数据分析

Numpy 和 Pandas

Numpy是基于C语言编写，比Python内置的list和set运算速度更快

用到矩阵计算，方便，快速

Numpy 和 Pandas安装

```
pip3 install numpy scipy matplotlib pandas -i https://pypi.tuna.tsinghua.edu.cn/simple
```

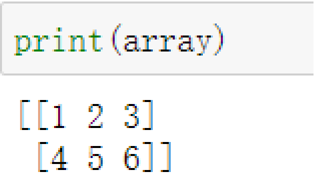
#anaconda 虚拟环境带有numpy和pandas

Numpy

1

打印np

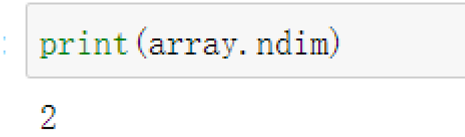
```
array = np.array([[1,2,3],[4,5,6]])
#输出的np没有逗号
```



```
print(array)
```

```
[[1 2 3]
 [4 5 6]]
```

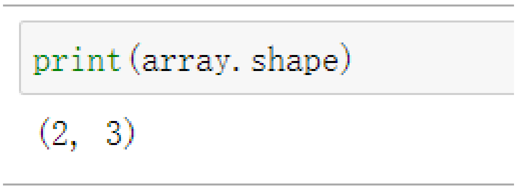
np的维度



```
print(array.ndim)
```

```
2
```

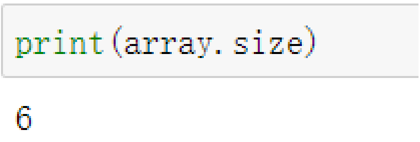
np的形状（2行3列）



```
print(array.shape)
```

```
(2, 3)
```

np的长度

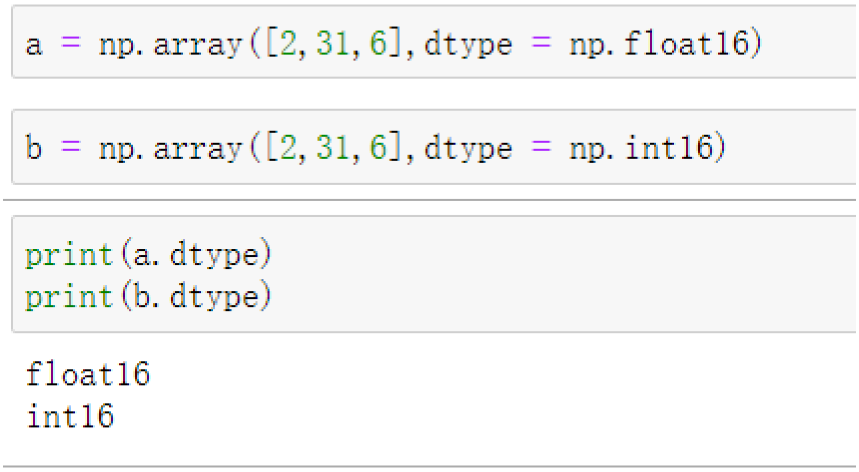


```
print(array.size)
```

```
6
```

2 基础

array赋值



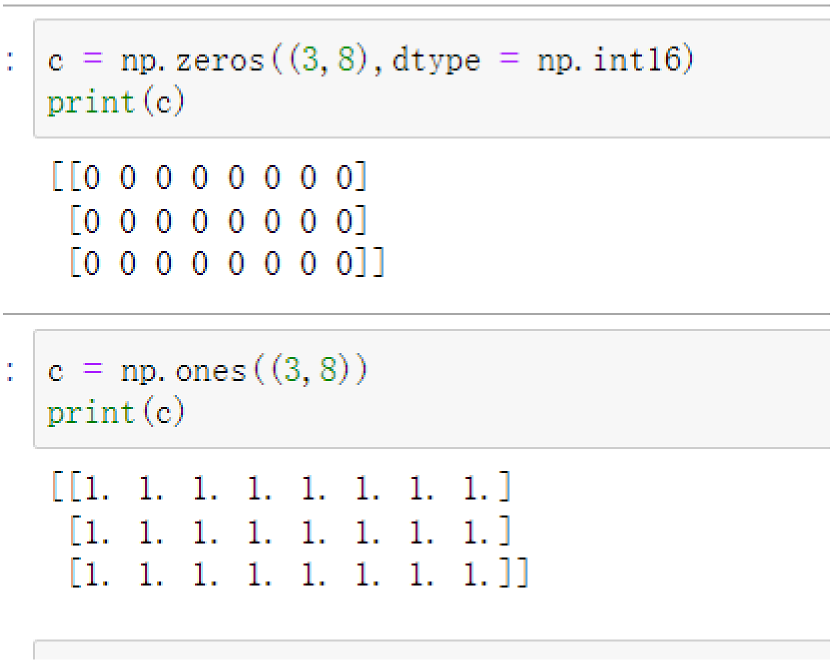
```
a = np.array([2,31,6],dtype = np.float16)
```

```
b = np.array([2,31,6],dtype = np.int16)
```

```
print(a.dtype)
print(b.dtype)
```

```
float16
int16
```

生成全0和全1矩阵



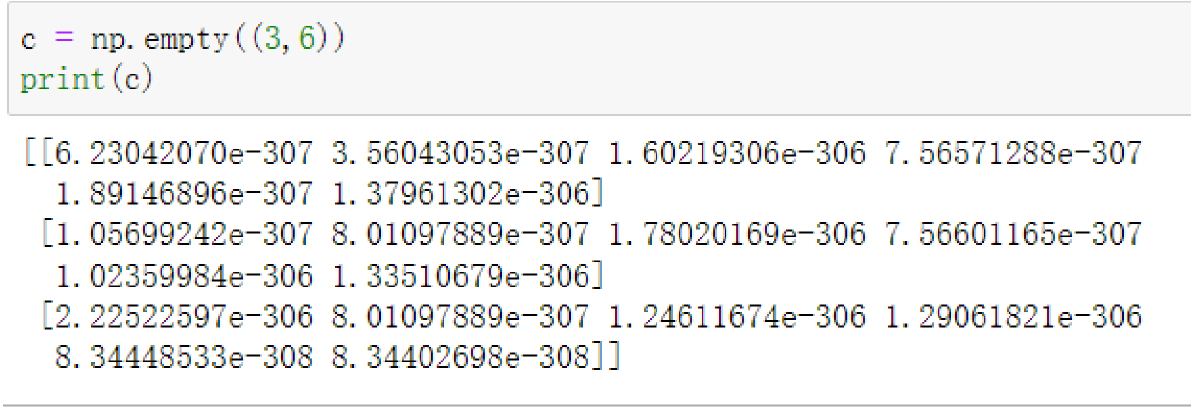
```
c = np.zeros((3,8),dtype = np.int16)
print(c)
```

```
[[0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]]
```

```
c = np.ones((3,8))
print(c)
```

```
[[1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1.]]
```

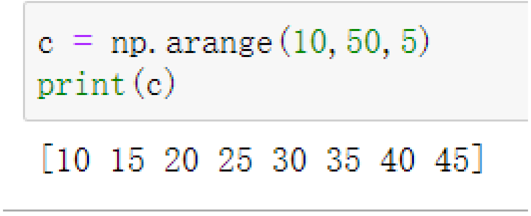
empty#生成空矩阵，元素非常接近0



```
c = np.empty((3,6))
print(c)
```

```
[[6.23042070e-307 3.56043053e-307 1.60219306e-306 7.56571288e-307
  1.89146896e-307 1.37961302e-306]
 [1.05699242e-307 8.01097889e-307 1.78020169e-306 7.56601165e-307
  1.02359984e-306 1.33510679e-306]
 [2.22522597e-306 8.01097889e-307 1.24611674e-306 1.29061821e-306
  8.34448533e-308 8.34402698e-308]]
```

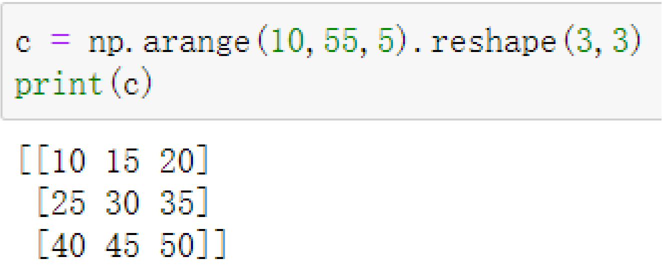
arange



```
c = np.arange(10,50,5)
print(c)
```

```
[10 15 20 25 30 35 40 45]
```

#重新定义形状

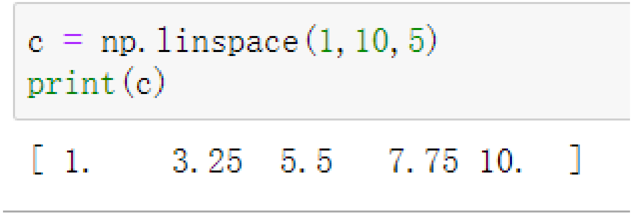


```
c = np.arange(10,55,5).reshape(3,3)
print(c)
```

```
[[10 15 20]
 [25 30 35]
 [40 45 50]]
```

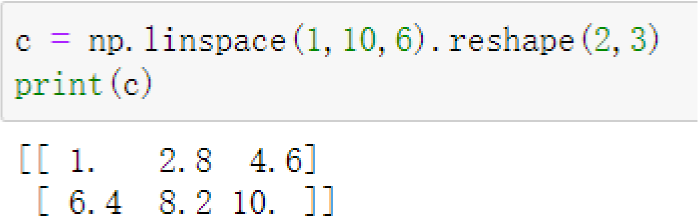
linspace

#在1到10之间，分割 step - 1 段



```
c = np.linspace(1,10,5)
print(c)
```

```
[ 1.    3.25  5.5   7.75 10.  ]
```

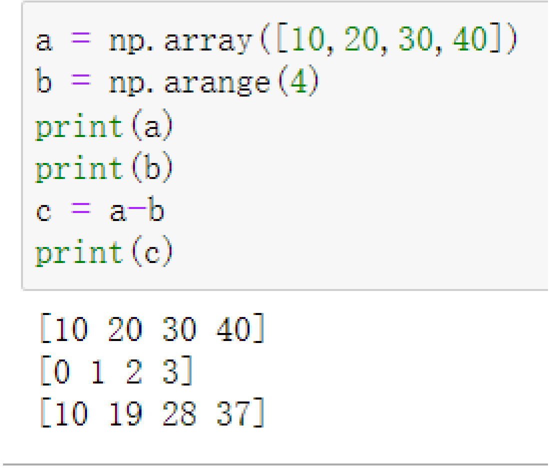


```
c = np.linspace(1,10,6).reshape(2,3)
print(c)
```

```
[[ 1.   2.8  4.6]
 [ 6.4  8.2 10. ]]
```

3

基础运算

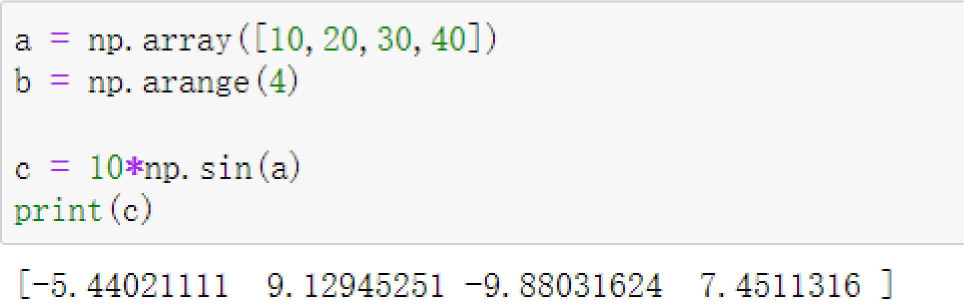


```
a = np.array([10,20,30,40])
b = np.arange(4)
print(a)
print(b)
c = a-b
print(c)
```

```
[10 20 30 40]
[0 1 2 3]
[10 19 28 37]
```

