

# AI基础: Pandas简易入门

机器学习初学者 11月23日

以下文章来源于光城，作者lightcity



光城

本公众号旨在推送一些个人学习c++、python的一些历程，包括python爬虫，机器学习...

## Pandas 简易入门

### 0. 导语

pandas 是基于 NumPy 的一种工具，该工具是为了解决数据分析任务而创建的。Pandas 纳入了大量库和一些标准的数据模型，提供了高效地操作大型数据集所需的工具。pandas 提供了大量能使我们快速便捷地处理数据的函数和方法。你很快就会发现，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。

在此之前，我已经写了一篇Numpy的快速入门：

**AI基础：Numpy简易入门**，[建议先看这篇](#)。

接下来怎么快速入门Pandas？

先完整运行本文的代码，预计用一天时间就够了，再尝试完成这篇文章的代码：

**Pandas 练习题-提高你的数据分析技能**，[巩固下](#)。

接着呢？可以练习这篇文章的代码：

**学完可以解决 90%以上的数据分析问题-利用 python 进行数据分析第二版（代码和中文笔记）**，[有了前面的基础，看起来会非常快，这个时候，pandas的基本操作都会了。](#)

备注：本文代码可以在github下载

<https://github.com/fengdu78/Data-Science-Notes/tree/master/3.pandas>

### 1. Series

```
import pandas as pd
```

```
import numpy as np

# Series

s = pd.Series([1,3,6,np.nan,44,1])
print(s)

# 默认index从0开始,如果想要按照自己的索引设置,则修改index参数,如:index=[3,4,3,7,8,9]
```

```
0    1.0
1    3.0
2    6.0
3    NaN
4   44.0
5    1.0
dtype: float64
```

## 2.DataFrame

### 2.1 DataFrame 的简单运用

```
# DataFrame

dates = pd.date_range('2018-08-19',periods=6)

# dates = pd.date_range('2018-08-19','2018-08-24') # 起始、结束 与上述等价
'''

numpy.random.randn(d0, d1, ..., dn)是从标准正态分布中返回一个或多个样本值。

numpy.random.rand(d0, d1, ..., dn)的随机样本位于[0, 1)中。

(6,4)表示6行4列数据
'''

df = pd.DataFrame(np.random.randn(6,4),index=dates,columns=['a','b','c','d'])
print(df)

# DataFrame既有行索引也有列索引, 它可以被看做由Series组成的大字典。
```

	a	b	c	d
2018-08-19	0.090400	-0.029562	-2.004038	2.686679
2018-08-20	-0.531038	-0.750023	0.662672	1.637006
2018-08-21	-1.040762	-0.005521	-0.531630	0.192298
2018-08-22	-0.388458	0.456383	0.412524	1.918840
2018-08-23	0.446538	1.062472	1.179866	-0.725910
2018-08-24	2.828722	1.234659	1.251329	0.620756

```
print(df['b'])
```

```
2018-08-19    -0.029562
2018-08-20    -0.750023
2018-08-21    -0.005521
```

```

2018-08-22    0.456383
2018-08-23    1.062472
2018-08-24    1.234659
Freq: D, Name: b, dtype: float64

```

# 未指定行标签和列标签的数据

```

df1 = pd.DataFrame(np.arange(12).reshape(3,4))
print(df1)

```

```

   0  1  2  3
0  0  1  2  3
1  4  5  6  7
2  8  9 10 11

```

# 另一种方式

```

df2 = pd.DataFrame({
    'A': [1,2,3,4],
    'B': pd.Timestamp('20180819'),
    'C': pd.Series([1,6,9,10],dtype='float32'),
    'D': np.array([3] * 4,dtype='int32'),
    'E': pd.Categorical(['test','train','test','train']),
    'F': 'foo'
})
print(df2)

```

```

   A      B      C  D      E      F
0  1 2018-08-19  1.0  3   test   foo
1  2 2018-08-19  6.0  3  train   foo
2  3 2018-08-19  9.0  3   test   foo
3  4 2018-08-19 10.0  3  train   foo

```

```

print(df2.index)

```

```

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

```

```

print(df2.columns)

```

```

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

```

```

print(df2.values)

```

```

[[1 Timestamp('2018-08-19 00:00:00') 1.0 3 'test' 'foo']
 [2 Timestamp('2018-08-19 00:00:00') 6.0 3 'train' 'foo']
 [3 Timestamp('2018-08-19 00:00:00') 9.0 3 'test' 'foo']
 [4 Timestamp('2018-08-19 00:00:00') 10.0 3 'train' 'foo']]

