数据分析工具Pandas

> Pandas的名称来自于面板数据（panel data）和Python数据分析（data analysis）。

Pandas是一个强大的分析结构化数据的工具集，基于NumPy构建，提供了 高级数据结构 和 数据操作工具，它是使Python成为强大而高效的数据分析环境的重要因素之一。

- 一个强大的分析和操作大型结构化数据集所需的工具集
- 基础是NumPy，提供了高性能矩阵的运算
- 提供了大量能够快速便捷地处理数据的函数和方法
- 应用于数据挖掘，数据分析
- 提供数据清洗功能

  http://pandas.pydata.org

1. 数据结构

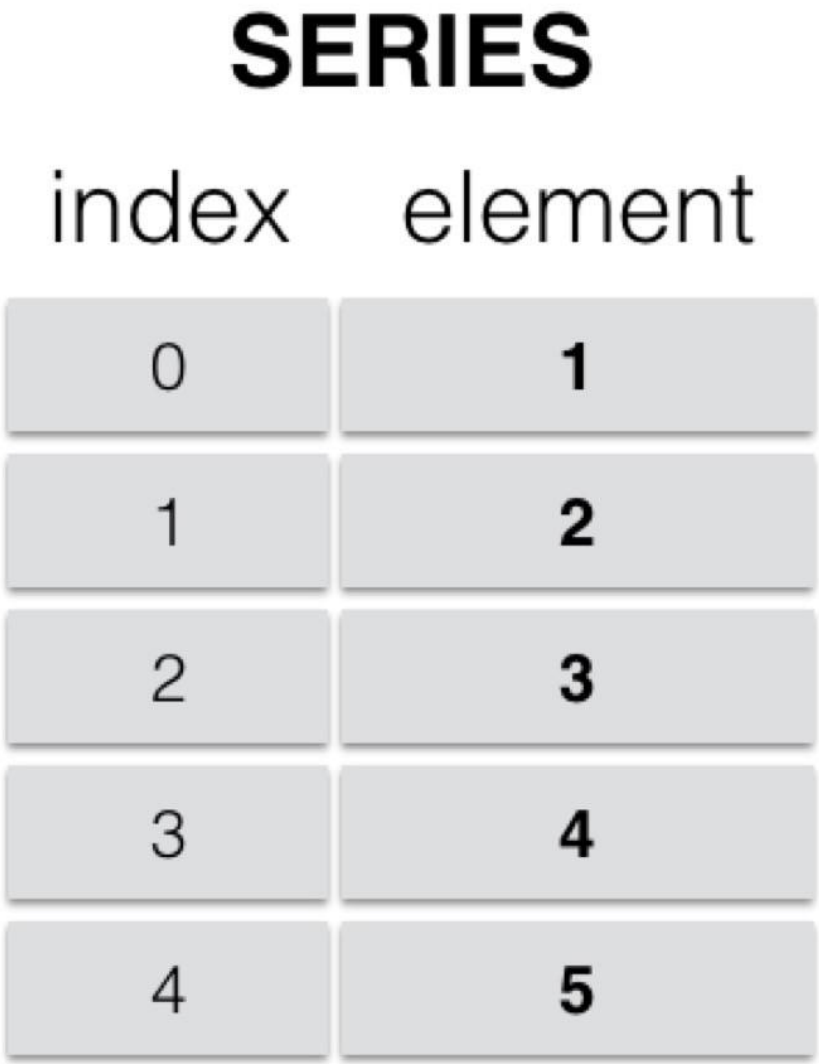
Pandas 的名称来自于面板数据（panel data）和Python数据分析(data analysis)，是一个强大的分析结构化数据的工具集，基于NumPy构建，提供了 高级数据结构 和 数据操作工具，它是使Python成为强大而高效的数据分析环境的重要因素之一。

1.1 Series

Series是一种类似于一维数组的 对象，由一组数据（各种NumPy数据类型）以及一组与之对应的索引（数据标签）组成。

- 类似一堆数组的对象
- 由数据和索引组成
  - 索引（index）在左，数据（values）在右

- 索引是自动创建的



1.1.1 Series 对象的创建

示例代码

```
import pandas as pd
#通过列表构建
lis = range(5,10) ser_obj1 =
pd.Series(lis)
#通过字典构建
dict_data = { "data" + str(i) : i for i in range(20,50,10)} ser_obj2 =
pd.Series(dict_data)
print(ser_obj1)
print('------------------')
print(ser_obj1.head(3))#查看前三个数据
print('------------------')
print(ser_obj2)
```