三角函数

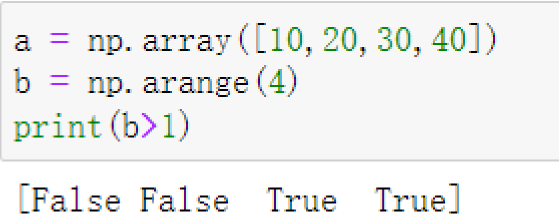
sin(a)   cos(a)   tan(a)



```
a = np.array([10,20,30,40])
b = np.arange(4)

c = 10*np.sin(a)
print(c)
```

```
[-5.44021111  9.12945251 -9.88031624  7.4511316 ]
```

判断运算

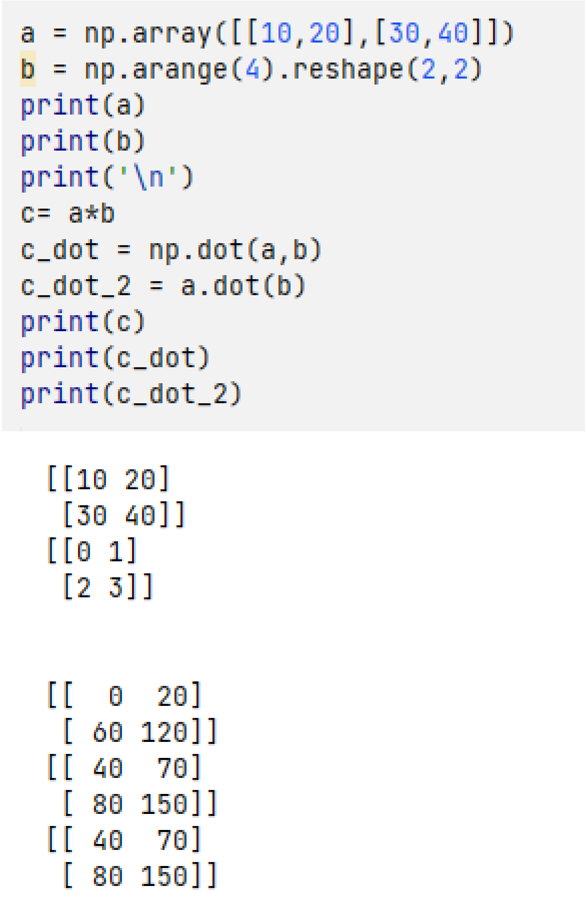


```
a = np.array([10,20,30,40])
b = np.arange(4)
print(b>1)
```

```
[False False  True  True]
```

矩阵运算

```
#矩阵中，对应位置的元素，逐个相乘
c= a*b
#矩阵乘法
c_dot = np.dot(a,b)
#矩阵乘法
c_dot_2 = a.dot(b)
```

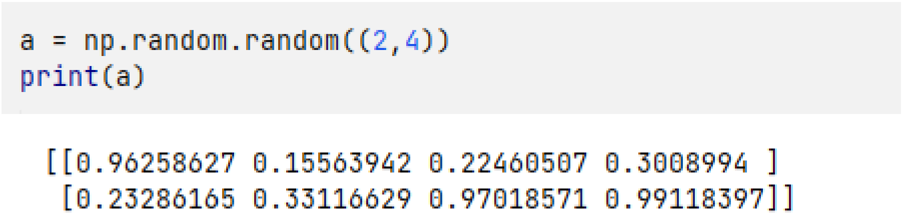


```
a = np.array([[10,20],[30,40]])
b = np.arange(4).reshape(2,2)
print(a)
print(b)
print('\n')
c= a*b
c_dot = np.dot(a,b)
c_dot_2 = a.dot(b)
print(c)
print(c_dot)
print(c_dot_2)
```

```
[[10 20]
 [30 40]]
[[0 1]
 [2 3]]


[[  0  20]
 [ 60 120]]
[[ 40  70]
 [ 80 150]]
[[ 40  70]
 [ 80 150]]
```

随机生成

#随机生成2行4列的数字（0-1）之间

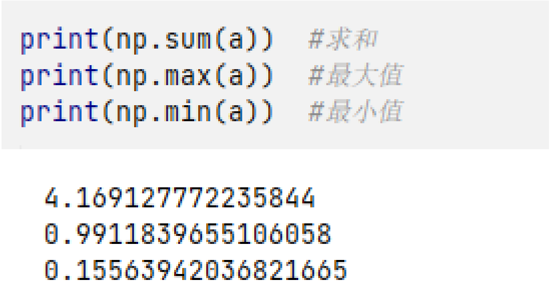


```
a = np.random.random((2,4))
print(a)
```

```
[[0.96258627 0.15563942 0.22460507 0.3008994 ]
 [0.23286165 0.33116629 0.97018571 0.99118397]]
```

计算

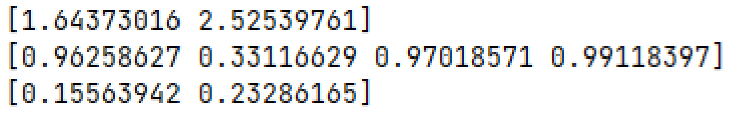
```
print(np.sum(a))   #求和
print(np.max(a))   #最大值
print(np.min(a))   #最小值
```



```
print(np.sum(a))  #求和
print(np.max(a))  #最大值
print(np.min(a))  #最小值
```

```
4.169127772235844
0.9911839655106058
0.15563942036821665
```

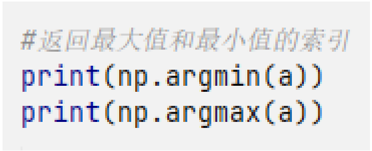
```
#axis = 1 水平方向
#axis = 0 垂直方向

print(np.sum(a,axis=1))  #求和：每行求和
print(np.max(a,axis=0))  #最大值：每列最大值
print(np.min(a,axis=1))  #最小值：每列最小值
```



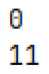
```
[1.64373016 2.52539761]
[0.96258627 0.33116629 0.97018571 0.99118397]
[0.15563942 0.23286165]
```

索引1

```
#返回最大值和最小值的索引
a = np.arange(2,14).reshape((3,4))
print(np.argmin(a))
print(np.argmax(a))
```



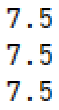
```
#返回最大值和最小值的索引
print(np.argmin(a))
print(np.argmax(a))
```



```
0
11
```

平均值和中位数

```
#计算矩阵平均值
a = np.arange(2,14).reshape((3,4))
print(np.mean(a))
print(a.mean())
print(np.average(a))
#print(a.average())这个不能用
```



```
7.5
7.5
7.5
```

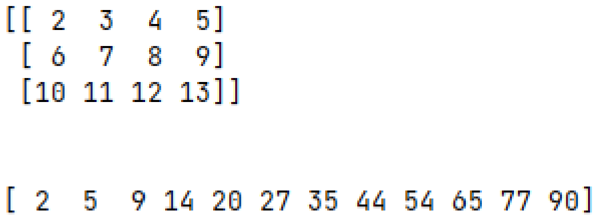
```
#按行和列求平均值
a = np.arange(2,14).reshape((3,4))
print(a)
print(np.mean(a,axis=0)) #求列平均值
print(np.mean(a,axis=1)) #求行平均值
```

```
#计算中位数
a = np.arange(2,14).reshape((3,4))
print(np.median(a))
```

```
7.5
```

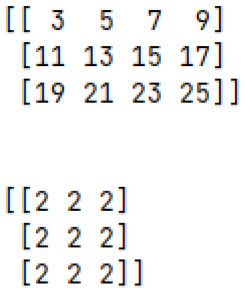
累加

```python
#累加
print(a)
print('\n')
print(np.cumsum(a))
```



```
[[ 2  3  4  5]
 [ 6  7  8  9]
 [10 11 12 13]]


[ 2  5  9 14 20 27 35 44 54 65 77 90]
```

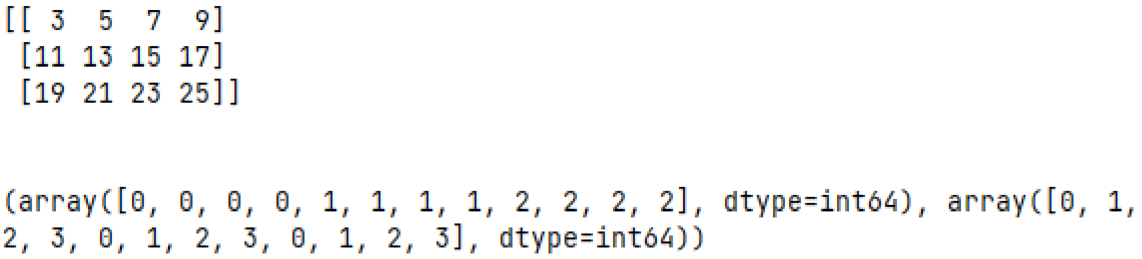
```python
#相邻两数差
a = np.arange(3,27,2).reshape((3,4))
print(a)
print('\n')
print(np.diff(a))
```



```
[[ 3  5  7  9]
 [11 13 15 17]
 [19 21 23 25]]


[[2 2 2]
 [2 2 2]
 [2 2 2]]
```

输出非0

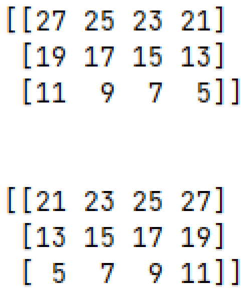
```python
#输出非0的坐标
#第一个数组是行，第二个数组是列
a = np.arange(3,27,2).reshape((3,4))
print(a)
print('\n')
print(np.nonzero(a))
```



```
[[ 3  5  7  9]
 [11 13 15 17]
 [19 21 23 25]]


(array([0, 0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2], dtype=int64), array([0, 1,
2, 3, 0, 1, 2, 3, 0, 1, 2, 3], dtype=int64))
```

排序

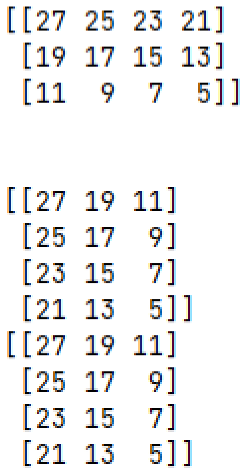
```
#逐行排序
a = np.arange(27,3,-2).reshape((3,4))
print(a)
print('\n')
print(np.sort(a))
```



```
[[27 25 23 21]
 [19 17 15 13]
 [11  9  7  5]]


[[21 23 25 27]
 [13 15 17 19]
 [ 5  7  9 11]]
```

转置

```
#转置，行列转换
a = np.arange(27,3,-2).reshape((3,4))
print(a)
print('\n')
print(np.transpose(a))
print(a.T)
```



```
[[27 25 23 21]
 [19 17 15 13]
 [11  9  7  5]]


[[27 19 11]
 [25 17  9]
 [23 15  7]
 [21 13  5]]
[[27 19 11]
 [25 17  9]
 [23 15  7]
 [21 13  5]]
```

截取

```
#小于12全变为12，大于18全变为18
a = np.arange(25,1,-2).reshape((3,4))
print(a)
print('\n')
print(np.clip(a,12,18))
```

```
[[25 23 21 19]
 [17 15 13 11]
 [ 9  7  5  3]]


[[18 18 18 18]
 [17 15 13 12]
 [12 12 12 12]]
```

索引2

```
#取值
a = np.arange(3,15)
print(a)
print(a[3])
```

```
[ 3  4  5  6  7  8  9 10 11 12 13 14]
6
```

```
#二维取值和列表相同
a = np.arange(3,15).reshape(3,4)
print(a)
print(a[1][2])
print(a[1,2])
```

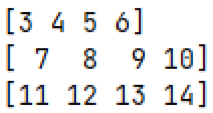
```
[[ 3  4  5  6]
 [ 7  8  9 10]
 [11 12 13 14]]
9
9
```

```
print(a[1,:])    #1行 所有值
print(a[1,1:3])    #1行，第1-2
print(a[:,1:3])  #所有行，第1-2
print(a[:,::2])  #所有行，步长2
```

```
[ 7  8  9 10]
[8 9]
[[ 4  5]
 [ 8  9]
 [12 13]]
[[ 3  5]
 [ 7  9]
 [11 13]]
```