```

# 数据总结

```
print(df2.describe())
```

	A	C	D
count	4.000000	4.000000	4.0
mean	2.500000	6.500000	3.0
std	1.290994	4.041452	0.0
min	1.000000	1.000000	3.0
25%	1.750000	4.750000	3.0
50%	2.500000	7.500000	3.0
75%	3.250000	9.250000	3.0
max	4.000000	10.000000	3.0

```
# 翻转数据
print(df2.T)
# print(np.transpose(df2)) 等价于上述操作
```

	0	1	2 \
A	1	2	3
B	2018-08-19 00:00:00	2018-08-19 00:00:00	2018-08-19 00:00:00
C	1	6	9
D	3	3	3
E	test	train	test
F	foo	foo	foo

	3
A	4
B	2018-08-19 00:00:00
C	10
D	3
E	train
F	foo

```
'''
axis=1表示行
axis=0表示列
默认ascending(升序)为True
ascending=True表示升序,ascending=False表示降序
下面两行分别表示按行升序与按行降序
'''
print(df2.sort_index(axis=1,ascending=True))
```

	A	B	C	D	E	F
0	1	2018-08-19	1.0	3	test	foo
1	2	2018-08-19	6.0	3	train	foo
2	3	2018-08-19	9.0	3	test	foo
3	4	2018-08-19	10.0	3	train	foo

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

	F	E	D	C	B	A
0	foo	test	3	1.0	2018-08-19	1
1	foo	train	3	6.0	2018-08-19	2
2	foo	test	3	9.0	2018-08-19	3
3	foo	train	3	10.0	2018-08-19	4

# 表示按列降序与按列升序

```
print(df2.sort_index(axis=0,ascending=False))
```

	A	B	C	D	E	F
3	4	2018-08-19	10.0	3	train	foo
2	3	2018-08-19	9.0	3	test	foo
1	2	2018-08-19	6.0	3	train	foo
0	1	2018-08-19	1.0	3	test	foo

```
print(df2.sort_index(axis=0,ascending=True))
```

	A	B	C	D	E	F
0	1	2018-08-19	1.0	3	test	foo
1	2	2018-08-19	6.0	3	train	foo
2	3	2018-08-19	9.0	3	test	foo
3	4	2018-08-19	10.0	3	train	foo

# 对特定列数值排列

# 表示对C列降序排列

```
print(df2.sort_values(by='C',ascending=False))
```

	A	B	C	D	E	F
3	4	2018-08-19	10.0	3	train	foo
2	3	2018-08-19	9.0	3	test	foo
1	2	2018-08-19	6.0	3	train	foo
0	1	2018-08-19	1.0	3	test	foo

## 3.pandas 选择数据

### 3.1 实战筛选

```
import pandas as pd
import numpy as np

dates = pd.date_range('20180819', periods=6)

df = pd.DataFrame(np.arange(24).reshape((6,4)),index=dates, columns=['A','B','C','D'])
print(df)
```

	A	B	C	D
--	---	---	---	---

2018-08-19	0	1	2	3
2018-08-20	4	5	6	7
2018-08-21	8	9	10	11
2018-08-22	12	13	14	15
2018-08-23	16	17	18	19
2018-08-24	20	21	22	23

```
# 检索A列  
print(df['A'])
```

2018-08-19	0
2018-08-20	4
2018-08-21	8
2018-08-22	12
2018-08-23	16
2018-08-24	20

Freq: D, Name: A, dtype: int32

```
print(df.A)
```

2018-08-19	0
2018-08-20	4
2018-08-21	8
2018-08-22	12
2018-08-23	16
2018-08-24	20

Freq: D, Name: A, dtype: int32

```
# 选择跨越多行或多列  
# 选取前3行  
print(df[0:3])
```

	A	B	C	D
2018-08-19	0	1	2	3
2018-08-20	4	5	6	7
2018-08-21	8	9	10	11

```
print(df['2018-08-19':'2018-08-21'])
```

	A	B	C	D
2018-08-19	0	1	2	3
2018-08-20	4	5	6	7
2018-08-21	8	9	10	11

```
# 根据标签选择数据  
# 获取特定行或列  
# 指定行数据  
print(df.loc['20180819'])
```

```
A    0
B    1
C    2
D    3
```