打印结果

```
0    5
1    6
2    7
3    8
4    9
dtype: int32
---------- 0 5
1    6
2    7
dtype: int32
----------
data20    20
data30    30
data40    40
dtype: int64
```

1.1.2 Series 通过索引获取数据

示例代码

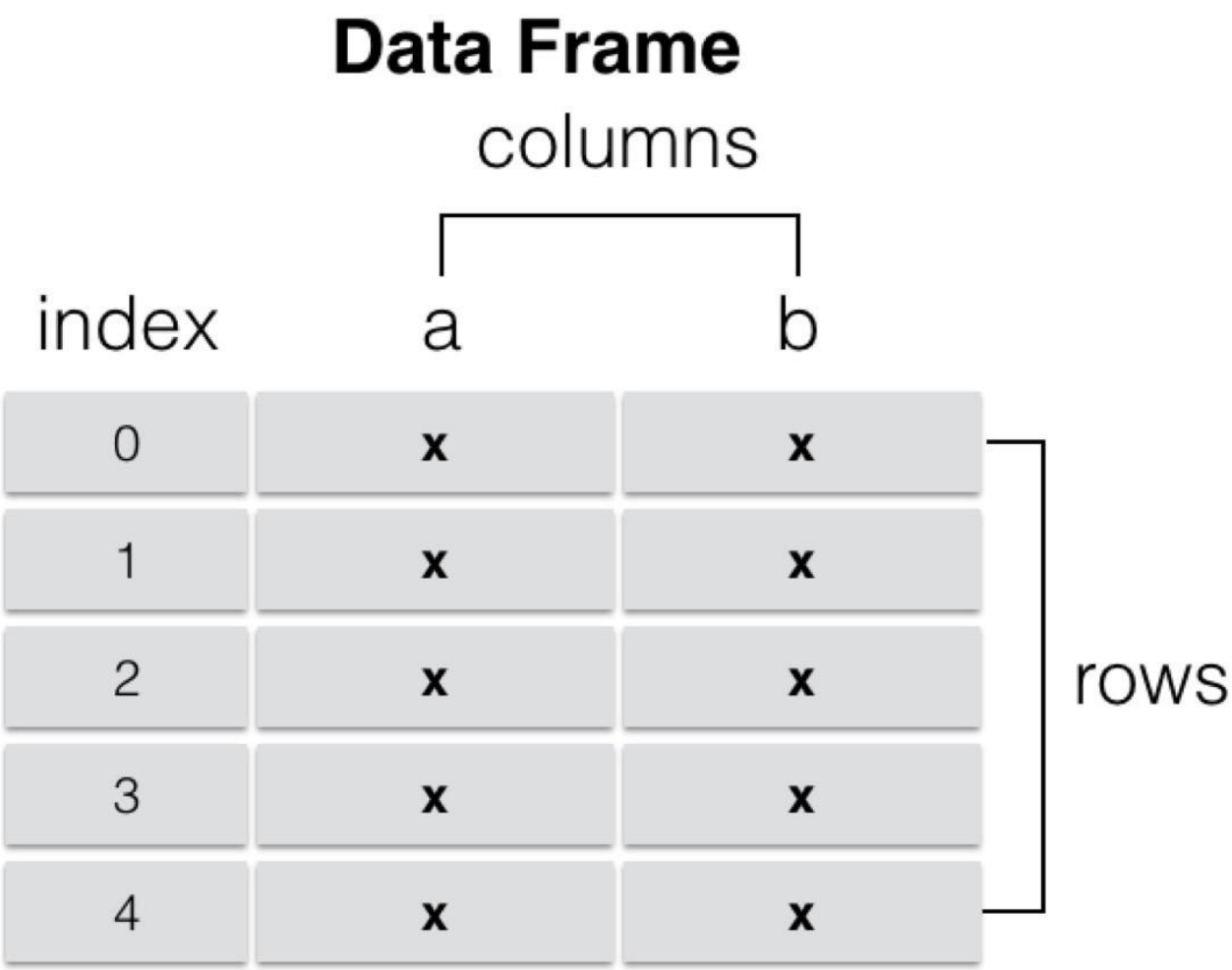
```
print('查看数据：',ser_obj1.values) print('-
'*30)
print('查看索引：',ser_obj1.index)
print('-'*30)
print('通过索引获取数据：',ser_obj1[3]) print('-
'*30)
print('通过索引获取数据：',ser_obj2[2])
print('-'*30) ser_obj2.index.name =
"序列" ser_obj2.name = "测试"
print('给Seris和索引命名:\n',ser_obj2)
```