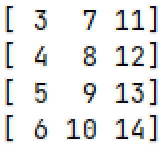
遍历

```
#遍历行
a = np.arange(3,15).reshape(3,4)
for row in a:
    print(row)
```



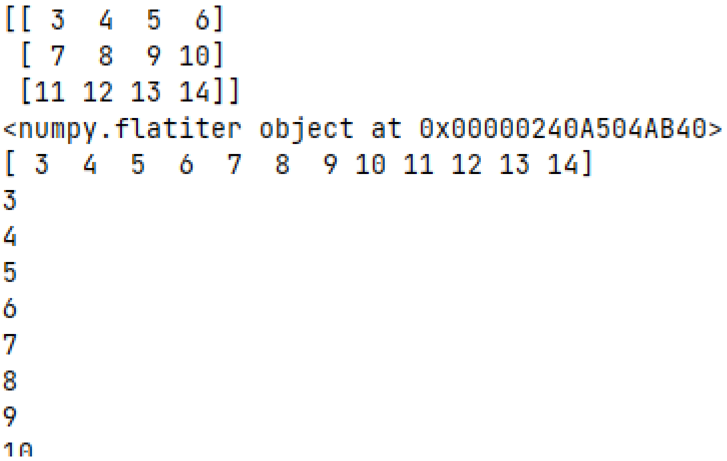
```
[3 4 5 6]
[ 7  8  9 10]
[11 12 13 14]
```

```python
#遍历列
a = np.arange(3,15).reshape(3,4)
for column in a.T:
    print(column)
```



```
[ 3  7 11]
[ 4  8 12]
[ 5  9 13]
[ 6 10 14]
```

```python
#遍历各项
a = np.arange(3,15).reshape(3,4)
print(a)
print(a.flat)
print(a.flatten())
for item in a.flat:
    print(item)
```

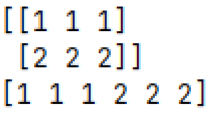


```
[[ 3  4  5  6]
 [ 7  8  9 10]
 [11 12 13 14]]
<numpy.flatiter object at 0x00000240A504AB40>
[ 3  4  5  6  7  8  9 10 11 12 13 14]
3
4
5
6
7
8
9
10
```

4

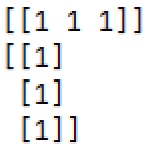
合并

```python
a = np.array([1,1,1])
b = np.array([2,2,2])
print(np.vstack((a,b)))  #上下合并
print(np.hstack((a,b)))  #左右合并
```



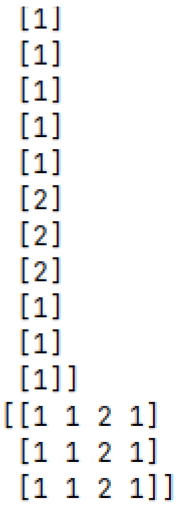
```
[[1 1 1]
 [2 2 2]]
[1 1 1 2 2 2]
```

```python
#横向、纵向转换
a = np.array([1,1,1])
print(np.mat(a))
a = np.mat(a).T
print(a)
```



```
[[1 1 1]]
[[1]
 [1]
 [1]]
```

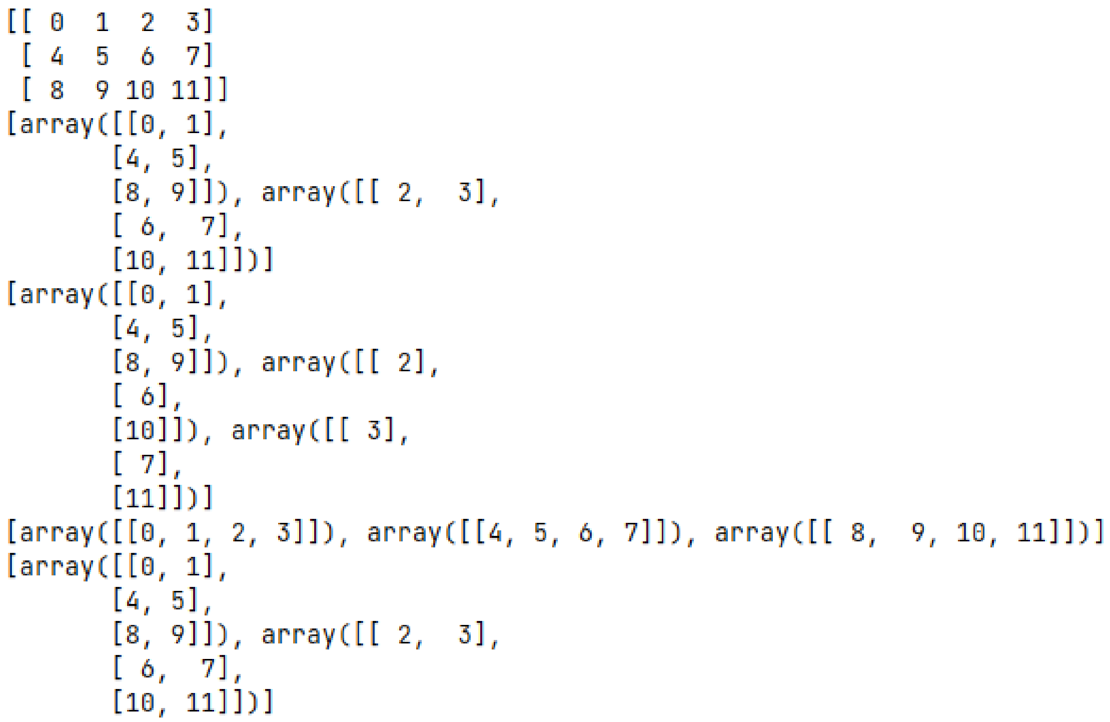
```python
a = np.array([1,1,1])[:,np.newaxis]
b = np.array([2,2,2])[:,np.newaxis]
#既可以横向也可以纵向
c = np.concatenate((a,a,b,a),axis=0)
print(c)
d = np.concatenate((a,a,b,a),axis=1)
print(d)
```



```
 [1]
 [1]
 [1]
 [1]
 [1]
 [2]
 [2]
 [2]
 [1]
 [1]
 [1]]
[[1 1 2 1]
 [1 1 2 1]
 [1 1 2 1]]
```

分割

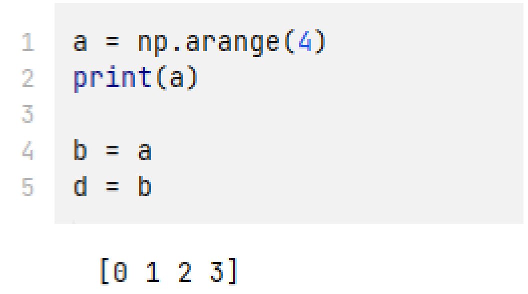
```python
a = np.arange(12).reshape(3,4)
print(a)
#  第二个参数，表示分割为几块,只能实现等量分割
print(np.split(a,2,axis=1))
#  第二个参数，表示分割为几块,可以实现不等量分割
print(np.array_split(a,3,axis=1))
#根据垂直和水平方向进行分割,只能实现等量分割
print(np.vsplit(a,3))
print(np.hsplit(a,2))
```



```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
[array([[0, 1],
       [4, 5],
       [8, 9]]), array([[ 2,  3],
       [ 6,  7],
       [10, 11]])]
[array([[0, 1],
       [4, 5],
       [8, 9]]), array([[ 2],
       [ 6],
       [10]]), array([[ 3],
       [ 7],
       [11]])]
[array([[0, 1, 2, 3]]), array([[4, 5, 6, 7]]), array([[ 8,  9, 10, 11]])]
[array([[0, 1],
       [4, 5],
       [8, 9]]), array([[ 2,  3],
       [ 6,  7],
       [10, 11]])]
```

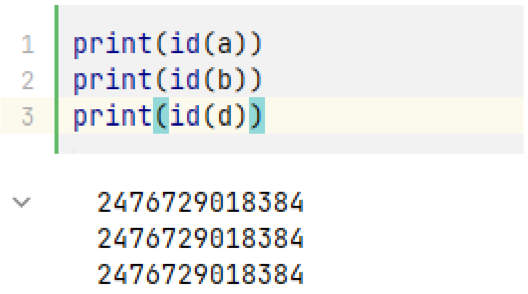
5

赋值



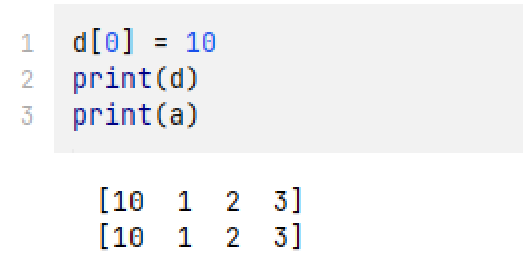
```python
a = np.arange(4)
print(a)

b = a
d = b
```

```
[0 1 2 3]
```



```python
print(id(a))
print(id(b))
print(id(d))
```

```
2476729018384
2476729018384
2476729018384
```



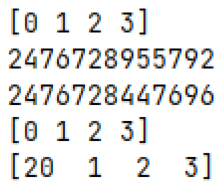
```python
d[0] = 10
print(d)
print(a)
```

```
[10  1  2  3]
[10  1  2  3]
```

所有变量都是指向同一地址。

copy

```python
a = np.arange(4)
print(a)


b = a.copy()
print(id(a))
print(id(b))


b[0]=20
print(a)
print(b)
```



```
[0 1 2 3]
2476728955792
2476728447696
[0 1 2 3]
[20  1  2  3]
```

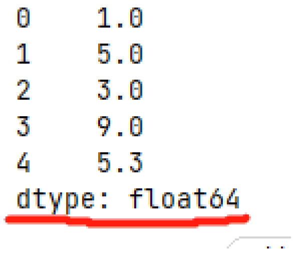
使用copy，重新开辟地址并赋值

pandas

1

Series

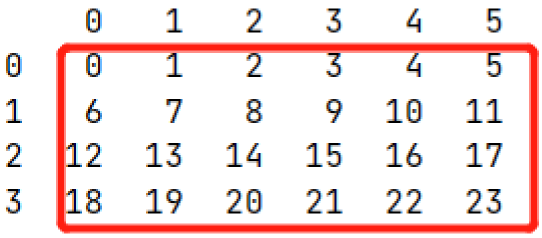
```python
s = pd.Series([1,5,3,9,5.3])
print(s)
```



```
0    1.0
1    5.0
2    3.0
3    9.0
4    5.3
dtype: float64
```

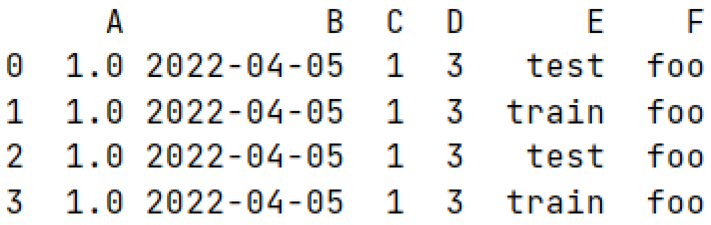
Dataframe

```python
df =pd.DataFrame(np.arange(24).reshape((4,6)))
print(df)
```



