# 三，pandas

Pandas的数据结构，缺失数据处理，层次化索引等等

**Pandas的数据结构**

**Series:**

Series数据的建立 ->

In [**1**]: import numpy as np

In [**2**]: import pandas as pd

In [**3**]: pddata\_1=pd.Series([1,2,3])

In [**4**]: pddata\_1

Out[**4**]:

0 1

1 2

2 3

dtype: int64

数据取值和获取搜索索引

In [**7**]: pddata\_1.values

Out[**7**]: array([1, 2, 3], dtype=int64)

In [**8**]: pddata\_1.index

Out[**8**]: RangeIndex(start=0, stop=3, step=1)

例子：

In [**9**]: pddata\_2=pd.Series(['this','is','pandas'])

In [**10**]: pddata\_2.values

Out[**10**]: array(['this', 'is', 'pandas'], dtype=object)

In [**11**]: pddata\_2.index

Out[**11**]: RangeIndex(start=0, stop=3, step=1)

带有独特索引的Series数据：

In [**15**]: pddata\_3=pd.Series([344,33,-456],index=['andereas','hachenberger','dieter'])

In [**16**]: pddata\_3

Out[**16**]:

andereas 344

hachenberger 33

dieter -456

dtype: int64

In [**17**]: pddata\_3.index,pddata\_3.values

Out[**17**]:

(Index(['andereas', 'hachenberger', 'dieter'], dtype='object'),

array([ 344, 33, -456], dtype=int64))

例子：

In [**21**]: testa=[245,788,90]

In [**22**]: pddata\_4=pd.Series(testa,index=['wir','sind','Menschen'])

In [**23**]: pddata\_4

Out[**23**]:

wir 245

sind 788

Menschen 90

dtype: int64

In [**25**]: pddata\_4['wir']

Out[**25**]: 245

In [**27**]: pddata\_4[['wir','sind','Menschen']]

Out[**27**]:

wir 245

sind 788

Menschen 90

dtype: int64

例子：

In [**31**]: seriesdata\_1=pd.Series(['Profession','student','bechelor','professor','informatics developer'],index=['Name','Linker','Hamacher','Schuhmacher','Heintze'])

In [**32**]: seriesdata\_1

Out[**32**]:

Name Profession

Linker student

Hamacher bechelor

Schuhmacher professor

Heintze informatics developer

dtype: object

Series数据中的过滤、计算。

In [**33**]: seriesdata\_2=pd.Series([34,-4,-45,-37,32,9,1,3],index=['a','d','e','y','f','t','u','o'])

In [**34**]: seriesdata\_2

Out[**34**]:

a 34

d -4

e -45

y -37

f 32

t 9

u 1

o 3

dtype: int64

In [**35**]: seriesdata\_2[seriesdata\_2>1]

Out[**35**]:

a 34

f 32

t 9

o 3

dtype: int64

In [**36**]: seriesdata\_2[seriesdata\_2>0]

Out[**36**]:

a 34

f 32

t 9

u 1

o 3

dtype: int64

In [**37**]: seriesdata\_2\*3

Out[**37**]:

a 102

d -12

e -135

y -111

f 96

t 27

u 3

o 9

dtype: int64

In [**38**]: np.sin(seriesdata\_2)

Out[**38**]:

a 0.529083

d 0.756802

e -0.850904

y 0.643538

f 0.551427

t 0.412118

u 0.841471

o 0.141120

dtype: float64

Serie 和字典十分相似，因此原本字典的函数也可以用：

我们先复习下字典：

例子：

In [**39**]: dic={'Winne':178,'Johanis':189,'Banach':186}

In [**40**]: dic['Li']=176

In [**41**]: dic

Out[**41**]: {'Banach': 186, 'Johanis': 189, 'Li': 176, 'Winne': 178}

例子：

In [**19**]: list11=['zhang','wang','li']

In [**20**]: list22=range(3)

In [**21**]: indirect=zip(list22,list11)

In [**22**]: diction={index:value for index,value in indirect}

In [**23**]: diction

Out[**23**]: {0: 'zhang', 1: 'wang', 2: 'li'}

例3：

In [**26**]: list11=['a','b','c']

In [**27**]: list22=[1,2,3]

In [**28**]: dfgeg=zip(list11,list22)

In [**29**]: sss=dict((index,value) for index,value in dfgeg)

In [**30**]: sss

Out[**30**]: {'a': 1, 'b': 2, 'c': 3}

应用字典函数“in”和“Update”到Series

例子

In [**34**]: datas\_pys=pd.Series(range(4),index=['i','want','to','do'])

In [**35**]: datas\_pys

Out[**35**]:

i 0

want 1

to 2

do 3

dtype: int32

In [**36**]: 'want' in datas\_pys

Out[**36**]: True

例子：

In [**56**]: datas\_pys.update(pd.Series([2,3,4],index=['want','to','do']))

In [**57**]: datas\_pys

Out[**57**]:

i 0

want 2

to 3

do 4

dtype: int32

In [**58**]: s1 = pd.Series([1, 2, 3])

In [**59**]: s2 = pd.Series([4, 5, 6])

In [**60**]: s3 = pd.Series([4, 5, 6], index=[3,4,5])

In [**61**]: s1.append(s2)

Out[**61**]:

0 1

1 2

2 3

0 4

1 5

2 6

dtype: int64

In [**62**]: s1.append(s3)

Out[**62**]:

0 1

1 2

2 3

3 4

4 5

5 6

dtype: int64

In [**63**]: s1.append(s2, ignore\_index=True)

Out[**63**]:

0 1

1 2

2 3

3 4

4 5

5 6

dtype: int64

重新编号

字典可以直接转化为Serie

In [**68**]: dicttna={1:'foo',3:'drt',8:'tyue'}

In [**69**]: serie\_12=pd.Series(dicttna)

In [**70**]: serie\_12

Out[**70**]:

1 foo

3 drt

8 tyue

dtype: object

isnull和notnull函数可用于检测数据缺失。

In [**79**]: dit\_113={'lin':139,'zhang':134,'wang':173,'tan':None}

In [**80**]: serie\_123=pd.Series(dit\_113)

In [**81**]: serie\_123

Out[**81**]:

lin 139.0

tan NaN

wang 173.0

zhang 134.0

dtype: float64

In [**82**]: pd.isnull(serie\_123)

Out[**82**]:

lin False

tan True

wang False

zhang False

dtype: bool

In [**83**]: pd.notnull(serie\_123)

Out[**83**]:

lin True

tan False

wang True

zhang True

dtype: bool

serie\_123.notnull()

Out[85]:

lin True

tan False

wang True

zhang True

dtype: bool

Series 的索引可以修改：

In [**87**]: serie\_123.index

Out[**87**]: Index(['lin', 'tan', 'wang', 'zhang'], dtype='object')

In [**88**]: serie\_123.index=['lin', 'tan', 'shan', 'zhang']

In [**90**]: serie\_123

Out[**90**]:

lin 139.0

tan NaN

shan 173.0

zhang 134.0

dtype: float64

**DataFrame**

简单数据框的构成：

由字典直接形成

In [**93**]: sales=[12365,34563,45673,23461,89034]

In [**94**]: seller=['zhanghui','dongyibo','yangqian','liujuntao','zhangshanshan']

In [**95**]: sales\_quantity=[213,305,452,302,190]

In [**96**]: table\_sales={'Seller':seller,'Sales':sales,'SalesQuantity':sales\_quantity}

In [**97**]: framesample\_1=pd.DataFrame(table\_sales)

In [**98**]: framesample\_1

Out[**98**]:

Sales SalesQuantity Seller

0 12365 213 zhanghui

1 34563 305 dongyibo

2 45673 452 yangqian

3 23461 302 liujuntao

4 89034 190 zhangshanshan

与Series一样，索引会自动加上。

我们还可以指定列序列的左右顺序。

In [**99**]: framesample\_1=pd.DataFrame(table\_sales,columns=['SalesQuantity','Seller','Sales'])

In [**100**]: framesample\_1

Out[**100**]:

SalesQuantity Seller Sales

0 213 zhanghui 12365

1 305 dongyibo 34563

2 452 yangqian 45673

3 302 liujuntao 23461

4 190 zhangshanshan 89034

如果数据框中不包含所要找的值，自动返回NaN

In [**101**]: framesample\_1=pd.DataFrame(table\_sales,columns=['SalesQuantity','Seller','Sales','Profit'])

In [**102**]: framesample\_1

Out[**102**]:

SalesQuantity Seller Sales Profit

0 213 zhanghui 12365 NaN

1 305 dongyibo 34563 NaN

2 452 yangqian 45673 NaN

3 302 liujuntao 23461 NaN

4 190 zhangshanshan 89034 NaN

索引DataFrame 的数据。类似字典

列数据索引

In [**104**]: framesample\_1['Seller']

Out[**104**]:

0 zhanghui

1 dongyibo

2 yangqian

3 liujuntao

4 zhangshanshan

Name: Seller, dtype: object

行数据索引：

In [**106**]: framesample\_1.ix[2]

Out[**106**]:

SalesQuantity 452

Seller yangqian

Sales 45673

Prodit NaN

Name: 2, dtype: object

Dateframe的索引也可以改变。

In [**108**]: framesample\_1.index=[5,6,7,8,9]

In [**109**]: framesample\_1

Out[**109**]:

SalesQuantity Seller Sales Prodit

5 213 zhanghui 12365 NaN

6 305 dongyibo 34563 NaN

7 452 yangqian 45673 NaN

8 302 liujuntao 23461 NaN

9 190 zhangshanshan 89034 NaN

NAN值可以通过赋值语句替换。

In [**13**]: framesample\_1['Profit']=[278,967,654,234,432]

In [**14**]: framesample\_1

Out[**14**]:

SalesQuantity Seller Sales Profit

5 213 zhanghui 12365 278

6 305 dongyibo 34563 967

7 452 yangqian 45673 654

8 302 liujuntao 23461 234

9 190 zhangshanshan 89034 432

当用Series赋值时，可以精确地赋值到行列交叉位，没有指定的行列交叉位不会被赋值，将会以NAN的形式显示。

In [**17**]: val\_Series=pd.Series([0,1,1],index=[6,7,9])

In [**19**]: framesample\_1['Profit']=val\_Series

In [**20**]: framesample\_1

Out[**20**]:

SalesQuantity Seller Sales Profit

5 213 zhanghui 12365 NaN

6 305 dongyibo 34563 0.0

7 452 yangqian 45673 1.0

8 302 liujuntao 23461 NaN

9 190 zhangshanshan 89034 1.0

为不存在的列赋值会产生新的列。

In [**21**]: framesample\_1['loss']=pd.Series([165,0,0,34,0],index=range(5,10,1))

In [**22**]: framesample\_1

Out[**22**]:

SalesQuantity Seller Sales Profit loss

5 213 zhanghui 12365 NaN 165

6 305 dongyibo 34563 0.0 0

7 452 yangqian 45673 1.0 0

8 302 liujuntao 23461 NaN 34

9 190 zhangshanshan 89034 1.0 0

可用del删除列

In [**25**]: del framesample\_1['Profit']

In [**26**]: framesample\_1

Out[**26**]:

SalesQuantity Seller Sales loss

5 213 zhanghui 12365 165

6 305 dongyibo 34563 0

7 452 yangqian 45673 0

8 302 liujuntao 23461 34

9 190 zhangshanshan 89034 0

嵌套字典：

In [**28**]: Qtditc={'TeacherLiu':{'height':172,'weight':67,'age':34},'TeacherHuang':{'height':182,'weight':77,'age':36},'TeacherTao':{'height':192,'weight':98,'age':56} }

In [**29**]: Qtditc

Out[**29**]:

{'TeacherHuang': {'age': 36, 'height': 182, 'weight': 77},

'TeacherLiu': {'age': 34, 'height': 172, 'weight': 67},

'TeacherTao': {'age': 56, 'height': 192, 'weight': 98}}

如果把嵌套字典传给数据框，构造数据框，外键为列，内键为行索引。

In [**30**]: GetNewFrame=pd.DataFrame(Qtditc)

In [**31**]: GetNewFrame

Out[**31**]:

TeacherHuang TeacherLiu TeacherTao

age 36 34 56

height 182 172 192

weight 77 67 98

数据框的转置

In [**32**]: GetNewFrame.T

Out[**32**]:

age height weight

TeacherHuang 36 182 77

TeacherLiu 34 172 67

TeacherTao 56 192 98

Series组成的字典也可以直接转化为数据框：

In [**33**]: Popdic={'LinFeng':pd.Series([23,45,165],index=['age','weight','height']),'Zhangduoli':pd.Series([43,75,175],index=['age','weight','height']),'JinChang':pd.Series([51,46,185],index=['age','weight','height'])}

In [**35**]: Getnumerpop=pd.DataFrame(Popdic)

In [**36**]: Getnumerpop

Out[**36**]:

JinChang LinFeng Zhangduoli

age 51 23 43

weight 46 45 75

height 185 165 175

数据框名字的添加：

In [**45**]: Getnumerpop.columns.name='Name'

In [**46**]: Getnumerpop.index.name='personal information'

In [**47**]: Getnumerpop

Out[**47**]:

Name JinChang LinFeng Zhangduoli

personal information

age 51 23 43

weight 46 45 75

height 185 165 175

数据框的值：

In [**53**]: Getnumerpop.values

Out[**53**]:

array([[ 51, 23, 43],

[ 46, 45, 75],

[185, 165, 175]], dtype=int64)

数据框的index：

例1：

Getnumerpop.index=['p\_age','p\_weight','p\_height']

Getnumerpop.index

Out[57]: Index(['p\_age', 'p\_weight', 'p\_height'], dtype='object')

Getnumerpop

Out[58]:

Name JinChang LinFeng Zhangduoli

p\_age 51 23 43

p\_weight 46 45 75

p\_height 185 165 175

In [64]:index1=Getnumerpop.index

In [65]:index2=GetNewFrame.index

In [66]:index1.append(index2)

Out[66]: Index(['p\_age', 'p\_weight', 'p\_height', 'age', 'height', 'weight'], dtype='object')

例2

index2.intersection(index1)

Out[82]: Index([], dtype='object')

例3：

index2.union(index1)

Out[86]: Index(['age', 'height', 'p\_age', 'p\_height', 'p\_weight', 'weight'], dtype='object')

例4

index1.delete(1)

Out[94]: Index(['p\_age', 'p\_height'], dtype='object')

例5

index1.insert(1,'pheight')

Out[97]: Index(['p\_age', 'pheight', 'p\_weight', 'p\_height'], dtype='object')

index1.is\_monotonic

Out[104]: False

index1.is\_unique

Out[105]: True

index1.unique()

Out[107]: Index(['p\_age', 'p\_weight', 'p\_height'], dtype='object')

# Series索引重建

# In [**2**]: import numpy as np

# In [**3**]: import pandas as pd

In [**4**]: Seriestest\_1=pd.Series([10,-34,-89,36,50],index=['a','b','c','d','e'])

In [**5**]: Seriestest\_1

Out[**5**]:

a 10

b -34

c -89

d 36

e 50

dtype: int64

In [**7**]: Seriestest\_2=Seriestest\_1.reindex(['c','b','d','a','e','f'])

In [**8**]: Seriestest\_2

Out[**8**]:

c -89.0

b -34.0

d 36.0

a 10.0

e 50.0

f NaN

dtype: float64

NAN值可以被替换

In [**10**]: Seriestest\_2=Seriestest\_1.reindex(['c','b','d','a','e','f'],fill\_value=0)

In [**11**]: Seriestest\_2

Out[**11**]:

c -89

b -34

d 36

a 10

e 50

f 0

dtype: int64

使用ffill 和 bfill函数可以自动向前和向后补充缺失索引。**（缺失值补充**）

In [**6**]: testarray1=pd.Series([23,22,34],index=[3,7,9])

In [**10**]: test\_12=testarray1.reindex(range(11),method='ffill')

In [**11**]: test\_12

Out[**11**]:

0 NaN

1 NaN

2 NaN

3 23.0

4 23.0

5 23.0

6 23.0

7 22.0

8 22.0

9 34.0

10 34.0

dtype: float64

In [**12**]: test\_12=testarray1.reindex(range(11),method='bfill')

In [**13**]: test\_12

Out[**13**]:

0 23.0

1 23.0

2 23.0

3 23.0

4 22.0

5 22.0

6 22.0

7 22.0

8 34.0

9 34.0

10 NaN

dtype: float64

数据框的索引修改：（**reindex可以任意删除，添加，交换行列**）

对于数据框，reindex可以修改行索引和列，或两个都更改。

In [**14**]: #这里我用一个新的方法构造数据框

In [**18**]: frame\_1=pd.DataFrame(np.arange(9).reshape(3,3),index=['row1','row2','row3'],columns=['one','two','three'])

In [**19**]: frame\_1

Out[**19**]:

one two three

row1 0 1 2

row2 3 4 5

row3 6 7 8

In [**20**]: frame2=frame\_1.reindex(['row0','row1','row2','row3'])

In [**21**]: frame2

Out[**21**]:

one two three

row0 NaN NaN NaN

row1 0.0 1.0 2.0

row2 3.0 4.0 5.0

row3 6.0 7.0 8.0

In [**22**]: #reindex 修改列

In [**28**]: frame3=frame2.reindex(columns=['four','three'])

In [**29**]: frame3

Out[**29**]:

four three

row0 NaN NaN

row1 NaN 2.0

row2 NaN 5.0

row3 NaN 8.0

In [**30**]: #同时修改列和行索引

In [**31**]: frame4=frame\_1.reindex(['row1','row2','row3','row4'],columns=['five','three','six'])

In [**32**]: frame\_1

Out[**32**]:

one two three

row1 0 1 2

row2 3 4 5

row3 6 7 8

In [**33**]: frame4

Out[**33**]:

five three six

row1 NaN 2.0 NaN

row2 NaN 5.0 NaN

row3 NaN 8.0 NaN

row4 NaN NaN NaN

利用ix函数修改数据框行索引与列，快捷整洁，节约时间。但是Ix已经几乎被启用，现在用loc函数。

In [**35**]: frame5=frame\_1.reindex(columns=['one','four','two','three'])

In [**36**]: frame5

Out[**36**]:

one four two three

row1 0 NaN 1 2

row2 3 NaN 4 5

row3 6 NaN 7 8

In [**40**]: frame5['four']=pd.Series([34,56,78],index=['row1','row2','row3'])

In [**41**]: frame5

Out[**41**]:

one four two three

row1 0 34 1 2

row2 3 56 4 5

row3 6 78 7 8

In [**42**]: frame5.ix[['row2','row1','row3'],['one','two','four','three']]

C:\Users\dongfeng\Anaconda3\lib\site-packages\ipykernel\_launcher.py:1: DeprecationWarning:

.ix is deprecated. Please use

.loc for label based indexing or

.iloc for positional indexing

See the documentation here:

http://pandas.pydata.org/pandas-docs/stable/indexing.html#ix-indexer-is-deprecated

"""Entry point for launching an IPython kernel.

Out[**42**]:

one two four three

row2 3 4 56 5

row1 0 1 34 2

row3 6 7 78 8

#现在用loc的比较多，ix已经不再推荐使用。

In [**44**]: frame5.loc[['row2','row1','row3'],['one','two','four','three']]

Out[**44**]:

one two four three

row2 3 4 56 5

row1 0 1 34 2

row3 6 7 78 8

指定轴上项目的丢弃：

Drop 函数应用到“Series”

In [**46**]: Seriestest\_1=pd.Series([233,356,997],index=['as','a','sample'])

In [**47**]: Seriestest\_1

Out[**47**]:

as 233

a 356

sample 997

dtype: int64

Seriestest\_1.drop('a')

Out[95]:

as 233

sample 997

dtype: int64

In [**48**]: #drop函数或者说方法返回的是一个在指定轴上删除指定值的新对象

Drop函数应用到“DataFrame”

Drop函数删除指定值时必须指定轴，如果不指定轴，默认零轴。也就是说删除“1”轴上的指定项时必须指定轴号。删除“0”轴上的指定项时不需要指定轴号。

In [**53**]: Dataframe\_1\_test=pd.DataFrame(np.floor(np.random.randn(4,4)),index=['xu','liu','zhang','feng'],columns=['stufe','klasse','degree','group'])

In [**54**]: Dataframe\_1\_test

Out[**54**]:

stufe klasse degree group

xu -1.0 0.0 -1.0 1.0

liu 0.0 -2.0 -1.0 -1.0

zhang 0.0 -3.0 -2.0 1.0

feng -1.0 0.0 -1.0 0.0

In [**57**]: Dataframe\_1\_test.drop(['stufe','klasse'],axis=1)

Out[**57**]:

degree group

xu -1.0 1.0

liu -1.0 -1.0

zhang -2.0 1.0

feng -1.0 0.0

In [**59**]: Dataframe\_1\_test.drop(['xu','feng'])

Out[**59**]:

stufe klasse degree group

liu 0.0 -2.0 -1.0 -1.0

zhang 0.0 -3.0 -2.0 1.0

In [**60**]: Dataframe\_1\_test.drop(['liu'])

Out[**60**]:

stufe klasse degree group

xu -1.0 0.0 -1.0 1.0

zhang 0.0 -3.0 -2.0 1.0

feng -1.0 0.0 -1.0 0.0

索引与选取

Series的索引和字典没有太大不同,唯一的区别是Series不仅可以通过设置的索引进行检索，也可以用默认的索引进行检索。另外Series也可以做切片，下面我们先看几个例子：

In [**6**]: Testser\_1=pd.Series([23,43,789,674,90,65],index=['apple','pear','persimmon','watermelon','strawberry','orange'])

In [**7**]: Testser\_1

Out[**7**]:

apple 23

pear 43

persimmon 789

watermelon 674

strawberry 90

orange 65

dtype: int64

#按默认索引检索

In [**9**]: Testser\_1[2:4]

Out[**9**]:

persimmon 789

watermelon 674

dtype: int64

#按给定索引检索

In [**25**]: Testser\_1['apple':'watermelon']

Out[**25**]:

apple 23

pear 43

persimmon 789

watermelon 674

dtype: int64

#注意，按给定标签（也叫索引）检索切片末端不会减一末端是被包含的，这就是说末端是封闭区间。（是’]’，不是‘）’）

In [**26**]: Testser\_1[['apple','watermelon']]

Out[**26**]:

apple 23

watermelon 674

dtype: int64

#注意上面两个例子的区别

#也可单个索引，按默认和按给定索引（也叫标签）均可检索

In [**27**]: Testser\_1['apple']

Out[**27**]: 23

In [**28**]: Testser\_1[0]

Out[**28**]: 23

#也可以按照bool值进行索引，此时检索不再按照给定的默认索引，而是与Series中的数据进行比较。

In [**30**]: Testser\_1[Testser\_1>100]

Out[**30**]:

persimmon 789

watermelon 674

dtype: int64

给Series赋值：

In [**31**]: Testser\_1['apple':'watermelon']=[112,22,32,42]

In [**32**]: Testser\_1

Out[**32**]:

apple 112

pear 22

persimmon 32

watermelon 42

strawberry 90

orange 65

dtype: int64

In [**34**]: Testser\_1['apple':'watermelon']=56

In [**35**]: Testser\_1

Out[**35**]:

apple 56

pear 56

persimmon 56

watermelon 56

strawberry 90

orange 65

dtype: int64

例子：

In [**37**]: Testser\_1[[2,3]]=34

In [**38**]: Testser\_1

Out[**38**]:

apple 56

pear 56

persimmon 34

watermelon 34

strawberry 90

orange 65

dtype: int64

In [**39**]: Testser\_1['apple']=34

In [**40**]: Testser\_1

Out[**40**]:

apple 34

pear 56

persimmon 34

watermelon 34

strawberry 90

orange 65

dtype: int64

DataFrame的索引

In [**5**]: Data=pd.DataFrame(np.arange(16).reshape(4,4),index=('one','two','three','four'),columns=('wir','sie','ihr','ich'))

