimiter : *str, default* `None`

Alias for sep.

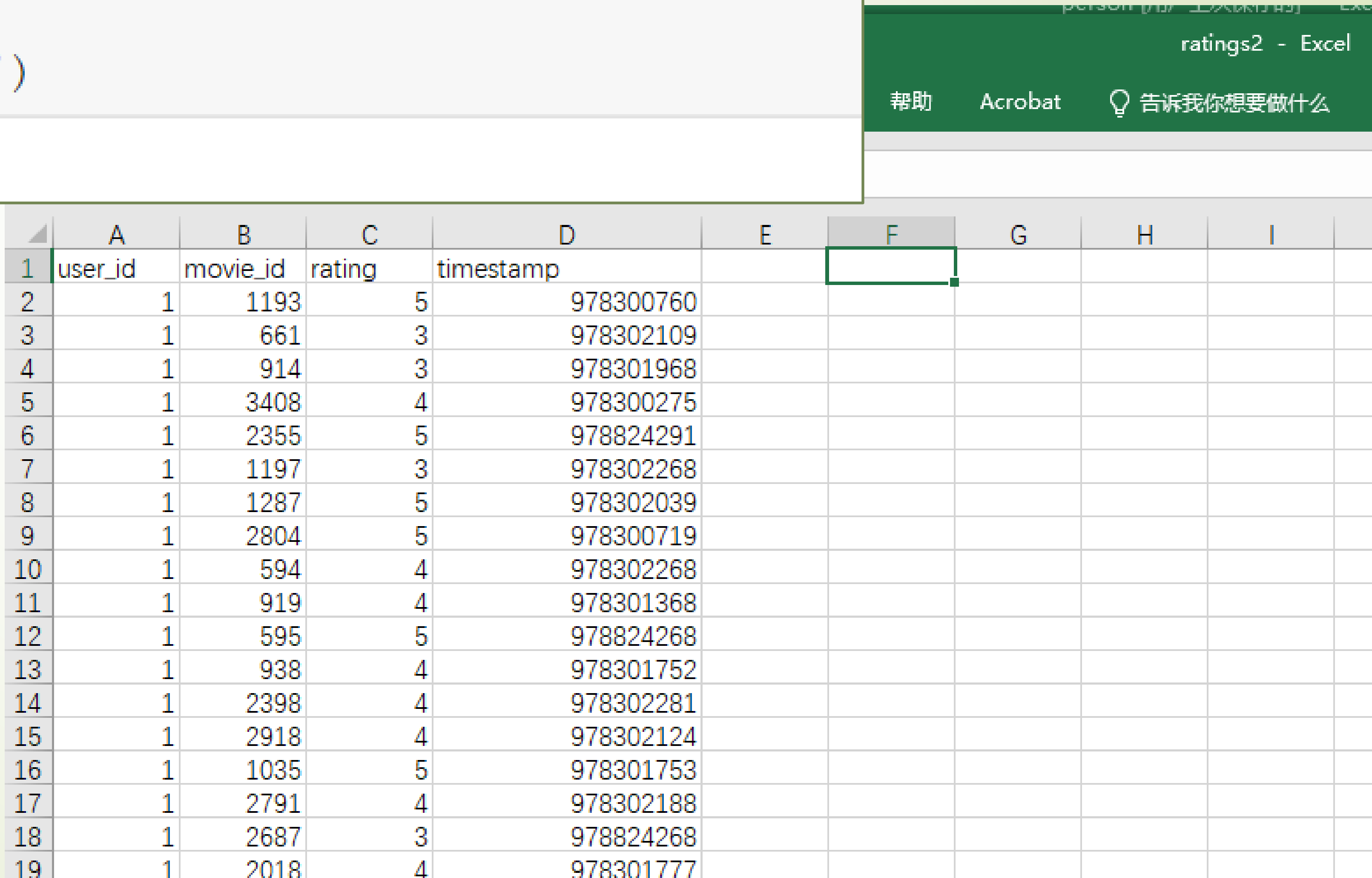
header : *int, list of int, None, default 'infer'*

写出数据表文件



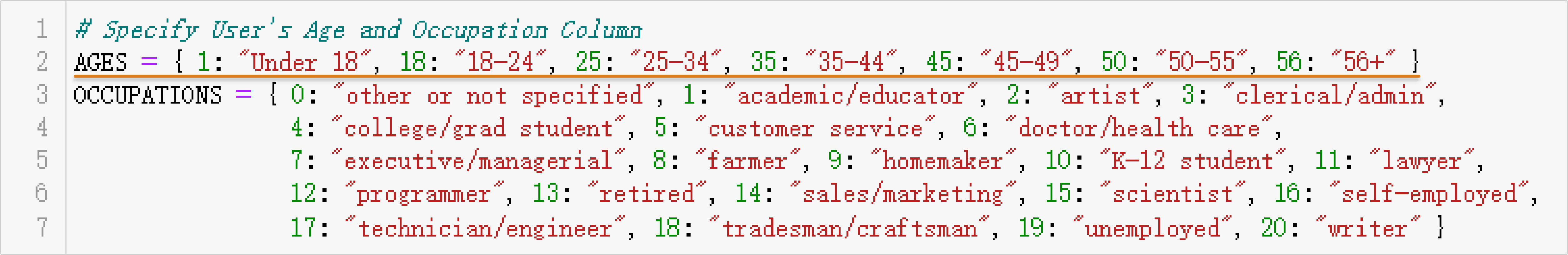
```
# Save into ratings-2.csv
ratings.to_csv('ratings2.csv',
               header = True,
               encoding = 'latin-1',
               index = False
               #columns=['user_id', 'movie_id', 'rating', 'timestamp']
              )
print('Saved to', 'ratings2.csv')
```

```
Saved to ratings2.csv
```

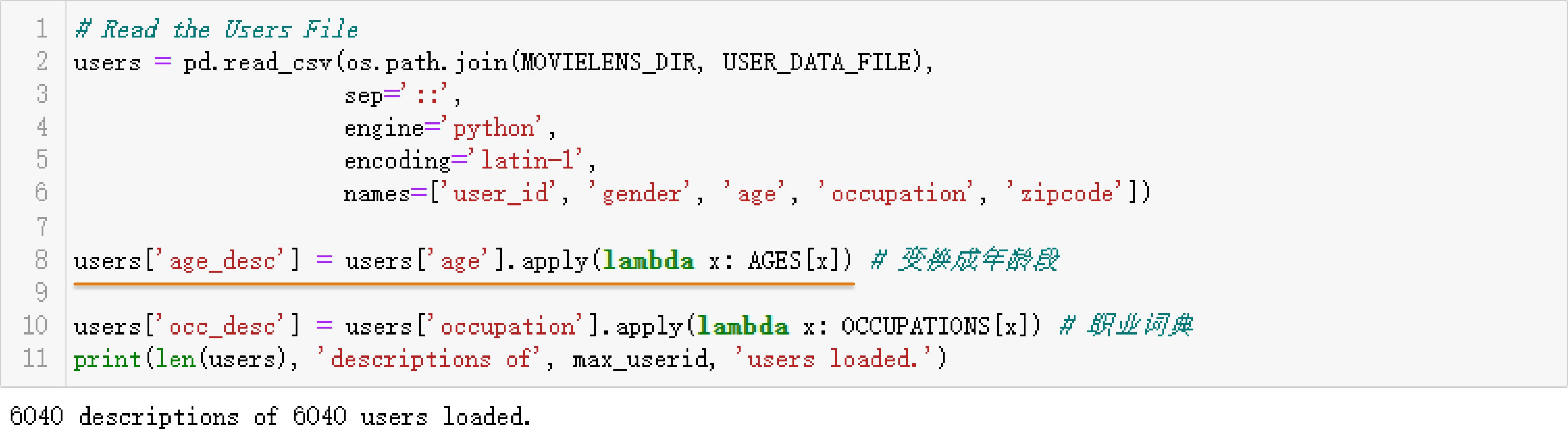


	A	B	C	D	E	F	G	H	I
1	user_id	movie_id	rating	timestamp					
2	1	1193	5	978300760					
3	1	661	3	978302109					
4	1	914	3	978301968					
5	1	3408	4	978300275					
6	1	2355	5	978824291					
7	1	1197	3	978302268					
8	1	1287	5	978302039					
9	1	2804	5	978300719					
10	1	594	4	978302268					
11	1	919	4	978301368					
12	1	595	5	978824268					
13	1	938	4	978301752					
14	1	2398	4	978302281					
15	1	2918	4	978302124					
16	1	1035	5	978301753					
17	1	2791	4	978302188					
18	1	2687	3	978824268					
19	1	2018	4	978301777					