Name: 2018-08-19 00:00:00, dtype: int32

```
# 指定列
# 两种方式
print(df.loc[:, 'A':'C'])
```

```
      A  B  C
2018-08-19  0  1  2
2018-08-20  4  5  6
2018-08-21  8  9 10
2018-08-22 12 13 14
2018-08-23 16 17 18
2018-08-24 20 21 22
```

```
print(df.loc[:, ['A', 'C']])
```

```
      A  C
2018-08-19  0  2
2018-08-20  4  6
2018-08-21  8 10
2018-08-22 12 14
2018-08-23 16 18
2018-08-24 20 22
```

```
# 行与列同时检索
print(df.loc['20180819', ['A', 'B']])
```

```
A    0
B    1
```

Name: 2018-08-19 00:00:00, dtype: int32

```
# 根据序列loc
# 获取特定位置的值
print(df.iloc[3,1])
```

13

```
print(df.iloc[3:5,1:3]) # 不包含末尾5或3，同列表切片
```

```
      B  C
2018-08-22 13 14
2018-08-23 17 18
```

```
# 跨行操作
```

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

	B	C
2018-08-20	5	6
2018-08-22	13	14
2018-08-24	21	22

```
# 混合选择
```

```
print(df.ix[:3,['A','C']])
```

	A	C
2018-08-19	0	2
2018-08-20	4	6
2018-08-21	8	10

```
print(df.iloc[:3,[0,2]]) # 结果同上
```

	A	C
2018-08-19	0	2
2018-08-20	4	6
2018-08-21	8	10

```
# 通过判断的筛选
```

```
print(df[df.A>8])
```

	A	B	C	D
2018-08-22	12	13	14	15
2018-08-23	16	17	18	19
2018-08-24	20	21	22	23

```
# 通过判断的筛选
```

```
print(df.loc[df.A>8])
```

	A	B	C	D
2018-08-22	12	13	14	15
2018-08-23	16	17	18	19
2018-08-24	20	21	22	23

## • 3.2 筛选总结

### 1.iloc 与 ix 区别





总结:相同点: `iloc` 可以取相应的值, 操作方便, 与 `ix` 操作类似。

//



不同点: `ix` 可以混合选择, 可以填入 `column` 对应的字符选择, 而 `iloc` 只能采用 `index` 索引, 对于列数较多情况下, `ix` 要方便操作许多。

//

## 2.loc 与 iloc 区别



总结: 相同点: 都可以索引处块数据

//



不同点: `iloc` 可以检索对应值, 两者操作不同。

//

## 3.ix 与 loc、iloc 三者的区别



n 总结: `ix` 是混合 `loc` 与 `iloc` 操作

//

如下:对比三者操作,输出结果相同

```
print(df.loc['20180819', 'A': 'B'])
print(df.iloc[0, 0:2])
print(df.ix[0, 'A': 'B'])
```

```
A    0
B    1
Name: 2018-08-19 00:00:00, dtype: int32
A    0
B    1
Name: 2018-08-19 00:00:00, dtype: int32
A    0
B    1
Name: 2018-08-19 00:00:00, dtype: int32
```

## 4.Pandas 设置值

### 4.1 创建数据

```
import pandas as pd
import numpy as np

# 创建数据

dates = pd.date_range('20180820', periods=6)

df = pd.DataFrame(np.arange(24).reshape(6,4), index=dates, columns=['A', 'B', 'C', 'D'])
print(df)
```

	A	B	C	D
2018-08-20	0	1	2	3
2018-08-21	4	5	6	7
2018-08-22	8	9	10	11
2018-08-23	12	13	14	15
2018-08-24	16	17	18	19
2018-08-25	20	21	22	23

### 4.2 根据位置设置 loc 和 iloc

```
# 根据位置设置loc和iloc

df.iloc[2,2] = 111

df.loc['20180820', 'B'] = 2222

print(df)
```

	A	B	C	D
2018-08-20	0	2222	2	3
2018-08-21	4	5	6	7
2018-08-22	8	9	111	11
2018-08-23	12	13	14	15
2018-08-24	16	17	18	19
2018-08-25	20	21	22	23

### 4.3 根据条件设置

```
# 根据条件设置

# 更改B中的数，而更改的位置取决于A的位置，并设相应位置的数为0

df.B[df.A>4] = 0

print(df)
```

	A	B	C	D
2018-08-20	0	2222	2	3
2018-08-21	4	5	6	7
2018-08-22	8	0	111	11
2018-08-23	12	0	14	15
2018-08-24	16	0	18	19
2018-08-25	20	0	22	23

```
df.B.loc[df.A>4] = 0
print(df)
```