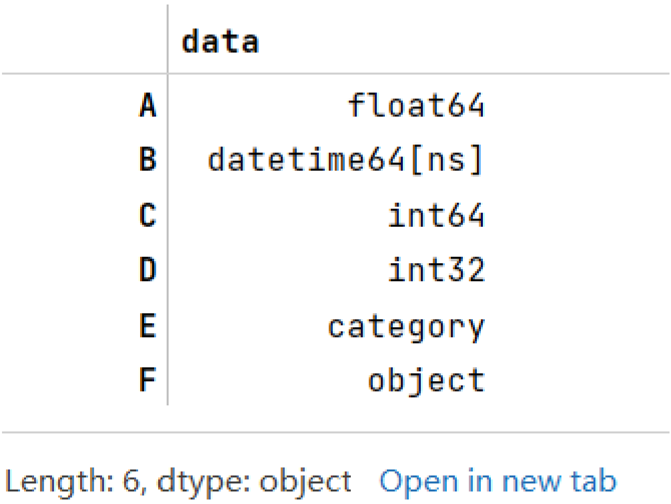
```
    0   1   2   3   4   5
0   0   1   2   3   4   5
1   6   7   8   9  10  11
2  12  13  14  15  16  17
3  18  19  20  21  22  23
```

```
df =pd.DataFrame({'A':1.,
                  'B':pd.Timestamp('20220405'),
                  'C':pd.Series(1,index=list(range(4))),
                  'D':np.array([3]*4,dtype='int32'),
                  'E':pd.Categorical(['test','train','test','train']),
                  'F':'foo'
                  })
print(df)
```



```
     A          B  C  D      E    F
0  1.0 2022-04-05  1  3   test  foo
1  1.0 2022-04-05  1  3  train  foo
2  1.0 2022-04-05  1  3   test  foo
3  1.0 2022-04-05  1  3  train  foo
```

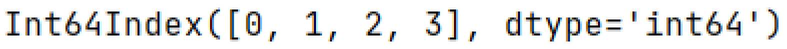
```
df.dtypes
```



	data
A	float64
B	datetime64[ns]
C	int64
D	int32
E	category
F	object

Length: 6, dtype: object Open in new tab

```
#查看行
df.index
```



```
Int64Index([0, 1, 2, 3], dtype='int64')
```

```
#查看列
df.columns
```

```
Index(['A', 'B', 'C', 'D', 'E', 'F'], dtype='object')
```

```
#查看所有值
df.values
```

	0	1	2	3	4	5
0	1.0	2022-04-05	1	3	test	foo
1	1.0	2022-04-05	1	3	train	foo
2	1.0	2022-04-05	1	3	test	foo
3	1.0	2022-04-05	1	3	train	foo

4 rows × 6 columns Open in new tab

```
#计算数字类型的列的 数量、平均值、方差等等
df.describe()
```

	A	C	D
count	4.0	4.0	4.0
mean	1.0	1.0	3.0
std	0.0	0.0	0.0
min	1.0	1.0	3.0
25%	1.0	1.0	3.0
50%	1.0	1.0	3.0
75%	1.0	1.0	3.0
max	1.0	1.0	3.0

```
#行列转置
df.T
```

	0	1	2	3
A	1.0	1.0	1.0	1.0
B	2022-04-05 00:00:00	2022-04-05 00:00:00	2022-04-05 00:00:00	2022-04-05 00:00:00
C	1	1	1	1
D	3	3	3	3
E	test	train	test	train
F	foo	foo	foo	foo

排序

```python
#排序  axis=0 根据列排序，ascending=False 倒序

df1 = df.sort_index(axis=1,ascending=False)
print(df1)
df2 = df.sort_index(axis=1,ascending=True)
print(df2)
```

```
     F      E  D  C          B    A
0  foo   test  3  1 2022-04-05  1.0
1  foo  train  3  1 2022-04-05  1.0
2  foo   test  3  1 2022-04-05  1.0
3  foo  train  3  1 2022-04-05  1.0
     A          B  C  D      E    F
0  1.0 2022-04-05  1  3   test  foo
1  1.0 2022-04-05  1  3  train  foo
2  1.0 2022-04-05  1  3   test  foo
3  1.0 2022-04-05  1  3  train  foo
```

```python
df1 = df.sort_index(axis=0,ascending=False)
print(df1)
df2 = df.sort_index(axis=0,ascending=True)
print(df2)
```

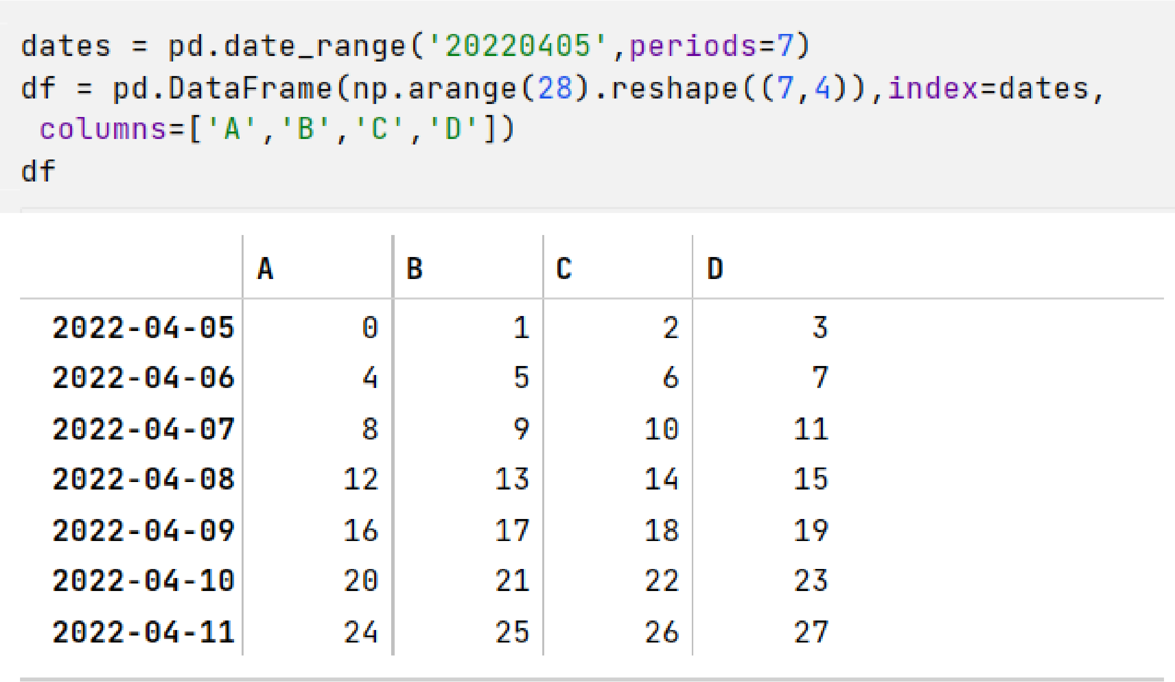
```
     A          B  C  D      E    F
3  1.0 2022-04-05  1  3  train  foo
2  1.0 2022-04-05  1  3   test  foo
1  1.0 2022-04-05  1  3  train  foo
0  1.0 2022-04-05  1  3   test  foo
     A          B  C  D      E    F
0  1.0 2022-04-05  1  3   test  foo
1  1.0 2022-04-05  1  3  train  foo
2  1.0 2022-04-05  1  3   test  foo
3  1.0 2022-04-05  1  3  train  foo
```

```python
#根据 列 的值进行排序
df.sort_values(by='E')
```

	A	B	C	D	E	F
0	1.0	2022-04-05	1	3	test	foo
2	1.0	2022-04-05	1	3	test	foo
1	1.0	2022-04-05	1	3	train	foo
3	1.0	2022-04-05	1	3	train	foo

2

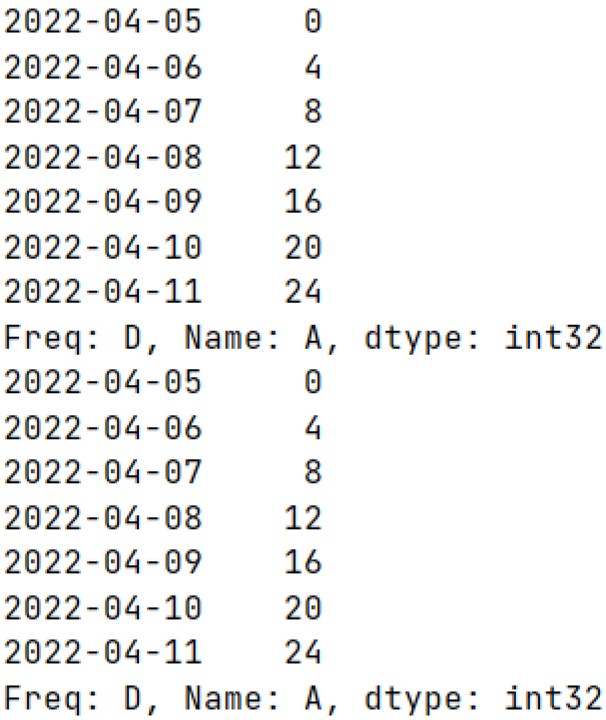
选择数据



```
dates = pd.date_range('20220405',periods=7)
df = pd.DataFrame(np.arange(28).reshape((7,4)),index=dates,
 columns=['A','B','C','D'])
df
```

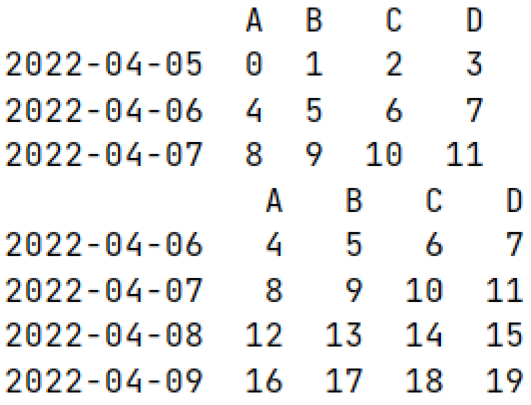
	A	B	C	D
2022-04-05	0	1	2	3
2022-04-06	4	5	6	7
2022-04-07	8	9	10	11
2022-04-08	12	13	14	15
2022-04-09	16	17	18	19
2022-04-10	20	21	22	23
2022-04-11	24	25	26	27

```
#两种方法，调用某列
print(df['A'])
print(df.A)
```



```
2022-04-05     0
2022-04-06     4
2022-04-07     8
2022-04-08    12
2022-04-09    16
2022-04-10    20
2022-04-11    24
Freq: D, Name: A, dtype: int32
2022-04-05     0
2022-04-06     4
2022-04-07     8
2022-04-08    12
2022-04-09    16
2022-04-10    20
2022-04-11    24
Freq: D, Name: A, dtype: int32
```

```
#两种方法，选择行数据
print(df[0:3])
print(df['2022-04-06':'2022-04-09']) #前后都选择到
```

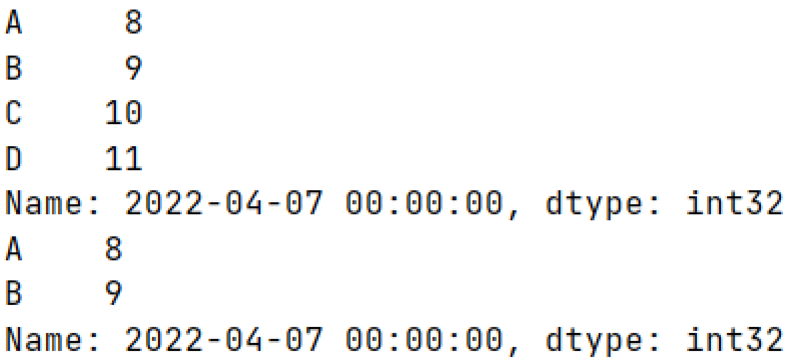


```
            A  B   C   D
2022-04-05  0  1   2   3
2022-04-06  4  5   6   7
2022-04-07  8  9  10  11
             A   B   C   D
2022-04-06   4   5   6   7
2022-04-07   8   9  10  11
2022-04-08  12  13  14  15
2022-04-09  16  17  18  19
```

df.loc

#根据lable进行筛选

```
print(df.loc['20220407'])  #横向
print(df.loc['20220407',['A','B']])
```