打印结果

```
查看数据：  [5 6 7 8 9]
--------------------------------
查看索引：  RangeIndex(start=0, stop=5, step=1)
--------------------------------
通过索引获取数据：  8
--------------------------------
通过索引获取数据：  40
--------------------------------
给Seris和索引命名:
 序列
data20    20
data30    30
data40    40
Name: 测试, dtype: int64
```

1.2 DataFrame

DataFrame是一个表格型的数据结构，它含有一组有序的列，每列可以是不同类型的值。DataFrame既有行索引也有列索引，它可以被看做是由Series组成的字典（共用同一个索引），数据是以二维结构存放的。 - 类似多维数组/表格数据（如，excel, R中的data.frame） - 每列数据可以是不同的类型 - 索引包括列索引和行索引



1.2.1 DataFrame 对象的创建

示例代码

```
import pandas as pd import
numpy as np

# 通过二维数组创建DataFrame对象
arr = np.random.randint(2, 6, (3,4)) print(arr)
df_obj1 = pd.DataFrame(arr) print('---------
----------')
print(df_obj1)

# 通过字典创建DataFrame对象
dict_data = {"A": 100, "B": pd.Timestamp("20171016"),
```

```
                    "C": "Xianyu", "D": np.array([3] * 4),
                    "E": ["C++", "Python", "Java","PHP"]} print('----
----------')
df_obj2 = pd.DataFrame(dict_data) # 创建DataFrame对象
print(df_obj2)
```

打印结果：

```
[[4 5 3 3]
 [5 3 3 4]
 [5 4 4 2]]
--------------------------
   0  1  2  3
0  4  5  3  3
1  5  3  3  4
2  5  4  4  2
--------------------------
     A          B       C  D       E
0  100 2017-10-16  Xianyu  3     C++
1  100 2017-10-16  Xianyu  3  Python
2  100 2017-10-16  Xianyu  3    Java
3  100 2017-10-16  Xianyu  3     PHP
```

1.2.2 DataFrame 通过索引获取数据

示例代码

```
import pandas as pd import
numpy as np

# 通过字典创建DataFrame对象
dict_data = {"A": 100, "B": pd.Timestamp("20171016"), "C":
                    "Xianyu", "D": np.array([3] * 4),
                    "E": ["C++", "Python", "Java","PHP"]}

df_obj2 = pd.DataFrame(dict_data) # 创建DataFrame对象

print(df_obj2["E"])
print(df_obj2.E)
```

打印结果：

```
0          C++
1       Python
2         Java
3          PHP
Name: E, dtype:object 0
           C++
1       Python
2         Java
3          PHP
Name: E, dtype: object
```

1.2.3 DataFrame 增加列数据或删除

示例代码

```
import pandas as pd import
numpy as np

# 通过字典创建DataFrame对象
dict_data = {"A": 100, "B": pd.Timestamp("20171016"), "C":
                    "Xianyu", "D": np.array([3] * 4),
                    "E": ["C++", "Python", "Java","PHP"]}

df_obj2 = pd.DataFrame(dict_data) # 创建DataFrame对象

df_obj2["F"] = df_obj2["D"] + 10 # 增加一列，数据为D列的每个数字加10
print(df_obj2)
del (df_obj2["D"])  #删除D列
print(df_obj2)
```

打印结果：

```
     A          B       C  D       E   F
0  100 2017-10-16  Xianyu  3     C++  13
1  100 2017-10-16  Xianyu  3  Python  13
2  100 2017-10-16  Xianyu  3    Java  13
3  100 2017-10-16  Xianyu  3     PHP  13
     A          B       C       E   F
0  100 2017-10-16  Xianyu     C++  13
1  100 2017-10-16  Xianyu  Python  13
2  100 2017-10-16  Xianyu    Java  13
3  100 2017-10-16  Xianyu     PHP  13
```