	A	B	C	D
2018-08-20	0	2222	2	3
2018-08-21	4	5	6	7
2018-08-22	8	0	111	11
2018-08-23	12	0	14	15
2018-08-24	16	0	18	19
2018-08-25	20	0	22	23

## 4.4 按行或列设置

```
# 按行或列设置
# 列批处理, F列全改为NaN
df['F'] = np.nan
print(df)
```

	A	B	C	D	F
2018-08-20	0	2222	2	3	NaN
2018-08-21	4	5	6	7	NaN
2018-08-22	8	0	111	11	NaN
2018-08-23	12	0	14	15	NaN
2018-08-24	16	0	18	19	NaN
2018-08-25	20	0	22	23	NaN

## 4.5 添加 Series 序列(长度必须对齐)

```
df['E'] = pd.Series([1,2,3,4,5,6], index=pd.date_range('20180820',periods=6))
print(df)
```

	A	B	C	D	F	E
2018-08-20	0	2222	2	3	NaN	1
2018-08-21	4	5	6	7	NaN	2
2018-08-22	8	0	111	11	NaN	3
2018-08-23	12	0	14	15	NaN	4
2018-08-24	16	0	18	19	NaN	5
2018-08-25	20	0	22	23	NaN	6

## 4.6 设定某行某列为特定值

```
# 设定某行某列为特定值

df.ix['20180820','A'] = 56
print(df)

#ix 以后要剥离了，尽量不要用了
```

	A	B	C	D	F	E
2018-08-20	56	2222	2	3	NaN	1
2018-08-21	4	5	6	7	NaN	2
2018-08-22	8	0	111	11	NaN	3
2018-08-23	12	0	14	15	NaN	4
2018-08-24	16	0	18	19	NaN	5
2018-08-25	20	0	22	23	NaN	6

```
df.loc['20180820','A'] = 67
print(df)
```

	A	B	C	D	F	E
2018-08-20	67	2222	2	3	NaN	1
2018-08-21	4	5	6	7	NaN	2
2018-08-22	8	0	111	11	NaN	3
2018-08-23	12	0	14	15	NaN	4
2018-08-24	16	0	18	19	NaN	5
2018-08-25	20	0	22	23	NaN	6

```
df.iloc[0,0] = 76
print(df)
```

	A	B	C	D	F	E
2018-08-20	76	2222	2	3	NaN	1
2018-08-21	4	5	6	7	NaN	2
2018-08-22	8	0	111	11	NaN	3
2018-08-23	12	0	14	15	NaN	4
2018-08-24	16	0	18	19	NaN	5
2018-08-25	20	0	22	23	NaN	6

## 4.7 修改一整行数据

```
# 修改一整行数据

df.iloc[1] = np.nan # df.iloc[1,:]=np.nan
print(df)
```

	A	B	C	D	F	E
2018-08-20	76.0	2222.0	2.0	3.0	NaN	1.0
2018-08-21	NaN	NaN	NaN	NaN	NaN	NaN

2018-08-22	8.0	0.0	111.0	11.0	NaN	3.0
2018-08-23	12.0	0.0	14.0	15.0	NaN	4.0
2018-08-24	16.0	0.0	18.0	19.0	NaN	5.0
2018-08-25	20.0	0.0	22.0	23.0	NaN	6.0

```
df.loc['20180820'] = np.nan # df.loc['20180820,:']=np.nan
print(df)
```

	A	B	C	D	F	E
2018-08-20	NaN	NaN	NaN	NaN	NaN	NaN
2018-08-21	NaN	NaN	NaN	NaN	NaN	NaN
2018-08-22	8.0	0.0	111.0	11.0	NaN	3.0
2018-08-23	12.0	0.0	14.0	15.0	NaN	4.0
2018-08-24	16.0	0.0	18.0	19.0	NaN	5.0
2018-08-25	20.0	0.0	22.0	23.0	NaN	6.0

```
df.ix[2] = np.nan # df.ix[2,:]=np.nan
print(df)
```

	A	B	C	D	F	E
2018-08-20	NaN	NaN	NaN	NaN	NaN	NaN
2018-08-21	NaN	NaN	NaN	NaN	NaN	NaN
2018-08-22	NaN	NaN	NaN	NaN	NaN	NaN
2018-08-23	12.0	0.0	14.0	15.0	NaN	4.0
2018-08-24	16.0	0.0	18.0	19.0	NaN	5.0
2018-08-25	20.0	0.0	22.0	23.0	NaN	6.0

```
df.ix['20180823'] = np.nan
print(df)
```