连续值属性量化



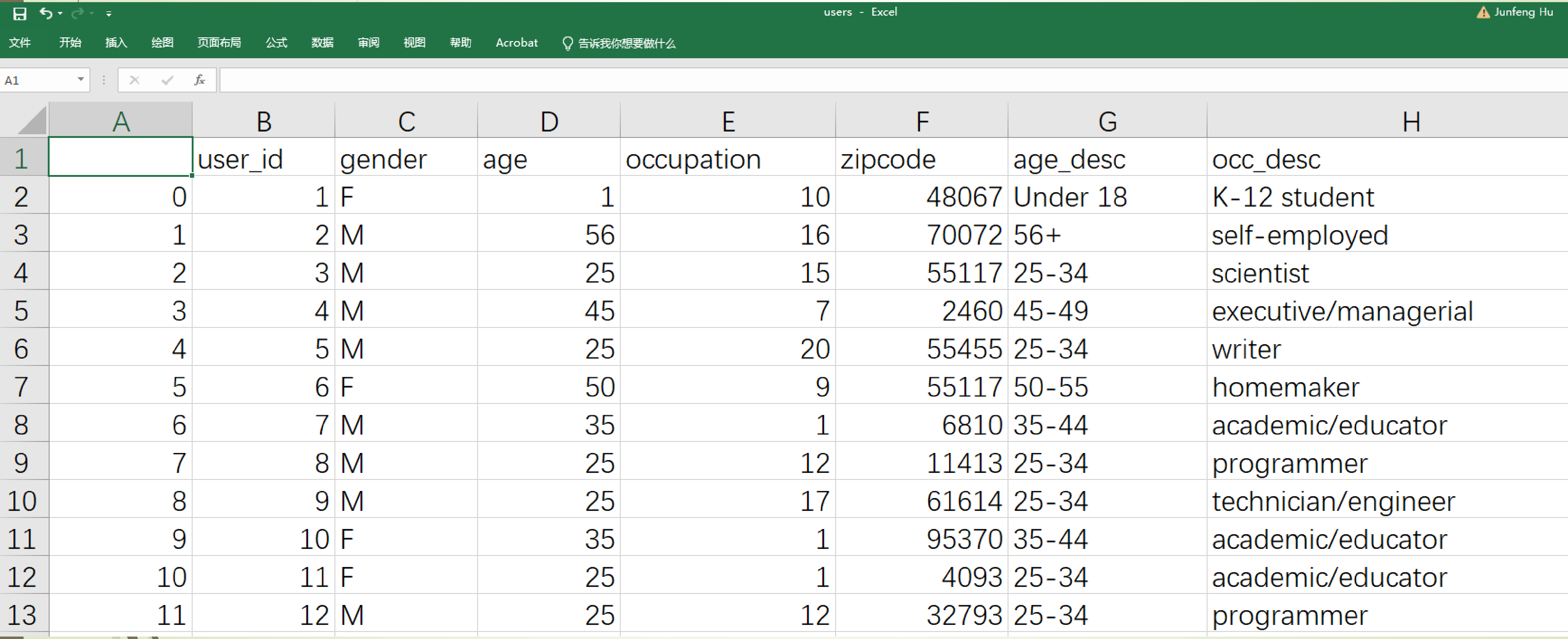
```python
# Specify User's Age and Occupation Column
AGES = { 1: "Under 18", 18: "18-24", 25: "25-34", 35: "35-44", 45: "45-49", 50: "50-55", 56: "56+" }
OCCUPATIONS = { 0: "other or not specified", 1: "academic/educator", 2: "artist", 3: "clerical/admin",
                4: "college/grad student", 5: "customer service", 6: "doctor/health care",
                7: "executive/managerial", 8: "farmer", 9: "homemaker", 10: "K-12 student", 11: "lawyer",
                12: "programmer", 13: "retired", 14: "sales/marketing", 15: "scientist", 16: "self-employed",
                17: "technician/engineer", 18: "tradesman/craftsman", 19: "unemployed", 20: "writer" }
```



```python
# Read the Users File
users = pd.read_csv(os.path.join(MOVIELENS_DIR, USER_DATA_FILE),
                    sep='::',
                    engine='python',
                    encoding='latin-1',
                    names=['user_id', 'gender', 'age', 'occupation', 'zipcode'])

users['age_desc'] = users['age'].apply(lambda x: AGES[x]) # 变换成年龄段

users['occ_desc'] = users['occupation'].apply(lambda x: OCCUPATIONS[x]) # 职业词典
print(len(users), 'descriptions of', max_userid, 'users loaded.')
```

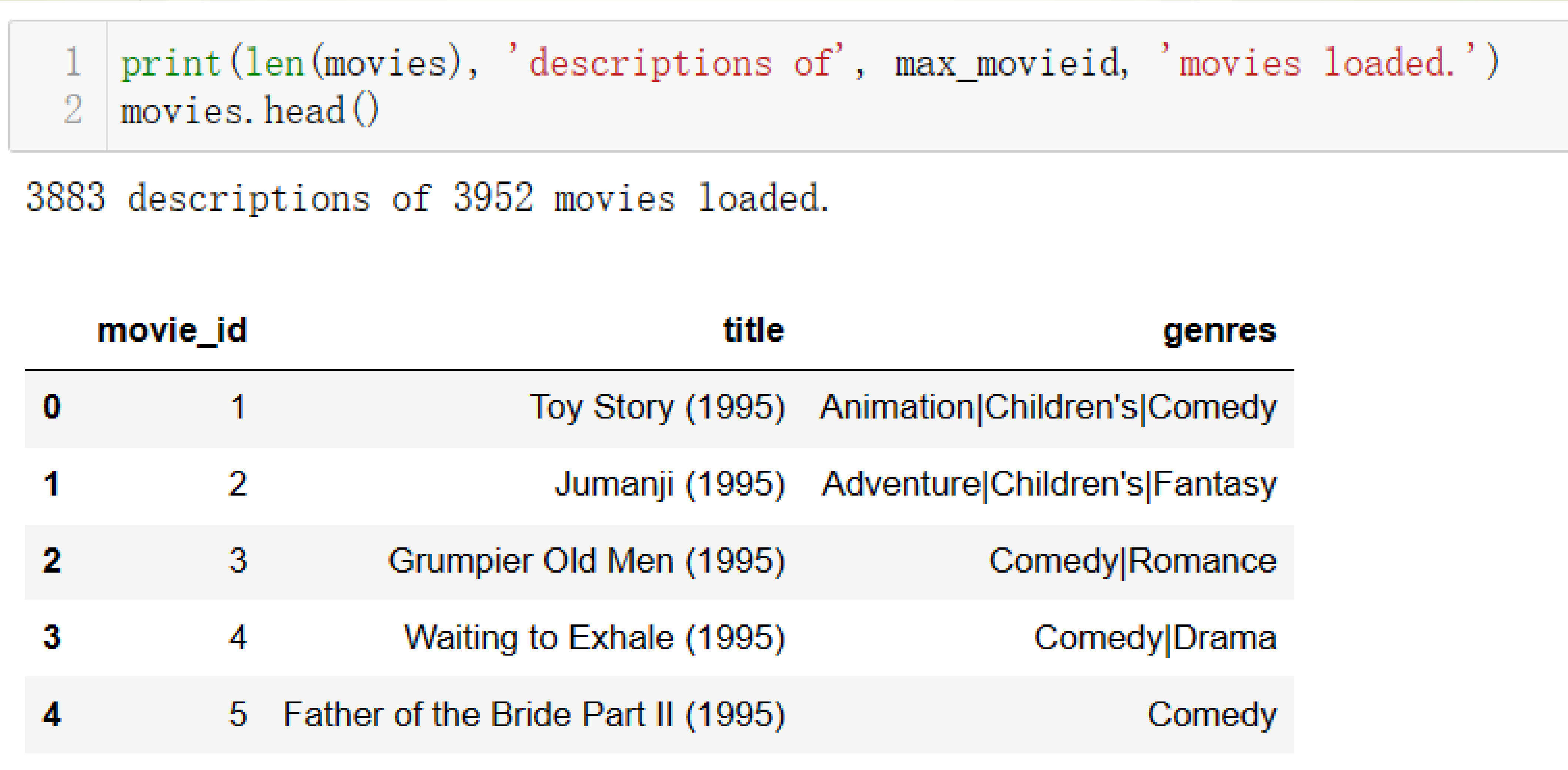
```
6040 descriptions of 6040 users loaded.
```

加工后的数据表：



	user_id	gender	age	occupation	zipcode	age_desc	occ_desc
0	1	F	1	10	48067	Under 18	K-12 student
1	2	M	56	16	70072	56+	self-employed
2	3	M	25	15	55117	25-34	scientist
3	4	M	45	7	2460	45-49	executive/managerial
4	5	M	25	20	55455	25-34	writer
5	6	F	50	9	55117	50-55	homemaker
6	7	M	35	1	6810	35-44	academic/educator
7	8	M	25	12	11413	25-34	programmer
8	9	M	25	17	61614	25-34	technician/engineer
9	10	F	35	1	95370	35-44	academic/educator
10	11	F	25	1	4093	25-34	academic/educator
11	12	M	25	12	32793	25-34	programmer