2. 索引操作

创建一个Dataframe对象

```
import pandas as pd import
numpy as np

#index指定索引行名，columns指定索引列名
df_obj = pd.DataFrame(np.random.rand(3, 4), index=['A', 'B', 'C'], columns=['a', 'b', 'c', 'd']) print(df_obj)
```

```
          a         b         c         d
A  0.469455  0.637635  0.603428  0.214533
B  0.610082  0.452971  0.035141  0.840517
C  0.330020  0.194059  0.054812  0.389160
```

2.1 索引取值

```
import pandas as pd import
numpy as np

#index指定索引行名，columns指定索引列名
df_obj = pd.DataFrame(np.random.rand(3, 4), index=['A', 'B', 'C'], columns=['a', 'b', 'c', 'd']) print(df_obj['b'])
```

```
A    0.212508
B    0.829791
C    0.214478
Name: b, dtype:float64
```

2.2 不连续索引

```
import pandas as pd import
numpy as np

#index指定索引行名，columns指定索引列名
df_obj = pd.DataFrame(np.random.rand(3, 4), index=['A', 'B', 'C'], columns=['a', 'b', 'c', 'd'])

# 取出指定的列数据
print(df_obj[['a', 'c','d']])
```

```
          a         c         d
A  0.363549  0.650133  0.596102
B  0.433886  0.339270  0.466601
C  0.676856  0.979104  0.390176
```

2.3 高级索引

loc 根据标签做切片

DataFrame 不能直接切片，可以通过loc来做切片

loc是基于标签名的索引，也就是我们自定义的索引名

```
import pandas as pd import
numpy as np

#index指定索引行名，columns指定索引列名
df_obj = pd.DataFrame(np.random.rand(3, 4), index=['A', 'B', 'C'], columns=['a', 'b', 'c', 'd'])

#loc包含两个参数，第一个为行的索引，第二个为列的索引
print(df_obj.loc['A':'C','c':'d'])
```

```
          c         d
A  0.032798  0.999552
B  0.753305  0.309328
C  0.444314  0.007701
```

iloc 根据索引下表做切片

作用和loc一样，不过是基于索引编号来索引

```
import pandas as pd import
numpy as np

#index指定索引行名，columns指定索引列名
df_obj = pd.DataFrame(np.random.rand(3, 4), index=['A', 'B', 'C'], columns=['a', 'b', 'c', 'd'])

print(df_obj.iloc[:,2:4])
```

```
          c         d
A  0.325931  0.683291
B  0.200420  0.127396
C  0.172634  0.947526
```

ix 根据索引/下标，混合做切片

ix是以上二者的综合，既可以使用索引编号，又可以使用自定义索引，要视情况不同来使用，

如果索引既有数字又有英文，那么这种方式是不建议使用的，容易导致定位的混乱。

```
import pandas as pd import
numpy as np

#index指定索引行名，columns指定索引列名
df_obj = pd.DataFrame(np.random.rand(3, 4), index=['A', 'B', 'C'], columns=['a', 'b', 'c', 'd'])

print(df_obj.ix[1:3])
```

```
          a         b         c         d
B  0.174557  0.045430  0.274111  0.970873
C  0.996574  0.556244  0.971159  0.212596
```

3. 对齐运算

是数据清洗的重要过程，可以按索引对齐进行运算，如果没对齐的位置则补NaN，最后也可以填充NaN

3.1 Series的对齐运算

Series 按行、索引对齐

示例代码：

```
import pandas as pd

s1 = pd.Series(range(10, 20), index=range(10)) s2 =
pd.Series(range(20, 25), index=range(5))

print('s1: ') print(s1)

print('')

print('s2: ') print(s2)
```

运行结果：

```
s1:
0    10
1    11
2    12
3    13
4    14
5    15
6    16
7    17
8    18
9    19
dtype: int64

s2:
0    20
1    21
2    22
3    23
4    24
dtype: int64
```

Series的对齐运算

示例代码：

```
# Series 对齐运算
print('s1 + s2')
```

运行结果：

```
0    30.0
1    32.0
2    34.0
3    36.0
4    38.0
5     NaN
6     NaN
7     NaN
8     NaN
9     NaN
dtype: float64
```