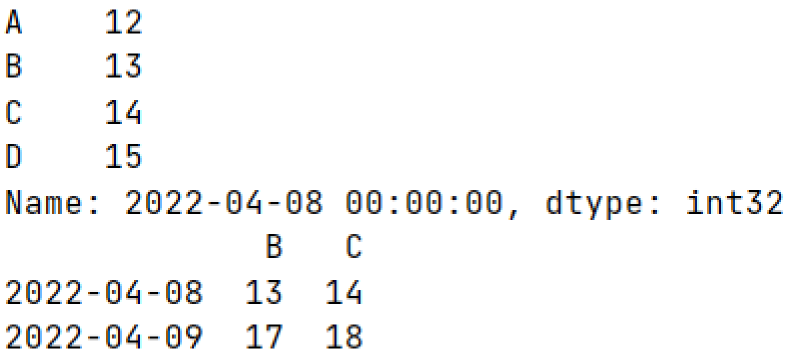


```
A     8
B     9
C    10
D    11
Name: 2022-04-07 00:00:00, dtype: int32
A    8
B    9
Name: 2022-04-07 00:00:00, dtype: int32
```

df.iloc

#根据position进行筛选

```
print(df.iloc[3])  #第3行
print(df.iloc[3:5,1:3])
```



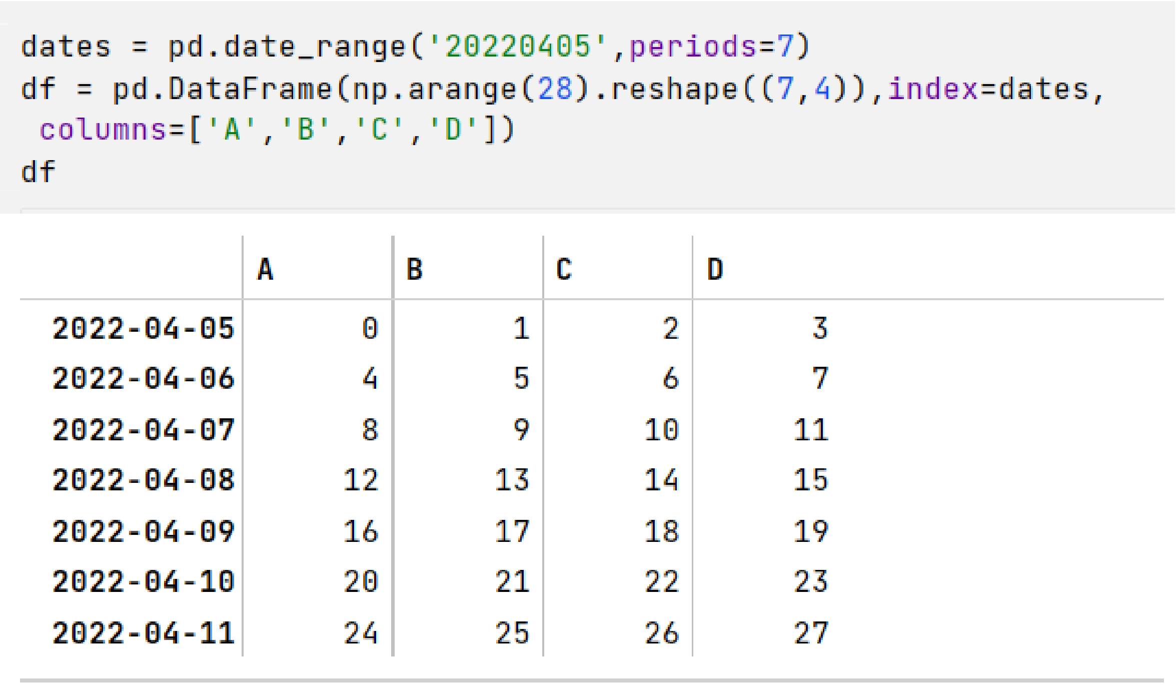
```
A    12
B    13
C    14
D    15
Name: 2022-04-08 00:00:00, dtype: int32
             B   C
2022-04-08  13  14
2022-04-09  17  18
```

ix

#新pandas已经弃用ix

3

#示例df

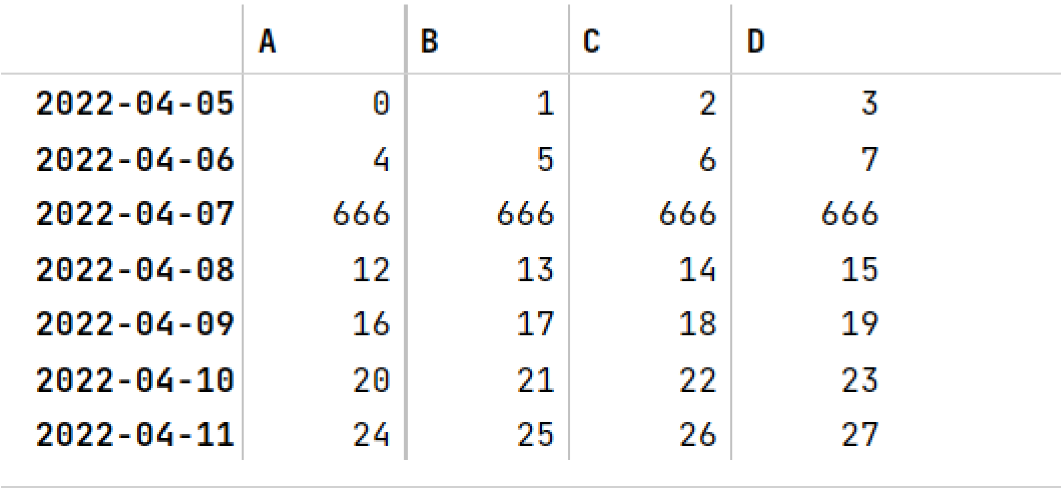


```
dates = pd.date_range('20220405',periods=7)
df = pd.DataFrame(np.arange(28).reshape((7,4)),index=dates,
 columns=['A','B','C','D'])
df
```

	A	B	C	D
2022-04-05	0	1	2	3
2022-04-06	4	5	6	7
2022-04-07	8	9	10	11
2022-04-08	12	13	14	15
2022-04-09	16	17	18	19
2022-04-10	20	21	22	23
2022-04-11	24	25	26	27

取值

```
#整行赋值
df.iloc[2:3] = 666
df
```



	A	B	C	D
2022-04-05	0	1	2	3
2022-04-06	4	5	6	7
2022-04-07	666	666	666	666
2022-04-08	12	13	14	15
2022-04-09	16	17	18	19
2022-04-10	20	21	22	23
2022-04-11	24	25	26	27

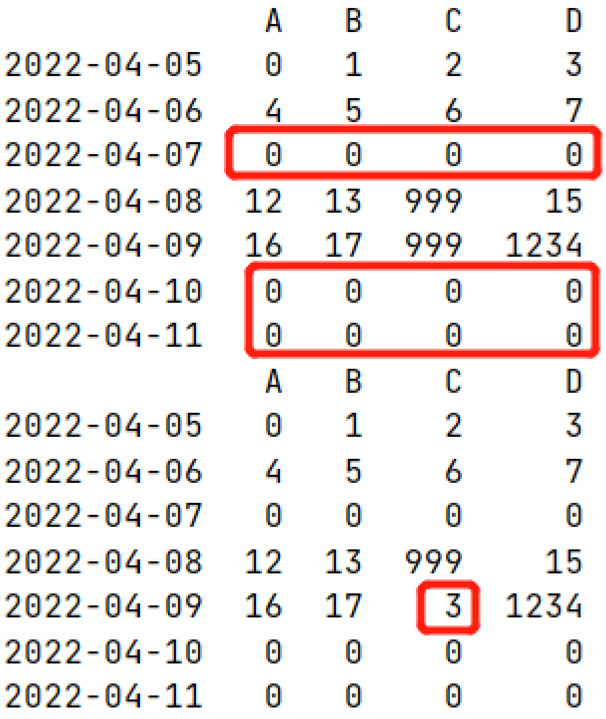
```
#范围赋值
df.iloc[3:5,2] = 999
df
```

	A	B	C	D
2022-04-05	0	1	2	3
2022-04-06	4	5	6	7
2022-04-07	666	666	666	666
2022-04-08	12	13	999	15
2022-04-09	16	17	999	19
2022-04-10	20	21	22	23
2022-04-11	24	25	26	27

```
#根据lable进行范围赋值
df.loc['20220409','D'] = 1234
df
```

	A	B	C	D
2022-04-05	0	1	2	3
2022-04-06	4	5	6	7
2022-04-07	666	666	666	666
2022-04-08	12	13	999	15
2022-04-09	16	17	999	1234
2022-04-10	20	21	22	23
2022-04-11	24	25	26	27

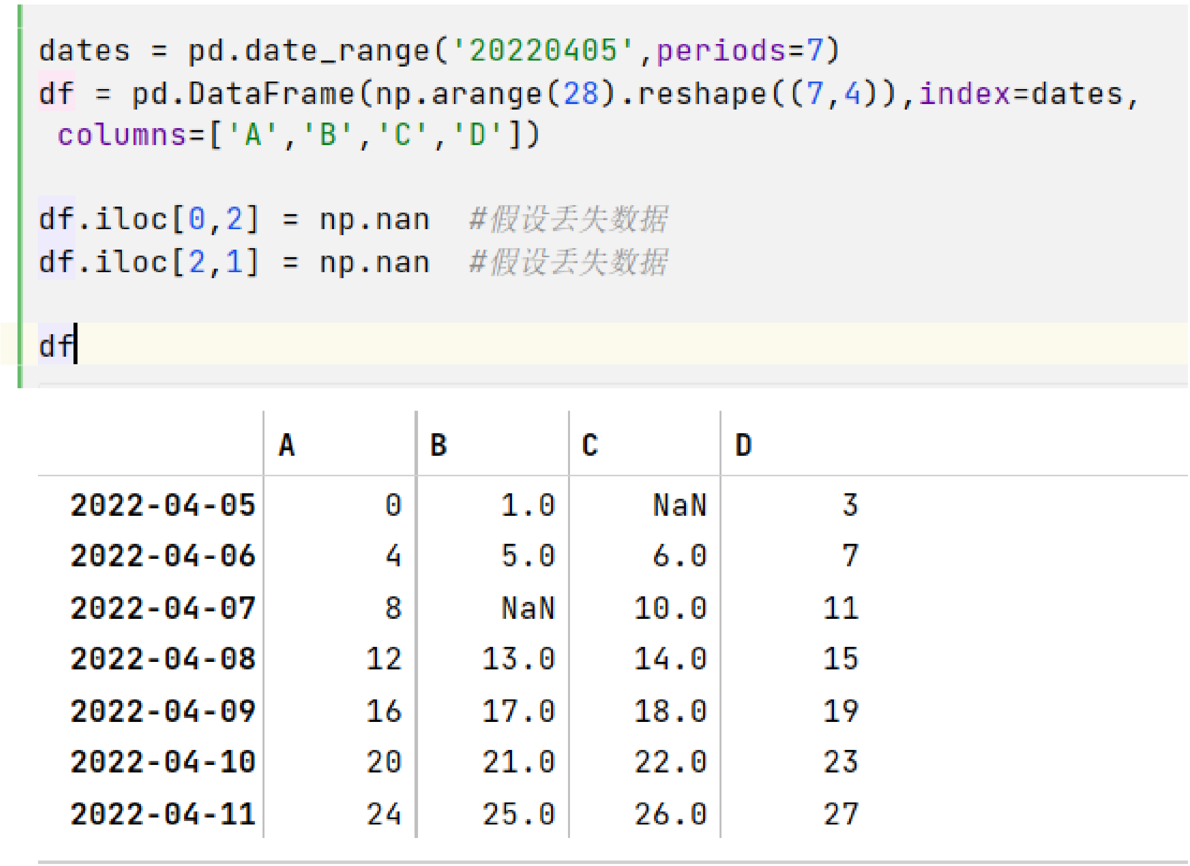
```
df[df.A>16] = 0 #根据条件，修改所有的值
print(df)
df.C[df.A>12] = 3 #根据条件，修改某列的值
print(df)
```



```
             A   B    C     D
2022-04-05   0   1    2     3
2022-04-06   4   5    6     7
2022-04-07   0   0    0     0
2022-04-08  12  13  999    15
2022-04-09  16  17  999  1234
2022-04-10   0   0    0     0
2022-04-11   0   0    0     0
             A   B    C     D
2022-04-05   0   1    2     3
2022-04-06   4   5    6     7
2022-04-07   0   0    0     0
2022-04-08  12  13  999    15
2022-04-09  16  17    3  1234
2022-04-10   0   0    0     0
2022-04-11   0   0    0     0
```