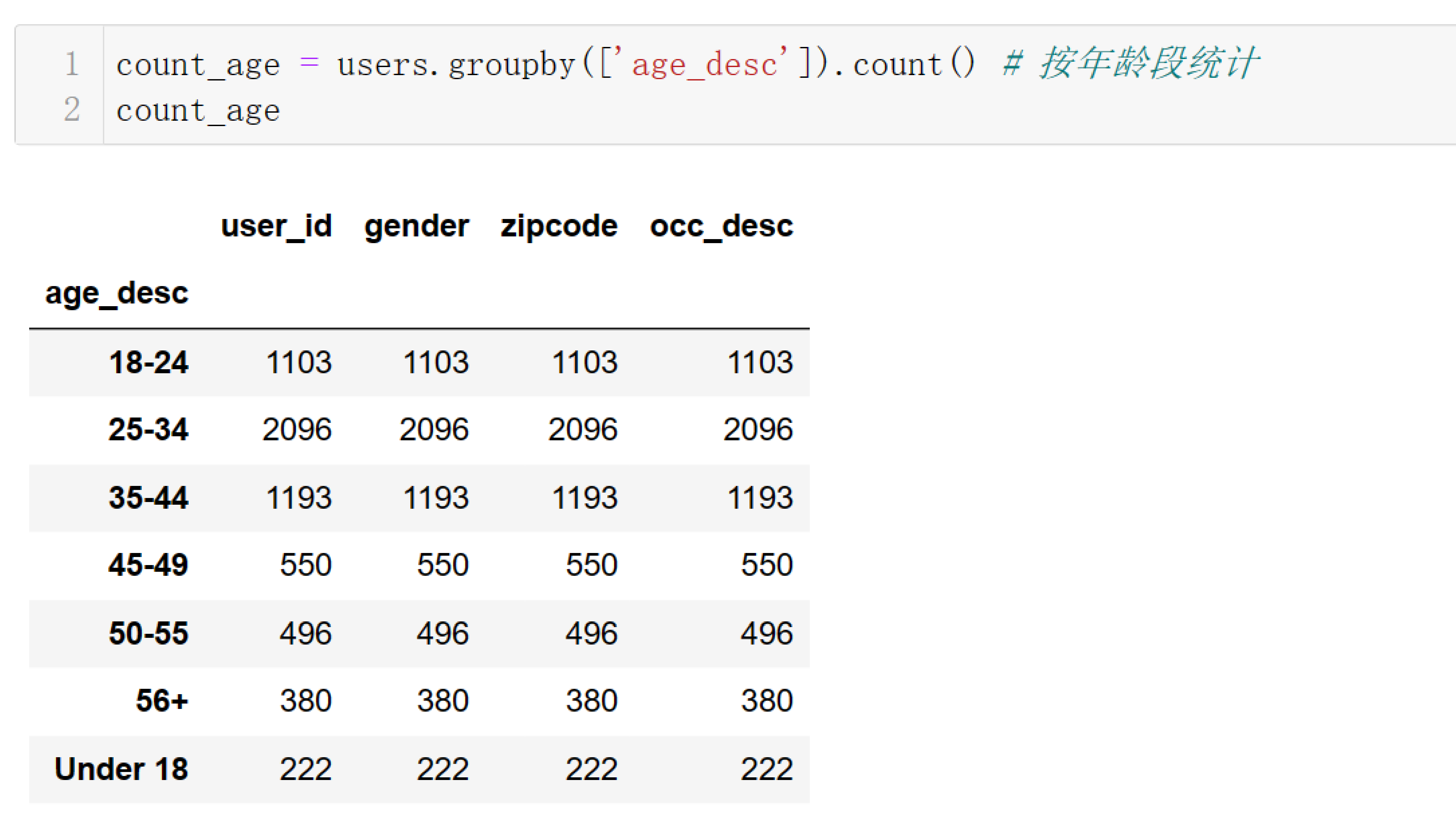
电影信息表：



```
print(len(movies), ’descriptions of’, max_movieid, ’movies loaded.’)
movies.head()
```

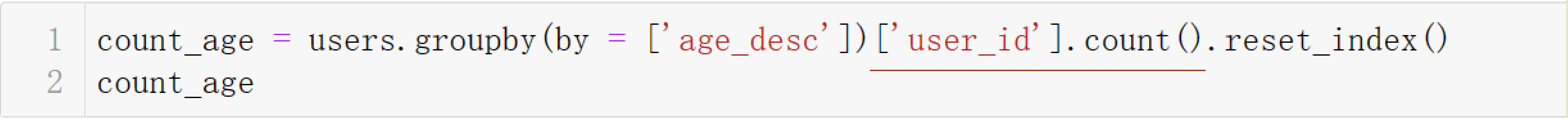
```
3883 descriptions of 3952 movies loaded.
```

	movie_id	title	genres
0	1	Toy Story (1995)	Animation|Children's|Comedy
1	2	Jumanji (1995)	Adventure|Children's|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama
4	5	Father of the Bride Part II (1995)	Comedy

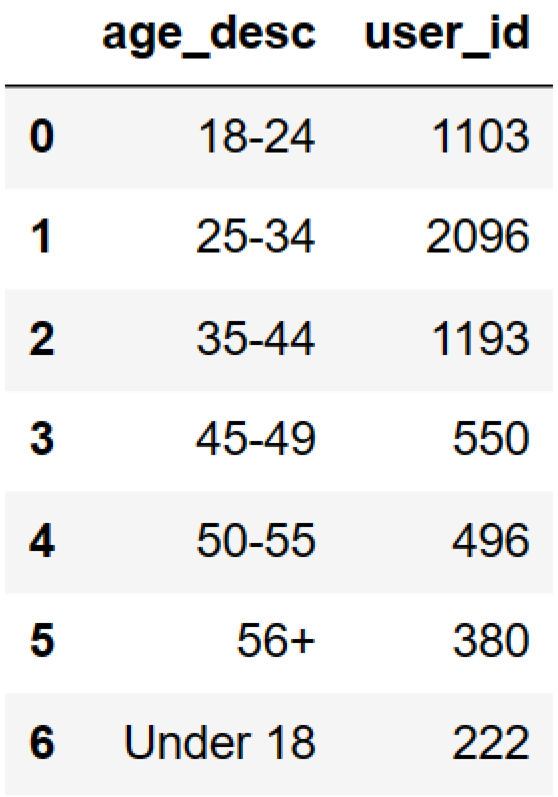


```python
count_age = users.groupby(['age_desc']).count() # 按年龄段统计
count_age
```

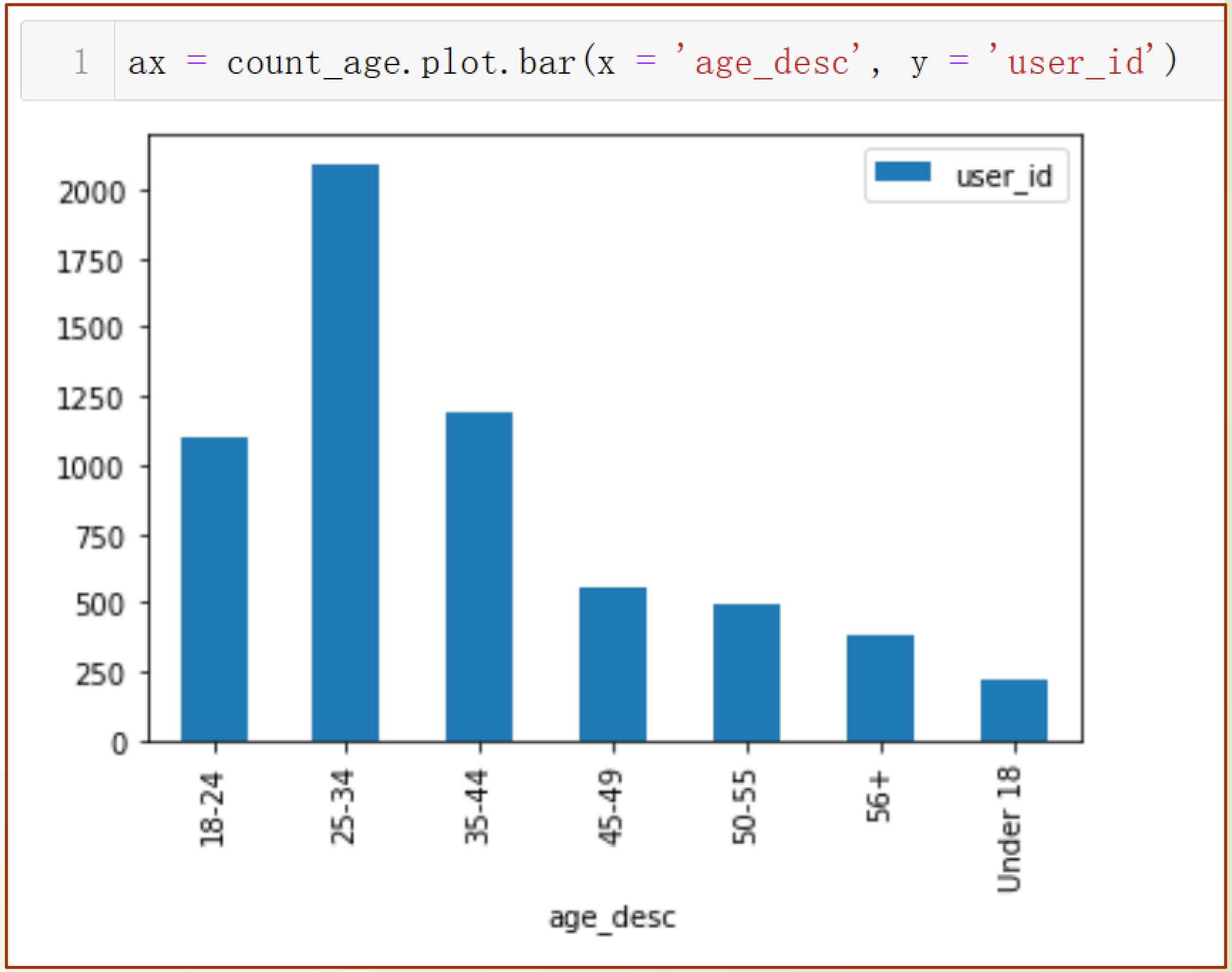
	user_id	gender	zipcode	occ_desc
age_desc				
18-24	1103	1103	1103	1103
25-34	2096	2096	2096	2096
35-44	1193	1193	1193	1193
45-49	550	550	550	550
50-55	496	496	496	496
56+	380	380	380	380
Under 18	222	222	222	222