	A	B	C	D	F	E
2018-08-20	NaN	NaN	NaN	NaN	NaN	NaN
2018-08-21	NaN	NaN	NaN	NaN	NaN	NaN
2018-08-22	NaN	NaN	NaN	NaN	NaN	NaN
2018-08-23	NaN	NaN	NaN	NaN	NaN	NaN
2018-08-24	16.0	0.0	18.0	19.0	NaN	5.0
2018-08-25	20.0	0.0	22.0	23.0	NaN	6.0

## 5.Pandas 处理丢失数据

### 5.1 创建含 NaN 的矩阵

```
# Pandas 处理丢失数据

import pandas as pd
import numpy as np

# 创建含NaN的矩阵
```

# 如何填充和删除NaN数据?

```
dates = pd.date_range('20180820', periods=6)
df = pd.DataFrame(np.arange(24).reshape((6,4)), index=dates, columns=['A', 'B', 'C', 'D'])
print(df)
```

	A	B	C	D
2018-08-20	0	1	2	3
2018-08-21	4	5	6	7
2018-08-22	8	9	10	11
2018-08-23	12	13	14	15
2018-08-24	16	17	18	19
2018-08-25	20	21	22	23

# a.reshape(6,4) 等价于a.reshape((6,4))

```
df.iloc[0,1] = np.nan
df.iloc[1,2] = np.nan
print(df)
```

	A	B	C	D
2018-08-20	0	NaN	2.0	3
2018-08-21	4	5.0	NaN	7
2018-08-22	8	9.0	10.0	11
2018-08-23	12	13.0	14.0	15
2018-08-24	16	17.0	18.0	19
2018-08-25	20	21.0	22.0	23

## 5.2 删除掉有 NaN 的行或列

# 删除掉有NaN的行或列

```
print(df.dropna()) # 默认是删除掉含有NaN的行
```

	A	B	C	D
2018-08-22	8	9.0	10.0	11
2018-08-23	12	13.0	14.0	15
2018-08-24	16	17.0	18.0	19
2018-08-25	20	21.0	22.0	23

```
print(df.dropna(
    axis=0, # 0对行进行操作;1对列进行操作
    how='any' # 'any':只要存在NaN就drop掉; 'all':必须全部是NaN才drop
))
```

	A	B	C	D
2018-08-22	8	9.0	10.0	11
2018-08-23	12	13.0	14.0	15

2018-08-24	16	17.0	18.0	19
2018-08-25	20	21.0	22.0	23

```
# 删除掉所有含有NaN的列
print(df.dropna(
    axis=1,
    how='any'
))
```

	A	D
2018-08-20	0	3
2018-08-21	4	7
2018-08-22	8	11
2018-08-23	12	15
2018-08-24	16	19
2018-08-25	20	23

### 5.3 替换 NaN 值为 0 或者其他

```
# 替换NaN值为0或者其他
print(df.fillna(value=0))
```

	A	B	C	D
2018-08-20	0	0.0	2.0	3
2018-08-21	4	5.0	0.0	7
2018-08-22	8	9.0	10.0	11
2018-08-23	12	13.0	14.0	15
2018-08-24	16	17.0	18.0	19
2018-08-25	20	21.0	22.0	23

### 5.4 是否有缺失数据 NaN

```
# 是否有缺失数据NaN
# 是否为空
print(df.isnull())
```

	A	B	C	D
2018-08-20	False	True	False	False
2018-08-21	False	False	True	False
2018-08-22	False	False	False	False
2018-08-23	False	False	False	False
2018-08-24	False	False	False	False
2018-08-25	False	False	False	False

```
# 是否为NaN
```

```
print(df.isna())
```

	A	B	C	D
2018-08-20	False	True	False	False
2018-08-21	False	False	True	False
2018-08-22	False	False	False	False
2018-08-23	False	False	False	False
2018-08-24	False	False	False	False
2018-08-25	False	False	False	False

```
# 检测某列是否有缺失数据NaN
print(df.isnull().any())
```

```
A    False
B     True
C     True
D    False
dtype: bool
```

```
# 检测数据中是否存在NaN, 如果存在就返回True
print(np.any(df.isnull())==True)
```

```
True
```