4

#示例df

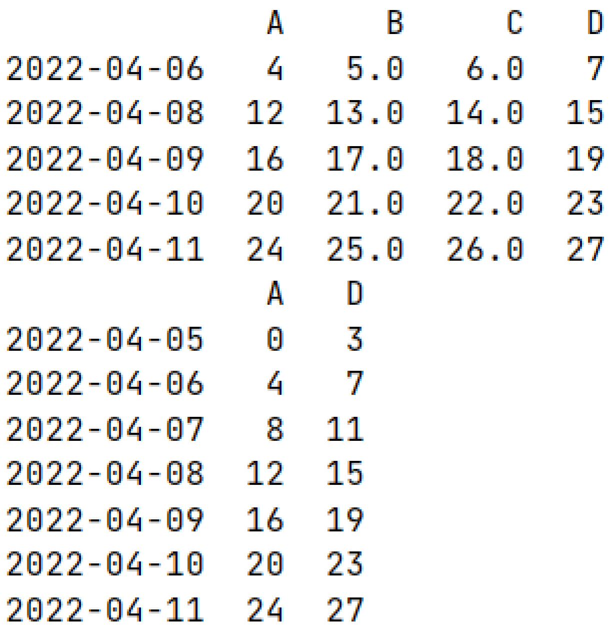


```python
dates = pd.date_range('20220405',periods=7)
df = pd.DataFrame(np.arange(28).reshape((7,4)),index=dates,
 columns=['A','B','C','D'])

df.iloc[0,2] = np.nan  #假设丢失数据
df.iloc[2,1] = np.nan  #假设丢失数据

df
```

	A	B	C	D
2022-04-05	0	1.0	NaN	3
2022-04-06	4	5.0	6.0	7
2022-04-07	8	NaN	10.0	11
2022-04-08	12	13.0	14.0	15
2022-04-09	16	17.0	18.0	19
2022-04-10	20	21.0	22.0	23
2022-04-11	24	25.0	26.0	27

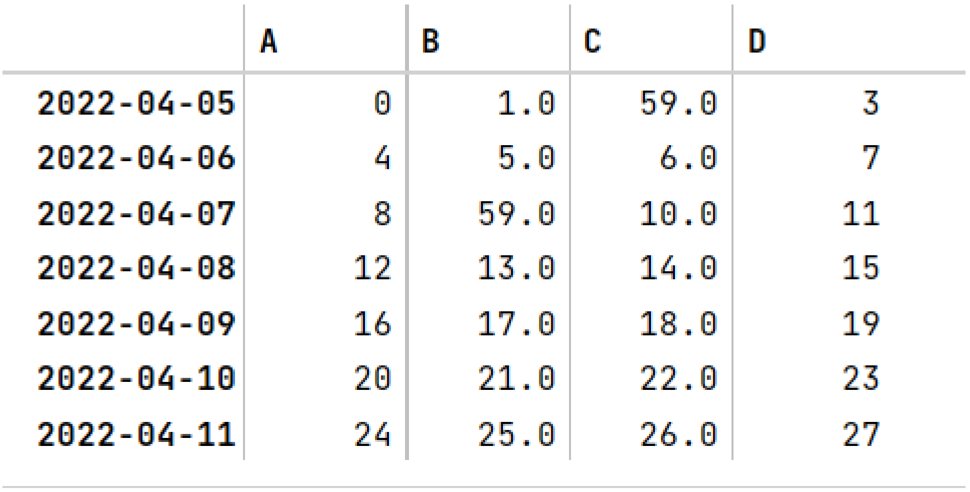
处理丢失数据

```python
# how={'any','all'}
# any行或列出现过任意NaN就丢掉
# all行或列所有值都是NaN才丢掉
# 出现NaN 丢掉行

df1 = df.dropna(axis=0,how='any')
print(df1)
df2 = df.dropna(axis=1,how='any')
print(df2)
```



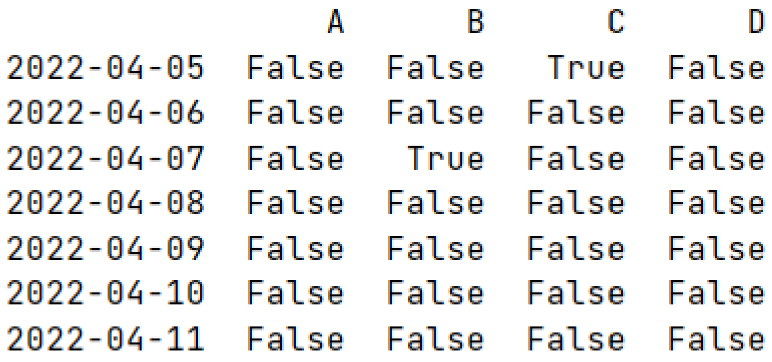
```
             A     B     C   D
2022-04-06   4   5.0   6.0   7
2022-04-08  12  13.0  14.0  15
2022-04-09  16  17.0  18.0  19
2022-04-10  20  21.0  22.0  23
2022-04-11  24  25.0  26.0  27
             A   D
2022-04-05   0   3
2022-04-06   4   7
2022-04-07   8  11
2022-04-08  12  15
2022-04-09  16  19
2022-04-10  20  23
2022-04-11  24  27
```

```python
#替换NaN为value
df3 = df.fillna(value=59)
df3
```



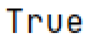
	A	B	C	D
2022-04-05	0	1.0	59.0	3
2022-04-06	4	5.0	6.0	7
2022-04-07	8	59.0	10.0	11
2022-04-08	12	13.0	14.0	15
2022-04-09	16	17.0	18.0	19
2022-04-10	20	21.0	22.0	23
2022-04-11	24	25.0	26.0	27

```python
#判断值是否为NaN
print(df.isnull())
```



```
                A      B      C      D
2022-04-05  False  False   True  False
2022-04-06  False  False  False  False
2022-04-07  False   True  False  False
2022-04-08  False  False  False  False
2022-04-09  False  False  False  False
2022-04-10  False  False  False  False
2022-04-11  False  False  False  False
```

```
#判断范围内是否有NaN
print(    np.any(df.isnull()) == True    )
```

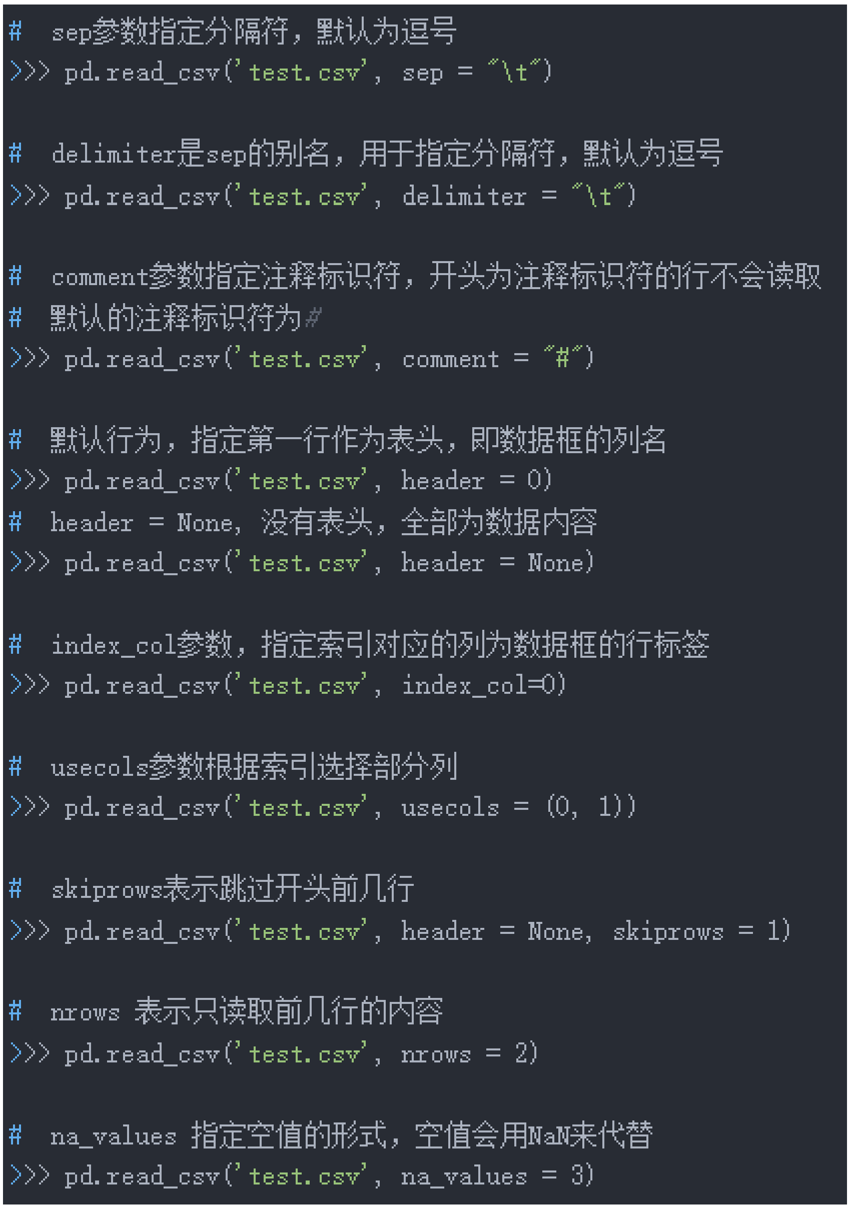


```
True
```

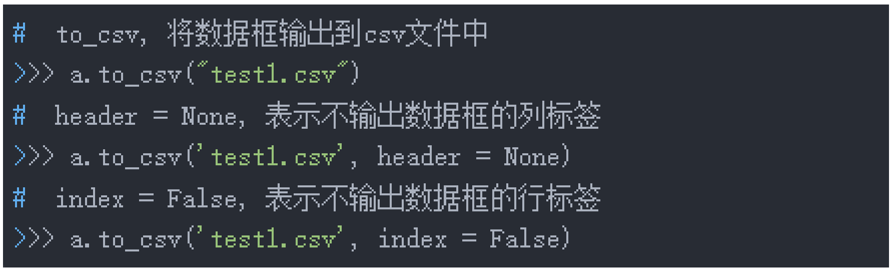
5

读写CSV

```
data = pd.read_csv('path.csv')

data.to_csv('path.csv')
```



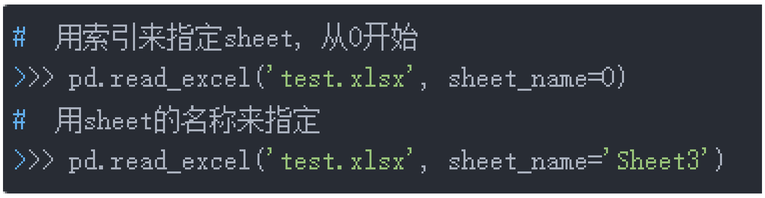
```
#  sep参数指定分隔符，默认为逗号
>>> pd.read_csv(’test.csv’, sep = ”\t”)

#  delimiter是sep的别名，用于指定分隔符，默认为逗号
>>> pd.read_csv(’test.csv’, delimiter = ”\t”)

#  comment参数指定注释标识符，开头为注释标识符的行不会读取
#  默认的注释标识符为#
>>> pd.read_csv(’test.csv’, comment = ”#”)

#  默认行为，指定第一行作为表头，即数据框的列名
>>> pd.read_csv(’test.csv’, header = 0)
#  header = None, 没有表头，全部为数据内容
>>> pd.read_csv(’test.csv’, header = None)

#  index_col参数，指定索引对应的列为数据框的行标签
>>> pd.read_csv(’test.csv’, index_col=0)

#  usecols参数根据索引选择部分列
>>> pd.read_csv(’test.csv’, usecols = (0, 1))

#  skiprows表示跳过开头前几行
>>> pd.read_csv(’test.csv’, header = None, skiprows = 1)

#  nrows 表示只读取前几行的内容
>>> pd.read_csv(’test.csv’, nrows = 2)

#  na_values 指定空值的形式，空值会用NaN来代替
>>> pd.read_csv(’test.csv’, na_values = 3)
```