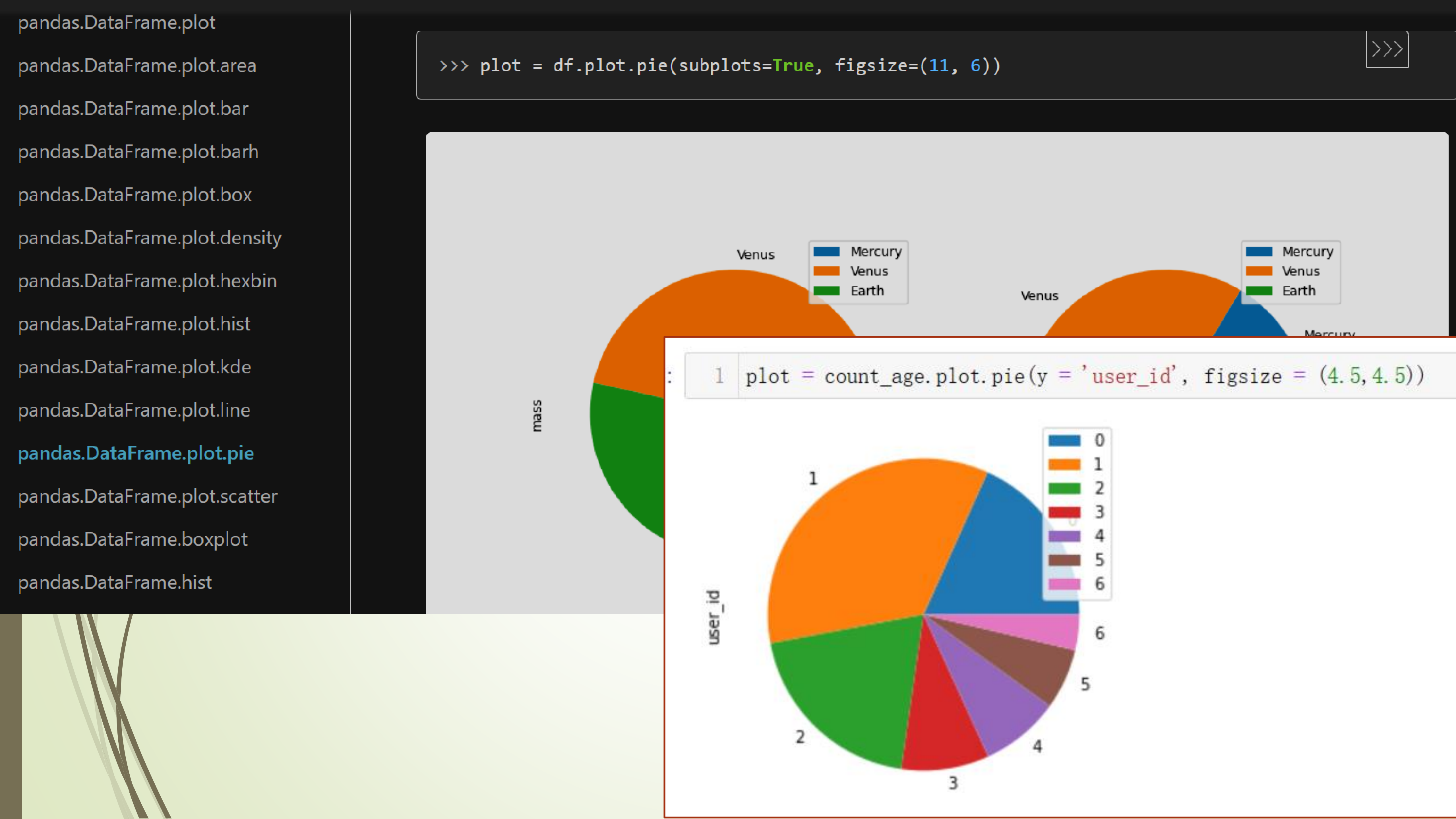
```
count_age = users.groupby(by = ['age_desc'])['user_id'].count().reset_index()
count_age
```



	age_desc	user_id
0	18-24	1103
1	25-34	2096
2	35-44	1193
3	45-49	550
4	50-55	496
5	56+	380
6	Under 18	222



```
ax = count_age.plot.bar(x = 'age_desc', y = 'user_id')
```



pandas.DataFrame.plot

pandas.DataFrame.plot.area

pandas.DataFrame.plot.bar

pandas.DataFrame.plot.barh

pandas.DataFrame.plot.box

pandas.DataFrame.plot.density

pandas.DataFrame.plot.hexbin

pandas.DataFrame.plot.hist

pandas.DataFrame.plot.kde

pandas.DataFrame.plot.line

pandas.DataFrame.plot.pie

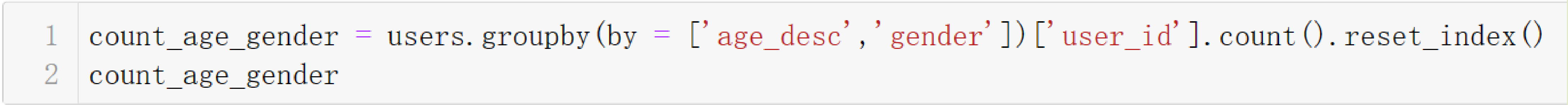
pandas.DataFrame.plot.scatter

pandas.DataFrame.boxplot

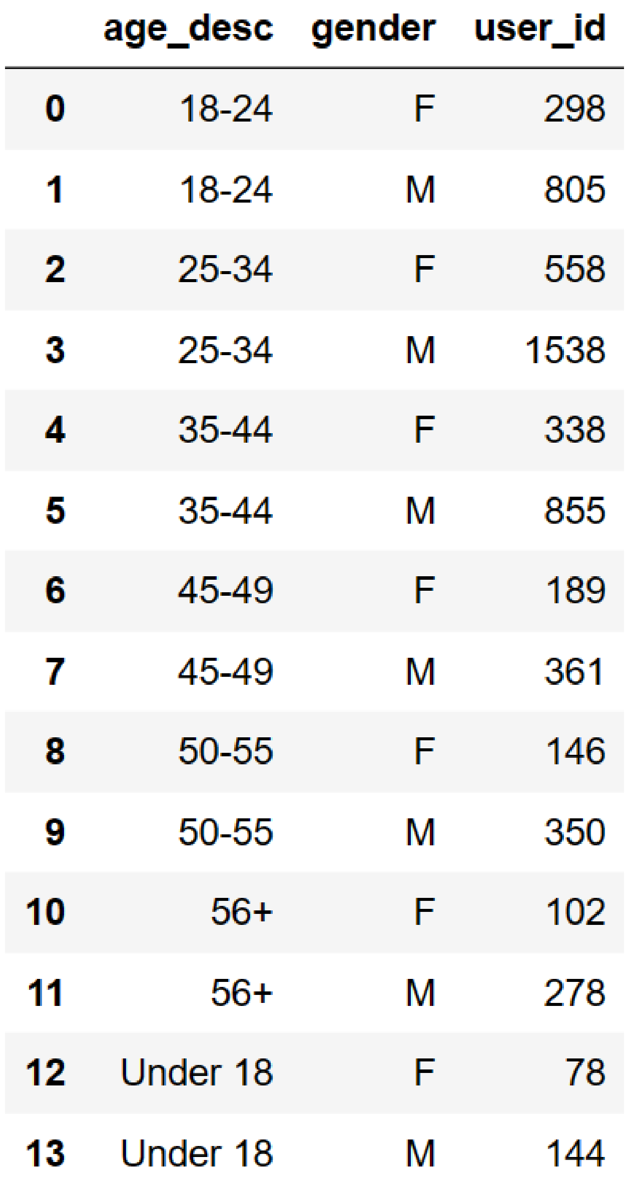
pandas.DataFrame.hist

```
>>> plot = df.plot.pie(subplots=True, figsize=(11, 6))
```