## 6.Pandas 导入导出

### 6.1 导入数据

```
import pandas as pd # 加载模块
# 读取csv
data = pd.read_csv('student.csv')
# 打印出data
print(data)
```

	Student ID	name	age	gender
0	1100	Kelly	22	Female
1	1101	Clo	21	Female
2	1102	Tilly	22	Female
3	1103	Tony	24	Male
4	1104	David	20	Male
5	1105	Catty	22	Female
6	1106	M	3	Female
7	1107	N	43	Male
8	1108	A	13	Male
9	1109	S	12	Male
10	1110	David	33	Male



11	1111	Dw	3	Female
12	1112	Q	23	Male
13	1113	W	21	Female

```
# 前三行
```

```
print(data.head(3))
```

	Student ID	name	age	gender
0	1100	Kelly	22	Female
1	1101	Clo	21	Female
2	1102	Tilly	22	Female

```
# 后三行
```

```
print(data.tail(3))
```

	Student ID	name	age	gender
11	1111	Dw	3	Female
12	1112	Q	23	Male
13	1113	W	21	Female

## 6.2 导出数据

```
# 将资料存取成pickle
```

```
data.to_pickle('student.pickle')
```

```
# 读取pickle文件并打印
```

```
print(pd.read_pickle('student.pickle'))
```

	Student ID	name	age	gender
0	1100	Kelly	22	Female
1	1101	Clo	21	Female
2	1102	Tilly	22	Female
3	1103	Tony	24	Male
4	1104	David	20	Male
5	1105	Catty	22	Female
6	1106	M	3	Female
7	1107	N	43	Male
8	1108	A	13	Male
9	1109	S	12	Male
10	1110	David	33	Male
11	1111	Dw	3	Female
12	1112	Q	23	Male
13	1113	W	21	Female

## 7. Pandas 合并操作

## 7.1 Pandas 合并 concat

```
import pandas as pd
import numpy as np

# 定义资料集

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)
```

```
   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
```

```
print(df2)
```

```
   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
```

```
print(df3)
```

```
   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
```

```
# concat纵向合并

res = pd.concat([df1,df2,df3],axis=0)

# 打印结果
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
```

```
# 上述合并过程中, index重复, 下面给出重置index方法
# 只需要将index_ignore设定为True即可

res = pd.concat([df1,df2,df3],axis=0,ignore_index=True)

# 打印结果
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

```
# join 合并方式
#定义资料集

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=['b','c','d','e'], index=[2,3,4])
print(df1)
```

	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

```
print(df2)
```

	b	c	d	e
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

```
'''
join='outer',函数默认为join='outer'。此方法是依照column来做纵向合并, 有相同的column上下合并在一起,
其他独自的column各自成列, 原来没有值的位置皆为NaN填充。
'''
# 纵向"外"合并df1与df2

res = pd.concat([df1,df2],axis=0,join='outer')

print(res)
```

	a	b	c	d	e
1	0.0	0.0	0.0	0.0	NaN

```

2  0.0  0.0  0.0  0.0  NaN
3  0.0  0.0  0.0  0.0  NaN
2  NaN  1.0  1.0  1.0  1.0
3  NaN  1.0  1.0  1.0  1.0
4  NaN  1.0  1.0  1.0  1.0

```

```

# 修改index

res = pd.concat([df1,df2],axis=0,join='outer',ignore_index=True)

print(res)

```

```

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

```

```

# join='inner' 合并相同的字段
# 纵向"内"合并df1与df2

res = pd.concat([df1,df2],axis=0,join='inner')

# 打印结果

print(res)

```

```

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

```

```

# join_axes( 依照axes 合并)
#定义资料集

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=['b','c','d','e'], index=[2,3,4])
print(df1)

```

```

      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

```

```
print(df2)
```

```

      b      c      d      e
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
```

```
# 依照df1.index进行横向合并

res = pd.concat([df1,df2],axis=1,join_axes=[df1.index])
print(res)
```

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

```
# 移除join_axes参数,打印结果

res = pd.concat([df1,df2],axis=1)
print(res)
```

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

```
# append(添加数据)

# append只有纵向合并,没有横向合并

#定义资料集

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'])
s1 = pd.Series([1,2,3,4], index=['a','b','c','d'])

# 将df2合并到df1下面,以及重置index,并打印出结果

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
```

```
# 合并多个df,将df2与df3合并至df1的下面,以及重置index,并打印出结果

res = df1.append([df2,df3], 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

```

*# 合并series, 将s1合并至df1, 以及重置index, 并打印结果*