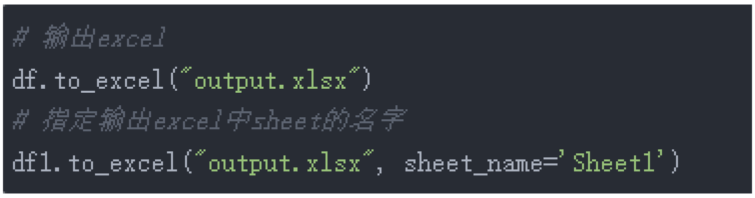
```
#  to_csv, 将数据框输出到csv文件中
>>> a.to_csv(”test1.csv”)
#  header = None, 表示不输出数据框的列标签
>>> a.to_csv(’test1.csv’, header = None)
#  index = False, 表示不输出数据框的行标签
>>> a.to_csv(’test1.csv’, index = False)
```

读写EXCEL

```
pd.read_excel('path.xlsx')

pd.to_excel('path.xlsx')
```



```
#  用索引来指定sheet, 从0开始
>>> pd.read_excel(’test.xlsx’, sheet_name=0)
#  用sheet的名称来指定
>>> pd.read_excel(’test.xlsx’, sheet_name=’Sheet3’)
```



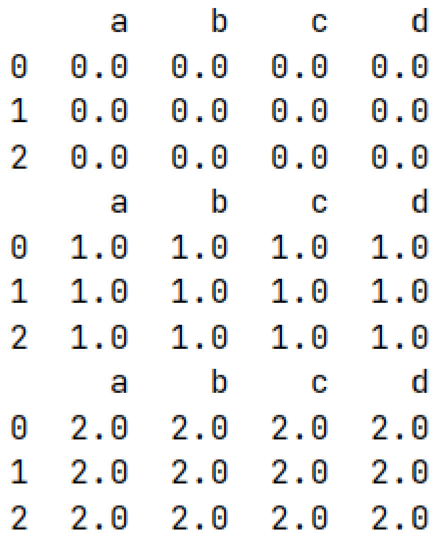
```
# 输出excel
df.to_excel(″output.xlsx″)
# 指定输出excel中sheet的名字
df1.to_excel(″output.xlsx″, sheet_name='Sheet1')
```

6

合并—concat

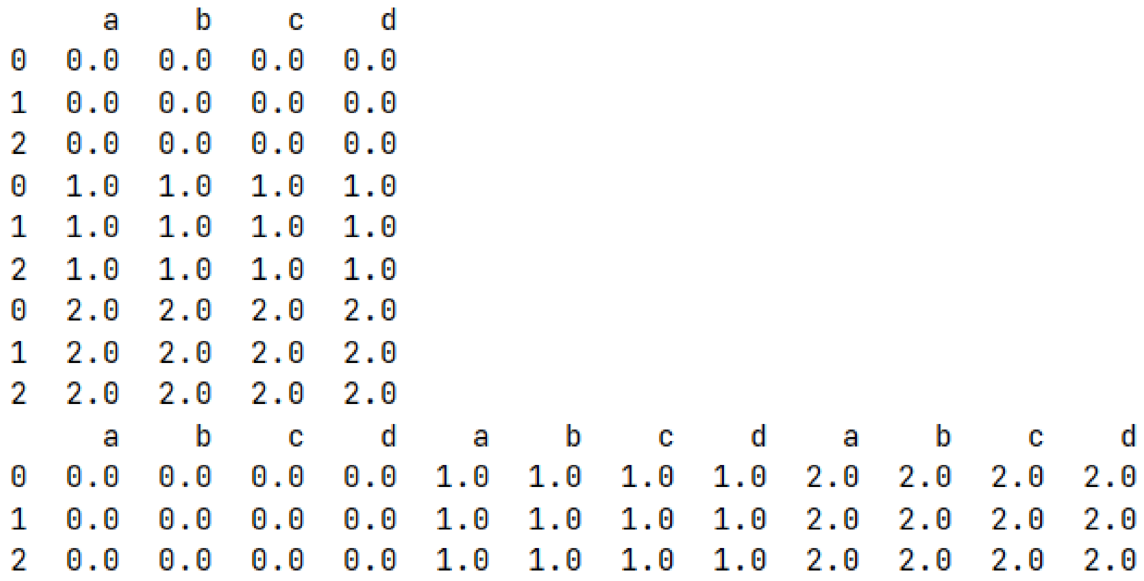
#示例DataFrame数据

```
df1 = pd.DataFrame(np.ones((3,4))*0,columns = ['a','b','c','d'])
df2 = pd.DataFrame(np.ones((3,4))*1,columns = ['a','b','c','d'])
df3 = pd.DataFrame(np.ones((3,4))*2,columns = ['a','b','c','d'])
print(df1)
print(df2)
print(df3)
```



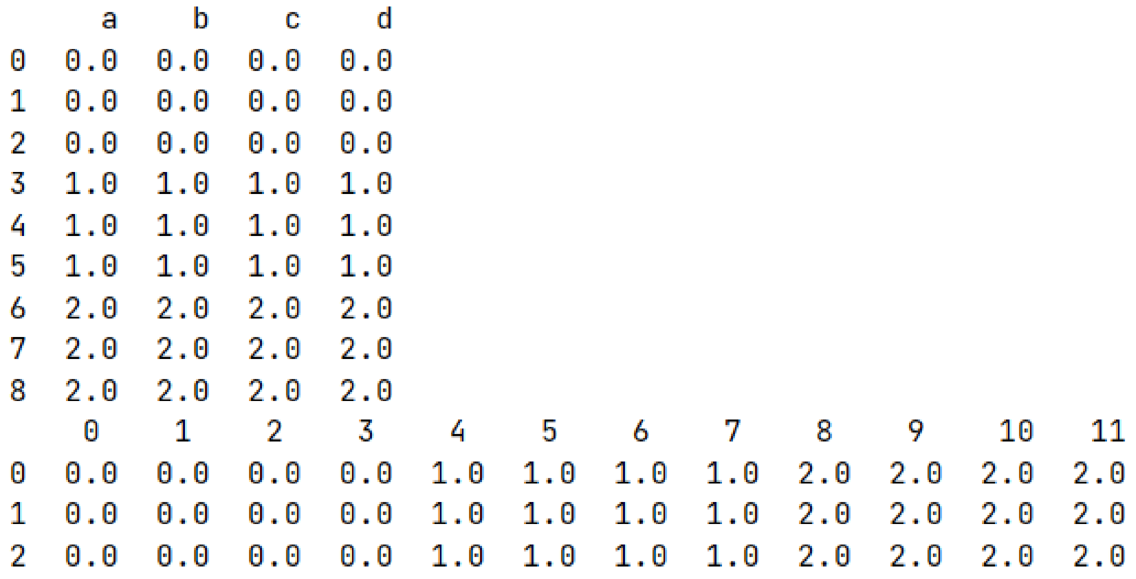
```
     a    b    c    d
0  0.0  0.0  0.0  0.0
1  0.0  0.0  0.0  0.0
2  0.0  0.0  0.0  0.0
     a    b    c    d
0  1.0  1.0  1.0  1.0
1  1.0  1.0  1.0  1.0
2  1.0  1.0  1.0  1.0
     a    b    c    d
0  2.0  2.0  2.0  2.0
1  2.0  2.0  2.0  2.0
2  2.0  2.0  2.0  2.0
```

```
#垂直合并
res = pd.concat([df1,df2,df3],axis=0)
print(res)

#水平合并
res = pd.concat([df1,df2,df3],axis=1)
print(res)
```



```
     a    b    c    d
0  0.0  0.0  0.0  0.0
1  0.0  0.0  0.0  0.0
2  0.0  0.0  0.0  0.0
0  1.0  1.0  1.0  1.0
1  1.0  1.0  1.0  1.0
2  1.0  1.0  1.0  1.0
0  2.0  2.0  2.0  2.0
1  2.0  2.0  2.0  2.0
2  2.0  2.0  2.0  2.0
     a    b    c    d    a    b    c    d    a    b    c    d
0  0.0  0.0  0.0  0.0  1.0  1.0  1.0  1.0  2.0  2.0  2.0  2.0
1  0.0  0.0  0.0  0.0  1.0  1.0  1.0  1.0  2.0  2.0  2.0  2.0
2  0.0  0.0  0.0  0.0  1.0  1.0  1.0  1.0  2.0  2.0  2.0  2.0
```

```
#垂直合并   忽略索引
res = pd.concat([df1,df2,df3],axis=0,ignore_index=True)
print(res)

#水平合并   忽略索引
res = pd.concat([df1,df2,df3],axis=1,ignore_index=True)
print(res)
```

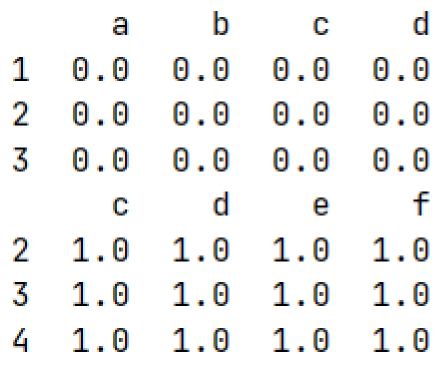


```
     a    b    c    d
0  0.0  0.0  0.0  0.0
1  0.0  0.0  0.0  0.0
2  0.0  0.0  0.0  0.0
3  1.0  1.0  1.0  1.0
4  1.0  1.0  1.0  1.0
5  1.0  1.0  1.0  1.0
6  2.0  2.0  2.0  2.0
7  2.0  2.0  2.0  2.0
8  2.0  2.0  2.0  2.0
    0    1    2    3    4    5    6    7    8    9    10   11
0  0.0  0.0  0.0  0.0  1.0  1.0  1.0  1.0  2.0  2.0  2.0  2.0
1  0.0  0.0  0.0  0.0  1.0  1.0  1.0  1.0  2.0  2.0  2.0  2.0
2  0.0  0.0  0.0  0.0  1.0  1.0  1.0  1.0  2.0  2.0  2.0  2.0
```

join=

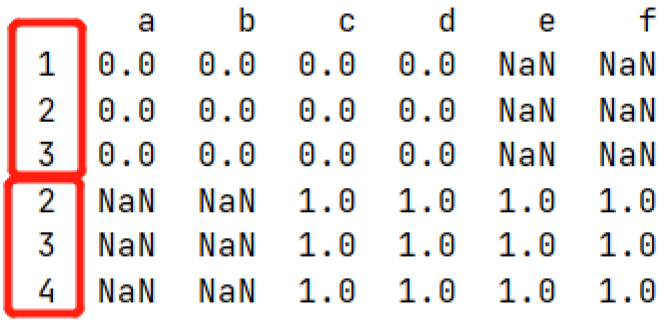
#示例DataFrame数据

```
df1 = pd.DataFrame(np.ones((3,4))*0,columns = ['a','b','c','d'],index=
[1,2,3])
df2 = pd.DataFrame(np.ones((3,4))*1,columns = ['c','d','e','f'],index=
[2,3,4])
print(df1)
print(df2)
```



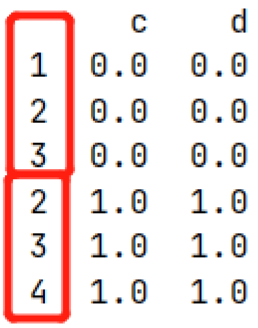
```
     a    b    c    d
1  0.0  0.0  0.0  0.0
2  0.0  0.0  0.0  0.0
3  0.0  0.0  0.0  0.0
     c    d    e    f
2  1.0  1.0  1.0  1.0
3  1.0  1.0  1.0  1.0
4  1.0  1.0  1.0  1.0
```

```
#直接合并  默认模式outer
res = pd.concat([df1,df2])
# 相当于  res = pd.concat([df1,df2],join='outer')
print(res)
```



```
     a    b    c    d    e    f
1  0.0  0.0  0.0  0.0  NaN  NaN
2  0.0  0.0  0.0  0.0  NaN  NaN
3  0.0  0.0  0.0  0.0  NaN  NaN
2  NaN  NaN  1.0  1.0  1.0  1.0
3  NaN  NaN  1.0  1.0  1.0  1.0
4  NaN  NaN  1.0  1.0  1.0  1.0
```

```
# join = 'inner' 模式
res = pd.concat([df1,df2],join='inner')
print(res)
```



```
     c    d
1  0.0  0.0
2  0.0  0.0
3  0.0  0.0
2  1.0  1.0
3  1.0  1.0
4  1.0  1.0
```