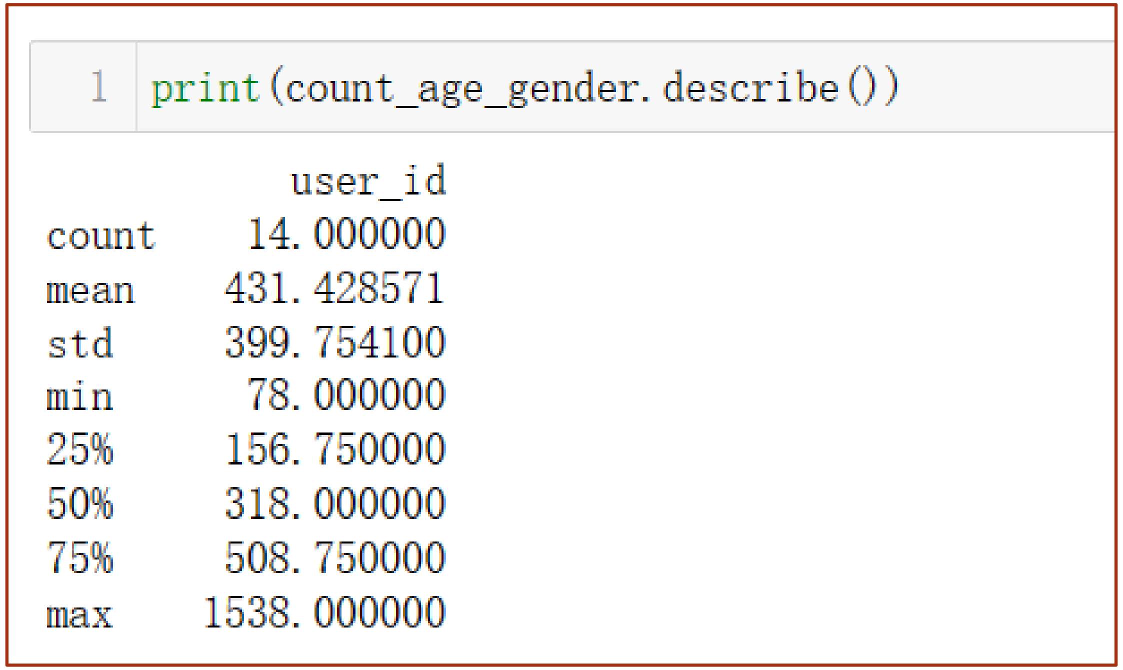
```
plot = count_age.plot.pie(y = 'user_id', figsize = (4.5,4.5))
```



```python
count_age_gender = users.groupby(by = ['age_desc','gender'])['user_id'].count().reset_index()
count_age_gender
```



	age_desc	gender	user_id
0	18-24	F	298
1	18-24	M	805
2	25-34	F	558
3	25-34	M	1538
4	35-44	F	338
5	35-44	M	855
6	45-49	F	189
7	45-49	M	361
8	50-55	F	146
9	50-55	M	350
10	56+	F	102
11	56+	M	278
12	Under 18	F	78
13	Under 18	M	144

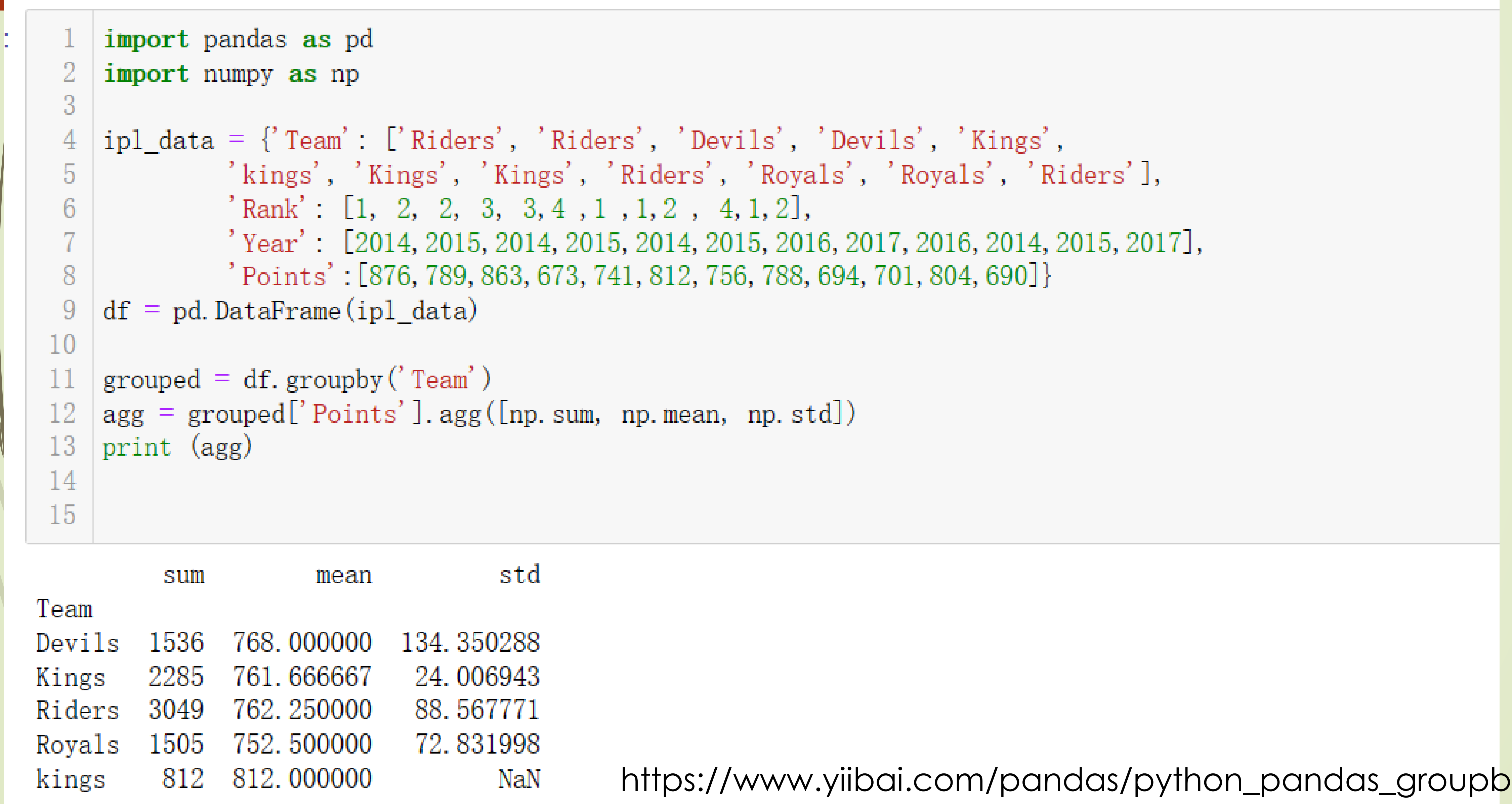


```python
print(count_age_gender.describe())
```

```
            user_id
count     14.000000
mean     431.428571
std      399.754100
min       78.000000
25%      156.750000
50%      318.000000
75%      508.750000
max     1538.000000
```

agg ()

聚合函数为每个组返回单个聚合值。 通过分组系列，还可以传递函数的列表或字典来进行聚合，并生成DataFrame



```
import pandas as pd
import numpy as np

ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
         'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
         'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
         'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
         'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}
df = pd.DataFrame(ipl_data)

grouped = df.groupby('Team')
agg = grouped['Points'].agg([np.sum, np.mean, np.std])
print (agg)