In [**6**]: Data

Out[**6**]:

wir sie ihr ich

one 0 1 2 3

two 4 5 6 7

three 8 9 10 11

four 12 13 14 15

In [**7**]: Data['wir']

Out[**7**]:

one 0

two 4

three 8

four 12

Name: wir, dtype: int32

#单个索引时只能索引列，因此用“bool”值来检索时只能先检索出列数据，然后再与设定值比较，得出bool值Series，见下例：

In [**22**]: Data[Data['ihr']>6]

Out[**22**]:

wir sie ihr ich

three 8 9 10 11

four 12 13 14 15

Bool Series是可以当做索引的，见下

**Data[pd.Series([True,False,True,False],index=['one','two','three','four'])]**

**Out[44]:**

**wir sie ihr ich**

**one 0 1 2 3**

**three 8 9 10 11**

但这种索引容易出错，往往结果与我们的目的不一致，比如：

ata=pd.DataFrame([[1,2,3,4],[0,4,8,2],[0.3,0.6,2.3,4],[3,8,9,0]],index=('one','two','three','four'),columns=('wir','sie','ihr','ich'))

ata

Out[54]:

wir sie ihr ich

one 1.0 2.0 3.0 4

two 0.0 4.0 8.0 2

three 0.3 0.6 2.3 4

four 3.0 8.0 9.0 0

ata[ata['ihr']>3]

Out[55]:

wir sie ihr ich

two 0.0 4.0 8.0 2

four 3.0 8.0 9.0 0

我们想选出大于‘3’，而结果并非如此。

In [**8**]: Data[['wir','ihr']]

Out[**8**]:

wir ihr

one 0 2

two 4 6

three 8 10

four 12 14

In [**13**]: #DataFrame 切片只能依靠行索引

In [**16**]: Data['one':'three']

Out[**16**]:

wir sie ihr ich

one 0 1 2 3

two 4 5 6 7

three 8 9 10 11

In [**17**]: Data[1:3]

Out[**17**]:

wir sie ihr ich

two 4 5 6 7

three 8 9 10 11

bool值索引：

In [**23**]: Data>8

Out[**23**]:

wir sie ihr ich

one False False False False

two False False False False

three False True True True

four True True True True

In [**25**]: Data[Data>8]=0

In [**26**]: Data

Out[**26**]:

wir sie ihr ich

one 0 1 2 3

two 4 5 6 7

three 8 0 0 0

four 0 0 0 0

上面我们已经讲过单个索引时我们只能索引列，为了解决这个问题numpy创造了ix方法。但是ix函数或许太老了，目前已经有新的函数loc来替代它。通过这个函数你几乎可以为所欲为。loc函数只能用给定行列索引进行检索

In [**31**]: Data.loc['two']

Out[**31**]:

wir 4

sie 5

ihr 6

ich 7

Name: two, dtype: int32

In [**33**]: Data.loc['four','wir':'ihr']

Out[**33**]:

wir 12

sie 13

ihr 14

Name: four, dtype: int32

In [**34**]: Data.loc['four',['wir','ihr']]

Out[**34**]:

wir 12

ihr 14

Name: four, dtype: int32

Data.loc[['one','four'],['wir','ich']]

Out[136]:

wir ich

one 0 3

four 12 15

In [**35**]: Data.loc['one':'three','ich']

Out[**35**]:

one 3

two 7

three 11

Name: ich, dtype: int32

In [**38**]: Data.loc['one':,'wir']

Out[**38**]:

one 0

two 4

three 8

four 12

Name: wir, dtype: int32

bool值索引

In [**45**]: Data.loc[Data.wir>5,:'ich']

Out[**45**]:

wir sie ihr ich

three 8 9 10 11

four 12 13 14 15

Reindex 方法：

In [**50**]: Data

Out[**50**]:

wir sie ihr ich

one 0 1 2 3

two 4 5 6 7

three 8 9 10 11

four 12 13 14 15

In [**51**]: Data.reindex(index=['one','three','four','two'])

Out[**51**]:

wir sie ihr ich

one 0 1 2 3

three 8 9 10 11

four 12 13 14 15

two 4 5 6 7

In [**52**]: Data.reindex(columns=['wir','ihr','sie','ich'])

Out[**52**]:

wir ihr sie ich

one 0 2 1 3

two 4 6 5 7

three 8 10 9 11

four 12 14 13 15

Xs方法：根据标签选取单行和单列，返回‘Series’

In [**66**]: Data.xs('one')

Out[**66**]:

wir 0

sie 1

ihr 2

ich 3

Name: one, dtype: int32

In [**69**]: Data.xs('sie',axis=1)

Out[**69**]:

one 1

two 5

three 9

four 13

Name: sie, dtype: int32

Get\_value 方法得到数据框内单个值：

Data.get\_value('three','ihr')

Out[72]: 10

简单的算数运算和数据对齐

加法

Series数据相加必须按照索引一一相加，如果索引不能配对，将返回空值。

In [**1**]: import numpy as np

In [**2**]: import pandas as pd

In [**3**]: Se1=pd.Series([2.3,78,2.5,3.8],index=['as','a','pupil','and'])

In [**4**]: Se2=pd.Series([2,7.7,2.1,3.3,9,12],index=['as','a','studen','teacher','and','or'])

In [**6**]: tesseq\_1=Se1+Se2

In [**7**]: tesseq\_1

Out[**7**]:

a 85.7

and 12.8

as 4.3

or NaN

pupil NaN

studen NaN

teacher NaN

dtype: float64

与Series相同，DataFrame数据相加必须按照行索引和列一一相加，如果索引和列不能配对（也就是两个数据的行列标签不相等时），将返回空值。（先行索引匹配，然后在进行列匹配）

In [**22**]: add1=pd.DataFrame(np.arange(9).reshape(3,3),index=['ein','zwei','drei'],columns=list('bde'))

In [**23**]: add2=pd.DataFrame(np.arange(12).reshape(4,3),index=['zwei','sechs','drei','seven'],columns=list('bef'))

In [**26**]: add1

Out[**26**]:

b d e

ein 0 1 2

zwei 3 4 5

drei 6 7 8

In [**27**]: add2

Out[**27**]:

b e f

zwei 0 1 2

sechs 3 4 5

drei 6 7 8

seven 9 10 11

In [**24**]: Test\_add=add1+add2

In [**25**]: Test\_add

Out[**25**]:

b d e f

drei 12.0 NaN 15.0 NaN

ein NaN NaN NaN NaN

sechs NaN NaN NaN NaN

seven NaN NaN NaN NaN

zwei 3.0 NaN 6.0 NaN

这里我们发现如果我们没有把两个数据框的行标签和列完全一一对应或者说一一相等的话，就会出现很多空值。空值的重新赋值比较麻烦，所以尽量不要出现这种情况。

接下来我们尝试给这个数据框赋值。也就是说消除空值.

首先我们根据数据缺失的实际状况构造出一个新的数据框：

In [**9**]: Newframe=pd.DataFrame(np.array([[2,45,11,30],[-34,45,89,63],[44,90,36,27]]),index=['ein','sechs','seven'],columns=['b','d','e','f'])

In [**10**]: Test\_add.add(Newframe,fill\_value=0)

Out[**10**]:

b d e f

drei 12.0 NaN 15.0 NaN

ein 2.0 45.0 11.0 30.0

sechs -34.0 45.0 89.0 63.0

seven 44.0 90.0 36.0 27.0

zwei 3.0 NaN 6.0 NaN

In [**12**]: Diframe=Test\_add.add(Newframe,fill\_value=0)

In [**13**]: Diframe

Out[**13**]:

b d e f

drei 12.0 NaN 15.0 NaN

ein 2.0 45.0 11.0 30.0

sechs -34.0 45.0 89.0 63.0

seven 44.0 90.0 36.0 27.0

zwei 3.0 NaN 6.0 NaN

In [**26**]: Diframe.loc['drei','d']=36

In [**28**]: Diframe.loc['zwei','d']=0

In [**29**]: Diframe.loc['drei','f']=0

In [**30**]: Diframe.loc['zwei','f']=0

In [**31**]: Diframe

Out[**31**]:

b d e f

drei 12.0 36.0 15.0 0.0

ein 2.0 45.0 11.0 30.0

sechs -34.0 45.0 89.0 63.0

seven 44.0 90.0 36.0 27.0

zwei 3.0 0.0 6.0 0.0

减法乘法和除法

In [**32**]: S1=pd.Series([3,6,9])

In [**34**]: S2=pd.Series([4,3.2,3])

S3=S1-S2

S3

Out[38]:

0 -1.0

1 2.8

2 6.0

dtype: float64

S4=S1\*S2

S4

Out[40]:

0 12.0

1 19.2

2 27.0

dtype: float64

S5=S1/S2

S5

Out[42]:

0 0.750

1 1.875

2 3.000

dtype: float64

In [**43**]: Frame1=pd.DataFrame([[2,2,1],[1,1,3],[0,2,3]],index=['Nr1','Nr2','Nr3'],columns=['mon','tue','wen'])

In [**44**]: Frame2=pd.DataFrame([[1,3,1],[0,1,2],[2,0,1]],index=['Nr1','Nr2','Nr3'],columns=['mon','tue','wen'])

In [**45**]: Frame3=Frame1-Frame2

In [**46**]: Frame3

Out[**46**]:

mon tue wen

Nr1 1 -1 0

Nr2 1 0 1

Nr3 -2 2 2

In [**47**]: Frame4=Frame1\*Frame2

In [**48**]: Frame4

Out[**48**]:

mon tue wen

Nr1 2 6 1

Nr2 0 1 6

Nr3 0 0 3

In [**49**]: Frame5=Frame1/Frame2

In [**50**]: Frame5

Out[**50**]:

mon tue wen

Nr1 2.000000 0.666667 1.0

Nr2 inf 1.000000 1.5

Nr3 0.000000 inf 3.0

DataFrame 和 Series之间运算

数据框与Serie是之间的加减运算分为沿行和沿列运算的。

匹配列索引，沿行运算

In [**1**]: import numpy as np

In [**2**]: import pandas as pd

In [**3**]: Frame1=pd.DataFrame([[2,2,1],[1,1,3],[0,2,3]],index=['Nr1','Nr2','Nr3'],columns=['mon','tue','wen'])

In [**4**]: Frame1

Out[**4**]:

mon tue wen

Nr1 2 2 1

Nr2 1 1 3

Nr3 0 2 3

In [**7**]: S1=pd.Series([2,2,1],index=['mon','tue','wen'])

In [**8**]: Frame1-S1

Out[**8**]:

mon tue wen

Nr1 0 0 0

Nr2 -1 -1 2

Nr3 -2 0 2

这里我们要注意到数据框沿着行一行行的被Series减去。这种计算方法在Python里被称作广播。

如果某个索引不出现在在数据框的列索引或Series的索引里就会出现NaN。这对数据分析也是不利的，因此要求我们在设置索引时一定要一一对应。

In [**9**]: S1=pd.Series([2,2,1],index=['mon','wen','fri'])

In [**10**]: Frame1-S1

Out[**10**]:

fri mon tue wen

Nr1 NaN 0.0 NaN -1.0

Nr2 NaN -1.0 NaN 1.0

Nr3 NaN -2.0 NaN 1.0

匹配行索引，沿列运算(广播）

In [**11**]: S2=pd.Series([2,1,0],index=['Nr1','Nr2','Nr3'])

In [**15**]: Frame1.sub(S2,axis=0)

Out[**15**]:

mon tue wen

Nr1 0 0 -1

Nr2 0 0 2

Nr3 0 2 3

(乘法和除法可以按照.Multiply 和.devide方法就行求解！！！）

函数如何应用到数据框

适合数组的方法与函数，也可应用到pandas的数据结构上。

In [**17**]: frame12=pd.DataFrame(np.arange(12).reshape(4,3),index=['r1','r2','r3','r4'],columns=['c1','c2','c3'])

In [**18**]: frame12

Out[**18**]:

c1 c2 c3

r1 0 1 2

r2 3 4 5

r3 6 7 8

r4 9 10 11

In [**20**]: frame13=frame12\*-1

In [**21**]: frame13

Out[**21**]:

c1 c2 c3

r1 0 -1 -2

r2 -3 -4 -5

r3 -6 -7 -8

r4 -9 -10 -11

In [**22**]: np.abs(frame13)

Out[**22**]:

c1 c2 c3

r1 0 1 2

r2 3 4 5

r3 6 7 8

r4 9 10 11

In [**23**]: f=lambda x:x.max()-x.min()

In [**24**]: frame12.apply(f)

Out[**24**]:

c1 9

c2 9

c3 9

dtype: int64

In [**25**]: frame12.apply(f,axis=1)

Out[**25**]:

r1 2

r2 2

r3 2

r4 2

dtype: int64

.Min（.max）方法和min（max）函数在计算上没有区别，只是方法有更多的选择性。

In [**26**]: frame12.min()

Out[**26**]:

c1 0

c2 1

c3 2

dtype: int32

#不做声明，默认轴为‘0’意思为沿行标签操作

#更多选择性

In [**37**]: frame12.min(0)

Out[**37**]:

c1 0

c2 1

c3 2

dtype: int32

In [**38**]: frame12.min(1)

Out[**38**]:

r1 0

r2 3

r3 6

r4 9

dtype: int32

In [**28**]: np.min(frame12)

Out[**28**]:

c1 0

c2 1

c3 2

dtype: int32

Lamda 函数应用：

In [**23**]: f=lambda x:x.max()-x.min()

In [**24**]: frame12.apply(f)

Out[**24**]:

c1 9

c2 9

c3 9

dtype: int64

In [**25**]: frame12.apply(f,axis=1)

Out[**25**]:

r1 2

r2 2

r3 2

r4 2

dtype: int64

#应用apply方法可以得到更为整齐的结果，试比较例1和例2

In [**6**]: frame11=pd.DataFrame(np.arange(12).reshape(3,4),index=['r1','r2','r3'],columns=['c1','c2','c3','c4'])

In [**13**]: def table\_extrem (x):

    ...: return pd.Series([x.min(),x.max()],index=['min','max'])

    ...:

例子1：

In [**14**]: table\_extrem(frame11)

Out[**14**]:

min c1 0

c2 1

c3 2

c4 3

dtype: int32

max c1 8

c2 9

c3 10

c4 11

dtype: int32

dtype: object

例子2.

In [**15**]: frame11.apply(table\_extrem)

Out[**15**]:

c1 c2 c3 c4

min 0 1 2 3

max 8 9 10 11

# DataFrame 的格式化（十分有用）

In [**16**]: frame11=pd.DataFrame(np.random.randn(3,4),index=['r1','r2','r3'],columns=['c1','c2','c3','c4'])

In [**17**]: frame11

Out[**17**]:

c1 c2 c3 c4

r1 -2.061714 0.584658 -0.540976 0.090904

r2 -0.517271 -0.077818 0.163807 0.418174

r3 0.321513 0.769480 2.131075 -0.535560

In [**18**]: formatierung=lambda x:'%.2f' % x

In [**19**]: frame11.applymap(formatierung)

Out[**19**]:

c1 c2 c3 c4

r1 -2.06 0.58 -0.54 0.09

r2 -0.52 -0.08 0.16 0.42

r3 0.32 0.77 2.13 -0.54

排序和排名

Series 排序：

In [**118**]: import numpy as np

In [**119**]: import pandas as pd

In [**120**]: project\_1=pd.Series(np.arange(1,5),index=['a','d','e','f'])

In [**121**]: project\_1

Out[**121**]:

a 1

d 2

e 3

f 4

dtype: int32

In [**122**]: project\_1.sort\_index()

Out[**122**]:

a 1

d 2

e 3

f 4

dtype: int32

DataFrame 排序：

In [**124**]: DataFrame\_12=pd.DataFrame(np.arange(8).reshape(2,4),index=['r1','r2'],columns=['ted','lie','qiu','send'])

In [**125**]: DataFrame\_12.sort\_index()

Out[**125**]:

ted lie qiu send

r1 0 1 2 3

r2 4 5 6 7

#给行标签排序

In [**126**]: DataFrame\_12.sort\_index(axis=1)

Out[**126**]:

lie qiu send ted

r1 1 2 3 0

r2 5 6 7 4

#给列标签排序

上面的程序行标签和列标签（也叫列索引）都是按升序进行排列。我们也可以按降序排列数据框。

In [**7**]: DataFrame\_12.sort\_index(axis=1,ascending=False)

Out[**7**]:

ted send qiu lie

r1 0 3 2 1

r2 4 7 6 5

我们上面都是按标签（索引）进行排序，其实我们还可以用sort\_values方法对Series按值进行排序：

In [**15**]: Test\_series\_1=pd.Series([3,6,2,9,1,0],index=['r1','r2','r3','r4','r5','r6'])

In [**16**]: Test\_series\_1

Out[**16**]:

r1 3

r2 6

r3 2

r4 9

r5 1

r6 0

dtype: int64

In [**17**]: Test\_series\_1.sort\_values()

Out[**17**]:

r6 0

r5 1

r3 2

r1 3

r2 6

r4 9

dtype: int64

我们也可以对DataFrame进行按值排序

In [**18**]: Dataffdic=pd.DataFrame({'liu':[-1,0,2,-3],'wang':[3,2,-5,7],'jiang':[1,0,1,5]})

In [**19**]: Dataffdic

Out[**19**]:

jiang liu wang

0 1 -1 3

1 0 0 2

2 1 2 -5

3 5 -3 7

In [**20**]: Dataffdic.sort\_index(by='liu')#对'liu'列进行排序

C:\Users\dongfeng\Anaconda3\lib\site-packages\ipykernel\_launcher.py:1: FutureWarning: by argument to sort\_index is deprecated, pls use .sort\_values(by=...)

"""Entry point for launching an IPython kernel.

Out[**20**]:

jiang liu wang

3 5 -3 7

0 1 -1 3

1 0 0 2

2 1 2 -5

In [**21**]: #目前sort\_index方法对数据框某列的值进行排序已经落后，现在用新的方法sort\_values 对列值进行排序。

In [**22**]: Dataffdic.sort\_values(by='liu')

Out[**22**]:

jiang liu wang

3 5 -3 7

0 1 -1 3

1 0 0 2

2 1 2 -5

#由于按‘liu’列进行排序，所以我们看到‘liu’列的数值排序是正确的。由于排序是整行移动，所以在对‘liu’列进行排序时，‘jiang’列和‘wang’列数值顺序一定会随其变得杂乱无章。

下面我们讲讲排名，排名和排序从结果上讲是完全不同的，排序是把一个没有顺序的标签或数据按照符号的自然排序规则（比如如果升序排列字符串“as”在顺序排位上应该在“at”之前）或者数值的太小进行排列。排名是通过打破数据的平级关系，从而产生一种有级别差的顺序。通俗的解释。一组没有排名的数据他们在级别上等级的，然后我们通过某种运算规则，使每个数据变成一个名次数值。

另外，排序可以对数据值本身和标签进行操作。而排名通常对数据本身操作。

In [**1**]: import numpy as np

In [**2**]: import pandas as pd

In [**3**]: Series\_1=pd.Series(np.array([3,8,34,33]),index=['r1','r2','r3','r4'])

In [**4**]: Frame\_1=pd.DataFrame(np.array([[3,2,5,0],[7,4,12,13],[1,0,1,-3],[3,6,2,0]]),index=['r1','r2','r3','r4'],columns=['a','b','c','d'])

Series排名的四种方法

1. ‘average’法

In [**6**]: Series\_1.rank(method='average')

Out[**6**]:

r1 1.0

r2 2.0

r3 4.0

r4 3.0

dtype: float64

1. ‘min’法

In [**8**]: Series\_1.rank(method='min')

Out[**8**]:

r1 1.0

r2 2.0

r3 4.0

r4 3.0

dtype: float64

1. ‘max’法

In [**7**]: Series\_1.rank(method='max')

Out[**7**]:

r1 1.0

r2 2.0

r3 4.0

r4 3.0

dtype: float64

1. ‘first’法

In [**9**]: Series\_1.rank(method='first')

Out[**9**]:

r1 1.0

r2 2.0

r3 4.0

r4 3.0

dtype: float64

上面所有的方法都是产生升序排名的结果，他们也可以产生降序的效果，如果我们填加“ascending”方法。

Series\_1.rank(ascending=False, method='first')

Out[19]:

r1 4.0

r2 3.0

r3 1.0

r4 2.0

dtype: float64

DataFrame的排名

在Series排名上，我们没有发现这四种方法的区别，他们一定有区别，只是我们这个Series的排名结果就是这样，没有显现出区别。我们下面通过DataFrame来区别这四种方法。

In [**14**]: Frame\_1.rank(method='first')

Out[**14**]:

a b c d

r1 2.0 2.0 3.0 2.0

r2 4.0 3.0 4.0 4.0

r3 1.0 1.0 1.0 1.0

r4 3.0 4.0 2.0 3.0

In [**15**]: Frame\_1.rank(method='average')

Out[**15**]:

a b c d

r1 2.5 2.0 3.0 2.5

r2 4.0 3.0 4.0 4.0

r3 1.0 1.0 1.0 1.0

r4 2.5 4.0 2.0 2.5

In [**16**]: Frame\_1.rank(method='min')

Out[**16**]:

a b c d

r1 2.0 2.0 3.0 2.0

r2 4.0 3.0 4.0 4.0

r3 1.0 1.0 1.0 1.0

r4 2.0 4.0 2.0 2.0

In [**17**]: Frame\_1.rank(method='max')

Out[**17**]:

a b c d

r1 3.0 2.0 3.0 3.0

r2 4.0 3.0 4.0 4.0

r3 1.0 1.0 1.0 1.0

r4 3.0 4.0 2.0 3.0

我们发现这四种排名的方式是不一样的，尽管是不一样的，但是排名这个目的都十分正确的实现了。仅仅是穿了不同外壳。

上面都是逐行排名，实际上我们还可以逐列排名。

In [**20**]: Frame\_1.rank(axis=1,method='max')

Out[**20**]:

a b c d

r1 3.0 2.0 4.0 1.0

r2 2.0 1.0 3.0 4.0

r3 4.0 2.0 4.0 1.0

r4 3.0 4.0 2.0 1.0

In [**21**]: Frame\_1.rank(axis=1,method='first')

Out[**21**]:

a b c d

r1 3.0 2.0 4.0 1.0

r2 2.0 1.0 3.0 4.0

r3 3.0 2.0 4.0 1.0

r4 3.0 4.0 2.0 1.0

上面就是逐列排名。个人认为‘first’法排名最为规整，且符合我们的习惯思维。

名词解释：

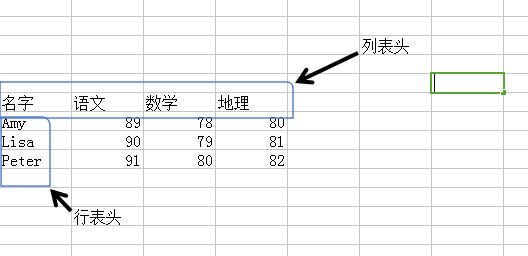
逐列排名（或排序）实际上对一行数据进行排名。

逐行排名（或排序）实际上对一列数据进行排名。

计算机默认逐行排名。

带有重复值的轴索引

对于数据框来说，轴索引就是指的行标签（或者叫行索引）或列标签（或者叫列索引），也就是我们Excel里的行表头和列表头。如下图



很多时候我们要求轴索引是唯一的，但这个要求并不是强制性的，很多时候它可

以重复，重复的轴索引对我们数据分析人员来说，不见得是坏事。

例子:

In [**1**]: import numpy as np

In [**2**]: import pandas as pd

In [**3**]: framne\_repeat=pd.DataFrame(np.arange(16).reshape(4,4),index=['r1','r2','r3','r1'],columns=['a','b','b','c'])

In [**4**]: framne\_repeat

Out[**4**]:

a b b c

r1 0 1 2 3

r2 4 5 6 7

r3 8 9 10 11

r1 12 13 14 15

In [**5**]: framne\_repeat['b']