```

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

```

*# 总结: 两种常用合并方式*

```

res = pd.concat([df1, df2, df3], axis=0, ignore_index=True)
res1 = df1.append([df2, df3], ignore_index=True)
print(res)
print(res1)

```

```

      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

      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

```

## 7.2.Pandas 合并 merge

## 7.2.1 定义资料集并打印出

```
import pandas as pd

# 依据一组key合并
# 定义资料集并打印出

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)
```

```
   A  B key
0  A0 B0 K0
1  A1 B1 K1
2  A2 B2 K2
3  A3 B3 K3
```

```
print(right)
```

```
   C  D key
0  C0 D0 K0
1  C1 D1 K1
2  C2 D2 K2
3  C3 D3 K3
```

## 7.2.2 依据 key column 合并,并打印

```
# 依据key column合并,并打印

res = pd.merge(left,right,on='key')

print(res)
```

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

```
# 依据两组key合并
#定义资料集并打印出

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)

```

```

   A  B key1 key2
0  A0 B0  K0  K0
1  A1 B1  K0  K1
2  A2 B2  K1  K0
3  A3 B3  K2  K1

```

```
print(right)
```

```

   C  D key1 key2
0  C0 D0  K0  K0
1  C1 D1  K1  K0
2  C2 D2  K1  K0
3  C3 D3  K2  K0

```

## 7.2.3 两列合并

```

# 依据key1与key2 columns进行合并, 并打印出四种结果['left', 'right', 'outer', 'inner']

res = pd.merge(left, right, on=['key1', 'key2'], how='inner')
print(res)

```

```

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

```

```

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

```

```

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

```



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

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

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

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

## 7.2.4 Indicator 设置合并列名称

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

	col1	col_left
0	0	a
1	1	b

```
print(df2)
```

	col1	col_right
0	1	2
1	2	2
2	2	2

```
# 依据col1进行合并, 并启用indicator=True, 最后打印
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

```
# 自定义indicator column的名称,并打印出
```

```
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

## 7.2.5 依据 index 合并

```
# 依据index合并
```

```
#定义资料集并打印出
```

```
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)
```

	A	B
K0	A0	B0
K1	A1	B1
K2	A2	B2

```
print(right)
```

	C	D
K0	C0	D0
K2	C2	D2
K3	C3	D3

```
# 依据左右资料集的index进行合并,how='outer',并打印
```

```
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

```
# 依据左右资料集的index进行合并,how='inner',并打印
res = pd.merge(left,right,left_index=True,right_index=True,how='inner')
print(res)
```

	A	B	C	D
K0	A0	B0	C0	D0
K2	A2	B2	C2	D2

## 7.2.6 解决 overlapping 的问题

```
# 解决overlapping的问题
#定义资料集
boys = pd.DataFrame({'k': ['K0', 'K1', 'K2'], 'age': [1, 2, 3]})
girls = pd.DataFrame({'k': ['K0', 'K0', 'K3'], 'age': [4, 5, 6]})
print(boys)
```

	age	k
0	1	K0
1	2	K1
2	3	K2

```
print(girls)
```

	age	k
0	4	K0
1	5	K0
2	6	K3

```
# 使用suffixes解决overlapping的问题
# 比如将上面两个合并时,age重复了,则可通过suffixes设置,以此保证不重复,不同名
res = pd.merge(boys,girls,on='k',suffixes=['_boy','_girl'],how='inner')
print(res)
```

	age_boy	k	age_girl
0	1	K0	4
1	1	K0	5

## 8.Pandas plot 出图

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
data = pd.Series(np.random.randn(1000), index=np.arange(1000))
print(data)
```

```
0      0.143408
1     -1.936116
2     -1.488609
```

中间数据略

```
998    -1.617468
999    -0.115447
```

Length: 1000, dtype: float64

```
print(data.cumsum())
```

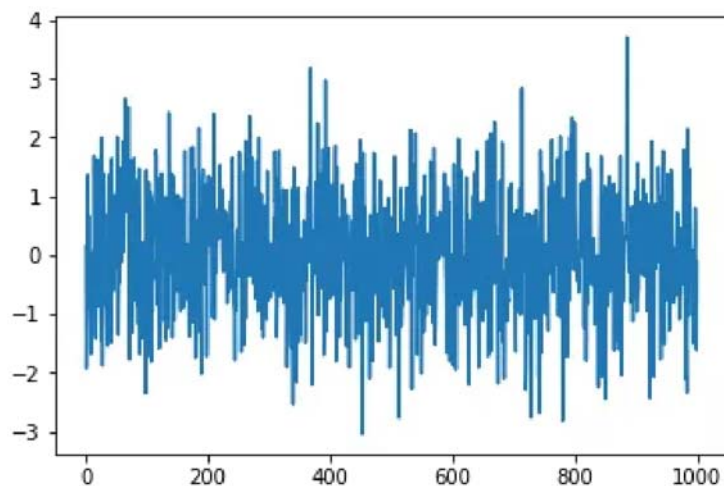
```
0      0.143408
1     -1.792708
2     -3.281317
```

中间数据略