3.2 DataFrame的对齐运算

DataFrame按行、列索引对齐

示例代码：

```
import pandas as pd import
numpy as np

df1 = pd.DataFrame(np.ones((2, 2)), columns=['a', 'b'])
df2 = pd.DataFrame(np.ones((3, 3)), columns=['a', 'b', 'c'])

print('df1: ') print(df1)

print('')
print('df2: ') print(df2)
```

运行结果：

```
df1:
     a    b
0  1.0  1.0
1  1.0  1.0

df2:
     a    b    c
0  1.0  1.0  1.0
1  1.0  1.0  1.0
2  1.0  1.0  1.0
```

DataFrame的对齐运算

示例代码：

```
# DataFrame对齐操作
df1 + df2
```

运行结果：

```
     a    b   c
0  2.0  2.0 NaN
1  2.0  2.0 NaN
2  NaN  NaN NaN
```

3.3 填充未对齐的数据进行运算

fill_value

使用add, sub, div, mul的同时，

通过fill_value指定填充值，未对齐的数据将和填充值做运

算示例代码：

```
import pandas as pd import
numpy as np

df1 = pd.DataFrame(np.ones((2, 2)), columns=['a', 'b'])
df2 = pd.DataFrame(np.ones((3, 3)), columns=['a', 'b', 'c'])

s1 = pd.Series(range(10, 20), index=range(10)) s2 =
pd.Series(range(20, 25), index=range(5))

print(s1) print(s2)

print(s1.add(s2,fill_value=-1))

print(df1)
```

```
print(df2)

print(df1.sub(df2,fill_value=2.))
```

运行结果：

```
# print(s1) 0
     10
1    11
2    12
3    13
4    14
5    15
6    16
7    17
8    18
9    19
dtype: int64

# print(s2) 0
     20
1    21
2    22
3    23
4    24
dtype: int64

# s1.add(s2, fill_value =-1) 0
     30.0
1    32.0
2    34.0
3    36.0
4    38.0
5    14.0
6    15.0
7    16.0
8    17.0
9    18.0
dtype: float64


# print(df1)
     a    b
0  1.0  1.0
1  1.0  1.0

# print(df2)
     a    b    c
0  1.0  1.0  1.0
1  1.0  1.0  1.0
2  1.0  1.0  1.0
```

```
# df1.sub(df2, fill_value =2.)
```

	a	b	c
0	0.0	0.0	1.0
1	0.0	0.0	1.0
2	1.0	1.0	1.0

4. 函数应用

apply 和 applymap：

可直接使用NumPy的函数

示例代码：

```
# Numpy ufunc 函数
df = pd.DataFrame(np.random.randn(5,4) - 1)
print(df)

print(np.abs(df))
```

运行结果：

	0	1	2	3
0	-1.715441	-0.179837	-0.145211	-0.728561
1	-1.306256	-1.295332	-2.025300	-0.688926
2	0.425331	-1.720239	-1.321167	-0.531384
3	-0.460321	-3.278391	1.789114	-1.192318
4	-1.257866	-2.024370	-1.740532	-0.864311

	0	1	2	3
0	1.715441	0.179837	0.145211	0.728561
1	1.306256	1.295332	2.025300	0.688926
2	0.425331	1.720239	1.321167	0.531384
3	0.460321	3.278391	1.789114	1.192318
4	1.257866	2.024370	1.740532	0.864311

通过apply将函数应用到列或行上

示例代码：

```
# 使用apply应用行或列数据
#f = lambda x : x.max()
print(df.apply(lambda x :x.max()))
```

运行结果：

```
0       0.051188
1       1.037033
2      -0.669072
3      -0.076978
dtype: float64
```

注意指定轴的方向，默认axis=0，方向是列

示例代码：

```
# 指定轴方向，axis=1，方向是行
print(df.apply(lambda x : x.max(), axis=1))
```

运行结果：

```
0       0.359039
1       0.190150
2      -0.561958
3      -0.517184
4       0.164946
dtype: float64
```

通过**applymap**将函数应用到每个数据上示例代码：

```
#使用applymap应用到每个数据
f2 = lambda x : '%.2f' % x
print(df.applymap(f2))
```

运行结果：

```
      0      1      2      3
0 -1.52  -0.05  -1.56  -0.68
1 -1.96  -3.04   0.08  -0.87
2  0.52  -1.99   0.78  -0.21
3  0.02  -1.44  -2.11  -1.03
4 -1.33  -1.29  -1.82  -1.35
```