Out[**5**]:

b b

r1 1 2

r2 5 6

r3 9 10

r1 13 14

In [**8**]: framne\_repeat.loc['r1']

Out[**8**]:

a b b c

r1 0 1 2 3

r1 12 13 14 15

In [**12**]: framne\_repeat.loc[('r1')]

Out[**12**]:

a b b c

r1 0 1 2 3

r1 12 13 14 15

上面两个例子可以看出framne\_repeat.loc['r1']和framne\_repeat.loc[('r1')]运行结果是一样的。也就是说加不加括号都无所谓。

带有重复轴索引的Series

In [**7**]: testserie\_1=pd.Series(np.array([1,2,3,9,10,3]),index=['a','b','a','d','e','a'])

In [**8**]: testserie\_1

Out[**8**]:

a 1

b 2

a 3

d 9

e 10

a 3

dtype: int32

In [**9**]: testserie\_1['a']

Out[**9**]:

a 1

a 3

a 3

dtype: int32

可以通过is\_unique 函数来判断是否存在重复轴索引。

In [**1**]: import pandas as pd

In [**2**]: import numpy as np

In [**3**]: testserie\_1=pd.Series(np.array([1,2,3,9,10,3]),index=['a','b','a','d','e','a'])

In [**4**]: testserie\_1.is\_unique

Out[**4**]: False

In [**5**]: #False值说明有重复轴索引

In [**6**]: #如果对没有重复值的索引进行索引时，返回一个标量。

In [**7**]: testserie\_1['b']

Out[**7**]: 2

Serie 和 Dataframe 的计算

dataFrame\_1=pd.DataFrame([range(4),[np.nan,2,3.6,0.9],[1.2,4,6,7],[3.4,7.9,0.4,8]],index=['r1','r2','r3','r4'],columns=['c1','c2','c3','c4'])

dataFrame\_1

Out[9]:

c1 c2 c3 c4

r1 0.0 1.0 2.0 3.0

r2 NaN 2.0 3.6 0.9

r3 1.2 4.0 6.0 7.0

r4 3.4 7.9 0.4 8.0

dataFrame\_1.sum()

Out[12]:

c1 4.6

c2 14.9

c3 12.0

c4 18.9

dtype: float64

dataFrame\_1.sum(1)

Out[13]:

r1 6.0

r2 6.5

r3 18.2

r4 19.7

dtype: float64

只要不是整行都是NaN值，python自动排除NAN值然后进行计算。通过skipna可以禁止该功能。

dataFrame\_1.sum(axis=1,skipna=False)

Out[22]:

r1 6.0

r2 NaN

r3 18.2

r4 19.7

dtype: float64

dataFrame\_1.idxmax()

Out[23]:

c1 r4

c2 r4

c3 r3

c4 r4

dtype: object

#求出每列最大值

dataFrame\_1.idxmax(1)

Out[7]:

r1 c4

r2 c3

r3 c4

r4 c4

dtype: object

#求出每行最大值

dataFrame\_1.idxmin(1)

Out[9]:

r1 c1

r2 c4

r3 c1

r4 c3

dtype: object

求出每行最小值，空值不参与比较。

dataFrame\_1.idxmin(axis=1,skipna=False)

Out[10]:

r1 c1

r2 NaN

r3 c1

r4 c3

dtype: object

#修改skipna的默认值”True”,可以修改NaN值自动忽略功能。

#累计加

按列累计加，遇到NaN值自动忽略，不参与运算。

dataFrame\_1.cumsum()

Out[8]:

c1 c2 c3 c4

r1 0.0 1.0 2.0 3.0

r2 NaN 3.0 5.6 3.9

r3 1.2 7.0 11.6 10.9

r4 4.6 14.9 12.0 18.9

dataFrame\_1.cumsum(1)

Out[10]:

c1 c2 c3 c4

r1 0.0 1.0 3.0 6.0

r2 NaN 2.0 5.6 6.5

r3 1.2 5.2 11.2 18.2

r4 3.4 11.3 11.7 19.7

dataFrame\_1.describe()

Out[12]:

c1 c2 c3 c4

count 3.000000 4.000 4.000000 4.00000

mean 1.533333 3.725 3.000000 4.72500

std 1.724336 3.050 2.388863 3.34203

min 0.000000 1.000 0.400000 0.90000

25% 0.600000 1.750 1.600000 2.47500

50% 1.200000 3.000 2.800000 5.00000

75% 2.300000 4.975 4.200000 7.25000

max 3.400000 7.900 6.000000 8.00000

#上面的分位数我们后面会有介绍

去重，值计数以及值资格判断

series\_3=pd.Series(['c','d','we','a','c','d','e','f','we'])

series\_3.unique()

Out[25]: array(['c', 'd', 'we', 'a', 'e', 'f'], dtype=object)

#通过uique方法的调用，我们立即可以去除Series中的重复的项目

serie\_unique=series\_3.unique()

serie\_unique.sort()

serie\_unique

Out[40]: array(['a', 'c', 'd', 'e', 'f', 'we'], dtype=object)

#通常去重后要进行排序

通过value\_counts()方法对Series中的值进行计数

series\_3.value\_counts()

Out[43]:

we 2

c 2

d 2

a 1

f 1

e 1

dtype: int64

我们可以通过这种方法求出指定元素出现的次数。

serie\_nr=series\_3.value\_counts()

serie\_nr['we']

Out[46]: 2

上面我们看到都是value\_counts()作为方法在应用，其实它也可以作为函数（或者叫Pandas的方法）单独应用。

pd.value\_counts(series\_3.values)

Out[47]:

we 2

c 2

d 2

a 1

f 1

e 1

dtype: int64

也可以不把计算内容做降序处理：

pd.value\_counts(series\_3.values,sort=False)

Out[49]:

e 1

d 2

f 1

a 1

c 2

we 2

dtype: int64

value\_counts()作为单独函数使用时，是一种超级牛的函数，他可以使用到任何序列和数组。

看下面例子：

list\_1=[1,3,4,5,2,6,8,4,3,8,9]

pd.value\_counts(list\_1)

Out[51]:

8 2

4 2

3 2

9 1

6 1

5 1

2 1

1 1

dtype: int64

tuple\_1=1,2,4,3,2,5,6,3,9

pd.value\_counts(tuple\_1)

Out[54]:

3 2

2 2

9 1

6 1

5 1

4 1

1 1

dtype: int64

array\_1=[1,2,3,4,52,3,2]

pd.value\_counts(array\_1)

Out[56]:

3 2

2 2

4 1

1 1

52 1

dtype: int64

series\_3

Out[60]:

0 c

1 d

2 we

3 a

4 c

5 d

6 e

7 f

8 we

dtype: object

series\_3.isin(['a','c'])

Out[61]:

0 True

1 False

2 False

3 True

4 True

5 False

6 False

7 False

8 False

dtype: bool

series\_3.isin(['a'])

Out[63]:

0 False

1 False

2 False

3 True

4 False

5 False

6 False

7 False

8 False

dtype: bool

#这是一个布尔Series，我们可以把他作为角码选出我们指定值在Series里的所有信息。

boolmatrix\_2=series\_3.isin(['a','c'])

series\_3[boolmatrix\_2]

Out[68]:

0 c

3 a

4 c

dtype: object

我们也可以把value\_counts()用到数据框，会得到一个你预料不到的结果。尽管这个结果很奇怪，但他却是巧妙的导出频数分布图。

dataFrame\_1

Out[69]:

c1 c2 c3 c4

r1 0.0 1.0 2.0 3.0

r2 NaN 2.0 3.6 0.9

r3 1.2 4.0 6.0 7.0

r4 3.4 7.9 0.4 8.0

dataFrame\_1.fillna(0)

Out[71]:

c1 c2 c3 c4

r1 0.0 1.0 2.0 3.0

r2 0.0 2.0 3.6 0.9

r3 1.2 4.0 6.0 7.0

r4 3.4 7.9 0.4 8.0

dataFrame\_2=dataFrame\_1.fillna(0)

result=dataFrame\_2.apply(pd.value\_counts).fillna(0)

result

Out[74]:

c1 c2 c3 c4

0.0 2.0 0.0 0.0 0.0

0.4 0.0 0.0 1.0 0.0

0.9 0.0 0.0 0.0 1.0

1.0 0.0 1.0 0.0 0.0

1.2 1.0 0.0 0.0 0.0

2.0 0.0 1.0 1.0 0.0

3.0 0.0 0.0 0.0 1.0

3.4 1.0 0.0 0.0 0.0

3.6 0.0 0.0 1.0 0.0

4.0 0.0 1.0 0.0 0.0

6.0 0.0 0.0 1.0 0.0

7.0 0.0 0.0 0.0 1.0

7.9 0.0 1.0 0.0 0.0

8.0 0.0 0.0 0.0 1.0

NAN数据处理，

在数据处理工作中，我们经常遇到缺失数据，它们的处理往往很费时间。Pandas设计之初，就已经考虑到这种情况。可以说，快速轻松地处理缺失数据是pandas最大优点之一。

#NaN值和None值都可以被当做空值处理：

series\_1=pd.Series([1,np.nan,3,None,2.3,6])

series\_1

Out[5]:

0 1.0

1 NaN

2 3.0

3 NaN

4 2.3

5 6.0

dtype: float64

缺失值得快速补充法

frame\_123=pd.DataFrame([[1,np.nan,np.nan,2,3],[1,2,3,None,8],[6,2.3,8,None,3],[2,3,2,3,2],[3.2,8.9,3,4.5,3],[2,3,0.45,None,8]])

frame\_123

Out[8]:

0 1 2 3 4

0 1.0 NaN NaN 2.0 3

1 1.0 2.0 3.00 NaN 8

2 6.0 2.3 8.00 NaN 3

3 2.0 3.0 2.00 3.0 2

4 3.2 8.9 3.00 4.5 3

5 2.0 3.0 0.45 NaN 8

pd.isnull(frame\_123)

Out[9]:

0 1 2 3 4

0 False True True False False

1 False False False True False

2 False False False True False

3 False False False False False

4 False False False False False

5 False False False True False

boolmatrix\_1=pd.isnull(frame\_123)

frame\_123[boolmatrix\_1]=0

frame\_123

Out[13]:

0 1 2 3 4

0 1.0 0.0 0.00 2.0 3

1 1.0 2.0 3.00 0.0 8

2 6.0 2.3 8.00 0.0 3

3 2.0 3.0 2.00 3.0 2

4 3.2 8.9 3.00 4.5 3

5 2.0 3.0 0.45 0.0 8

Dropna过滤NaN数据

series\_23=pd.Series([1,4,3,np.nan,4.5,None,3.4,4.5,6,np.nan,4,5,3])

series\_23

Out[20]:

0 1.0

1 4.0

2 3.0

3 NaN

4 4.5

5 NaN

6 3.4

7 4.5

8 6.0

9 NaN

10 4.0

11 5.0

12 3.0

dtype: float64

series\_23.dropna()

Out[21]:

0 1.0

1 4.0

2 3.0

4 4.5

6 3.4

7 4.5

8 6.0

10 4.0

11 5.0

12 3.0

dtype: float64

也可以通过Notnull

series\_23[series\_23.notnull()]

Out[22]:

0 1.0

1 4.0

2 3.0

4 4.5

6 3.4

7 4.5

8 6.0

10 4.0

11 5.0

12 3.0

dtype: float64

Dropna 用于DataFrame和用于Series稍有不同。用于DataFrame，含有NaN值的行会被删去。

frame\_123=pd.DataFrame([[1,np.nan,np.nan,2,3],[1,2,3,None,8],[6,2.3,8,None,3],[2,3,2,3,2],[3.2,8.9,3,4.5,3],[2,3,0.45,None,8]])

frame\_123

Out[28]:

0 1 2 3 4

0 1.0 NaN NaN 2.0 3

1 1.0 2.0 3.00 NaN 8

2 6.0 2.3 8.00 NaN 3

3 2.0 3.0 2.00 3.0 2

4 3.2 8.9 3.00 4.5 3

5 2.0 3.0 0.45 NaN 8

frame\_123.dropna()

Out[27]:

0 1 2 3 4

3 2.0 3.0 2.0 3.0 2

4 3.2 8.9 3.0 4.5 3

使用how=’all’只会消除全为NaN的行：

matrix\_2=frame\_123.dropna(how='all')

matrix\_2

Out[32]:

0 1 2 3 4

0 1.0 NaN NaN 2.0 3

1 1.0 2.0 3.00 NaN 8

2 6.0 2.3 8.00 NaN 3

3 2.0 3.0 2.00 3.0 2

4 3.2 8.9 3.00 4.5 3

5 2.0 3.0 0.45 NaN 8

列行的添加和丢弃

frame\_123

Out[47]:

0 1 2 3 4

0 1.0 NaN NaN 2.0 3

1 1.0 2.0 3.00 NaN 8

2 6.0 2.3 8.00 NaN 3

3 2.0 3.0 2.00 3.0 2

4 3.2 8.9 3.00 4.5 3

5 2.0 3.0 0.45 NaN 8

frame\_123[5]=[1,3,56,32,np.nan,0.7]

frame\_123

Out[49]:

0 1 2 3 4 5

0 1.0 NaN NaN 2.0 3 1.0

1 1.0 2.0 3.00 NaN 8 3.0

2 6.0 2.3 8.00 NaN 3 56.0

3 2.0 3.0 2.00 3.0 2 32.0

4 3.2 8.9 3.00 4.5 3 NaN

5 2.0 3.0 0.45 NaN 8 0.7

#列的添加

frame\_123

Out[47]:

0 1 2 3 4

0 1.0 NaN NaN 2.0 3

1 1.0 2.0 3.00 NaN 8

2 6.0 2.3 8.00 NaN 3

3 2.0 3.0 2.00 3.0 2

4 3.2 8.9 3.00 4.5 3

5 2.0 3.0 0.45 NaN 8

frame\_123.loc[6,:]=[np.nan,2,2.4,5,6]

frame\_123

Out[53]:

0 1 2 3 4

0 1.0 NaN NaN 2.0 3.0

1 1.0 2.0 3.00 NaN 8.0

2 6.0 2.3 8.00 NaN 3.0

3 2.0 3.0 2.00 3.0 2.0

4 3.2 8.9 3.00 4.5 3.0

5 2.0 3.0 0.45 NaN 8.0

6 NaN 2.0 2.40 5.0 6.0

#行的添加

frame\_123

Out[55]:

0 1 2 3 4

0 1.0 NaN NaN 2.0 3

1 1.0 2.0 3.00 NaN 8

2 6.0 2.3 8.00 NaN 3

3 2.0 3.0 2.00 3.0 2

4 3.2 8.9 3.00 4.5 3

5 2.0 3.0 0.45 NaN 8

frame\_123[5]=np.nan

frame\_123

Out[57]:

0 1 2 3 4 5

0 1.0 NaN NaN 2.0 3 NaN

1 1.0 2.0 3.00 NaN 8 NaN

2 6.0 2.3 8.00 NaN 3 NaN

3 2.0 3.0 2.00 3.0 2 NaN

4 3.2 8.9 3.00 4.5 3 NaN

5 2.0 3.0 0.45 NaN 8 NaN

frame\_123[5]=np.nan

frame\_123

Out[57]:

0 1 2 3 4 5

0 1.0 NaN NaN 2.0 3 NaN

1 1.0 2.0 3.00 NaN 8 NaN

2 6.0 2.3 8.00 NaN 3 NaN

3 2.0 3.0 2.00 3.0 2 NaN

4 3.2 8.9 3.00 4.5 3 NaN

5 2.0 3.0 0.45 NaN 8 NaN

frame\_123.dropna(axis=1,how='all')

Out[60]:

0 1 2 3 4

0 1.0 NaN NaN 2.0 3

1 1.0 2.0 3.00 NaN 8

2 6.0 2.3 8.00 NaN 3

3 2.0 3.0 2.00 3.0 2

4 3.2 8.9 3.00 4.5 3

5 2.0 3.0 0.45 NaN 8

frame\_1=pd.DataFrame(np.random.randn(7,4))

frame\_1

Out[65]:

0 1 2 3

0 -0.668146 0.034772 0.482339 1.444138

1 -1.167959 -0.703595 0.641404 -1.100771

2 -1.657068 -2.038607 0.141572 2.525831

3 -1.869547 -0.291923 0.275511 -0.459739

4 -0.287525 -0.966589 2.145633 0.703735

5 0.499207 -0.385792 -1.192131 -1.679805

6 -0.529885 2.053872 0.970785 -0.733382

frame\_1.loc[:4,1]=np.nan;frame\_1.loc[:2,2]=np.nan

frame\_1

Out[67]:

0 1 2 3

0 -0.668146 NaN NaN 1.444138

1 -1.167959 NaN NaN -1.100771

2 -1.657068 NaN NaN 2.525831

3 -1.869547 NaN 0.275511 -0.459739

4 -0.287525 NaN 2.145633 0.703735

5 0.499207 -0.385792 -1.192131 -1.679805

6 -0.529885 2.053872 0.970785 -0.733382

[:4,1]注意这里切片是切到“4”，不是到“3”

frame\_1.dropna()

Out[72]:

0 1 2 3

5 0.499207 -0.385792 -1.192131 -1.679805

6 -0.529885 2.053872 0.970785 -0.733382

#把不想看的数据行部分赋值NaN，然后在通过dropna语句把含有NaN的行删除掉。

缺失数据的补充：

通过fillna指令我们可以更快地补充缺失数据。这与通过isnull把数据框转化为bool数据框，然后把bool数据框作为索引来赋值‘0’要快的多。（前面已经讲过）

但是这个指令只能赋一个值。通过字典，我们可以给每一列赋不同的值。

frame\_1.fillna({1:5,2:4})

Out[7]:

0 1 2 3

0 1.651360 5.000000 4.000000 -0.600997

1 -0.020649 5.000000 4.000000 -0.529843

2 -0.850476 5.000000 4.000000 -0.893813

3 0.812226 5.000000 0.614715 0.112509

4 0.074320 5.000000 1.145727 0.168718

5 -0.516371 -1.022050 0.774645 0.705847

6 0.617606 0.881843 -0.619870 -1.527961

\_=frame\_1.fillna(0,inplace=True)

frame\_1

Out[13]:

0 1 2 3

0 1.651360 0.000000 0.000000 -0.600997

1 -0.020649 0.000000 0.000000 -0.529843

2 -0.850476 0.000000 0.000000 -0.893813

3 0.812226 0.000000 0.614715 0.112509

4 0.074320 0.000000 1.145727 0.168718

5 -0.516371 -1.022050 0.774645 0.705847

6 0.617606 0.881843 -0.619870 -1.527961

#可以直接对数据框修改，也就是说可以直接改变源数据，不产生副本。

frame\_1=pd.DataFrame(np.random.randn(7,4));frame\_1.loc[:4,1]=np.nan;frame\_1.loc[:2,2]=np.nan

frame\_1

Out[107]:

0 1 2 3

0 -0.748195 NaN NaN -0.539246

1 -2.360077 NaN NaN -0.061078

2 -1.743350 NaN NaN -0.233060

3 -2.100371 NaN 0.187509 0.893321

4 -0.477253 NaN 1.876733 0.208389

5 0.770411 -2.262797 0.321646 -2.490215

6 1.265209 -0.453140 0.045817 0.951077

frame\_1.fillna({1:pd.Series([2,3,7,9,0],index=[0,1,2,3,4]),2:pd.Series([1,9,8],index=[0,1,2])})

Out[108]:

0 1 2 3

0 -0.748195 2.000000 1.000000 -0.539246

1 -2.360077 3.000000 9.000000 -0.061078

2 -1.743350 7.000000 8.000000 -0.233060

3 -2.100371 9.000000 0.187509 0.893321

4 -0.477253 0.000000 1.876733 0.208389

5 0.770411 -2.262797 0.321646 -2.490215

6 1.265209 -0.453140 0.045817 0.951077

层次化索引：

series\_1=pd.Series(np.random.randn(12),index=[['r1','r1','r1','t1','t1','s1','s1','s1','s1','p1','q1','q1'],[9,8,7,6,5,4,3,2,1,0,12,11]])

series\_1

Out[8]:

r1 9 1.536272

8 0.378965

7 -0.292986

t1 6 -0.254990

5 0.765191

s1 4 2.204928

3 0.662662

2 -0.595029

1 -1.905753

p1 0 -1.029524

q1 12 -1.038748

11 -1.192589

dtype: float64

series\_1.index

Out[9]:

MultiIndex(levels=[['p1', 'q1', 'r1', 's1', 't1'], [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12]],

labels=[[2, 2, 2, 4, 4, 3, 3, 3, 3, 0, 1, 1], [9, 8, 7, 6, 5, 4, 3, 2, 1, 0, 11, 10]])

可以直接索引外层标签，也就是第一层标签。

series\_1['s1']

Out[6]:

4 -0.012306

3 0.800679

2 -0.862986

1 0.048458

dtype: float64

**也可以做切片。但是外层标签必须是**

1. **经过排序的，必须！**
2. **每个外层标签的第一个字母不能不一致，要大写，全大写，要小写全小写。**

Series\_123=pd.Series(np.random.randint(10),index=[['apple','apple','apple','apricot','apricot','apricot','banana','banana','blackberry','cherry'],[3,2,5,6,0,4,9,11,21,17]])

Series\_123['apple']

Out[4]:

3 9

2 9

5 9

dtype: int64

Series\_123['apple':'banana']

Out[64]:

apple 3 5

2 5

5 5

apricot 6 5

0 5

4 5

banana 9 5

11 5

dtype: int64

Series\_123['apple':'apricot']

Out[65]:

apple 3 5

2 5

5 5

apricot 6 5

0 5

4 5

dtype: int64

Series\_123['apple':'blackberry']

Out[67]:

apple 3 5

2 5

5 5

apricot 6 5

0 5

4 5

banana 9 5

11 5

blackberry 21 5

dtype: int64

Series\_67=pd.Series(np.random.randint(10),index=[['apple','apple','apple','apricot','apricot','apricot','banana','banana','blackberry','cherry'],[3,2,5,6,5,4,9,5,21,17]])

Series\_67

Out[7]:

apple 3 4

2 4

5 4

apricot 6 4

5 4

4 4

banana 9 4

5 4

blackberry 21 4

cherry 17 4

dtype: int64

Series\_67[:,5]

Out[8]:

apple 4

apricot 4

banana 4

dtype: int64

#对于Series，中括号的第一个位置是代表外层标签，第二位置代表内存标签

例子：

Series\_67['apple',5]

Out[10]: 4

带有层次化索引的Series可以立马转化为数据框，反之，数据框也可以转化为带有层次化索引的Series

Series\_67.unstack()

Out[20]:

2 3 4 5 6 9 17 21

apple 4.0 4.0 NaN 4.0 NaN NaN NaN NaN

apricot NaN NaN 4.0 4.0 4.0 NaN NaN NaN

banana NaN NaN NaN 4.0 NaN 4.0 NaN NaN

blackberry NaN NaN NaN NaN NaN NaN NaN 4.0

cherry NaN NaN NaN NaN NaN NaN 4.0 NaN

#Series的内层索引变化为列标签，外层索引转化为行标签,并且重复索引删除，索引保持唯一性。

Series\_67.unstack(0)

Out[22]:

apple apricot banana blackberry cherry

2 4.0 NaN NaN NaN NaN

3 4.0 NaN NaN NaN NaN

4 NaN 4.0 NaN NaN NaN

5 4.0 4.0 4.0 NaN NaN

6 NaN 4.0 NaN NaN NaN

9 NaN NaN 4.0 NaN NaN

17 NaN NaN NaN NaN 4.0

21 NaN NaN NaN 4.0 NaN

#axis=0时，Series的内层索引变化为行标签，外层索引转化为列标签

Series\_67.unstack(1)

Out[24]:

2 3 4 5 6 9 17 21

apple 4.0 4.0 NaN 4.0 NaN NaN NaN NaN

apricot NaN NaN 4.0 4.0 4.0 NaN NaN NaN

banana NaN NaN NaN 4.0 NaN 4.0 NaN NaN

blackberry NaN NaN NaN NaN NaN NaN NaN 4.0

cherry NaN NaN NaN NaN NaN NaN 4.0 NaN

Series\_67.unstack().stack()

Out[29]:

apple 2 4.0

3 4.0

5 4.0

apricot 4 4.0

5 4.0

6 4.0

banana 5 4.0

9 4.0

blackberry 21 4.0

cherry 17 4.0

dtype: float64

Gamedata=pd.DataFrame(np.array([[31,27,24,60],[26,28,13,29],[27,17,12,27],[29,9,5,18],[31,12,4,70],[45,11,12,12]]),index=[['Morning','Morning','Morning','Afternoon','Afternoon','Afternoon'],[1,2,3,1,2,3]],columns=[['Junior','Junior','Youth','Youth'],['Zhang','Wang','Li','Zhao']])

Gamedata

Out[35]:

Junior Youth

Zhang Wang Li Zhao

Morning 1 31 27 24 60

2 26 28 13 29

3 27 17 12 27

Afternoon 1 29 9 5 18

2 31 12 4 70

3 45 11 12 12

Gamedata=pd.DataFrame(np.array([[31,27,24,60],[26,28,13,29],[27,17,12,27],[29,9,5,18],[31,12,4,70],[45,11,12,12]]),index=[['A\_Morning','A\_Morning','A\_Morning','B\_Afternoon','B\_Afternoon','B\_Afternoon'],[1,2,3,1,2,3]],columns=[['Junior','Junior','Youth','Youth'],['Zhang','Wang','Li','Zhao']])

Gamedata.index.names=['time','sequence']

#因为行有双层索引，因此需要两个名字

#列也是如此，也是两个名字

Gamedata.columns.names=['age','name']

Gamedata

Out[50]:

age Junior Youth

name Zhang Wang Li Zhao

time sequence

A\_Morning 1 31 27 24 60

2 26 28 13 29

3 27 17 12 27

B\_Afternoon 1 29 9 5 18

2 31 12 4 70

3 45 11 12 12

Gamedata['Junior']

Out[51]:

name Zhang Wang

time sequence

A\_Morning 1 31 27

2 26 28

3 27 17

B\_Afternoon 1 29 9

2 31 12

3 45 11

Gamedata.loc['Morning']

Out[31]:

Age Junior Youth

Name Zhang Wang Li Zhao

series\_nr

1 31 27 24 60

2 26 28 13 29

3 27 17 12 27

Gamedata.loc['Morning',2]

Out[32]:

Age Name

Junior Zhang 26

Wang 28

Youth Li 13

Zhao 29

Name: (Morning, 2), dtype: int32

多重标签的顺序互换：

Gamedata.swaplevel('sequence','time')

Out[52]:

age Junior Youth

name Zhang Wang Li Zhao

sequence time

1 A\_Morning 31 27 24 60

2 A\_Morning 26 28 13 29

3 A\_Morning 27 17 12 27

1 B\_Afternoon 29 9 5 18

2 B\_Afternoon 31 12 4 70

3 B\_Afternoon 45 11 12 12

Gamedata.swaplevel('age','name',axis=1)

Out[60]:

name Zhang Wang Li Zhao

age Junior Junior Youth Youth

time sequence

A\_Morning 1 31 27 24 60

2 26 28 13 29

3 27 17 12 27

B\_Afternoon 1 29 9 5 18

2 31 12 4 70

3 45 11 12 12

Gamedata.sortlevel(1)

C:\Users\dongfeng\Anaconda3\lib\site-packages\ipykernel\_launcher.py:1: FutureWarning: sortlevel is deprecated, use sort\_index(level= ...)

"""Entry point for launching an IPython kernel.

Out[55]:

age Junior Youth

name Zhang Wang Li Zhao

time sequence

A\_Morning 1 31 27 24 60

B\_Afternoon 1 29 9 5 18

A\_Morning 2 26 28 13 29

B\_Afternoon 2 31 12 4 70

A\_Morning 3 27 17 12 27

B\_Afternoon 3 45 11 12 12

#sortlevel则根据单个级别的值进行排序，上例根据sequence列数据。

Gamedata.swaplevel('age','name',axis=1).sortlevel(0)

C:\Users\dongfeng\Anaconda3\lib\site-packages\ipykernel\_launcher.py:1: FutureWarning: sortlevel is deprecated, use sort\_index(level= ...)

"""Entry point for launching an IPython kernel.

Out[61]:

name Zhang Wang Li Zhao

age Junior Junior Youth Youth

time sequence

A\_Morning 1 31 27 24 60

2 26 28 13 29

3 27 17 12 27

B\_Afternoon 1 29 9 5 18

2 31 12 4 70

3 45 11 12 12

#交换两层列标签，然后对第二行列标签排序

Gamedata

Out[62]:

age Junior Youth

name Zhang Wang Li Zhao

time sequence

A\_Morning 1 31 27 24 60

2 26 28 13 29

3 27 17 12 27

B\_Afternoon 1 29 9 5 18

2 31 12 4 70

3 45 11 12 12

Gamedata.swaplevel('age','name',axis=1).sortlevel(1)

C:\Users\dongfeng\Anaconda3\lib\site-packages\ipykernel\_launcher.py:1: FutureWarning: sortlevel is deprecated, use sort\_index(level= ...)

"""Entry point for launching an IPython kernel.

Out[63]:

name Zhang Wang Li Zhao

age Junior Junior Youth Youth

time sequence

A\_Morning 1 31 27 24 60

B\_Afternoon 1 29 9 5 18

A\_Morning 2 26 28 13 29

B\_Afternoon 2 31 12 4 70

A\_Morning 3 27 17 12 27

B\_Afternoon 3 45 11 12 12

#交换两层行标签，然后对第二列行标签排序

根据级别计算：

Gamedata

Out[62]:

age Junior Youth

name Zhang Wang Li Zhao

time sequence

A\_Morning 1 31 27 24 60

2 26 28 13 29

3 27 17 12 27

B\_Afternoon 1 29 9 5 18

2 31 12 4 70

3 45 11 12 12

Gamedata.sum(level='sequence')

Out[64]:

age Junior Youth

name Zhang Wang Li Zhao

sequence

1 60 36 29 78

2 57 40 17 99

3 72 28 24 39

#每个级别包含两个元素，（因为此级别标签分别重复两次。）

Gamedata.sum(level='name',axis=1)

Out[68]:

name Li Wang Zhang Zhao

time sequence

A\_Morning 1 24 27 31 60

2 13 28 26 29

3 12 17 27 27

B\_Afternoon 1 5 9 29 18

2 4 12 31 70

3 12 11 45 12

每个级别只包含一个元素，因此无法相加，只能排序。

Gamedata.sum(level='age',axis=1)

Out[69]:

age Junior Youth

time sequence

A\_Morning 1 58 84

2 54 42

3 44 39

B\_Afternoon 1 38 23

2 43 74

3 56 24

#每个级别包含两个元素，分别两两相加。

Gamedata.sum(level='time')

Out[71]:

age Junior Youth

name Zhang Wang Li Zhao

time

A\_Morning 84 72 49 116

B\_Afternoon 105 32 21 100

每个级别包含三个元素，分别相加即可

把数据框的列转化为行索引：

frame\_1123=pd.DataFrame({'a':range(4),'b':range(4,0,-1),'c':['one','one','two','two'],'d':[0,1,2,3]})

frame\_1123

Out[73]:

a b c d

0 0 4 one 0

1 1 3 one 1

2 2 2 two 2

3 3 1 two 3

frame\_1224=frame\_1123.set\_index(['c','d'])

frame\_1224

Out[76]:

a b

c d

one 0 0 4

1 1 3

two 2 2 2

3 3 1

#我们发现列从数据框中消失，变成了双重行索引。

然而而这些列也可以不消失。

frame\_1123.set\_index(['c','d'],drop=False)

Out[79]:

a b c d

c d

one 0 0 4 one 0

1 1 3 one 1

two 2 2 2 two 2

3 3 1 two 3

我们可以把双重索引再转化到数据框的数据区域。

frame\_1123.reset\_index()

Out[82]:

index a b c d

0 0 0 4 one 0

1 1 1 3 one 1

2 2 2 2 two 2

3 3 3 1 two 3

面板数据：

Panel实质上是一个三维的数据框，3维说明其有三个轴，每个轴的含义如下：

* **items:** 0轴， 每个项目对应其中的一个DataFrame
* \*\*major\_axis（主轴）: \*\* 1轴，它是每个DataFrame的index
* \*\*minor\_axis（副轴）: \*\* 2轴，它是每个DataFrame的column

创建panel：

1. 数组创建,三维数组

Dataframe\_3 = pd.Panel(np.random.randn(2,5,4),items=['Item1','Item2'],major\_axis=pd.date\_range('2000-01-01','2000-01-05'),minor\_axis=['A','B','C','D'])

Dataframe\_4=pd.Panel(np.array([[[1,4,67,45],[34,56,2,0],[78,90,3,4],[4,23,67,5],[34,89,67,1]],[[67,90,64,7],[789,345,6,2],[33,89,467,8],[43,93,2,8],[33,74,89,6]]]),items=['Item1','Item2'],major\_axis=pd.date\_range('2000-01-01','2000-01-05'),minor\_axis=['A','B','C','D'])

Dataframe\_3

Out[107]:

<class 'pandas.core.panel.Panel'>

Dimensions: 2 (items) x 5 (major\_axis) x 4 (minor\_axis)

Items axis: Item1 to Item2

Major\_axis axis: 2000-01-01 00:00:00 to 2000-01-05 00:00:00

Minor\_axis axis: A to D

Dataframe\_4

Out[108]:

<class 'pandas.core.panel.Panel'>

Dimensions: 2 (items) x 5 (major\_axis) x 4 (minor\_axis)

Items axis: Item1 to Item2

Major\_axis axis: 2000-01-01 00:00:00 to 2000-01-05 00:00:00

Minor\_axis axis: A to D

1. 通过字典创建

data = {'Item1' : pd.DataFrame(np.random.randn(4, 3),index=pd.date\_range('2017-09-05','2017-09-08'),columns=['X','D','F']),'Item2' : pd.DataFrame(np.random.randn(4, 2),index=pd.date\_range('2017-09-05','2017-09-08'),columns=['X','D'])}

panel\_1=pd.Panel(data)

panel\_1

Out[115]:

<class 'pandas.core.panel.Panel'>

Dimensions: 2 (items) x 4 (major\_axis) x 3 (minor\_axis)

Items axis: Item1 to Item2

Major\_axis axis: 2017-09-05 00:00:00 to 2017-09-08 00:00:00

Minor\_axis axis: D to X

1. 通过数据框创建

midx = pd.MultiIndex(levels=[['one', 'two'], ['x','y']], labels=[[1,1,0,0],[1,0,1,0]])

midx

Out[117]:

MultiIndex(levels=[['one', 'two'], ['x', 'y']],

labels=[[1, 1, 0, 0], [1, 0, 1, 0]])

第一层行标是['one', 'two']，labels[1, 1, 0, 0]；第二层行标是 ['x','y']，labels[1, 0, 1, 0]，注意’one’对应‘0’，’two’对应‘1’

df = pd.DataFrame({'A':[1,2,3,4],'B':[5,6,7,8]},index=midx)

df

Out[119]:

A B

two y 1 5

x 2 6

one y 3 7

x 4 8

df.to\_panel()

面板操作

1. 选取

Dataframe\_3['Item1']

Out[124]: Dataframe\_3

A B C D

2000-01-01 0.836830 -1.856472 0.345340 0.576771

2000-01-02 -1.529758 -1.646630 0.635996 -0.337408

2000-01-03 0.451765 0.156648 -1.225328 -0.177641

2000-01-04 1.123645 0.364549 0.684536 1.558884

2000-01-05 -0.082263 1.472391 -0.379373 2.410845

Dataframe\_3.major\_axis

Out[128]:

DatetimeIndex(['2000-01-01', '2000-01-02', '2000-01-03', '2000-01-04',

'2000-01-05'],

dtype='datetime64[ns]', freq='D')

Dataframe\_3['Item1']

Out[131]:

A B C D

2000-01-01 0.836830 -1.856472 0.345340 0.576771

2000-01-02 -1.529758 -1.646630 0.635996 -0.337408

2000-01-03 0.451765 0.156648 -1.225328 -0.177641

2000-01-04 1.123645 0.364549 0.684536 1.558884

2000-01-05 -0.082263 1.472391 -0.379373 2.410845

Dataframe\_3.major\_xs(Dataframe\_3.major\_axis[2])

Dataframe\_3.minor\_axis

Out[134]: Index(['A', 'B', 'C', 'D'], dtype='object')

Dataframe\_3.minor\_xs(Dataframe\_3.minor\_axis[3])

Out[137]:

Item1 Item2

2000-01-01 0.576771 1.387176

2000-01-02 -0.337408 -0.159394

2000-01-03 -0.177641 -1.622084

2000-01-04 1.558884 0.393800

2000-01-05 2.410845 -0.249931

转置：

Dataframe\_3=pd.Panel(np.random.randn(2,5,4),items=['Item1','Item2'],major\_axis=pd.date\_range('2000-01-01','2000-01-05'),minor\_axis=['A','B','C','D'])

Dataframe\_3

Out[5]:

<class 'pandas.core.panel.Panel'>

Dimensions: 2 (items) x 5 (major\_axis) x 4 (minor\_axis)

Items axis: Item1 to Item2

Major\_axis axis: 2000-01-01 00:00:00 to 2000-01-05 00:00:00

Minor\_axis axis: A to D

Dataframe\_3['Item1','2000-01-03','B']

Out[9]: 2.3125077493899666

#转置前的元素定位查询

Dataframe\_3.transpose(0,2,1)

Out[10]:

<class 'pandas.core.panel.Panel'>

Dimensions: 2 (items) x 4 (major\_axis) x 5 (minor\_axis)

Items axis: Item1 to Item2

Major\_axis axis: A to D

Minor\_axis axis: 2000-01-01 00:00:00 to 2000-01-05 00:00:00

Data\_transpose=Dataframe\_3.transpose(0,2,1)

Data\_transpose['Item1','B','2000-01-03']

Out[12]: 2.3125077493899666

#转置后的同一元素查询，两者查询结果相同，说明转置成功。

面板转化为分层索引数据框

Dataframe\_3.to\_frame()

Out[16]:

Item1 Item2

major minor

2000-01-01 A 1.345174 -1.719012

B -1.075240 1.258681

C 0.366470 -0.004046

D -0.149155 -0.405295

2000-01-02 A 0.005256 0.204166

B -0.032269 -0.667655

C 1.825649 1.050139

D -1.505179 0.534874

2000-01-03 A 0.876495 -0.414982

B 2.312508 -0.731893

C -0.129701 -1.470191

D -0.637856 -0.083188

2000-01-04 A -0.287276 0.633456

B -0.623472 -0.229308

C 0.530747 -0.896306

D 0.229674 0.288064

2000-01-05 A 0.163915 -1.836235

B 0.618230 1.353955

C 0.260995 0.808007

D 1.673127 -0.275785

切片：

Dataframe\_3['Item2',:,'B']

Out[18]:

2000-01-01 1.258681

2000-01-02 -0.667655

2000-01-03 -0.731893

2000-01-04 -0.229308

2000-01-05 1.353955

Freq: D, Name: B, dtype: float64

Dataframe\_3['Item2','2000-01-02':'2000-01-04','B']

Out[19]:

2000-01-02 -0.667655

2000-01-03 -0.731893

2000-01-04 -0.229308

Freq: D, Name: B, dtype: float64

Dataframe\_3['Item2','2000-01-02']

Out[20]:

A 0.204166

B -0.667655

C 1.050139

D 0.534874

Name: 2000-01-02 00:00:00, dtype: float64

Dataframe\_3['Item2','2000-01-02':'2000-01-04']

Out[21]:

A B C D

2000-01-02 0.204166 -0.667655 1.050139 0.534874

2000-01-03 -0.414982 -0.731893 -1.470191 -0.083188

2000-01-04 0.633456 -0.229308 -0.896306 0.288064

Dataframe\_3['Item1':'Item2','2000-01-02':'2000-01-04','C']

Out[22]:

Item1 Item2

2000-01-02 1.825649 1.050139

2000-01-03 -0.129701 -1.470191

2000-01-04 0.530747 -0.896306

Dataframe\_3[:,'2000-01-02':'2000-01-04','C':'D']

Out[28]:

<class 'pandas.core.panel.Panel'>

Dimensions: 2 (items) x 3 (major\_axis) x 2 (minor\_axis)

Items axis: Item1 to Item2

Major\_axis axis: 2000-01-02 00:00:00 to 2000-01-04 00:00:00

Minor\_axis axis: C to D

#切出新面板。

# 数据加载，存储，清理，转换，合并与重塑

数据的加载与存储

Python在文本文件的加载与存储方面极其方便，这是它成为深受大家喜爱语言的原因之一。

Pandas 提供了一些直接将表格文件读取为DataFrame对象的函数

下面我们一一讲解：

A, read\_csv

import pandas as pd

!type example\_1.txt

系统找不到指定的文件。我们做一下简单修改

!type Desktop\example\_1.txt

Liu,Zhang,Wang,Li,Class

23,34,78,32,'primary'

77,32,89,66,'intermediate'

99,34,78,66,'senior'

66,34,6,33,'intermediate'

也可以直接读取‘CSV’文件

!type Desktop\example\_1.csv

Liu,Zhang,Wang,Li,Class

23,34,78,32,'primary'

77,32,89,66,'intermediate'

99,34,78,66,'senior'

66,34,6,33,'intermediate'

直接读取的数据相当于源数据，即没有转化为数据框的数据。

下面我们直接把CSV文件读成数据框：

frame\_1=pd.read\_csv('Desktop\example\_1.csv')

frame\_1

Out[10]:

Liu Zhang Wang Li Class

0 23 34 78 32 'primary'

1 77 32 89 66 'intermediate'

2 99 34 78 66 'senior'

3 66 34 6 33 'intermediate'

#我们发现我们得到一个完美的数据框

下面我们尝试用‘read\_table’

frame\_1=pd.read\_table('Desktop\example\_1.csv')

frame\_1

Out[15]:

Liu,Zhang,Wang,Li,Class

0 23,34,78,32,'primary'

1 77,32,89,66,'intermediate'

2 99,34,78,66,'senior'

3 66,34,6,33,'intermediate'

我们发现被读成的DataFrame排列很混乱，下面我们加上分割符号‘，’再运行一次。

frame\_1=pd.read\_table('Desktop\example\_1.csv',sep=',')

frame\_1

Out[13]:

Liu Zhang Wang Li Class

0 23 34 78 32 'primary'

1 77 32 89 66 'intermediate'

2 99 34 78 66 'senior'

3 66 34 6 33 'intermediate'

总结：用read\_table一定要加分割符号。

上面的例子中的csv数据我们都加了列名，如果没有列名会怎么样呢?

frame\_2=pd.read\_table('Desktop\example\_2.csv',sep=',')

frame\_2

Out[19]:

23 34 78 32 'primary'

0 77 32 89 66 'intermediate'

1 99 34 78 66 'senior'

2 66 34 6 33 'intermediate'

我们发现计算机自动把第一行数据当做列名，这样的话我们的源数据就遭到破坏。

为了避免这种错误，我们加入‘header’属性。让计算机自动加列名或者自己显性指定：

frame\_3=pd.read\_csv('Desktop\example\_2.csv',header=None)

frame\_3

Out[21]:

0 1 2 3 4

0 23 34 78 32 'primary'

1 77 32 89 66 'intermediate'

2 99 34 78 66 'senior'

3 66 34 6 33 'intermediate'

frame\_3=pd.read\_csv('Desktop\example\_2.csv',names=['a','b','c','d','class'])

frame\_3

Out[23]:

a b c d class

0 23 34 78 32 'primary'

1 77 32 89 66 'intermediate'

2 99 34 78 66 'senior'

3 66 34 6 33 'intermediate'

源文件中去掉双引号，这里双引号也会消失！

我们可以把最右边的列直接转化成行索引列。

frame\_3=pd.read\_csv('Desktop\example\_2.csv',names=['a','b','c','d','class'],index\_col='class')

frame\_3

Out[28]:

a b c d

class

'primary' 23 34 78 32

'intermediate' 77 32 89 66

'senior' 99 34 78 66

'intermediate' 66 34 6 33

除了可以读取为普通的数据框，还可以读取为带有重索引的数据框：

pd.read\_csv('Desktop\example\_5.csv',index\_col=['group','games'])

Out[45]:

a b c a.1 b.1 c.1 a.2 b.2 c.2

group games

one Att 4 3 4 2 4 3 4 2 1

Cgt 4 3 4 2 4 3 4 2 1

Aer 4 3 4 2 4 3 4 2 1

two Att 4 3 4 2 4 3 4 2 1

Cgt 4 3 4 2 4 3 4 2 1

Aer 4 3 4 2 4 3 4 2 1

对于那些用数量不等空格或者字符串隔开的数据，我们可以通过正则操作符一步处理到位，把数据处理整齐：

!type Desktop\exampel\_6.csv

A B C'

aaa -3.45 2.36 8.90'

bbb 0.334 0.457 -4.5'

ccc 0.76 -7.34 -8.99'

ddd 0.37 -7.8 -4.45'

pd.read\_csv('Desktop\exampel\_6.csv',sep='\s+')

Out[63]:

A B C'

aaa -3.450 2.360 8.90'

bbb 0.334 0.457 -4.5'

ccc 0.760 -7.340 -8.99'

ddd 0.370 -7.800 -4.45'

如果这里的源数据没有引号的话，这里引号也会自动消失。

利用skiprows语句可以跳过任意无意义行信息。

!type Desktop\example\_9.csv

title:Good Boy

Little Robert asked his mother for two cents. "What did you do with the money I gave you yesterday?"

a,b,c,d,name

"I gave it to a poor old woman," he answered.