```

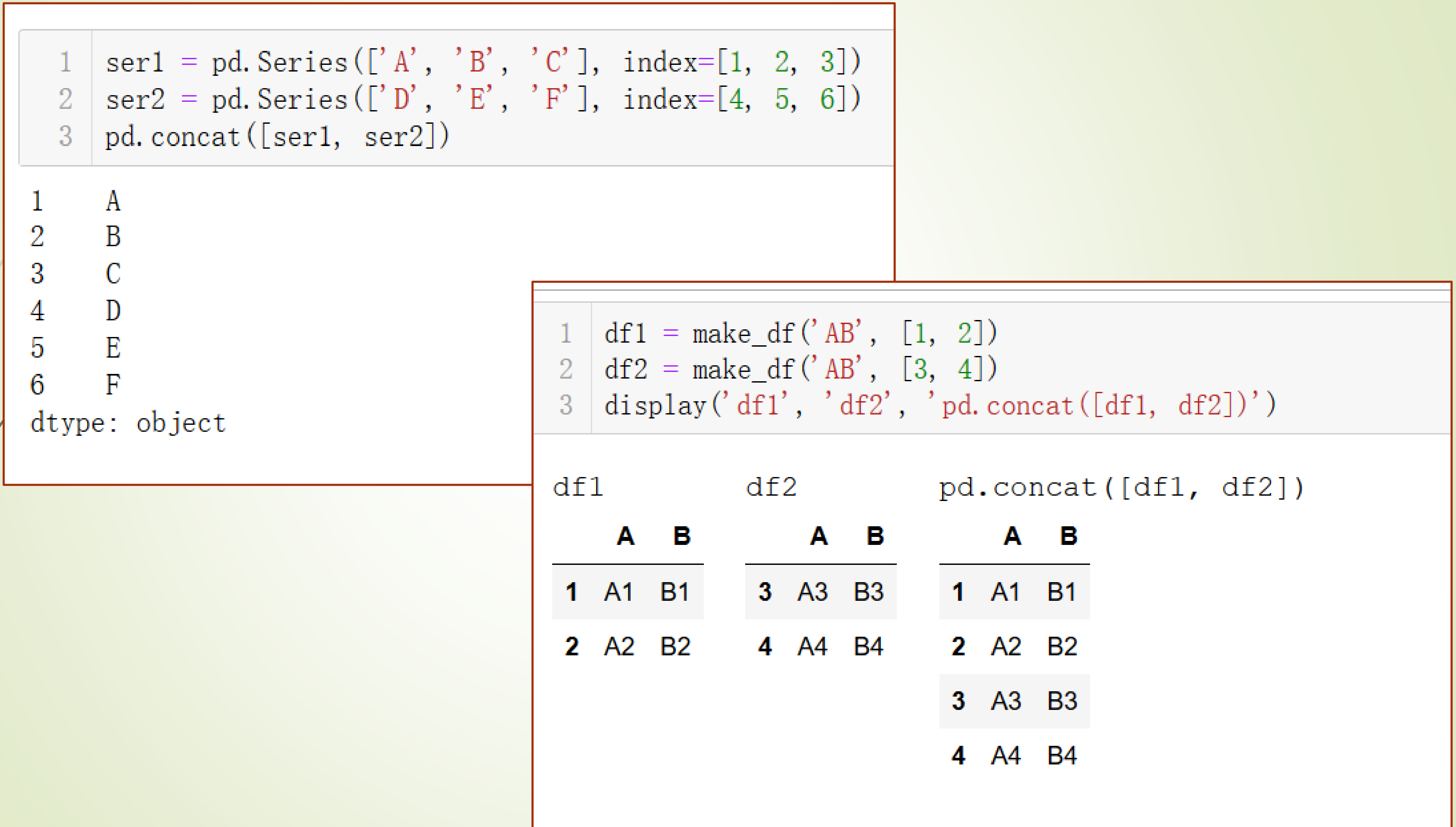
```
         sum        mean         std
Team
Devils  1536  768.000000  134.350288
Kings   2285  761.666667   24.006943
Riders  3049  762.250000   88.567771
Royals  1505  752.500000   72.831998
kings    812  812.000000         NaN
```

https://www.yiibai.com/pandas/python_pandas_groupby.html

Combining Datasets: Merge and Join

- *one-to-one*
- *many-to-one*
- *many-to-many* joins.

Combining Datasets: Concat and Append



```
ser1 = pd.Series(['A', 'B', 'C'], index=[1, 2, 3])
ser2 = pd.Series(['D', 'E', 'F'], index=[4, 5, 6])
pd.concat([ser1, ser2])
```

```
1     A
2     B
3     C
4     D
5     E
6     F
dtype: object
```

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

df1

	A	B
1	A1	B1
2	A2	B2

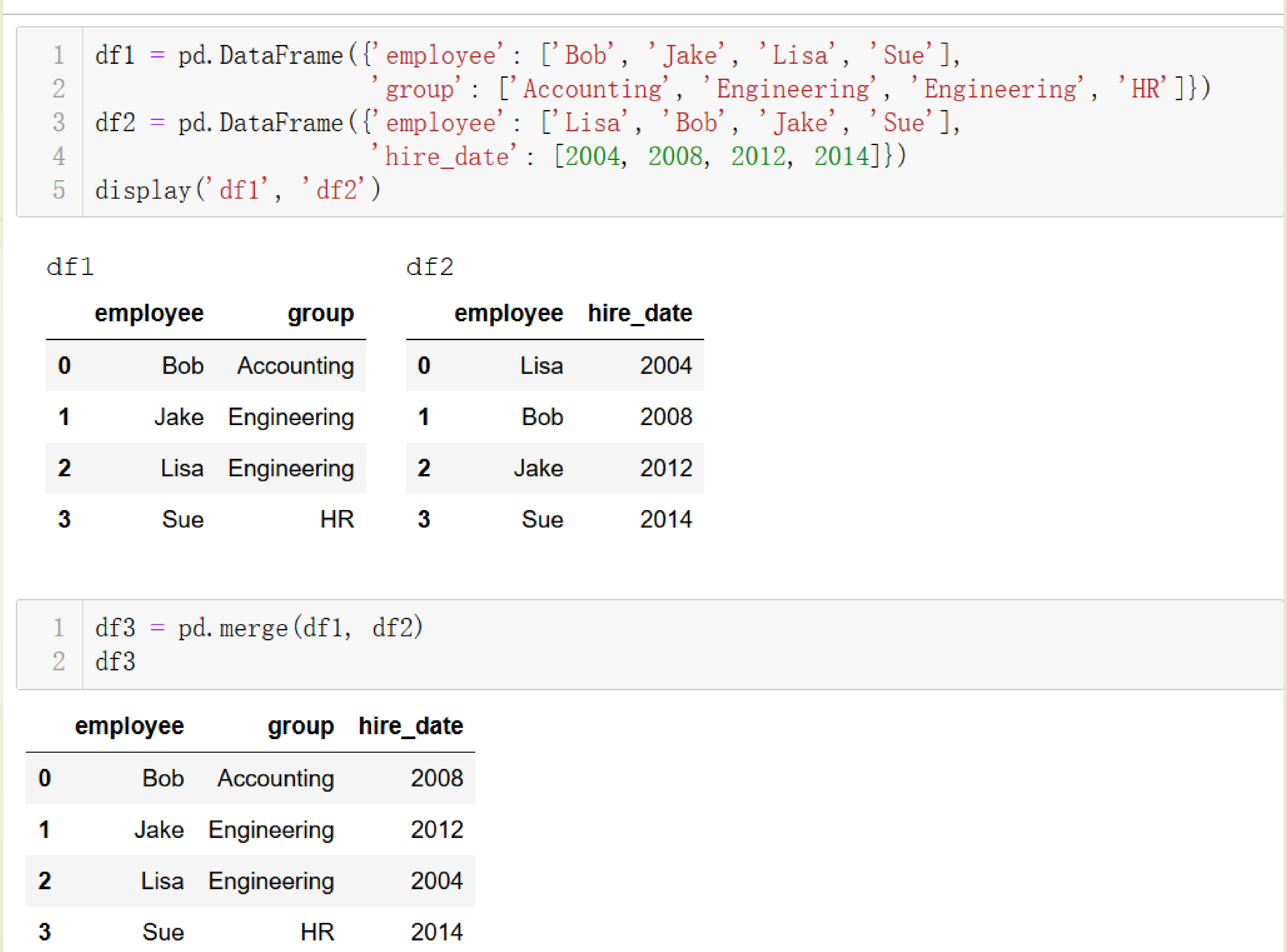
df2

	A	B
3	A3	B3
4	A4	B4

pd.concat([df1, df2])