5. 排序

5.1 索引排序

`sort_index()`

排序默认使用升序排序，`ascending=False` 为降序排

序示例代码：

```
import pandas as pd import
numpy as np

# Series
s4 = pd.Series(range(10, 15), index=np.random.randint(5, size=5)) print(s4)

# 索引排序
print(s4.sort_index())  # 0 0 1 33
```

运行结果：

```
2    10
2    11
1    12
1    13
3    14
dtype: int64

1    12
1    13
2    10
2    11
3    14
dtype: int64
```

> 对DataFrame操作时注意轴方向

示例代码：

```
import pandas as pd import
numpy as np

# DataFrame
df4 = pd.DataFrame(np.random.randn(3, 5),
                   index=np.random.randint(3, size=3),
                   columns=np.random.randint(5,  size=5))
print(df4)

df4_isort=df4.sort_index(axis=1,ascending=False)
print(df4_isort)  # 4 2 1 10
```

运行结果：

```
          4         1         4         0         0
2  1.178415 -0.772974  0.097241  0.352205  0.507060
2 -0.053121  0.991102 -1.947110  0.316616 -0.462781
0  0.014143  0.916708  1.003581  0.314284 -1.336108
          4         4         1         0         0
2  1.178415  0.097241 -0.772974  0.352205  0.507060
2 -0.053121 -1.947110  0.991102  0.316616 -0.462781
0  0.014143  1.003581  0.916708  0.314284 -1.336108
```

5.2 按值排序

`sort_values(by='column name')`

根据某个唯一的列名进行排序，如果有其他相同列名则报错。

示例代码：

```
import pandas as pd import
numpy as np

# DataFrame
df4 = pd.DataFrame(np.random.randn(3, 5),
                   index=np.random.randint(3, size=3),
                   columns=np.random.randint(5,  size=5))

# 按值排序
df4_vsort = df4.sort_values(by=0, ascending=False)
print(df4_vsort)
```

运行结果：

```
          1         2         4         0         2
2  0.942696 -0.359493 -1.070879 -1.450919  0.592629
0  0.870878 -0.444433 -1.731290 -1.463504 -1.284208
0 -1.119502 -0.033885 -0.944169 -2.247666  1.544891
```

6. 处理缺失数据

示例代码：

```
import numpy as np import
pandas as pd

df_data = pd.DataFrame([np.random.randn(3), [1., 2., np.nan],
                        [np.nan, 4., np.nan], [1., 2., 3.]])
print(df_data.head())
```

运行结果：

```
          0         1         2
0 -0.307292 -0.257819  0.517772
1  1.000000  2.000000       NaN
2       NaN  4.000000       NaN
3  1.000000  2.000000  3.000000
```

6.1 判断是否存在缺失值：isnull()

示例代码：

```
import numpy as np import
pandas as pd

df_data = pd.DataFrame([np.random.randn(3), [1., 2., np.nan],
                        [np.nan, 4., np.nan], [1., 2., 3.]])

# isnull print(df_data.isnull())
```

运行结果：

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

6.2 丢弃缺失数据：dropna()

根据axis轴方向，丢弃包含NaN的行或列。 示例代码：

```
import numpy as np import
pandas as pd

df_data = pd.DataFrame([np.random.randn(3), [1., 2., np.nan],
                        [np.nan, 4., np.nan], [1., 2., 3.]])

# dropna
print(df_data.dropna())

print(df_data.dropna(axis=1))
```

运行结果：

```
          0         1         2
0  0.585092 -0.033454-0.405858
3  1.000000  2.000000 3.000000
          1
0 -0.033454
1   2.000000
2   4.000000
3   2.000000
```

6.3 填充缺失数据：fillna()

示例代码：

```
import numpy as np import
pandas as pd

df_data = pd.DataFrame([np.random.randn(3), [1., 2., np.nan],
                                    [np.nan, 4., np.nan], [1., 2., 3.]])

# fillna print(df_data.fillna(-100.))
```

运行结果：

```
              0         1           2
0     -1.318291  2.001131   -0.875723
1      1.000000  2.000000 -100.000000
2   -100.000000  4.000000 -100.000000
3      1.000000  2.000000    3.000000
```