"You're a good boy," said the mother proudly. "Here are two cents more. But why are you so interested in the old woman?"

group1,12,2,2,3,robert

group2,10,19,18,17,linz

"She is the one who sells the candy."

group3,29,28,27,27,hans

group4,34,35,35,36,manfried

整理后数据

pd.read\_csv('Desktop\example\_9.csv',skiprows=[0,1,3,4,7])

Out[69]:

a b c d name

group1 12 2 2 3 robert

group2 10 19 18 17 linz

group3 29 28 27 27 hans

group4 34 35 35 36 manfried

缺失数据要么没有，要么用某个标记值表示。请看下面两个例子：

!type Desktop\example\_10.csv

Liu,Zhang,Wang,Li,Class

23,34,78,32,NA

77,32,,66,'intermediate'

99,34,78,66,'senior'

66,34,6,33,'intermediate'

pd.read\_csv('Desktop\example\_10.csv')

Out[83]:

Liu Zhang Wang Li Class

0 23 34 78.0 32 NaN

1 77 32 NaN 66 'intermediate'

2 99 34 78.0 66 'senior'

3 66 34 6.0 33 'intermediate'

#上面例子产生NAN值，（直接在数字处空一位，就会产生NaN，比如数据1,2,4对应列标签，A,B,C,D）

!type Desktop\example\_11.csv

23,34,78,32,' '

77,32,89,66,'intermediate'

99,34, ,66,'senior'

66,34,6,33,'intermediate'

pd.read\_csv('Desktop\example\_11.csv')

Out[86]:

23 34 78 32 ' '

0 77 32 89 66 'intermediate'

1 99 34 66 'senior'

2 66 34 6 33 'intermediate'

#上面的例子什么都不显示（只有空字符串和空格）

通过na\_values python可以用字符串标识NAN值。但要注意源数据中字符串上不能有引号。

!type Desktop\example\_12.csv

A,B,C,D

1,2,3,4

5,me,7,8

8,10,11,me

12,13,14,16

pd.read\_table('Desktop\example\_12.csv',na\_values=['me'],sep=',')

Out[14]:

A B C D

0 1 2.0 3 4.0

1 5 NaN 7 8.0

2 8 10.0 11 NaN

3 12 13.0 14 16.0

可以用字符串字典的形式标识多空值数据框

!type Desktop\example\_13.csv

A,B,C,D

1,2,3,4

too,to,56,33

45,dee,78,69

12,21,34,43

pd.read\_table('Desktop\example\_13.csv',na\_values={'A':'too','B':['to','dee']},sep=',')

Out[18]:

A B C D

0 1.0 2.0 3 4

1 NaN NaN 56 33

2 45.0 NaN 78 69

3 12.0 21.0 34 43

爬虫数据例子：

df=pd.read\_csv('Desktop\pachong.csv',encoding='gbk',sep=',',header=None, skiprows=[14,47,49,50,51,59,68,83,104,125,127,128,136,150,173,184])

用这个指令可以处理中文数据

**Pd.read\_csv读取excel数据：**

import pandas as pd

pd.read\_excel(r'Desktop\1.xls' ,sheetname=[0,1])

Out[2]:

OrderedDict([(0, 1 2 3 4

0 2 3 4 5

1 3 4 5 6

2 4 5 6 7

3 5 6 7 8

4 6 7 8 9

5 7 8 9 10

6 8 9 10 11

7 9 10 11 12

8 10 11 12 13

9 11 12 13 14

10 12 13 14 15

11 13 14 15 16

12 14 15 16 17

13 15 16 17 18

14 16 17 18 19

15 17 18 19 20

16 18 19 20 21

17 19 20 21 22

18 20 21 22 23

19 21 22 23 24), (1, 23 22 34 55 67

0 24 23 35 56 68

1 25 24 36 57 69

2 26 25 37 58 70

3 27 26 38 59 71

4 28 27 39 60 72

5 29 28 40 61 73

6 30 29 41 62 74

7 31 30 42 63 75

8 32 31 43 64 76

9 33 32 44 65 77

10 34 33 45 66 78

11 35 34 46 67 79

12 36 35 47 68 80

13 37 36 48 69 81

14 38 37 49 70 82

15 39 38 50 71 83

16 40 39 51 72 84

17 41 40 52 73 85

18 42 41 53 74 86

19 43 42 54 75 87

20 44 43 55 76 88

21 45 44 56 77 89

22 46 45 57 78 90)])

#sheetname=表格时，返回多个指定的在一个workbook里面的表格，sheetname=none，将返回一个workbook里的所有表格。sheetname=int时，int指的是表格索引号，sheetname=’sheet1‘返回指定表格。

pd.read\_excel(r'Desktop\1.xls' ,sheetname=None)

Out[4]:

OrderedDict([('Sheet1', 1 2 3 4

0 2 3 4 5

1 3 4 5 6

2 4 5 6 7

3 5 6 7 8

4 6 7 8 9

5 7 8 9 10

6 8 9 10 11

7 9 10 11 12

8 10 11 12 13

9 11 12 13 14

10 12 13 14 15

11 13 14 15 16

12 14 15 16 17

13 15 16 17 18

14 16 17 18 19

15 17 18 19 20

16 18 19 20 21

17 19 20 21 22

18 20 21 22 23

19 21 22 23 24), ('Sheet2', 23 22 34 55 67

0 24 23 35 56 68

1 25 24 36 57 69

2 26 25 37 58 70

3 27 26 38 59 71

4 28 27 39 60 72

5 29 28 40 61 73

6 30 29 41 62 74

7 31 30 42 63 75

8 32 31 43 64 76

9 33 32 44 65 77

10 34 33 45 66 78

11 35 34 46 67 79

12 36 35 47 68 80

13 37 36 48 69 81

14 38 37 49 70 82

15 39 38 50 71 83

16 40 39 51 72 84

17 41 40 52 73 85

18 42 41 53 74 86

19 43 42 54 75 87

20 44 43 55 76 88

21 45 44 56 77 89

22 46 45 57 78 90), ('Sheet3', 234 22 44 78 990

0 235 23 45 79 991

1 236 24 46 80 992

2 237 25 47 81 993

3 238 26 48 82 994

4 239 27 49 83 995

5 240 28 50 84 996

6 241 29 51 85 997

7 242 30 52 86 998

8 243 31 53 87 999

9 244 32 54 88 1000

10 245 33 55 89 1001

11 246 34 56 90 1002

12 247 35 57 91 1003

13 248 36 58 92 1004

14 249 37 59 93 1005

15 250 38 60 94 1006

16 251 39 61 95 1007

17 252 40 62 96 1008

18 253 41 63 97 1009

19 254 42 64 98 1010

20 255 43 65 99 1011)])

pd.read\_excel(r'Desktop\1.xls' ,sheetname=2)

Out[5]:

234 22 44 78 990

0 235 23 45 79 991

1 236 24 46 80 992

2 237 25 47 81 993

3 238 26 48 82 994

4 239 27 49 83 995

5 240 28 50 84 996

6 241 29 51 85 997

7 242 30 52 86 998

8 243 31 53 87 999

9 244 32 54 88 1000

10 245 33 55 89 1001

11 246 34 56 90 1002

12 247 35 57 91 1003

13 248 36 58 92 1004

14 249 37 59 93 1005

15 250 38 60 94 1006

16 251 39 61 95 1007

17 252 40 62 96 1008

18 253 41 63 97 1009

19 254 42 64 98 1010

20 255 43 65 99 1011

pd.read\_excel(r'Desktop\1.xls' ,sheetname=0)

Out[6]:

1 2 3 4

0 2 3 4 5

1 3 4 5 6

2 4 5 6 7

3 5 6 7 8

4 6 7 8 9

5 7 8 9 10

6 8 9 10 11

7 9 10 11 12

8 10 11 12 13

9 11 12 13 14

10 12 13 14 15

11 13 14 15 16

12 14 15 16 17

13 15 16 17 18

14 16 17 18 19

15 17 18 19 20

16 18 19 20 21

17 19 20 21 22

18 20 21 22 23

19 21 22 23 24

#sheetname=整数时，整数表示表格编号，编号从0开始。

pd.read\_excel(r'Desktop\1.xls' ,sheetname='Sheet3')

Out[9]:

234 22 44 78 990

0 235 23 45 79 991

1 236 24 46 80 992

2 237 25 47 81 993

3 238 26 48 82 994

4 239 27 49 83 995

5 240 28 50 84 996

6 241 29 51 85 997

7 242 30 52 86 998

8 243 31 53 87 999

9 244 32 54 88 1000

10 245 33 55 89 1001

11 246 34 56 90 1002

12 247 35 57 91 1003

13 248 36 58 92 1004

14 249 37 59 93 1005

15 250 38 60 94 1006

16 251 39 61 95 1007

17 252 40 62 96 1008

18 253 41 63 97 1009

19 254 42 64 98 1010

20 255 43 65 99 1011

#上面是用字符串‘Sheet3’来调用表格三数据

#header : int, list of ints, default 0 指定列索引行，默认0，即取第一行，即数据第一行被自动读取为列索引，这一点我们并不希望，因为会丢失掉第一行数据，因此通常设定 header = None，这种情况下，计算机自动为整个数据添加新的列索引。

pd.read\_excel(r'Desktop\1.xls' ,sheetname='Sheet3',header=None)

Out[10]:

0 1 2 3 4

0 234 22 44 78 990

1 235 23 45 79 991

2 236 24 46 80 992

3 237 25 47 81 993

4 238 26 48 82 994

5 239 27 49 83 995

6 240 28 50 84 996

7 241 29 51 85 997

8 242 30 52 86 998

9 243 31 53 87 999

10 244 32 54 88 1000

11 245 33 55 89 1001

12 246 34 56 90 1002

13 247 35 57 91 1003

14 248 36 58 92 1004

15 249 37 59 93 1005

16 250 38 60 94 1006

17 251 39 61 95 1007

18 252 40 62 96 1008

19 253 41 63 97 1009

20 254 42 64 98 1010

21 255 43 65 99 1011

#上例中我们发现计算机没有破坏原数据，额外为我们添加了一行列索引。

#skiprows : list-like，跳过指定行数的数据

pd.read\_excel(r'Desktop\1.xls' ,sheetname='Sheet3',header=None,skiprows=[1,5,19,10])

Out[11]:

0 1 2 3 4

0 234 22 44 78 990

1 236 24 46 80 992

2 237 25 47 81 993

3 238 26 48 82 994

4 240 28 50 84 996

5 241 29 51 85 997

6 242 30 52 86 998

7 243 31 53 87 999

8 245 33 55 89 1001

9 246 34 56 90 1002

10 247 35 57 91 1003

11 248 36 58 92 1004

12 249 37 59 93 1005

13 250 38 60 94 1006

14 251 39 61 95 1007

15 252 40 62 96 1008

16 254 42 64 98 1010

17 255 43 65 99 1011

#我们发现数据少了4行，原因是我们分别跳过了4行不同行

#skip\_footer : int,default 0, 省略从尾部数的整数行数据

pd.read\_excel(r'Desktop\1.xls' ,sheetname='Sheet3',header=None,skip\_footer=8)

Out[13]:

0 1 2 3 4

0 234 22 44 78 990

1 235 23 45 79 991

2 236 24 46 80 992

3 237 25 47 81 993

4 238 26 48 82 994

5 239 27 49 83 995

6 240 28 50 84 996

7 241 29 51 85 997

8 242 30 52 86 998

9 243 31 53 87 999

10 244 32 54 88 1000

11 245 33 55 89 1001

12 246 34 56 90 1002

13 247 35 57 91 1003

#从下向上数8行，然后跳过这8行

#index\_col : int, list of ints, default None指定某列为索引列

pd.read\_excel(r'Desktop\1.xls' ,sheetname='Sheet3',header=None,index\_col=4)

Out[15]:

0 1 2 3

4

990 234 22 44 78

991 235 23 45 79

992 236 24 46 80

993 237 25 47 81

994 238 26 48 82

995 239 27 49 83

996 240 28 50 84

997 241 29 51 85

998 242 30 52 86

999 243 31 53 87

1000 244 32 54 88

1001 245 33 55 89

1002 246 34 56 90

1003 247 35 57 91

1004 248 36 58 92

1005 249 37 59 93

1006 250 38 60 94

1007 251 39 61 95

1008 252 40 62 96

1009 253 41 63 97

1010 254 42 64 98

1011 255 43 65 99

#上面指定第四列为索引列

#names : array-like, default None, 给所有列索引重新命名。

pd.read\_excel(r'Desktop\1.xls' ,sheetname='Sheet3',names=['col\_1','col\_2','col\_3','col\_4','col\_5'])

Out[17]:

col\_1 col\_2 col\_3 col\_4 col\_5

0 235 23 45 79 991

1 236 24 46 80 992

2 237 25 47 81 993

3 238 26 48 82 994

4 239 27 49 83 995

5 240 28 50 84 996

6 241 29 51 85 997

7 242 30 52 86 998

8 243 31 53 87 999

9 244 32 54 88 1000

10 245 33 55 89 1001

11 246 34 56 90 1002

12 247 35 57 91 1003

13 248 36 58 92 1004

14 249 37 59 93 1005

15 250 38 60 94 1006

16 251 39 61 95 1007

17 252 40 62 96 1008

18 253 41 63 97 1009

19 254 42 64 98 1010

20 255 43 65 99 1011

#上面的列索引都被更换。

文本文件的块读取

如果只想读取数据一部分，或者数据过大，我们想逐步读取，我们可以通过附加属性函数来实现。

通过nrows属性函数可以选择所需要的行数

!type Desktop\example\_14.csv

one,two,three,four

0,1,2,3

4,5,6,7

8,9,10,11

12,13,14,15

16,17,18,19

20,21,22,23

12,13,14,15

16,17,18,19

20,21,22,23

12,13,14,15

16,17,18,19

20,21,22,23

12,13,14,15

16,17,18,19

20,21,22,23

0,1,2,3

4,5,6,7

8,9,10,11

12,13,14,15

16,17,18,19

20,21,22,23

12,13,14,15

16,17,18,19

pd.read\_table('Desktop\example\_14.csv',sep=',',nrows=8)

Out[21]:

one two three four

0 0 1 2 3

1 4 5 6 7

2 8 9 10 11

3 12 13 14 15

4 16 17 18 19

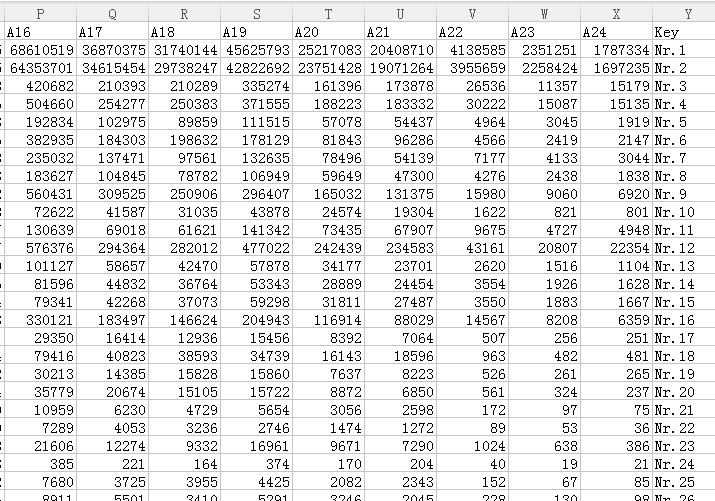
5 20 21 22 23

6 12 13 14 15

7 16 17 18 19

实验项目：数据的分块读取：

原始数据的部分截取：



block\_2=pd.read\_csv('Desktop\lockreading.csv',chunksize=59)

key\_quantity=pd.Series([])

#block\_2是对整个数据分块后的返回值，可看做一个三维数据框，key\_quantity是一个存储行编号数量的series

for block\_var in block\_2:

key\_quantity=key\_quantity.add(block\_var['Key'].value\_counts(),fill\_value=0)

**{千万注意：这里fill\_value很重要，因为在循环过程中，由于相加的两个Seires的行标签不可能每一次都完全匹配，行标签不完全匹配的两个Series相加如果会产生空值，这种情况下会影响下次循环series的相加，因为空值+实数会继续产生空值，从而计数错误。为了消除这种现象的产生，最好的办法是每一次相加前的空值立马填充为‘0’值，这样杜绝空值产生，不会影响下一步两个series的相加。**

**看下面例子：**

**import pandas**

**import pandas as pd**

**serie\_1=pd.Series([1,2,3],index=list('abc'))**

**serie\_2=pd.Series([4,5,8],index=list('abd'))**

**serie\_3=serie\_1+serie\_2**

**serie\_3**

**Out[6]:**

**a 5.0**

**b 7.0**

**c NaN**

**d NaN**

**dtype: float64**

**#假如我们添加’fill\_value=0‘**

**serie\_1.add(serie\_2,fill\_value=0)**

**Out[8]:**

**a 5.0**

**b 7.0**

**c 3.0**

**d 8.0**

**dtype: float64 }**

**接下来我们继续我们的项目：**

key\_quantity=key\_quantity.sort\_values(ascending=False)

key\_quantity

Out[16]:

Nr.36 10.0

Nr.30 10.0

Nr.32 10.0

Nr.33 10.0

Nr.34 10.0

Nr.35 10.0

Nr.37 10.0

Nr.38 10.0

Nr.39 10.0

Nr.29 10.0

Nr.40 10.0

Nr.41 10.0

Nr.28 10.0

Nr.27 10.0

Nr.26 10.0

Nr.25 10.0

Nr.24 10.0

Nr.23 10.0

Nr.31 10.0

Nr.42 9.0

Nr.21 7.0

Nr.20 7.0

Nr.2 7.0

Nr.19 7.0

Nr.18 7.0

Nr.17 7.0

Nr.22 7.0

Nr.15 7.0

Nr.14 7.0

Nr.3 7.0

Nr.13 7.0

Nr.12 7.0

Nr.11 7.0

Nr.10 7.0

Nr.16 7.0

Nr.9 7.0

Nr.8 7.0

Nr.52 7.0

Nr.7 7.0

Nr.6 7.0

Nr.59 7.0

Nr.58 7.0

Nr.57 7.0

Nr.56 7.0

Nr.55 7.0

Nr.54 7.0

Nr.53 7.0

Nr.51 7.0

Nr.4 7.0

Nr.50 7.0

Nr.5 7.0

Nr.49 7.0

Nr.48 7.0

Nr.47 7.0

Nr.46 7.0

Nr.45 7.0

Nr.44 7.0

Nr.43 7.0

Nr.1 7.0

dtype: float64

#上面这个Series左边是行编号，（注意不是行索引，行编号是key列中的值，它是每一行的编号），右边是每个编号在整个数据里出现的次数。

通过一个小函数我们能够瞬间遍查每个数据块寻找任何一个给定的行编号。

def search\_rowindizies(str\_test):

list\_1=[]

for block\_var in block\_2:

boll=block\_var['Key']==str\_test

st=block\_var.loc[boll,:'A24'].values

lst=st.astype(np.float32)

#alst=list(lst)

list\_1.append(lst)

return list\_1

block\_2=pd.read\_csv('Desktop\lockreading.csv',chunksize=59)

search\_rowindizies('Nr.42')

Out[19]:

[array([[ 4.57160000e+04, 2.33750000e+04, 2.23410000e+04,

1.60100000e+03, 6.03000000e+02, 9.98000000e+02,

2.27580000e+04, 1.01930000e+04, 1.25650000e+04,

1.37070000e+04, 7.65300000e+03, 6.05400000e+03,

4.13700000e+03, 2.58400000e+03, 1.55300000e+03,

2.97900000e+03, 2.01500000e+03, 9.64000000e+02,

5.19000000e+02, 3.16000000e+02, 2.03000000e+02,

1.50000000e+01, 1.10000000e+01, 4.00000000e+00]], dtype=float32),

array([[ 4.57160000e+04, 2.33750000e+04, 2.23410000e+04,

1.60100000e+03, 6.03000000e+02, 9.98000000e+02,

2.27580000e+04, 1.01930000e+04, 1.25650000e+04,

1.37070000e+04, 7.65300000e+03, 6.05400000e+03,

4.13700000e+03, 2.58400000e+03, 1.55300000e+03,

2.97900000e+03, 2.01500000e+03, 9.64000000e+02,

5.19000000e+02, 3.16000000e+02, 2.03000000e+02,

1.50000000e+01, 1.10000000e+01, 4.00000000e+00]], dtype=float32),

array([[ 4.57160000e+04, 2.33750000e+04, 2.23410000e+04,

1.60100000e+03, 6.03000000e+02, 9.98000000e+02,

2.27580000e+04, 1.01930000e+04, 1.25650000e+04,

1.37070000e+04, 7.65300000e+03, 6.05400000e+03,

4.13700000e+03, 2.58400000e+03, 1.55300000e+03,

2.97900000e+03, 2.01500000e+03, 9.64000000e+02,

5.19000000e+02, 3.16000000e+02, 2.03000000e+02,

1.50000000e+01, 1.10000000e+01, 4.00000000e+00]], dtype=float32),

array([[ 4.57160000e+04, 2.33750000e+04, 2.23410000e+04,

1.60100000e+03, 6.03000000e+02, 9.98000000e+02,

2.27580000e+04, 1.01930000e+04, 1.25650000e+04,

1.37070000e+04, 7.65300000e+03, 6.05400000e+03,

4.13700000e+03, 2.58400000e+03, 1.55300000e+03,

2.97900000e+03, 2.01500000e+03, 9.64000000e+02,

5.19000000e+02, 3.16000000e+02, 2.03000000e+02,

1.50000000e+01, 1.10000000e+01, 4.00000000e+00]], dtype=float32),

array([[ 4.57160000e+04, 2.33750000e+04, 2.23410000e+04,

1.60100000e+03, 6.03000000e+02, 9.98000000e+02,

2.27580000e+04, 1.01930000e+04, 1.25650000e+04,

1.37070000e+04, 7.65300000e+03, 6.05400000e+03,

4.13700000e+03, 2.58400000e+03, 1.55300000e+03,

2.97900000e+03, 2.01500000e+03, 9.64000000e+02,

5.19000000e+02, 3.16000000e+02, 2.03000000e+02,

1.50000000e+01, 1.10000000e+01, 4.00000000e+00]], dtype=float32),

array([[ 4.57160000e+04, 2.33750000e+04, 2.23410000e+04,

1.60100000e+03, 6.03000000e+02, 9.98000000e+02,

2.27580000e+04, 1.01930000e+04, 1.25650000e+04,

1.37070000e+04, 7.65300000e+03, 6.05400000e+03,

4.13700000e+03, 2.58400000e+03, 1.55300000e+03,

2.97900000e+03, 2.01500000e+03, 9.64000000e+02,

5.19000000e+02, 3.16000000e+02, 2.03000000e+02,

1.50000000e+01, 1.10000000e+01, 4.00000000e+00]], dtype=float32),

array([[ 4.57160000e+04, 2.33750000e+04, 2.23410000e+04,

1.60100000e+03, 6.03000000e+02, 9.98000000e+02,

2.27580000e+04, 1.01930000e+04, 1.25650000e+04,

1.37070000e+04, 7.65300000e+03, 6.05400000e+03,

4.13700000e+03, 2.58400000e+03, 1.55300000e+03,

2.97900000e+03, 2.01500000e+03, 9.64000000e+02,

5.19000000e+02, 3.16000000e+02, 2.03000000e+02,

1.50000000e+01, 1.10000000e+01, 4.00000000e+00]], dtype=float32),

array([[ 4.57160000e+04, 2.33750000e+04, 2.23410000e+04,

1.60100000e+03, 6.03000000e+02, 9.98000000e+02,

2.27580000e+04, 1.01930000e+04, 1.25650000e+04,

1.37070000e+04, 7.65300000e+03, 6.05400000e+03,

4.13700000e+03, 2.58400000e+03, 1.55300000e+03,

2.97900000e+03, 2.01500000e+03, 9.64000000e+02,

5.19000000e+02, 3.16000000e+02, 2.03000000e+02,

1.50000000e+01, 1.10000000e+01, 4.00000000e+00],

[ 4.57160000e+04, 2.33750000e+04, 2.23410000e+04,

1.60100000e+03, 6.03000000e+02, 9.98000000e+02,

2.27580000e+04, 1.01930000e+04, 1.25650000e+04,

1.37070000e+04, 7.65300000e+03, 6.05400000e+03,

4.13700000e+03, 2.58400000e+03, 1.55300000e+03,

2.97900000e+03, 2.01500000e+03, 9.64000000e+02,

5.19000000e+02, 3.16000000e+02, 2.03000000e+02,

1.50000000e+01, 1.10000000e+01, 4.00000000e+00]], dtype=float32)]

**#每次执行函数都要调用一次block\_2,因为数据用完一次后自动与源数据断开，Block\_2自动清零消失**

到此为止，我们的项目结束！！

把数据下载到csv 文件

DataFrame\_1=pd.DataFrame(np.arange(16).reshape(4,4),index=list('asdt'),columns=list('sgdt'))

DataFrame\_1

Out[5]:

s g d t

a 0 1 2 3

s 4 5 6 7

d 8 9 10 11

t 12 13 14 15

DataFrame\_1.to\_csv('Desktop\DataFrame\_1.csv')

#数据储存到桌面，文件名是DataFrame\_1。默认情况下，To\_csv方法只会把数据存为用逗号隔开的csv数据。我们可以通过!type函数直接调用这个已存到桌面上的文件，就会发现它的确如此，如下：

!type Desktop\DataFrame\_1.csv

,s,g,d,t

a,0,1,2,3

s,4,5,6,7

d,8,9,10,11

t,12,13,14,15

#上面是没有转化为数据框的源数据格式，也就是csv文件的直接调取。

当然，我们也可以直接将数据直接存其它分割方式，例如下面的例子

DataFrame\_1.to\_csv('Desktop\DataFrame\_2.csv',sep='#')