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

One-to-one joins :



```
df1 = pd.DataFrame({'employee': ['Bob', 'Jake', 'Lisa', 'Sue'],
                    'group': ['Accounting', 'Engineering', 'Engineering', 'HR']})
df2 = pd.DataFrame({'employee': ['Lisa', 'Bob', 'Jake', 'Sue'],
                    'hire_date': [2004, 2008, 2012, 2014]})
display('df1', 'df2')
```

df1

	employee	group
0	Bob	Accounting
1	Jake	Engineering
2	Lisa	Engineering
3	Sue	HR

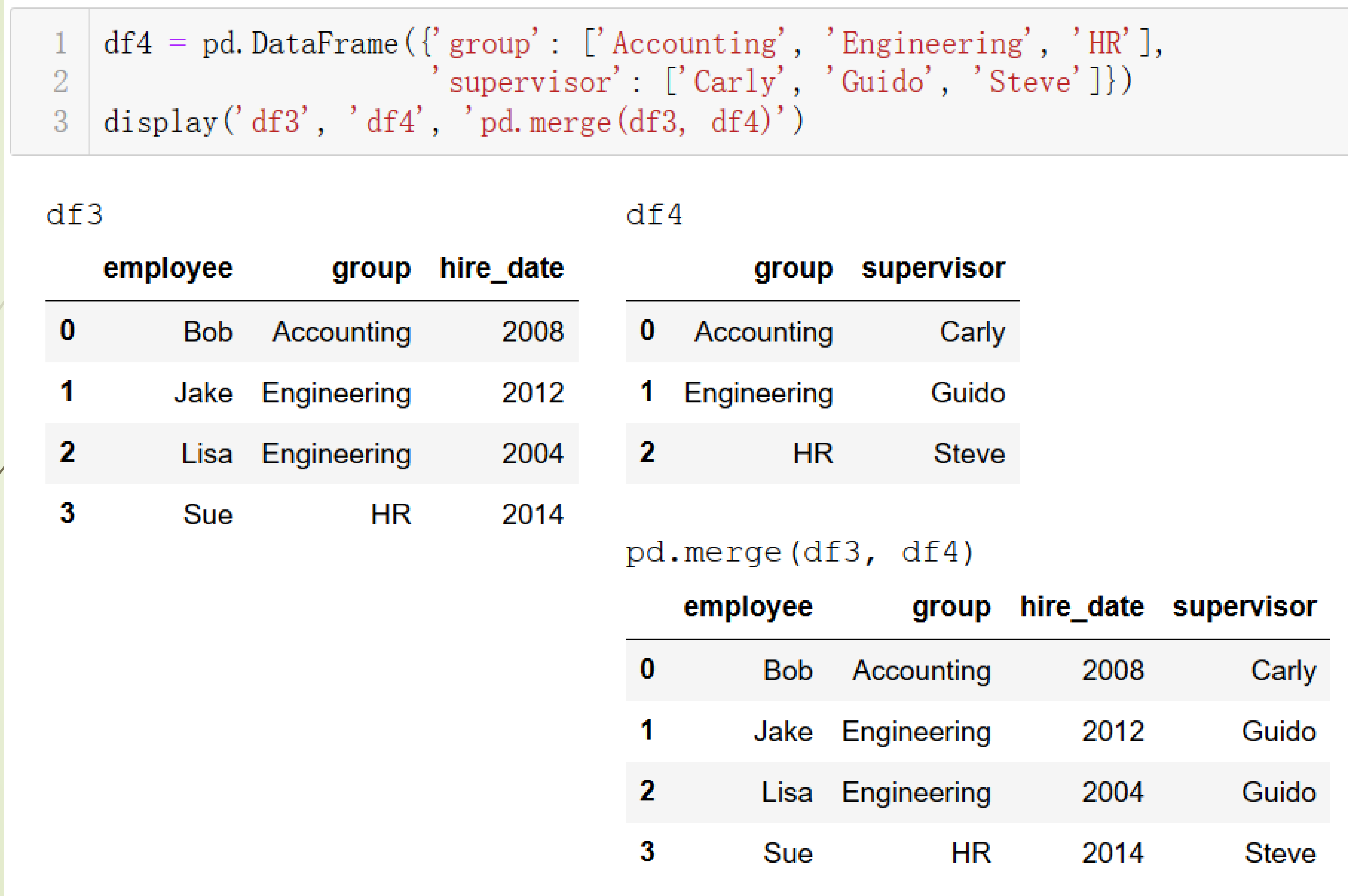
df2

	employee	hire_date
0	Lisa	2004
1	Bob	2008
2	Jake	2012
3	Sue	2014

```
df3 = pd.merge(df1, df2)
df3
```

	employee	group	hire_date
0	Bob	Accounting	2008
1	Jake	Engineering	2012
2	Lisa	Engineering	2004
3	Sue	HR	2014

Many-to-one joins 有重键值与对应的唯一键值进行join，单值对应的记录展开为多个：



```
df4 = pd.DataFrame({'group': ['Accounting', 'Engineering', 'HR'],
                    'supervisor': ['Carly', 'Guido', 'Steve']})
display('df3', 'df4', 'pd.merge(df3, df4)')
```

df3

	employee	group	hire_date
0	Bob	Accounting	2008
1	Jake	Engineering	2012
2	Lisa	Engineering	2004
3	Sue	HR	2014

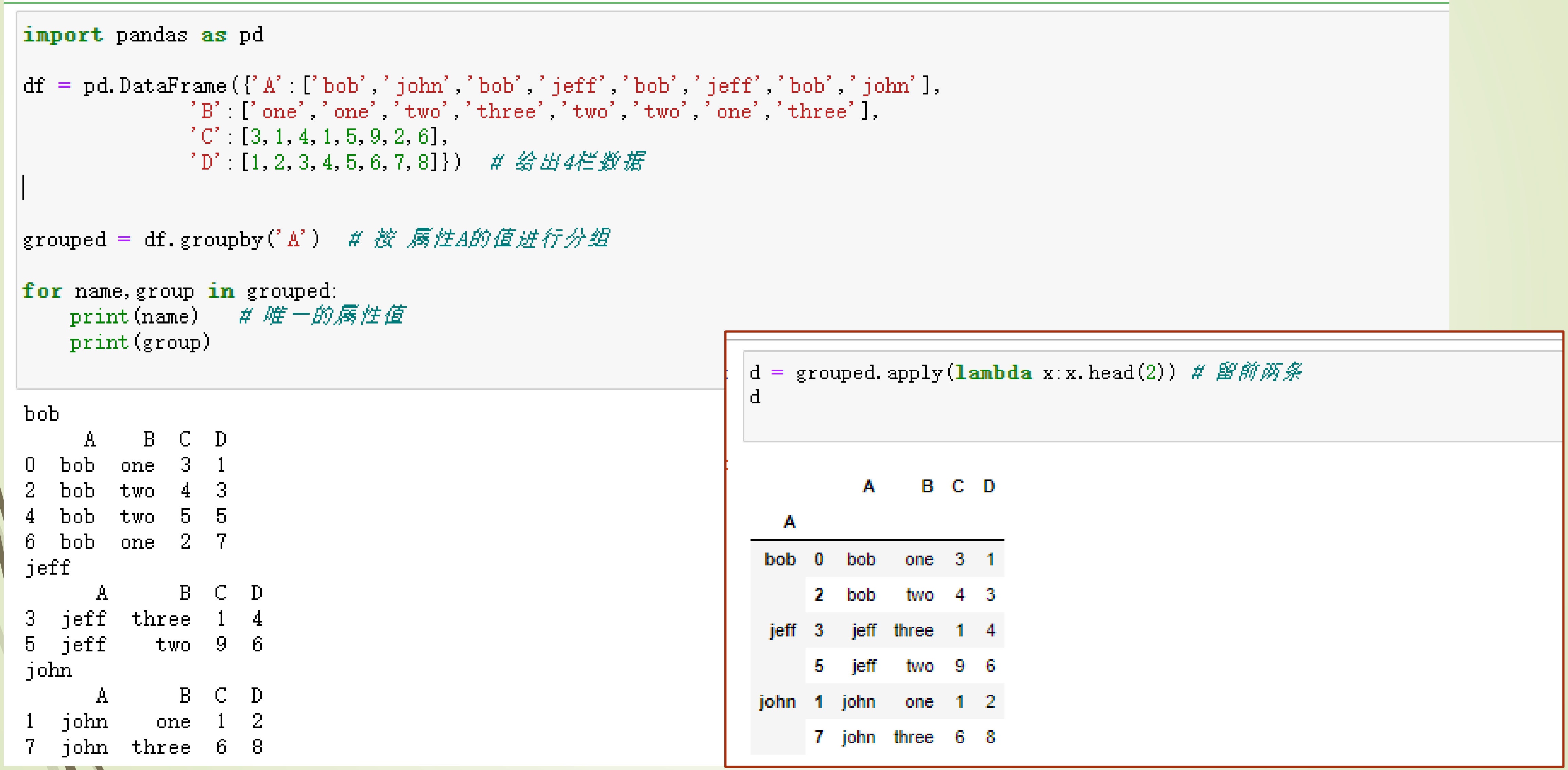
df4

	group	supervisor
0	Accounting	Carly
1	Engineering	Guido
2	HR	Steve

pd.merge(df3, df4)

	employee	group	hire_date	supervisor
0	Bob	Accounting	2008	Carly
1	Jake	Engineering	2012	Guido
2	Lisa	Engineering	2004	Guido
3	Sue	HR	2014	Steve