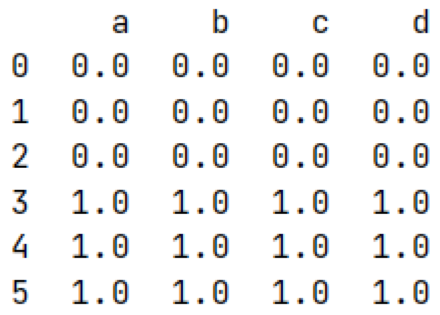
```
#如果需要修改索引，加入参数ignore_index=True
```

join_axes

#最新pandas已经不再支持

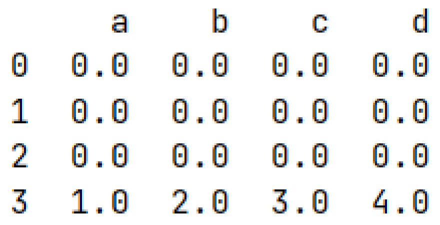
append

```
#默认垂直方向追加
df1 = pd.DataFrame(np.ones((3, 4)) * 0, columns=['a', 'b', 'c', 'd'])
df2 = pd.DataFrame(np.ones((3, 4)) * 1, columns=['a', 'b', 'c', 'd'])
res = df1.append(df2,ignore_index=True)
print(res)
```



```
     a    b    c    d
0  0.0  0.0  0.0  0.0
1  0.0  0.0  0.0  0.0
2  0.0  0.0  0.0  0.0
3  1.0  1.0  1.0  1.0
4  1.0  1.0  1.0  1.0
5  1.0  1.0  1.0  1.0
```

```
#默认垂直方向追加一个Series
s1 = pd.Series([1,2,3,4],index=['a', 'b', 'c', 'd'])
res = df1.append(s1,ignore_index=True)
print(res)
```



```
     a    b    c    d
0  0.0  0.0  0.0  0.0
1  0.0  0.0  0.0  0.0
2  0.0  0.0  0.0  0.0
3  1.0  2.0  3.0  4.0
```

merge-重点

#示例数据

```
left = pd.DataFrame({'key':['K0','K1','K2','K3'],
                     'A':['A0','A1','A2','A3'],
                     'B':['B0','B1','B2','B3'],})
right = pd.DataFrame({'key':['K0','K1','K2','K3'],
                     'C':['C0','C1','C2','C3'],
                     'D':['D0','D1','D2','D3'],})
print(left)
print(right)
```

```
  key   A   B
0  K0  A0  B0
1  K1  A1  B1
2  K2  A2  B2
3  K3  A3  B3
  key   C   D
0  K0  C0  D0
1  K1  C1  D1
2  K2  C2  D2
3  K3  C3  D3
```

```
#根据某列进行合并
res = pd.merge(left,right,on='key')
print(res)
```

```
  key   A   B   C   D
0  K0  A0  B0  C0  D0
1  K1  A1  B1  C1  D1
2  K2  A2  B2  C2  D2
3  K3  A3  B3  C3  D3
```

#示例数据

```
left = pd.DataFrame({'key1':['K0','K0','K1','K2'],
                    'key2':['K0','K1','K0','K1'],
                    'A':['A0','A1','A2','A3'],
                    'B':['B0','B1','B2','B3'],})
right = pd.DataFrame({'key1':['K0','K1','K1','K2'],
                     'key2':['K0','K0','K0','K0'],
                    'C':['C0','C1','C2','C3'],
                    'D':['D0','D1','D2','D3'],})
print(left)
print(right)
```

```
  key1 key2   A   B
0   K0   K0  A0  B0
1   K0   K1  A1  B1
2   K1   K0  A2  B2
3   K2   K1  A3  B3
  key1 key2   C   D
0   K0   K0  C0  D0
1   K1   K0  C1  D1
2   K1   K0  C2  D2
3   K2   K0  C3  D3
```

```python
#默认inner方式    类似等值连接
#how = ['left','right','outer','inner']
res = pd.merge(left,right,on=['key1','key2'])
print(res)
```

```
  key1 key2   A   B   C   D
0   K0   K0  A0  B0  C0  D0
1   K1   K0  A2  B2  C1  D1
2   K1   K0  A2  B2  C2  D2
```

```python
#outer方式
res = res = pd.merge(left,right,on=['key1','key2'],how='outer')
print(res)
```

```
  key1 key2    A    B    C    D
0   K0   K0   A0   B0   C0   D0
1   K0   K1   A1   B1  NaN  NaN
2   K1   K0   A2   B2   C1   D1
3   K1   K0   A2   B2   C2   D2
4   K2   K1   A3   B3  NaN  NaN
5   K2   K0  NaN  NaN   C3   D3
```

```python
#left方式
res = pd.merge(left,right,on=['key1','key2'],how='left')
print(res)
```

```
  key1 key2   A   B    C    D
0   K0   K0  A0  B0   C0   D0
1   K0   K1  A1  B1  NaN  NaN
2   K1   K0  A2  B2   C1   D1
3   K1   K0  A2  B2   C2   D2
4   K2   K1  A3  B3  NaN  NaN
```

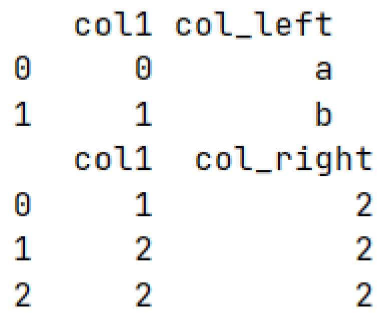
```python
#right方式
res = pd.merge(left,right,on=['key1','key2'],how='right')
print(res)
```

```
  key1 key2    A    B   C   D
0   K0   K0   A0   B0  C0  D0
1   K1   K0   A2   B2  C1  D1
2   K1   K0   A2   B2  C2  D2
3   K2   K0  NaN  NaN  C3  D3
```

indicator

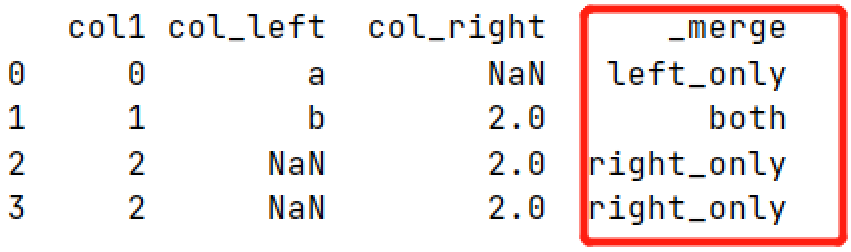
#示例数据

```
df1 = pd.DataFrame({'col1':[0,1],'col_left':['a','b']})
df2 = pd.DataFrame({'col1':[1,2,2],'col_right':[2,2,2]})
print(df1)
print(df2)
```



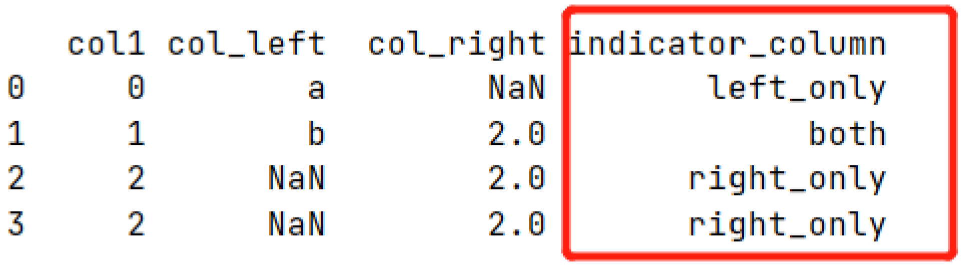
```
   col1 col_left
0     0        a
1     1        b
   col1  col_right
0     1          2
1     2          2
2     2          2
```

```
#显示merge的方式
res = pd.merge(df1,df2,on='col1',how='outer',indicator=True)
print(res)
```



```
   col1 col_left  col_right      _merge
0     0        a        NaN   left_only
1     1        b        2.0        both
2     2      NaN        2.0  right_only
3     2      NaN        2.0  right_only
```

```
#改名字
res = pd.merge(df1,df2,on='col1',how='outer',indicator='indicator_column')
print(res)
```

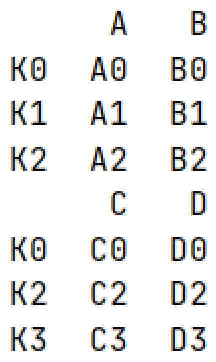


```
   col1 col_left  col_right indicator_column
0     0        a        NaN        left_only
1     1        b        2.0             both
2     2      NaN        2.0       right_only
3     2      NaN        2.0       right_only
```

index

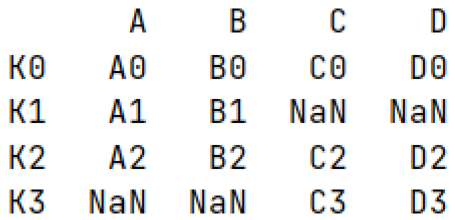
#示例数据

```
left = pd.DataFrame({'A':['A0','A1','A2'],
                     'B':['B0','B1','B2']},
                     index = ['K0','K1','K2'])
right = pd.DataFrame({'C':['C0','C2','C3'],
                     'D':['D0','D2','D3']},
                     index = ['K0','K2','K3'])
print(left)
print(right)
```



```
     A   B
K0  A0  B0
K1  A1  B1
K2  A2  B2
     C   D
K0  C0  D0
K2  C2  D2
K3  C3  D3
```

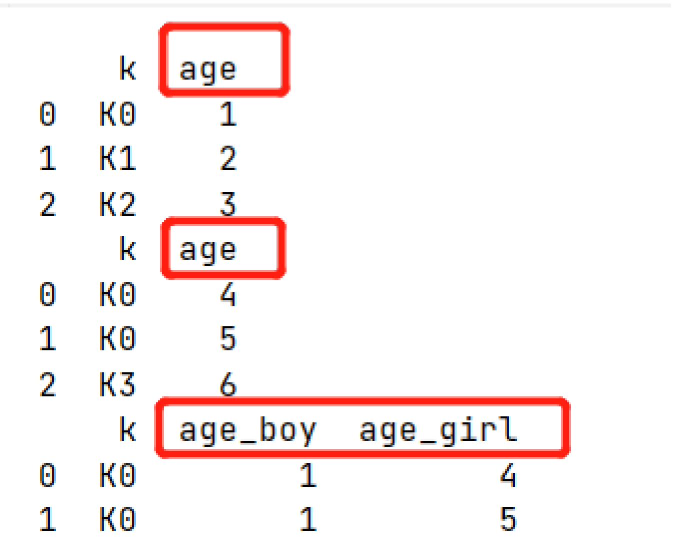
```
#left_index 和  right_index默认是False
res = pd.merge(left,right,left_index=True,right_index=True,how='outer')
print(res)
```



```
      A    B    C    D
K0   A0   B0   C0   D0
K1   A1   B1  NaN  NaN
K2   A2   B2   C2   D2
K3  NaN  NaN   C3   D3
```

overlapping

```
boys = pd.DataFrame({'k':['K0','K1','K2'],'age':[1,2,3]})
girls = pd.DataFrame({'k':['K0','K0','K3'],'age':[4,5,6]})
print(boys)
print(girls)
res = pd.merge(boys,girls,on='k',suffixes=['_boy','_girl'],how='inner')
print(res)
```



```
    k  age
0  K0    1
1  K1    2
2  K2    3
    k  age
0  K0    4
1  K0    5
2  K3    6
    k  age_boy  age_girl
0  K0        1         4
1  K0        1         5
```