!type Desktop\DataFrame\_2.csv

#s#g#d#t

a#0#1#2#3

s#4#5#6#7

d#8#9#10#11

t#12#13#14#15

DataFrame\_2=DataFrame\_1.copy()

DataFrame\_2.loc['a','s']=np.nan

DataFrame\_2

Out[32]:

s g d t

a NaN 1 2 3

s 4.0 5 6 7

d 8.0 9 10 11

t 12.0 13 14 15

DataFrame\_2.to\_csv('Desktop\DataFrame\_3.csv')

!type Desktop\DataFrame\_3.csv

,s,g,d,t

a,,1,2,3

s,4.0,5,6,7

d,8.0,9,10,11

t,12.0,13,14,15

#再重新通过！type指令调回已存储的文件后，发现数据框原有的NaN值变成空位置，然而，空位置在csv文件中并不好辨认，因此我们需要把含有nan值的数据框存储为nan值被指定的容易辨认的符号替代的csv文件。

DataFrame\_2

Out[50]:

s g d t

a NaN 1 2 3

s 4.0 5 6 7

d 8.0 9 10 11

t 12.0 13 14 15

DataFrame\_2.to\_csv('Desktop\DataFrame\_4.csv',na\_rep='Cat')

!type Desktop\DataFrame\_4.csv

,s,g,d,t

a,Cat,1,2,3

s,4.0,5,6,7

d,8.0,9,10,11

t,12.0,13,14,15

#用!type调用储存在桌面的文件DataFrame\_4.csv，我们会发现原来在数据框的NaN值在存储后被‘Cat’替代

这里我们要注意空位置与空字符串不一样

试比较：

pd.read\_csv('Desktop\example\_16.csv')

Out[42]:

a b c d

0 ' ' 123 456 89

1 22 65 ' ' 12

2 345 89 0 6

3 23 ' ' 33 66

!type Desktop\example\_16.csv

a,b,c,d

' ',123,456,89

22,65,' ',12

345,89,0,6

23,' ',33,66

上面是空字符串

pd.read\_csv('Desktop\example\_17.csv')

Out[44]:

a b c d

0 NaN 123.0 456.0 89

1 22.0 65.0 NaN 12

2 345.0 89.0 0.0 6

3 23.0 NaN 33.0 66

!type Desktop\example\_17.csv

a,b,c,d

,123,456,89

22,65,,12

345,89,0,6

23,,33,66

上面是空位置，空位置再读取成数据框时会产生NaN值。

如果没有特殊指定，数据框行和列的标签会随数据一起存储为csv文件，当然我们也可以不这样。如下：

DataFrame\_1

Out[51]:

s g d t

a 0 1 2 3

s 4 5 6 7

d 8 9 10 11

t 12 13 14 15

DataFrame\_1.to\_csv('Desktop\Dataframe\_12.csv',index=False,header=False)

!type Desktop\Dataframe\_12.csv

0,1,2,3

4,5,6,7

8,9,10,11

12,13,14,15

我们发现行标签和列标签都不见了。

我们还可以只存储数据框的部分列，并指定顺序：

DataFrame\_1.to\_csv('Desktop\Dataframe\_13.csv',index=False,columns=['s','t','d'])

调用存储文件，测试！！

!type Desktop\Dataframe\_13.csv

s,t,d

0,3,2

4,7,6

8,11,10

12,15,14

我们发现只有部分列被存储，且列的顺序按照指定。

Series 也可以通过to\_csv来存储信息到桌面。

dates\_1=pd.date\_range('01/06/2017',periods=14)

dates\_1

Out[60]:

DatetimeIndex(['2017-01-06', '2017-01-07', '2017-01-08', '2017-01-09',

'2017-01-10', '2017-01-11', '2017-01-12', '2017-01-13',

'2017-01-14', '2017-01-15', '2017-01-16', '2017-01-17',

'2017-01-18', '2017-01-19'],

dtype='datetime64[ns]', freq='D')

series\_12=pd.Series(np.arange(14),index=dates\_1)

series\_12.to\_csv('Desktop\series\_123.csv')

!type Desktop\series\_123.csv

2017-01-06,0

2017-01-07,1

2017-01-08,2

2017-01-09,3

2017-01-10,4

2017-01-11,5

2017-01-12,6

2017-01-13,7

2017-01-14,8

2017-01-15,9

2017-01-16,10

2017-01-17,11

2017-01-18,12

2017-01-19,13

csv文件也可以从桌面直接被读取成Series：

pd.Series.from\_csv('Desktop\series\_123.csv')

Out[71]:

2017-01-06 0

2017-01-07 1

2017-01-08 2

2017-01-09 3

2017-01-10 4

2017-01-11 5

2017-01-12 6

2017-01-13 7

2017-01-14 8

2017-01-15 9

2017-01-16 10

2017-01-17 11

2017-01-18 12

2017-01-19 13

dtype: int64

注：

1. Read\_csv可以读取文件，url，文件型对象，但被加载文件必须有分割符，默认的分割符为逗号。
2. Read\_table可以读取文件，url，文件型对象，但被加载文件必须有分割符，默认的分割符为制表符‘\t’。这一点与read\_csv不同，烦请再加载有逗号的数据时用seq属性注明，也即seq=‘，’！
3. Read\_fwf读取（或称加载）没有分隔符数据，但是各数据之间间距要恒定。
4. Read\_clipboard 通常用来读取网络数据,在使用前，必须把网页内容先复制到粘贴板上，例如：

pd.read\_clipboard(sep='\\s+')

Out[73]:

北 京 220956 209468 11488 697.02 264.30 253555 44087 701.42

0 天 津 93162 90080 3082 330.68 126.98 106063 15983 340.10

1 河 北 209740 200012 9728 963.39 327.27 262396 62384 982.31

2 山 西 131802 114466 17336 628.55 192.27 214625 100159 669.39

3 内蒙古 71196 65627 5569 306.82 110.15 101829 36202 308.10

4 　 　 　 　 　 　 　 　 　NaN

5 辽 宁 211502 199611 11891 838.09 305.58 268741 69130 846.25

上面函数常用关键字列表

|  |  |
| --- | --- |
| filepath\_or\_buffer: | 文件系统位置，url，文件型对象的字符串 |
| delimiter或sep | 源文件各数据间分隔符或正则表达式 |
| header | 上载数据成数据框时，数据框的列名，默认为上载数据的第一行。即‘0’行。如果不需要列标签，那么使header=None |
| index\_col | 就是给行层次化索引命名，行层次索引一般有两列，内列和外列。 |
| skiprows | 需要忽略的行数，跳过无用行 |
| na\_values | 实质就是把源数据中的指定字符串转化为空值。 |
| converters | 由列名或者说列号或者说列标签和函数组成字典，例如{‘A’:f}说明f函数应用到’A’列中的每一个数据。 |
| nrows | 需要读取的数据行数 |
| skip\_footer | 需要忽略的行数，注意从源数据最后一行向上数 |
| encoding | 用于指明unicode文本文件的文本编码格式 |
| Squeeze | 如果数据仅有一列，自动返回series |
| thousands | 千分位分隔符，如’.’或’,’ |

converters的应用：

1. 数据框的元素作为字符穿处理

In [**1**]: import pandas as pd

In [**2**]: import numpy as np

In [**3**]: Data\_1=pd.DataFrame([[1,2,1],[3,4,2],[3,5,4]],columns=['A','B','C'],index=['r1','r2','r3'])

In [**4**]: Data\_1

Out[**4**]:

A B C

r1 1 2 1

r2 3 4 2

r3 3 5 4

In [**7**]: Data\_1.to\_csv(r'Desktop\dong1234.csv',sep=',')

In [**8**]: pd.read\_csv('Desktop\dong1234.csv',delimiter=',',usecols=['A','B','C'])

Out[**8**]:

A B C

0 1 2 1

1 3 4 2

2 3 5 4

In [**10**]: pd.read\_csv('Desktop\dong123.csv',delimiter=',',usecols=['A','B','C'],converters={'B':lambda x:x\*2})

Out[**10**]:

A B C

0 1 22 1

1 3 44 2

2 3 55 4

In [**11**]: pd.read\_csv('Desktop\dong123.csv',delimiter=',',usecols=['A','B','C'],converters={'B':lambda x:x\*2+'1'})

Out[**11**]:

A B C

0 1 221 1

1 3 441 2

2 3 551 4

In [**12**]: pd.read\_csv('Desktop\dong123.csv',delimiter=',',usecols=['A','B','C'],converters={'B':lambda x:x\*2\*3+'1'})

    ...:

Out[**12**]:

A B C

0 1 2222221 1

1 3 4444441 2

2 3 5555551 4

2.数据框的元素作为数据处理

In [**14**]: pd.read\_csv('Desktop\dong123.csv',delimiter=',',usecols=['A','B','C'],converters={'B':lambda x:float(x)\*\*3+2\*float(x)})

Out[**14**]:

A B C

0 1 12.0 1

1 3 72.0 2

2 3 135.0 4

In [**15**]: import math

In [**21**]: converters\_1={i: lambda x:float(x)\*\*3+math.sin((np.pi/6)\*float(x)) for i in range(3)}

In [**22**]: converters\_1

Out[**22**]:

{0: <function \_\_main\_\_.<dictcomp>.<lambda>>,

1: <function \_\_main\_\_.<dictcomp>.<lambda>>,

2: <function \_\_main\_\_.<dictcomp>.<lambda>>}

In [**23**]: pd.read\_csv('Desktop\dong123.csv',delimiter=',',usecols=['A','B','C'],converters=converters\_1)

Out[**23**]:

A B C

0 1.5 8.866025 1

1 28.0 64.866025 2

2 28.0 125.500000 4

分隔符格式的手工处理

通常用read\_table或者read\_csv直接加载csv文件或者TXT文件都是没有问题的，但有时csv文件十分混乱，以致无法加载，也是十分常见的，这就要求我们在加载前要对原始csv文件进行处理，使其能够被顺利加载。

我们先看一个单字符分隔符文件

import pandas as pd

import numpy as np

!type Desktop\example\_20.csv

"f","g","h","d"

"1","1","4","5"

"7","9","0","4"

"2","8","11",0

import csv

调用内置csv模块，将已打开文件直接传给csv.reader

f=open('Desktop\example\_20.csv')

getting\_1=csv.reader(f)

getting\_1

Out[25]: <\_csv.reader at 0x3077b74320>

#我们无法看到getting\_1的庐山真面目，怎么办？通过for循环，我们看到getting\_1中的每个元素。这是一个十分好的方法，请大家务必注意：

for subgetting in getting\_1:

print(subgetting)

['f', 'g', 'h', 'd']

['1', '1', '4', '5']

['7', '9', '0', '4']

['2', '8', '11', '0']

我下面再举一个具体的例子：

range(1,9,1)

Out[27]: range(1, 9)

for subrange in range(1,9,1):

print(subrange)

1

2

3

4

5

6

7

8

当然这里我们还有一个终极解决方案，也就是通过list。

Gate\_1=list(csv.reader(open('Desktop\example\_20.csv')))

Gate\_1

Out[32]:

[['f', 'g', 'h', 'd'],

['1', '1', '4', '5'],

['7', '9', '0', '4'],

['2', '8', '11', '0']]

#直到这里我们终于看到了csv.reader()的庐山真面目。

header,values=Gate\_1[0],Gate\_1[1:]

dic\_new\_1={u:v for u,v in zip(header,zip(\*values))}

dic\_new\_1

Out[36]:

{'d': ('5', '4', '0'),

'f': ('1', '7', '2'),

'g': ('1', '9', '8'),

'h': ('4', '0', '11')}

最终我们把相对杂乱无章的数据转化为规整的字典。

解释：zip（\*2dim-Array）

这个操作相当于把嵌套列表每一行一一对应先组成元组，然后组成列表：

c=[[1,2,3],[4,7,0]]

zip(\*c)

Out[40]: <zip at 0x3077b9ce08>

list(zip(\*c))

Out[41]: [(1, 4), (2, 7), (3, 0)]

我们可以自定义csv文件的读取或写入格式：

Import csv

class mydialect(csv.Dialect):

lineterminator = '\n' #用于写操作时的行结束符，读操作时将忽略此项，也就是读操作时无需在文件数据行尾处加‘\n’

delimiter = '#' #可以读取用‘#’分割的字段（或者数据）

quotechar='"' #用于带有特殊字符的字段（通常指的是带有单引号的字符串）的引用符号，例如源数据这样的字段’12#3’通过csv.reader函数读取到python程序页面后会变成”’12”,”3’”。

quoting=csv.QUOTE\_ALL #读取所有字段（和数据）

skipinitialspace=True #或略分隔符后面的空格

doublequote=True #如果读取的字段含有引用符号，则整个字段加双引号。

下面是一个比较简单的例子：

Import csv

class mydialect(csv.Dialect):

lineterminator = '\n'

delimiter = '#'

quotechar='"'

quoting=csv.QUOTE\_ALL

skipinitialspace=True

doublequote=True

myfile\_1=open('Desktop\example\_223.csv')

content\_11=csv.reader(myfile\_1,dialect=mydialect)

for contents in content\_11:

print(contents)

['At', 'bee', 'cotton', 'death', "common'hero'tree"]

['1', '2', '3', '4']

['5', '6', '7', '8', '9']

['9', '10', '11', '12', "'12", "3'"]

!type Desktop\example\_223.csv

At#bee#cotton# death#common'hero'tree

1#2# 3#4

5#6#7#8#9

9#10#11#12#'12#3'

我们上面一直在讨论通过定义csv.Dialect的一个子类来读取独有格式的csv，下面我们谈谈写入

class mydialect(csv.Dialect):

lineterminator = '\n'

delimiter = '#'

quotechar='"'

quoting=csv.QUOTE\_ALL

skipinitialspace=True

doublequote=True

with open('Desktop\example\_25.csv','w') as newsetting\_1: #首先open()函数打开文件'Desktop\example\_25.csv'，如果不存在这个文件，则创建它，返回文件对象newsetting\_1。

writer\_object=csv.writer(newsetting\_1,dialect=mydialect) #创建一个写对象'writer\_object'，可理解成给打开文件添加独有编码风格“mydialect”。然后返回一个带有独有编码风格的新文件对象

writer\_object.writerow(["'one'",0, 8]) #可通过列表把内容一步步地写入到打开的文件

writer\_object.writerow(['two',62,43])

mylist=[['three',46,88],['four', 3,12]]

writer\_object.writerows(mylist)

!type Desktop\example\_25.csv

one'#"0"#"8"

two#"62"#"43"

three#"46"#"88"

four#"3"#"12"

JSON数据

JSON是一种数据格式，它是通过HTTP请求在web浏览器与其他应用程序之间数据传送格式之一。

例如：

Json\_documents="""

{"Name":"Wes","places\_lived":["United States","Spain","Germany","Japan"],"pet":null,"siblings":[{"name":"Scott","age":25,"pet":"Zuko","weight":56,"height":172},{"name":"Katie","age":33,"pet":"Cisco","weight":89,"height":189}]}

"""

这是一个Json数据格式字符串数据对象，可以通过python标准库中的json.loads函数即可将JSON字符串数据转化为python字符串数据。

import json

getting\_1=json.loads(Json\_documents)

getting\_1

Out[36]:

{'Name': 'Wes',

'pet': None,

'places\_lived': ['United States', 'Spain', 'Germany', 'Japan'],

'siblings': [{'age': 25,

'height': 172,

'name': 'Scott',

'pet': 'Zuko',

'weight': 56},

{'age': 33, 'height': 189, 'name': 'Katie', 'pet': 'Cisco', 'weight': 89}]}

试比较这两种数据结构，我们会发现：

1. JSON格式的字符串数据对象的字符串没有排序，确切的说字典或者说JSON对象的键和其所对应的子字典内容都没有任何排序。而python格式的字典缺有严格工整的排序，无论键还是键所对应的字典都有严格排序。
2. 所有字符串的双引号都变成单引号
3. JSON格式的字典最前面和最后面都有‘”””’（也可用“’”）标识，转化成Python格式后所有标识全部消失。
4. JSON的空值用null表达，Python用None表达。

有一点我们必须要注意，JSON格式的对象（JSON有对象（即字典）、数组、字符串、数字，bool值以及null等数据结构）的键必须是字符串。如果不是字符串将会在转换时出现错误提示:

Json\_documents="""

{23:"Wes","places\_lived":["United States","Spain","Germany","Japan"],"pet":null,"siblings":[{"name":"Scott","age":25,"pet":"Zuko","weight":56,"height":172},{"name":"Katie","age":33,"pet":"Cisco","weight":89,"height":189}]}

"""

getting\_1=json.loads(Json\_documents)

---------------------------------------------------------------------------

JSONDecodeError Traceback (most recent call last)

<ipython-input-38-66d19b29dad0> in <module>()

----> 1 getting\_1=json.loads(Json\_documents)

~\Anaconda3\lib\json\\_\_init\_\_.py in loads(s, encoding, cls, object\_hook, parse\_float, parse\_int, parse\_constant, object\_pairs\_hook, \*\*kw)

352 parse\_int is None and parse\_float is None and

.

.

.

357 raise JSONDecodeError("Expecting value", s, err.value) from None

JSONDecodeError: **Expecting property name enclosed in double quotes**: line 2 column 2 (char 2)

我们当然也可以把python格式的数据对象转化为JSON格式，使用python的函数json.dumps可以实现这一愿望：

例子：

python\_format\_1={'Name': 'Wes',

'pet': None,

'places\_lived': ['United States', 'Spain', 'Germany', 'Japan'],

'siblings': [{'age': 25,

'height': 172,

'name': 'Scott',

'pet': 'Zuko',

'weight': 56},

{'age': 33, 'height': 189, 'name': 'Katie', 'pet': 'Cisco', 'weight': 89}]}

json\_format\_1=pd.json.dumps(python\_format\_1)

json\_format\_1

Out[6]:'{"Name":"Wes","pet":null,"places\_lived":["United States","Spain","Germany","Japan"],"siblings":[{"age":25,"height":172,"name":"Scott","pet":"Zuko","weight":56},{"age":33,"height":189,"name":"Katie","pet":"Cisco","weight":89}]}'

再把json格式数据转化为python数据格式后，我们需要把数据作进一步处理，以便我们更好的处理数据。最常用的做法是把数据继续转化为数据框：

newDataFrame=pd.DataFrame(python\_format\_1['siblings'])

newDataFrame

Out[8]:

age height name pet weight

0 25 172 Scott Zuko 56

1 33 189 Katie Cisco 89

python\_format\_1['siblings']表示的是一个由字典组成的列表，通过它可以直接生成一个数据框，这是一个很好的数据框创建方法。我们下面一起做一些用这种模式创建数据框的练习：

例子1：

dict\_1={'c1':[1,2,3,4,5],'c2':[21,65,32,90,0],'c3':[90,67,5,6,3],'c4':[1,2,1,2,0],'c5':[9,0,8,7,0]}

dict\_2={'c1':[1,3,3,4,5],'c2':[21,5,32,90,3],'c3':[90,7,5,6,3],'c4':[1,2,155,2,0],'c5':[9,450,8,7,0]}

newframe\_0=pd.DataFrame(dict\_1)

newframe\_0

Out[19]:

c1 c2 c3 c4 c5

0 1 21 90 1 9

1 2 65 67 2 0

2 3 32 5 1 8

3 4 90 6 2 7

4 5 0 3 0 0

例子2：

newframe\_0=pd.DataFrame([dict\_1])

newframe\_0

Out[17]:

c1 c2 c3 c4 \

0 [1, 2, 3, 4, 5] [21, 65, 32, 90, 0] [90, 67, 5, 6, 3] [1, 2, 1, 2, 0]

c5

0 [9, 0, 8, 7, 0]

例子3：

newframe\_1=pd.DataFrame([dict\_1,dict\_2]):

newframe\_1

Out[15]:

c1 c2 c3 c4 \

0 [1, 2, 3, 4, 5] [21, 65, 32, 90, 0] [90, 67, 5, 6, 3] [1, 2, 1, 2, 0]

1 [1, 3, 3, 4, 5] [21, 5, 32, 90, 3] [90, 7, 5, 6, 3] [1, 2, 155, 2, 0]

c5

0 [9, 0, 8, 7, 0]

1 [9, 450, 8, 7, 0]

例子4

dic\_3={'c6':[1,3,3,4,5],'c7':[21,5,32,90,3],'c8':[90,7,5,6,3],'c9':[1,2,155,2,0],'c10':[9,450,8,7,0]}

newFrame\_4=pd.DataFrame([dict\_1,dic\_3])

newFrame\_4

Out[22]:

c1 c10 c2 c3 \

0 [1, 2, 3, 4, 5] NaN [21, 65, 32, 90, 0] [90, 67, 5, 6, 3]

1 NaN [9, 450, 8, 7, 0] NaN NaN

c4 c5 c6 c7 \

0 [1, 2, 1, 2, 0] [9, 0, 8, 7, 0] NaN NaN

1 NaN NaN [1, 3, 3, 4, 5] [21, 5, 32, 90, 3]

c8 c9

0 NaN NaN

1 [90, 7, 5, 6, 3] [1, 2, 155, 2, 0]

#数据框的列索引由字典的键确定，字典列表中的有多少字典键就有多少列，譬如字典列表的第一个字典（0位置字典）有5个键，第二个字典（1位置字典）有5个键，那么生成的数据框就有10个列。字典列表的长度代表了行的数量，就如例子中字典列表的长度是2，那么数据框就有两行，0行和1行。

通过columns我们可以指定需要显示的数据列：

newframe\_5=pd.DataFrame([dict\_1,dict\_2],columns=['c2','c3'])

newframe\_5

Out[24]:

c2 c3

0 [21, 65, 32, 90, 0] [90, 67, 5, 6, 3]

1 [21, 5, 32, 90, 3] [90, 7, 5, 6, 3]

数据的二进制格式存储

定义：将对象转换为可通过网络传输或可以存储到本地磁盘的数据格式的过程称为序列化；反之，则称为反序列化。Python内置的pickle序列化用于实现Python数据类型与Python特定二进制格式之间的转换。

import pandas as pd

frame\_1=pd.read\_csv('Desktop\exampel\_6.csv')

frame\_1=pd.read\_csv('Desktop\exampel\_6.csv',sep='\s+')

frame\_1

Out[6]:

A B C'

aaa -3.450 2.360 8.90'

bbb 0.334 0.457 -4.5'

ccc 0.760 -7.340 -8.99'

ddd 0.370 -7.800 -4.45'

frame\_1.to\_pickle('Desktop\frame\_pickle')

#上面我们把frame\_1存储成pickle格式

我们也可以把数据读回到python。

import pickle

load\_file=open('Desktop\Frame\_pickle','rb')

pickle.load(load\_file)

Out[17]:

A B C'

aaa -3.450 2.360 8.90'

bbb 0.334 0.457 -4.5'

ccc 0.760 -7.340 -8.99'

ddd 0.370 -7.800 -4.45'

**HDF5的存取**

Hierarchical Data Format(HDF)是一种针对大量数据进行组织和存储的文件格式。它包含了数据模型，库，和文件格式标准。以其便捷有效，移植性强，灵活可扩展的特点受到了广泛的关注和应用。

对大数据的组织和存储得益于HDF文件的系统式节点结构，这样的结构不仅支持元数据，而且使多数据集存储成为现实。

Python中的HDF5库有两个接口Pytables和h5py。

我们先看一下h5py，h5py是python的一种工具包，它提供了一种直接而高级的HDF5 API访问接口。

import h5py

import numpy as np

#HDF5的写入

Intentity\_12=np.ones((10,12,300,305))

data\_h5py\_1=h5py.File('Desktop\example\_29-1.h5','w')

#创建一个h5文件

data\_h5py\_1['data']=Intentity\_12

#把数据写入到data\_h5py\_1主键‘data’中

data\_h5py\_1['labels']=range(120)

#把数据写入到data\_h5py\_1主键‘labels’中

data\_h5py\_1.close()