Pandas apply方法



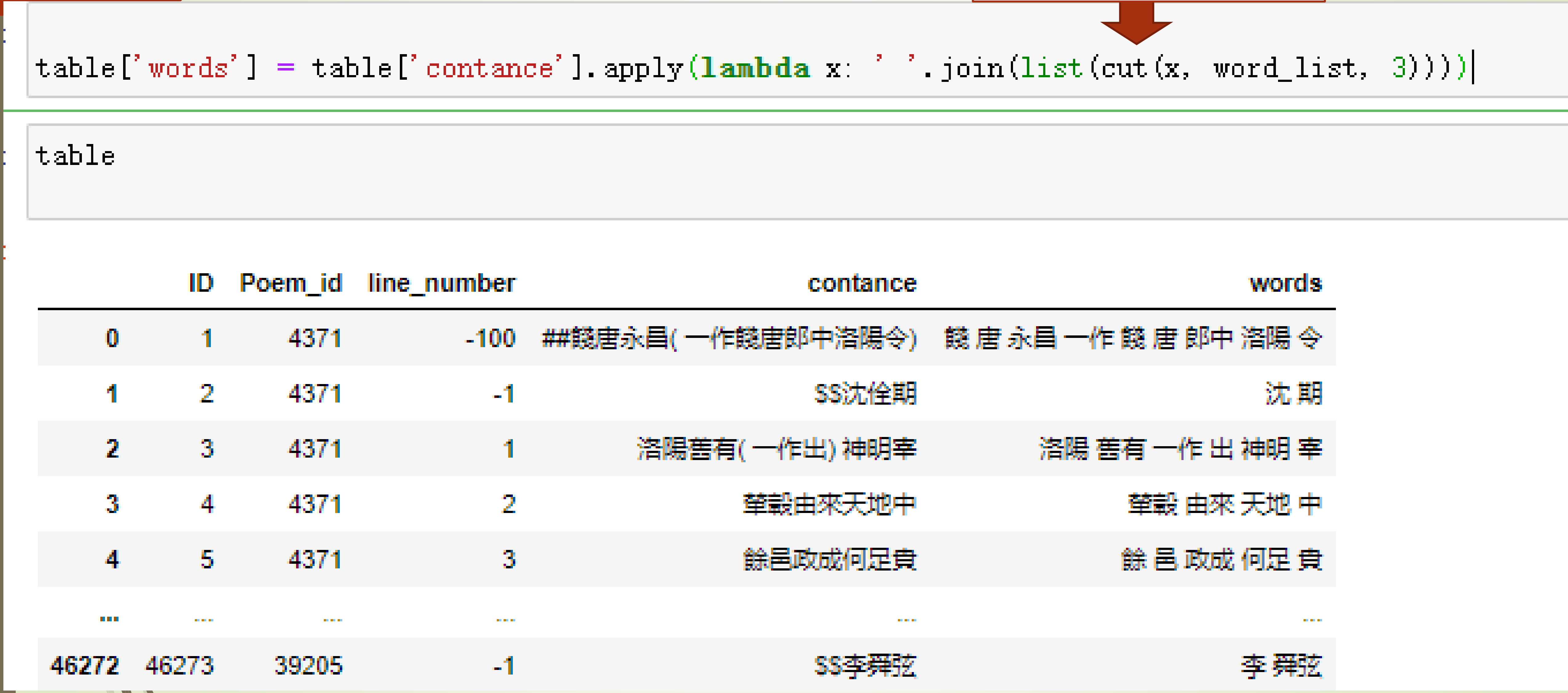
```
import pandas as pd

df = pd.DataFrame({'A':['bob','john','bob','jeff','bob','jeff','bob','john'],
                   'B':['one','one','two','three','two','two','one','three'],
                   'C':[3,1,4,1,5,9,2,6],
                   'D':[1,2,3,4,5,6,7,8]})  # 给出4栏数据


grouped = df.groupby('A')  # 按 属性A的值进行分组

for name,group in grouped:
    print(name)   # 唯一的属性值
    print(group)
```

```
bob
     A    B  C  D
0  bob  one  3  1
2  bob  two  4  3
4  bob  two  5  5
6  bob  one  2  7
jeff
      A      B  C  D
3  jeff  three  1  4
5  jeff    two  9  6
john
      A      B  C  D
1  john    one  1  2
7  john  three  6  8
```

```
d = grouped.apply(lambda x:x.head(2)) # 留前两条
d
```

		A	B	C	D
A					
bob	0	bob	one	3	1
	2	bob	two	4	3
jeff	3	jeff	three	1	4
	5	jeff	two	9	6
john	1	john	one	1	2
	7	john	three	6	8

自定义最大向前匹配函数

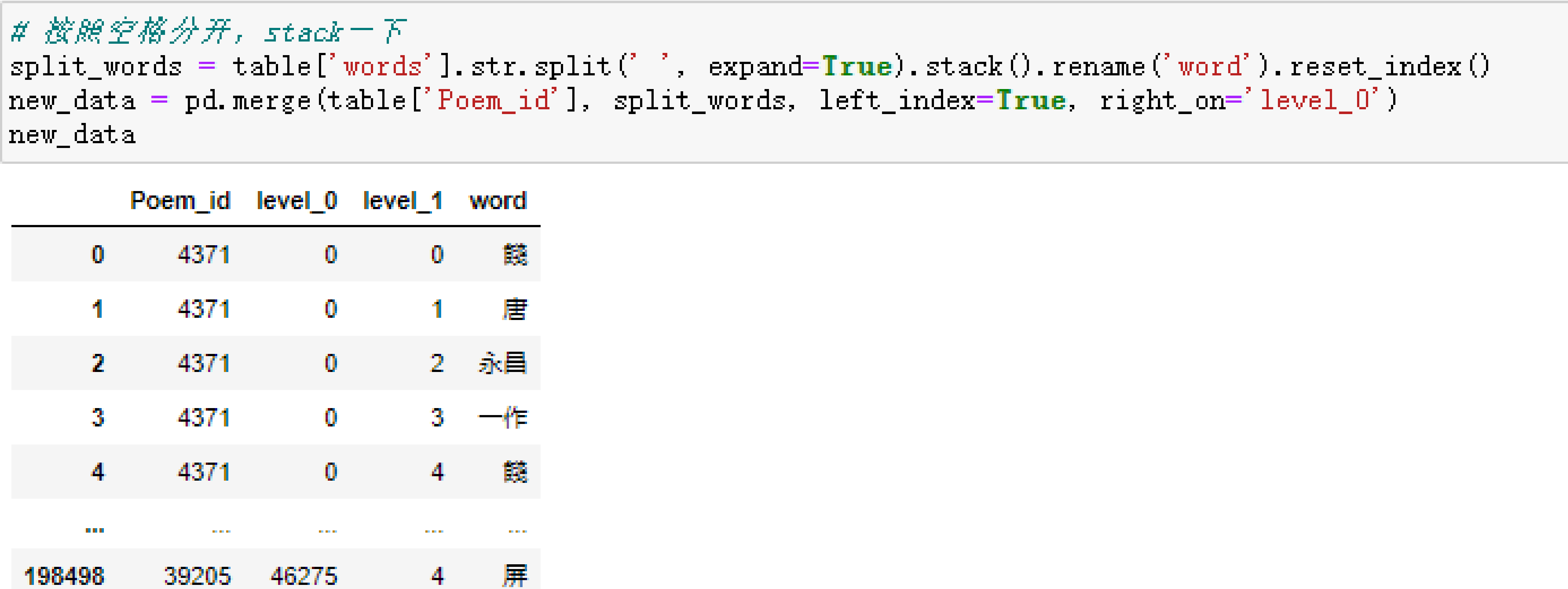


```
table['words'] = table['contance'].apply(lambda x: ' '.join(list(cut(x, word_list, 3))))
```

```
table
```

	ID	Poem_id	line_number	contance	words
0	1	4371	-100	##饒唐永昌(一作饒唐郎中洛陽令)	饒 唐 永昌 一作 饒 唐 郎中 洛陽 令
1	2	4371	-1	SS沈佺期	沈 期
2	3	4371	1	洛陽舊有(一作出) 神明宰	洛陽 舊有 一作 出 神明 宰
3	4	4371	2	輦轂由來天地中	輦轂 由來 天地 中
4	5	4371	3	餘邑政成何足貴	餘 邑 政成 何足 貴
...	...	...	...	...	...
46272	46273	39205	-1	SS李舜弦	李 舜弦

1.2 统计每个词的TF-IDF值



```
# 按照空格分开，stack一下
split_words = table['words'].str.split(' ', expand=True).stack().rename('word').reset_index()
new_data = pd.merge(table['Poem_id'], split_words, left_index=True, right_on='level_0')
new_data
```

	Poem_id	level_0	level_1	word
0	4371	0	0	餞
1	4371	0	1	唐
2	4371	0	2	永昌
3	4371	0	3	一作
4	4371	0	4	餞
...	...	...	...	...
198498	39205	46275	4	屏

Reference :

- Python Data Science Handbook :

  https://www.oreilly.com/library/view/python-data-science/9781491912126/