```
997    19.760929
998    18.143460
999    18.028013
```

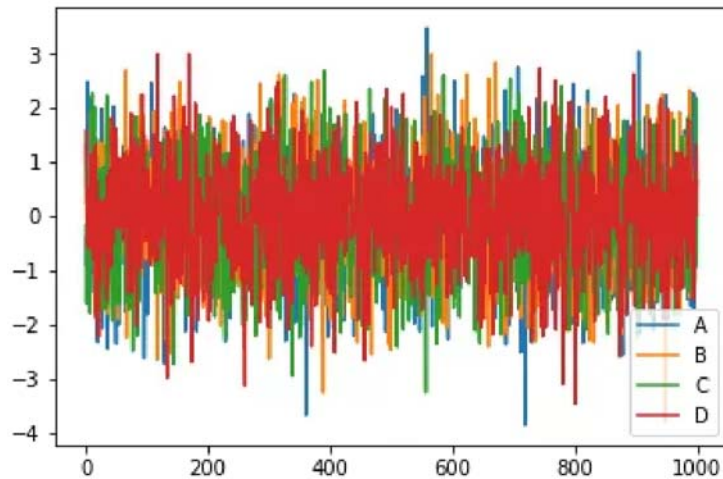
Length: 1000, dtype: float64

```
# data本来就是一个数据，所以我们可以直接plot
data.plot()
plt.show()
```

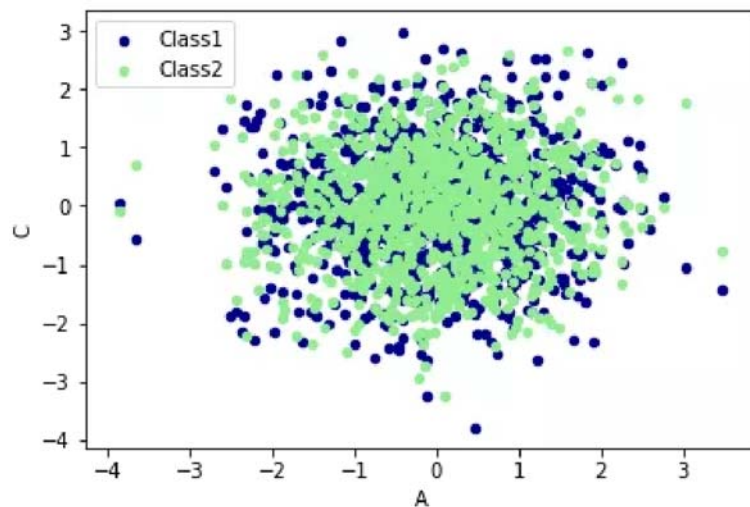


```
# np.random.randn(1000,4) 随机生成1000行4列数据
# list("ABCD")会变为['A','B','C','D']
data = pd.DataFrame(
    np.random.randn(1000,4),
    index=np.arange(1000),
    columns=list("ABCD")
)
data.cumsum()
```

```
data.plot()
plt.show()
```



```
ax = data.plot.scatter(x='A',y='B',color='DarkBlue',label='Class1')
# 将之下这个 data 画在上一个 ax 上面
data.plot.scatter(x='A',y='C',color='LightGreen',label='Class2',ax=ax)
plt.show()
```



## 9.学习来源

- 光城
- <https://morvanzhou.github.io/tutorials/data-manipulation/np-pd/>

备注：本文代码可以在github下载

<https://github.com/fengdu78/Data-Science-Notes/tree/master/3.pandas>



#### 往期回顾



- 那些年做的学术公益-你不是一个人在战斗
- 适合初学者入门人工智能的路线及资料下载
- 机器学习在线手册

备注：加入本站微信群或者qq群，请回复“加群”

加入知识星球（4500+用户，ID：92416895），请回复“知识星球”