#HDF5的读取

data\_h5py\_2=h5py.File('Desktop\example\_29-1.h5','r')

#h5文件的读取

key\_content=data\_h5py\_2.keys()

#所有键的查看

abc\_123=data\_h5py\_2['data'][:]

#查看主键‘data’的内容

data\_h5py\_2['labels'][:]

#查看主键‘labels’的内容

我们再看另一种接口Pytables，Pytables也是python的一种工具包，它抽像了许多HDF5的细节以提供多种灵活数据容器，及实现表索引和查询功能。

import pandas as pd

store=pd.HDFStore('Desktop\example\_30.h5','w')

#通过HDFStore创建一个h5文件

getframe\_1=pd.DataFrame(np.arange(16).reshape(4,4),index=['r1','r2','r3','r4'],columns=['c1','c2','c3','c4'])

store['layer\_1']=getframe\_1

store['layer\_1\_1']=getframe\_1['c2']

store

Out[50]:

<class 'pandas.io.pytables.HDFStore'>

File path: Desktop\example\_30.h5

/layer\_1 frame (shape->[4,4])

/layer\_1\_1 series (shape->[4])

store['layer\_1']['c4']

Out[51]:

r1 3

r2 7

r3 11

r4 15

Name: c4, dtype: int32

注意：

1. HDF5最好一次写入，不要多次写入,以免破坏文件
2. 创建H5文件时，各种文件类型。

r：只能读

**r+：可读可写，不会创建不存在的文件。如果直接写文件，则从顶部开始写，覆盖之前此位置的内容，如果先读后写，则会在文件最后追加内容。**

w：只能写，覆盖整个文件，文件不存在则创建   
a：只能写，从文件底部添加内容，文件不存在则创建

**数据的合并**

常用的数据合并函数有pandas.merge和pandas.contact以及combine\_first。它们分别有着各自的应用条件。我们根据实际情况选取合适的函数来进行数据合并。

1. pandas.merge

通过merge函数可以实现数据框的合并，但这种合并要经过一个或多个键的衔接，这就要求我们必须在数据框中专门添加一个‘Key’列用于衔接。

例1：

Frame\_1=pd.DataFrame({'schluessel':['Nr.1','Nr.2','Nr.3','Nr.4','Nr.5'],'values\_1':[1,2,2,8,9],'values\_2':[4,7,9,0,1]})

Frame\_2=pd.DataFrame({'schluessel':['Nr.1','Nr.1','Nr.3','Nr.3','Nr.5'],'values\_1':[1,2,2,8,9],'values\_2':[4,7,9,0,1]})

Frame\_1

Out[28]:

schluessel values\_1 values\_2

0 Nr.1 1 4

1 Nr.2 2 7

2 Nr.3 2 9

3 Nr.4 8 0

4 Nr.5 9 1

Frame\_2

Out[29]:

schluessel values\_1 values\_2

0 Nr.1 1 4

1 Nr.1 2 7

2 Nr.3 2 9

3 Nr.3 8 0

4 Nr.5 9 1

pd.merge(Frame\_2,Frame\_1,on='schluessel')

Out[27]:

schluessel values\_1\_x values\_2\_x values\_1\_y values\_2\_y

0 Nr.1 1 4 1 4

1 Nr.1 2 7 1 4

2 Nr.3 2 9 2 9

3 Nr.3 8 0 2 9

4 Nr.5 9 1 9 1

**多对一（重复键列对不重复键列）内连接注意事项**

每次合并前都要指定作为键的列，比如上面我们指定“schluessel”

作为键的列如有重复键要保留，不要去重，看另一个待合并数据框有没有这个键，如果有，把相同的键合并到一起。

如果某些键在各自数据框中都没有重复现象，但是两个数据框相比这些键相同，那么要把两个键合并到一起。

如果某些键在各自数据框中都没有重复现象，且两个数据框相比这些键也不相同，那么这些键将不出现在合并后的键列里

注意：默认是内连接

如果键名不同，我们可以通过关键字“left\_on”和“right\_on”分别安置：

frame\_2=pd.DataFrame({'rkey':['a','c','d'],'data\_2':[3,2,1]})

frame\_1=pd.DataFrame({'lkey':['a','a','c','a','b','c','d','b'],'data\_1':[2,6,9,3,2,1,0,5]})

pd.merge(frame\_1,frame\_2,left\_on='lkey',right\_on='rkey')

Out[5]:

data\_1 lkey data\_2 rkey

0 2 a 3 a

1 6 a 3 a

2 3 a 3 a

3 9 c 2 c

4 1 c 2 c

5 0 d 1 d

在默认的条件下，合并后的键是两个数据框的键的交集，例如上面的例子均是如此，这种键的合并方式被称作内连接，除了内连接还有左连接、右连接以及外连接。下满我们分别比较这几种方式：

frame\_2=pd.DataFrame({'sl':['a','c','d','f'],'data\_2':[3,2,1,7]})

frame\_1=pd.DataFrame({'sl':['a','a','c','a','b','c','d','b'],'data\_1':[2,6,9,3,2,1,0,5]})

pd.merge(frame\_1,frame\_2,on='sl',how='inner')

Out[16]:

data\_1 sl data\_2

0 2 a 3

1 6 a 3

2 3 a 3

3 9 c 2

4 1 c 2

5 0 d 1

#内连接是键的交集

pd.merge(frame\_1,frame\_2,on='sl',how='left')

Out[17]:

data\_1 sl data\_2

0 2 a 3.0

1 6 a 3.0

2 9 c 2.0

3 3 a 3.0

4 2 b NaN

5 1 c 2.0

6 0 d 1.0

7 5 b NaN

pd.merge(frame\_2,frame\_1,on='sl',how='left')

Out[6]:

data\_2 sl data\_1

0 3 a 2.0

1 3 a 6.0

2 3 a 3.0

3 2 c 9.0

4 2 c 1.0

5 1 d 0.0

6 7 f NaN

#左连接以第一个数据框的键列为合并参考键列，观察参考键列中的键是否在两个待合并数据框中有重复，有则在合并时也要重复。

pd.merge(frame\_1,frame\_2,on='sl',how='right')

Out[18]:

data\_1 sl data\_2

0 2.0 a 3

1 6.0 a 3

2 3.0 a 3

3 9.0 c 2

4 1.0 c 2

5 0.0 d 1

6 NaN f 7

#右连接以第二个数据框的键列为合并参考键列，观察参考键列中的键是否在两个待合并数据框中有重复，有则在合并时也要重复。

pd.merge(frame\_1,frame\_2,on='sl')

Out[19]:

data\_1 sl data\_2

0 2 a 3

1 6 a 3

2 3 a 3

3 9 c 2

4 1 c 2

5 0 d 1

我们可以看到，“how=’inner’”和how在缺失的状况下运行结果是一样的。

下面我们讨论多对多键的操作：

内连接

frame\_2=pd.DataFrame({'sl':['a','c','a','d','f'],'data\_2':[3,2,1,7,6]})

frame\_1=pd.DataFrame({'sl':['a','a','c','a','b','c','d','b'],'data\_1':[2,6,9,3,2,1,0,5]})

pd.merge(frame\_2,frame\_1,on='sl')

Out[9]:

data\_2 sl data\_1

0 3 a 2

1 3 a 6

2 3 a 3

3 1 a 2

4 1 a 6

5 1 a 3

6 2 c 9

7 2 c 1

8 7 d 0

**多对多键合并注意事项（内连接）:**

#指定合并键列

#作为键的列如有重复键要保留，不要去重，看另一个待合并数据框有没有这个键，如果有，把相同的键合并到一起并重复。如果在一个数据框重复的键在另一个数据框中也重复出现，那么合并后它重复的次数符合笛卡尔积。

如果某些键在各自数据框中都没有重复现象，但是两个数据框相比这些键相同，那么要把两个键合并到一起。

如果某些键在各自数据框中都没有重复现象，且两个数据框相比这些键也不相同，那么这些键将不出现在合并后的键列里

frame\_2

Out[17]:

data\_2 sl

0 3 a

1 2 c

2 1 a

3 7 d

4 6 f

frame\_1

Out[18]:

data\_1 sl

0 2 a

1 6 a

2 9 c

3 3 a

4 2 b

5 1 c

6 0 d

7 5 b

pd.merge(frame\_2,frame\_1,on='sl',how='left')

Out[16]:

data\_2 sl data\_1

0 3 a 2.0

1 3 a 6.0

2 3 a 3.0

3 2 c 9.0

4 2 c 1.0

5 1 a 2.0

6 1 a 6.0

7 1 a 3.0

8 7 d 0.0

9 6 f NaN

#左连接以第一个数据框的键列为合并参考键列，观察参考键列中的键是否在两个待合并数据框中有重复，有则在合并时也要重复。

如果在一个数据框重复的键在另一个数据框中也重复出现，那么合并后它重复的次数符合笛卡尔积。

pd.merge(frame\_2,frame\_1,on='sl',how='right')

Out[19]:

data\_2 sl data\_1

0 3.0 a 2

1 1.0 a 2

2 3.0 a 6

3 1.0 a 6

4 3.0 a 3

5 1.0 a 3

6 2.0 c 9

7 2.0 c 1

8 7.0 d 0

9 NaN b 2

10 NaN b 5

#右连接以第二个数据框的键列为合并参考键列，观察参考键列中的键是否在两个待合并数据框中有重复，有则在合并时也要重复。

如果在一个数据框重复的键在另一个数据框中也重复出现，那么合并后它重复的次数符合笛卡尔积。

外连接组合了左连接和右连接的效果。

pd.merge(frame\_2,frame\_1,on='sl',how='outer')

Out[20]:

data\_2 sl data\_1

0 3.0 a 2.0

1 3.0 a 6.0

2 3.0 a 3.0

3 1.0 a 2.0

4 1.0 a 6.0

5 1.0 a 3.0

6 2.0 c 9.0

7 2.0 c 1.0

8 7.0 d 0.0

9 6.0 f NaN

10 NaN b 2.0

11 NaN b 5.0

通过上面的一系列例子，我们发现，两个数据框的键列如果不完全一致，或者键不唯一，将是一件很麻烦的事，因此，我建议同学们除非万不得已还是应该给两个待合并的数据框设置带有唯一键的完全相同的键列。

frame\_2=pd.DataFrame({'sl':['a','b','c','d','f'],'data\_2':[3,2,1,7,6]})

frame\_1=pd.DataFrame({'sl':['a','b','c','d','f'],'data\_1':[2,6,9,3,2]})

pd.merge(frame\_2,frame\_1,on='sl')

Out[22]:

data\_2 sl data\_1

0 3 a 2

1 2 b 6

2 1 c 9

3 7 d 3

4 6 f 2

indirect\_Data=pd.merge(frame\_2,frame\_1,on='sl')

indirect\_Data.reindex(columns=['data\_1','data\_2','sl'])

Out[27]:

data\_1 data\_2 sl

0 2 3 a

1 6 2 b

2 9 1 c

3 3 7 d

4 2 6 f

带有双关键字列的数据框的合并

frame\_doppel\_1=pd.DataFrame({'key1':list('wwerrt'),'key2':list('onoodn'),'data1':[1,9,76,9,5,6]})

frame\_doppel\_2=pd.DataFrame({'key1':list('wwt'),'key2':list('onn'),'data1':[1,9,76]})

frame\_doppel\_2

Out[31]:

data1 key1 key2

0 1 w o

1 9 w n

2 76 t n

frame\_doppel\_1

Out[32]:

data1 key1 key2

0 1 w o

1 9 w n

2 76 e o

3 9 r o

4 5 r d

5 6 t n

pd.merge(frame\_doppel\_1,frame\_doppel\_2,on=['key1','key2'],how='outer')

Out[33]:

data1\_x key1 key2 data1\_y

0 1 w o 1.0

1 9 w n 9.0

2 76 e o NaN

3 9 r o NaN

4 5 r d NaN

5 6 t n 76.0

pd.merge(frame\_doppel\_1,frame\_doppel\_2,on=['key1','key2'],how='left')

Out[34]:

data1\_x key1 key2 data1\_y

0 1 w o 1.0

1 9 w n 9.0

2 76 e o NaN

3 9 r o NaN

4 5 r d NaN

5 6 t n 76.0

pd.merge(frame\_doppel\_1,frame\_doppel\_2,on=['key1','key2'],how='right')

Out[35]:

data1\_x key1 key2 data1\_y

0 1 w o 1

1 9 w n 9

2 6 t n 76

pd.merge(frame\_doppel\_1,frame\_doppel\_2,on=['key1','key2'])

Out[36]:

data1\_x key1 key2 data1\_y

0 1 w o 1

1 9 w n 9

2 6 t n 76

#对于有双从键列的数据框，我们先在各自的数据框内把两个键列组成一个元组序列键列，然后按单键列处理即可。（当然，真正的实际原理并非如此）。

frame\_123=pd.DataFrame({'data':[1,2,8,9,0,98,6,56],'key1':list('aacabbed')})

frame\_456=pd.DataFrame({'data\_1':[2,67,1]},index=['a','b','e'])

frame\_456

Out[39]:

data\_1

a 2

b 67

e 1

pd.merge(frame\_123,frame\_456,left\_on='key1',right\_index=True)

Out[40]:

data key1 data\_1

0 1 a 2

1 2 a 2

3 9 a 2

4 0 b 67

5 98 b 67

6 6 e 1

pd.merge(frame\_123,frame\_456,left\_on='key1',right\_index=True,how='outer')

Out[42]:

data key1 data\_1

0 1 a 2.0

1 2 a 2.0

3 9 a 2.0

2 8 c NaN

4 0 b 67.0

5 98 b 67.0

6 6 e 1.0

7 56 d NaN

firstar\_1=pd.DataFrame({'sl\_1':['doctor','doctor','doctor','patient','patient'],'sl\_2':[4,2,4,20,32],'data\_1':np.arange(5)})

secondar\_1=pd.DataFrame(np.arange(12).reshape(4,3),index=[['doctor','doctor','patient','patient'],[4,4,20,20]],columns=['day','hour','second'])

secondar\_1

Out[7]:

day hour second

doctor 4 0 1 2

4 3 4 5

patient 20 6 7 8

20 9 10 11

firstar\_1

Out[8]:

data\_1 sl\_1 sl\_2

0 0 doctor 4

1 1 doctor 2

2 2 doctor 4

3 3 patient 20

4 4 patient 32

combination\_1=pd.merge(firstar\_1,secondar\_1,left\_on=['sl\_1','sl\_2'],right\_index=True)

combination\_1

Out[11]:

data\_1 sl\_1 sl\_2 day hour second

0 0 doctor 4 0 1 2

0 0 doctor 4 3 4 5

2 2 doctor 4 0 1 2

2 2 doctor 4 3 4 5

3 3 patient 20 6 7 8

3 3 patient 20 9 10 11

对于带有重索引的数据框合并时，要先把重索引组成一个个元组对，然后与另一个数据框两个键列组成的元组队进行比较。上面的例子选出两个数据框元组对的交集。多对多相同时符合笛卡尔积

#下面是左连接效果

pd.merge(firstar\_1,secondar\_1,left\_on=['sl\_1','sl\_2'],right\_index=True,how='left')

Out[5]:

data\_1 sl\_1 sl\_2 day hour second

0 0 doctor 4 0.0 1.0 2.0

0 0 doctor 4 3.0 4.0 5.0

1 1 doctor 2 NaN NaN NaN

2 2 doctor 4 0.0 1.0 2.0

2 2 doctor 4 3.0 4.0 5.0

3 3 patient 20 6.0 7.0 8.0

3 3 patient 20 9.0 10.0 11.0

4 4 patient 32 NaN NaN NaN

#下面是左连接效果

pd.merge(firstar\_1,secondar\_1,left\_on=['sl\_1','sl\_2'],right\_index=True,how='right')

Out[6]:

data\_1 sl\_1 sl\_2 day hour second

0 0 doctor 4 0 1 2

2 2 doctor 4 0 1 2

0 0 doctor 4 3 4 5

2 2 doctor 4 3 4 5

3 3 patient 20 6 7 8

3 3 patient 20 9 10 11

#下面是外连接效果

pd.merge(firstar\_1,secondar\_1,left\_on=['sl\_1','sl\_2'],right\_index=True,how='outer')

Out[7]:

data\_1 sl\_1 sl\_2 day hour second

0 0 doctor 4 0.0 1.0 2.0

0 0 doctor 4 3.0 4.0 5.0

2 2 doctor 4 0.0 1.0 2.0

2 2 doctor 4 3.0 4.0 5.0

1 1 doctor 2 NaN NaN NaN

3 3 patient 20 6.0 7.0 8.0

3 3 patient 20 9.0 10.0 11.0

4 4 patient 32 NaN NaN NaN

同时使用双方索引直接合并也是没有问题的，我们看下例：

DataFrame\_1=pd.DataFrame([[2,89,0],[7,9,34],[3.4,5.6,8.9]],index=['a','g','f'],columns=['c1','c2','c3'])

DataFrame\_2=pd.DataFrame([[8,45,6,8],[4,5,3,23],[45,90,6.9,0.7],[4.5,6.8,9.2,7.6]],index=['g','g','a','f'],columns=['c3','c4','c5','c6'])

pd.merge(DataFrame\_1,DataFrame\_2,left\_index=True,right\_index=True,how='outer')

Out[12]:

c1 c2 c3\_x c3\_y c4 c5 c6

a 2.0 89.0 0.0 45.0 90.0 6.9 0.7

f 3.4 5.6 8.9 4.5 6.8 9.2 7.6

g 7.0 9.0 34.0 8.0 45.0 6.0 8.0

g 7.0 9.0 34.0 4.0 5.0 3.0 23.0

下面我们用join函数来实现按索引的合并：

利用join按行索引合并时，两个待合并数据框的列索引不能有重复。

DataFrame\_2\_cor=pd.DataFrame([[8,45,6,8],[4,5,3,23],[45,90,6.9,0.7],[4.5,6.8,9.2,7.6]],index=['g','g','c','b'],columns=['c7','c4','c5','c6'])

DataFrame\_2\_cor

Out[22]:

c7 c4 c5 c6

g 8.0 45.0 6.0 8.0

g 4.0 5.0 3.0 23.0

c 45.0 90.0 6.9 0.7

b 4.5 6.8 9.2 7.6

DataFrame\_1

Out[23]:

c1 c2 c3

a 2.0 89.0 0.0

g 7.0 9.0 34.0

f 3.4 5.6 8.9

DataFrame\_2\_cor.join(DataFrame\_1)

Out[24]:

c7 c4 c5 c6 c1 c2 c3

b 4.5 6.8 9.2 7.6 NaN NaN NaN

c 45.0 90.0 6.9 0.7 NaN NaN NaN

g 8.0 45.0 6.0 8.0 7.0 9.0 34.0

g 4.0 5.0 3.0 23.0 7.0 9.0 34.0

DataFrame\_2\_cor.join(DataFrame\_1,how='inner')

Out[25]:

c7 c4 c5 c6 c1 c2 c3

g 8.0 45.0 6.0 8.0 7.0 9.0 34.0

g 4.0 5.0 3.0 23.0 7.0 9.0 34.0

DataFrame\_2\_cor.join(DataFrame\_1,how='right')

Out[26]:

c7 c4 c5 c6 c1 c2 c3

a NaN NaN NaN NaN 2.0 89.0 0.0

f NaN NaN NaN NaN 3.4 5.6 8.9

g 8.0 45.0 6.0 8.0 7.0 9.0 34.0

g 4.0 5.0 3.0 23.0 7.0 9.0 34.0

DataFrame\_2\_cor.join(DataFrame\_1,how='outer')

Out[28]:

c7 c4 c5 c6 c1 c2 c3

a NaN NaN NaN NaN 2.0 89.0 0.0

b 4.5 6.8 9.2 7.6 NaN NaN NaN

c 45.0 90.0 6.9 0.7 NaN NaN NaN

f NaN NaN NaN NaN 3.4 5.6 8.9

g 8.0 45.0 6.0 8.0 7.0 9.0 34.0

g 4.0 5.0 3.0 23.0 7.0 9.0 34.0

#其运行结果完全按照merge.只不过默认是how=’left’。

轴向连接

在Numpy阶段我们讲过数组的合并，比如我们用concatenate连接两个数组：

arr=np.arange(12).reshape(3,4)

arr

Out[3]:

array([[ 0, 1, 2, 3],

[ 4, 5, 6, 7],

[ 8, 9, 10, 11]])

np.concatenate((arr,arr),axis=0)

Out[4]:

array([[ 0, 1, 2, 3],

[ 4, 5, 6, 7],

[ 8, 9, 10, 11],

[ 0, 1, 2, 3],

[ 4, 5, 6, 7],

[ 8, 9, 10, 11]])

np.concatenate((arr,arr),axis=1)

Out[5]:

array([[ 0, 1, 2, 3, 0, 1, 2, 3],

[ 4, 5, 6, 7, 4, 5, 6, 7],

[ 8, 9, 10, 11, 8, 9, 10, 11]])

接下来我们看一下两个或两个以上的Series是如何合并的。

ps\_1=pd.Series([0,1],index=['a','b'])

ps\_2=pd.Series([6,2],index=['c','d'])

ps\_3=pd.Series([7,0,8,6,0],index=['e','f','g','h','k'])

pd.concat((ps\_1,ps\_2,ps\_3))

Out[10]:

a 0

b 1

c 6

d 2

e 7

f 0

g 8

h 6

k 0

dtype: int64

接下来我们使 ’axis=1’，看看结果如何（注意：通常情况下，对于Series的轴是不允许其为“1”的）：

pd.concat((ps\_1,ps\_2,ps\_3),axis=1)

Out[11]:

0 1 2

a 0.0 NaN NaN

b 1.0 NaN NaN

c NaN 6.0 NaN

d NaN 2.0 NaN

e NaN NaN 7.0

f NaN NaN 0.0

g NaN NaN 8.0

h NaN NaN 6.0

k NaN NaN 0.0

pd.concat((ps\_2,ps\_4))

Out[16]:

c 6

d 2

a 0

b 8

c 6

d 2

dtype: int64

pd.concat((ps\_2,ps\_4),axis=1)

Out[17]:

0 1

a NaN 0

b NaN 8

c 6.0 6

d 2.0 2

pd.concat((ps\_2,ps\_4),axis=1,join='inner')

Out[18]:

0 1

c 6 6

d 2 2

我们还可以给合并后的Series指定索引名：

pd.concat((ps\_2,ps\_4),axis=1,join\_axes=[['a','b','u','v']])

Out[19]:

0 1

a NaN 0.0

b NaN 8.0

u NaN NaN

v NaN NaN

通过key参数可以实现把Series合并成层次化Series：

pd.concat((ps\_1,ps\_2\*3,ps\_3),keys=['ein','drei','fuenf'])

Out[6]:

ein a 0

b 1

drei c 18

d 6

fuenf e 7

f 0

g 8

h 6

k 0

dtype: int64

这里如果我们添加axis=1，结果会如何呢？

ein drei fuenf

a 0.0 NaN NaN

b 1.0 NaN NaN

c NaN 18.0 NaN

d NaN 6.0 NaN

e NaN NaN 7.0

f NaN NaN 0.0

g NaN NaN 8.0

h NaN NaN 6.0

k NaN NaN 0.0

我们发现此时原本Series的外层索引变成列索引。

对于有重复行索引的Series合并后结果会如何呢？

**很不幸，无法运行！**

到此，我们需要做一个用concat合并Series的总结：

总结：

待合并的每个series中不能出现重复的行标签

对于axis=0，Series合并后的效果是行标签的直接组合。

对于axis=1，要视合并模式而定，默认合并模式join=“outer”，按照单个的Seires，一列一列地排下去，join=“inner”模式意味着在outer模式的基础上只保留个Series共有部分，其他全删除。

下面我们把contact应用到数据框

import pandas as pd

import numpy as np

frame\_1=pd.DataFrame([[1000,2000],[279,1123],[721,877]],index=['salary','expenditure','surplus'],columns=['mattias','jennifer'])

frame\_2=pd.DataFrame(np.array([[1000,2000,3000],[279,1123,2000],[721,877,1000]])\*12,index=['salary per annum','expenditure per annum','surplus per annum'],columns=['mattias','jennifer','jocker'])

pd.concat((frame\_1,frame\_2),axis=1,keys=['works','boss'])

Out[2]:

works boss

mattias jennifer mattias jennifer jocker

expenditure 279.0 1123.0 NaN NaN NaN

expenditure per annum NaN NaN 3348.0 13476.0 24000.0

salary 1000.0 2000.0 NaN NaN NaN

salary per annum NaN NaN 12000.0 24000.0 36000.0

surplus 721.0 877.0 NaN NaN NaN

surplus per annum NaN NaN 8652.0 10524.0 12000.0

pd.concat((frame\_1,frame\_2),axis=0,keys=['works','boss'])

Out[3]:

jennifer jocker mattias

works salary 2000 NaN 1000

expenditure 1123 NaN 279

surplus 877 NaN 721

boss salary per annum 24000 36000.0 12000

expenditure per annum 13476 24000.0 3348

surplus per annum 10524 12000.0 8652

视axis=1和 =0 而定，Keys可以给数据框的列或行加上重索引

Keys所实现的功能也可以通过字典来实现：

pd.concat({'works':frame\_1,'boss':frame\_2},axis=1)

Out[6]:

boss works

mattias jennifer jocker mattias jennifer

expenditure NaN NaN NaN 279.0 1123.0

expenditure per annum 3348.0 13476.0 24000.0 NaN NaN

salary NaN NaN NaN 1000.0 2000.0

salary per annum 12000.0 24000.0 36000.0 NaN NaN

surplus NaN NaN NaN 721.0 877.0

surplus per annum 8652.0 10524.0 12000.0 NaN NaN

通过names可以给重索引进行命名：

pd.concat({'works':frame\_1,'boss':frame\_2},axis=1,names=['Identity','Names'])

Out[7]:

Identity boss works

Names mattias jennifer jocker mattias jennifer

expenditure NaN NaN NaN 279.0 1123.0

expenditure per annum 3348.0 13476.0 24000.0 NaN NaN

salary NaN NaN NaN 1000.0 2000.0

salary per annum 12000.0 24000.0 36000.0 NaN NaN

surplus NaN NaN NaN 721.0 877.0

surplus per annum 8652.0 10524.0 12000.0 NaN NaN

通过Ignore\_index=Ture可以把自行设置的行或列索引转变成python自动配置的行或列索引。

pd.concat((frame\_1,frame\_2),axis=0,ignore\_index=True)

Out[11]:

jennifer jocker mattias

0 2000 NaN 1000

1 1123 NaN 279

2 877 NaN 721

3 24000 36000.0 12000

4 13476 24000.0 3348

5 10524 12000.0 8652

pd.concat((frame\_1,frame\_2),axis=1,ignore\_index=True)

Out[12]:

0 1 2 3 4

expenditure 279.0 1123.0 NaN NaN NaN

expenditure per annum NaN NaN 3348.0 13476.0 24000.0

salary 1000.0 2000.0 NaN NaN NaN

salary per annum NaN NaN 12000.0 24000.0 36000.0

surplus 721.0 877.0 NaN NaN NaN

surplus per annum NaN NaN 8652.0 10524.0 12000.0

**如果axis=1，那么join是按0轴操作；同理，axis=1，那么join是按1轴操作。**

pd.concat((frame\_1,frame\_2),axis=1,join='outer')

Out[16]:

mattias jennifer mattias jennifer jocker

expenditure 279.0 1123.0 NaN NaN NaN

expenditure per annum NaN NaN 3348.0 13476.0 24000.0

salary 1000.0 2000.0 NaN NaN NaN

salary per annum NaN NaN 12000.0 24000.0 36000.0

surplus 721.0 877.0 NaN NaN NaN

surplus per annum NaN NaN 8652.0 10524.0 12000.0

pd.concat((frame\_1,frame\_2),axis=0,join='inner')

Out[17]:

mattias jennifer

salary 1000 2000

expenditure 279 1123

surplus 721 877

salary per annum 12000 24000

expenditure per annum 3348 13476

surplus per annum 8652 10524

ag\_1=pd.DataFrame(np.floor(np.random.randn(4,4)),index=list('gfas'),columns=list('xcvb'))

ag\_2=pd.DataFrame(np.floor(np.random.randint(2,8,12).reshape(4,3)),index=['xu','de','jiang','hu'],columns=list('xvb'))

pd.concat((ag\_1,ag\_2),keys=['Gr','A'],axis=1,join\_axes=[ag\_1.index])

Out[64]:

Gr A

x c v b x v b

g -2.0 0.0 0.0 1.0 NaN NaN NaN

f -2.0 -1.0 0.0 -1.0 NaN NaN NaN

a -2.0 0.0 0.0 2.0 NaN NaN NaN

s 0.0 -2.0 -1.0 -2.0 NaN NaN NaN

ag\_1=pd.DataFrame(np.floor(np.random.randn(4,4)),index=list('gfas'),columns=list('xcvb'))

ag\_2=pd.DataFrame(np.floor(np.random.randint(2,8,12).reshape(4,3)),index=['xu','de','jiang','hu'],columns=list('xvb'))

pd.concat((ag\_1,ag\_2),keys=['Gr','A'],axis=1,join\_axes=[ag\_2.index])

Out[65]:

Gr A

x c v b x v b

xu NaN NaN NaN NaN 3.0 3.0 4.0

de NaN NaN NaN NaN 5.0 2.0 6.0

jiang NaN NaN NaN NaN 7.0 2.0 3.0

hu NaN NaN NaN NaN 7.0 2.0 6.0

ag\_1=pd.DataFrame(np.floor(np.random.randn(4,4)),index=list('gfas'),columns=list('xcvb'))

ag\_2=pd.DataFrame(np.floor(np.random.randint(2,8,12).reshape(4,3)),index=['xu','de','jiang','hu'],columns=list('xvb'))

pd.concat((ag\_1,ag\_2),keys=['Gr','A'],axis=0,join\_axes=[ag\_2.columns])

Out[67]:

x v b

Gr g -1.0 -1.0 0.0

f -1.0 -1.0 -1.0

a 1.0 -1.0 1.0

s 0.0 -1.0 -1.0

A xu 7.0 3.0 3.0

de 6.0 6.0 3.0

jiang 2.0 7.0 5.0

hu 7.0 5.0 7.0

ag\_1=pd.DataFrame(np.floor(np.random.randn(4,4)),index=list('gfas'),columns=list('xcvb'))

ag\_2=pd.DataFrame(np.floor(np.random.randint(2,8,12).reshape(4,3)),index=['xu','de','jiang','hu'],columns=list('xvb'))

pd.concat((ag\_1,ag\_2),keys=['Gr','A'],axis=0,join\_axes=[ag\_1.columns])

Out[68]:

x c v b

Gr g 0.0 0.0 -1.0 2.0

f 0.0 -1.0 0.0 -2.0

a 0.0 -1.0 -2.0 0.0

s -1.0 -2.0 0.0 -1.0

A xu 2.0 NaN 2.0 6.0

de 6.0 NaN 4.0 2.0

jiang 7.0 NaN 6.0 5.0

hu 4.0 NaN 7.0 6.0

Concat的参数详解：

|  |  |
| --- | --- |
| objs | 参与连接的pandas对象的列表或者元组或者字典。是唯一必需的参数。 |
| axis | 指明按那个轴进行连接，默认为0 |
| join | 指明连接方式，“inner”或”outer”，默认“outer”。指明其他轴向上的索引是按交集（“inner”）还是并集(“outer”)进行合并。 |
| keys | 与连接对象有关的值，用于形成连接轴向上的层次化索引。可以是任意值的列表或数组、元组数组、数组列表等。 |
| join\_axes | 指定根据那个数据框的索引来对齐数据，这个索引不参与并\交运算 |
| Names | 给层索引每一层命名 |
| Ignore\_index | 不保留连接轴上的索引，产生一组新索引 |

**给数据打补丁：**

尽管我们学习各种各种的数据合并与连接（merge、join、concatenation等）。然而他们都仅仅是对索引的直接处理。很难对Series或者数据框所包含的数据直接处理。

通过np.where,可以实现数值的补空合并。

ser\_1=pd.Series([np.nan,45,np.nan,3.9,90,np.nan],index=list('abcdef'))

ser\_2=pd.Series(np.arange(len(ser\_1)),index=list('abcdef'))

ser\_1

Out[73]:

a NaN

b 45.0

c NaN

d 3.9

e 90.0

f NaN

dtype: float64

ser\_2

Out[74]:

a 0

b 1

c 2

d 3

e 4

f 5

dtype: int32

pd.Series(np.where(pd.isnull(ser\_1),ser\_2,ser\_1),index=list('abcdef'))

Out[79]:

a 0.0

b 45.0

c 2.0

d 3.9

e 90.0

f 5.0

dtype: float64

通过combine\_first来实现相同功能：

ser\_1.combine\_first(ser\_2)

Out[80]:

a 0.0

b 45.0

c 2.0

d 3.9

e 90.0

f 5.0

dtype: float64

#拿fra\_2值补fra\_1

fra\_1=pd.DataFrame({'a':[1.,np.nan,5.,np.nan],'b':[np.nan,2.,np.nan,6.],'c':range(2,9,2)})

fra\_2=pd.DataFrame({'a':[3,np.nan,6,3,8],'b':[np.nan,np.nan,2,6,8]})

fra\_1

Out[84]:

a b c

0 1.0 NaN 2

1 NaN 2.0 4

2 5.0 NaN 6

3 NaN 6.0 8

fra\_2

Out[85]:

a b

0 3.0 NaN

1 NaN NaN

2 6.0 2.0

3 3.0 6.0

4 8.0 8.0

fra\_1.combine\_first(fra\_2)

Out[86]:

a b c

0 1.0 NaN 2.0

1 NaN 2.0 4.0

2 5.0 2.0 6.0

3 3.0 6.0 8.0

4 8.0 8.0 NaN

#拿fra\_2值补fra\_1

# 数据的重塑（仔细读注解额IMG_256）

数据的重塑其本质就是对数据表格进行重排。所用到的函数通常被称为重塑函数或轴向旋转函数。

重塑层次化索引：

带有层次化索引数据框的重塑主要是通过以下函数来实现的：

Stack：将列索引旋转为行索引

Unstack：将行索引旋转为列索引

**注意：以上只适合重索引数据框**

我们先从一个简单的数据框开始。

data\_1=pd.DataFrame(np.arange(6).reshape(2,3),index=pd.Index(['XDL\_1','XDL\_2'],name='education\_group'),columns=pd.Index(['one','three','two'],name='nr.'))

data\_1

Out[4]:

nr. one three two

education\_group

XDL\_1 0 1 2

XDL\_2 3 4 5

data\_1.stack()

Out[5]:

education\_group nr.

XDL\_1 one 0

three 1

two 2

XDL\_2 one 3

three 4

two 5

dtype: int32

#**对于没有重索引的数据框，用stack方法会实现列索引成为内层行索引**

data\_1.unstack()

Out[14]:

nr. education\_group

one XDL\_1 0

XDL\_2 3

three XDL\_1 1

XDL\_2 4

two XDL\_1 2

XDL\_2 5

dtype: int32

#**对于没有重索引的数据框，用unstack方法会实现列索引成为外层行索引**

data\_double\_ser=data\_1.stack()

**#生成重索引series**

data\_double\_ser

Out[21]:

education\_group nr.

XDL\_1 one 0

three 1

two 2

XDL\_2 one 3

three 4

two 5

dtype: int32

data\_double\_ser.unstack()

Out[6]:

nr. one three two

education\_group

XDL\_1 0 1 2

XDL\_2 3 4 5

**#通过unstack把内层行索引旋转到列索引，默认旋转内层级行索引到列索引。**

data\_1.stack().unstack(0)

Out[10]:

education\_group XDL\_1 XDL\_2

nr.

one 0 3

three 1 4

two 2 5

**#默认按内层索引操作，我们可以指定操作级别。这里的‘0’代表外层级别。也可以用外层索引名来代替‘0’**

data\_1.stack().unstack('education\_group')

Out[13]:

education\_group XDL\_1 XDL\_2

nr.

one 0 3

three 1 4

two 2 5

**#指定重索引行外层级，并通过unstack把外层行索引旋转到列索引**

s1=pd.Series([0,3,4,-3],index=list('abcd'))

s2=pd.Series([6,2,9],index=list('bca'))

data\_ser\_double=pd.concat((s1,s2),keys=['one','two'])

data\_ser\_double

Out[25]:

one a 0

b 3

c 4

d -3

two b 6

c 2

a 9

dtype: int64

data\_ser\_double.unstack()

Out[26]:

a b c d

one 0.0 3.0 4.0 -3.0

two 9.0 6.0 2.0 NaN

#如果S1和S2的索引数目不等或者数目相等但索引内容不同都会产生空值，因此我们要求S1和S2的索引必须完全一致，才能在运算中不产生空值！！

#产生空值的情况下，如果我们设置dropna=True（默认）那么上面的运算可逆。如果我们设置dropna=False的情况下，上面的运算时不可逆的。

data\_ser\_double.unstack().stack()

Out[28]:

one a 0.0

b 3.0

c 4.0

d -3.0

two a 9.0

b 6.0

c 2.0

dtype: float64

data\_ser\_double.unstack().stack(dropna=False)

Out[29]:

one a 0.0

b 3.0

c 4.0

d -3.0

two a 9.0

b 6.0

c 2.0

d NaN

dtype: float64

**移除重复数据**

data=pd.DataFrame({'k1':['one']\*3+['two']\*4,'k2':[1,2,3,4,5,6,7]})

data

Out[8]:

k1 k2

0 one 1

1 one 2

2 one 3

3 two 4

4 two 5

5 two 6

6 two 7

利用duplicated可以判断重复行

#默认k1和k2列必须都重复才算重复行

data.duplicated()

Out[5]:

0 False

1 False

2 False

3 False

4 False

5 False

6 False

dtype: bool

data=pd.DataFrame({'k1':['one']\*3+['two']\*4,'k2':[2,3,2,3,6,6,6]})

data

Out[8]:

k1 k2

0 one 2

1 one 3

2 one 2

3 two 3

4 two 6

5 two 6

6 two 6

data.duplicated()

Out[9]:

0 False

1 False

2 True

3 False

4 False

5 True

6 True

dtype: bool

#注意，原项不会判断为True，只有真正的重复项才会，如上例子，红色不会判断为True，蓝色才会判断为True。

#通过drop\_duplicates方法，可以移除重复行，返回一个没有重复行的DataFrame。

data.drop\_duplicates()

Out[11]:

k1 k2

0 one 2

1 one 3

3 two 3

4 two 6

#可以指定某一列或几列中行重复就算重复行

data['k3'],data['k4'],data['k5']=[0,0,0,0,3,6,7],[4,8,6,7,6,7,7],[2,2,4,4,6,6,8]

data

Out[20]:

k1 k2 k3 k4 k5

0 one 2 0 4 2

1 one 3 0 8 2

2 one 2 0 6 4

3 two 3 0 7 4

4 two 6 3 6 6

5 two 6 6 7 6

6 two 6 7 7 8

data.drop\_duplicates(['k1'])

Out[22]:

k1 k2 k3 k4 k5

0 one 2 0 4 2

3 two 3 0 7 4

data.drop\_duplicates(['k1','k2'])

Out[24]:

k1 k2 k3 k4 k5

0 one 2 0 4 2

1 one 3 0 8 2

3 two 3 0 7 4

4 two 6 3 6 6

data

Out[27]:

k1 k2 k3 k4 k5

0 one 2 0 4 2

1 one 3 0 8 2

2 one 2 0 6 4

3 two 3 0 7 4

4 two 6 3 6 6

5 two 6 6 7 6

6 two 6 7 7 8

data.drop\_duplicates(['k1'],keep='last')

Out[29]:

k1 k2 k3 k4 k5

2 one 2 0 6 4

6 two 6 7 7 8

data.duplicated(['k1'],keep='last')

Out[34]:

0 True

1 True

2 False

3 True

4 True

5 True

6 False

dtype: bool

#默认保留重复行的第一行，也可以设置保留最后一个，keep='last',删除其他所有重复行。

**利用字典映射对数据进行操作：**

data\_1=pd.DataFrame({'food':['bacon火腿','sausage红肠','pulled pork手撕肉','sirloin牛里脊肉','beef jerky牛肉干','mutton shashlik烤羊肉串','Chicken salad鸡肉','Minced chicken鸡肉泥','dried squids鱿鱼干','dried fish鱼干','sausage红肠','bacon火腿','sausage红肠','bacon火腿','Chicken salad鸡肉','dried squids鱿鱼干','dried fish鱼干','mutton shashlik烤羊肉串','beef jerky牛肉干','Minced chicken鸡肉泥','pulled pork手撕肉','sirloin牛里脊肉','mutton shashlik烤羊肉串','Chicken salad鸡肉','Minced chicken鸡肉泥','bacon火腿','sausage红肠','pulled pork手撕肉','sirloin牛里脊肉','beef jerky牛肉干','mutton shashlik烤羊肉串','Chicken salad鸡肉','Minced chicken鸡肉泥','dried squids鱿鱼干','dried fish鱼干','sausage红肠','bacon火腿','sausage红肠','bacon火腿','Chicken salad鸡肉','dried squids鱿鱼干','dried fish鱼干','mutton shashlik烤羊肉串','beef jerky牛肉干','sirloin牛里脊肉','mutton shashlik烤羊肉串','Chicken salad鸡肉','Minced chicken鸡肉泥'],'weight(g)':abs((np.random.normal(0,1,size=48)\*24+10)).astype(np.int32)})

data\_1

Out[4]:

food weight(g)

0 bacon火腿 27

1 sausage红肠 41

2 pulled pork手撕肉 16

3 sirloin牛里脊肉 12

4 beef jerky牛肉干 0

5 mutton shashlik烤羊肉串 7

6 Chicken salad鸡肉 31

7 Minced chicken鸡肉泥 14

8 dried squids鱿鱼干 3

9 dried fish鱼干 38

10 sausage红肠 34

11 bacon火腿 12

12 sausage红肠 9

13 bacon火腿 23

14 Chicken salad鸡肉 34

15 dried squids鱿鱼干 1

16 dried fish鱼干 19

17 mutton shashlik烤羊肉串 6

18 beef jerky牛肉干 0

19 Minced chicken鸡肉泥 3

20 pulled pork手撕肉 22

21 sirloin牛里脊肉 9

22 mutton shashlik烤羊肉串 14

23 Chicken salad鸡肉 34

24 Minced chicken鸡肉泥 37

25 bacon火腿 27

26 sausage红肠 0

27 pulled pork手撕肉 32

28 sirloin牛里脊肉 0

29 beef jerky牛肉干 38

30 mutton shashlik烤羊肉串 16

31 Chicken salad鸡肉 35

32 Minced chicken鸡肉泥 4

33 dried squids鱿鱼干 5

34 dried fish鱼干 3

35 sausage红肠 26

36 bacon火腿 12

37 sausage红肠 2

38 bacon火腿 5

39 Chicken salad鸡肉 11

40 dried squids鱿鱼干 6

41 dried fish鱼干 24

42 mutton shashlik烤羊肉串 26

43 beef jerky牛肉干 56

44 sirloin牛里脊肉 2

45 mutton shashlik烤羊肉串 5

46 Chicken salad鸡肉 12

47 Minced chicken鸡肉泥 13

我们发现数据框‘food’列中有很多重复。如果我们给数据假如新列，并要求新列与food存在某种逻辑关系。例如给‘food’标记食材来源。

通过map函数可以很轻松实现上面的要求。

meat\_source={'bacon火腿':'猪','sausage红肠':'猪','pulled pork手撕肉':'猪','sirloin牛里脊肉':'牛','beef jerky牛肉干':'牛','mutton shashlik烤羊肉串':'羊','chicken salad鸡肉':'鸡','minced chicken鸡肉泥':'鸡','dried squids鱿鱼干':'鱿鱼','dried fish鱼干':'鱼'}

data\_1['animal']=data\_1['food'].map(meat\_source)

data\_1

Out[25]:

food weight(g) animal

0 bacon火腿 17 猪

1 sausage红肠 38 猪

2 pulled pork手撕肉 3 猪

3 sirloin牛里脊肉 12 牛

4 beef jerky牛肉干 31 牛

5 mutton shashlik烤羊肉串 16 羊

6 Chicken salad鸡肉 17 NaN

7 Minced chicken鸡肉泥 14 NaN

8 dried squids鱿鱼干 12 鱿鱼

9 dried fish鱼干 7 鱼

10 sausage红肠 25 猪

11 bacon火腿 30 猪

12 sausage红肠 12 猪

13 bacon火腿 32 猪

14 Chicken salad鸡肉 10 NaN

15 dried squids鱿鱼干 42 鱿鱼

16 dried fish鱼干 8 鱼

17 mutton shashlik烤羊肉串 13 羊

18 beef jerky牛肉干 34 牛

19 Minced chicken鸡肉泥 44 NaN

20 pulled pork手撕肉 3 猪

21 sirloin牛里脊肉 25 牛

22 mutton shashlik烤羊肉串 4 羊

23 Chicken salad鸡肉 39 NaN

24 Minced chicken鸡肉泥 20 NaN

25 bacon火腿 5 猪

26 sausage红肠 38 猪

27 pulled pork手撕肉 28 猪

28 sirloin牛里脊肉 12 牛

29 beef jerky牛肉干 21 牛

30 mutton shashlik烤羊肉串 13 羊

31 Chicken salad鸡肉 13 NaN

32 Minced chicken鸡肉泥 26 NaN

33 dried squids鱿鱼干 43 鱿鱼

34 dried fish鱼干 14 鱼

35 sausage红肠 25 猪

36 bacon火腿 19 猪

37 sausage红肠 24 猪

38 bacon火腿 57 猪

39 Chicken salad鸡肉 25 NaN

40 dried squids鱿鱼干 38 鱿鱼

41 dried fish鱼干 18 鱼

42 mutton shashlik烤羊肉串 54 羊

43 beef jerky牛肉干 24 牛

44 sirloin牛里脊肉 15 牛

45 mutton shashlik烤羊肉串 17 羊

46 Chicken salad鸡肉 61 NaN

47 Minced chicken鸡肉泥 29 NaN

#map的实质是把一种映射法则应用到所制定的列上，比如这里的‘food’列。这种映射法则通常通过字典与函数来体现。

#map把与meat\_source字典的键相对应food值一个个投影到字典建对应的值然后把投影结果赋值给data\_1[‘animal’]

#我们发现‘animal’列有空值。原因是meat\_source中的键与‘food’列值不完全一致。

#通过map(str.lower)把food列中的值的第一个字母转变成小写，这样meat\_source中的键与‘food’列值完全一致，这样Nan会自动消失。

data\_1['animal']=data\_1['food'].map(lambda x:meat\_source[x.lower()])

或

data\_1['animal']=data\_1['food'].map(str.lower).map(meat\_source)

data\_1

Out[23]:

food weight(g) animal

0 bacon火腿 17 猪

1 sausage红肠 38 猪

2 pulled pork手撕肉 3 猪

3 sirloin牛里脊肉 12 牛

4 beef jerky牛肉干 31 牛

5 mutton shashlik烤羊肉串 16 羊

6 Chicken salad鸡肉 17 鸡

7 Minced chicken鸡肉泥 14 鸡

8 dried squids鱿鱼干 12 鱿鱼

9 dried fish鱼干 7 鱼

10 sausage红肠 25 猪

11 bacon火腿 30 猪

12 sausage红肠 12 猪

13 bacon火腿 32 猪

14 Chicken salad鸡肉 10 鸡

15 dried squids鱿鱼干 42 鱿鱼

16 dried fish鱼干 8 鱼

17 mutton shashlik烤羊肉串 13 羊

18 beef jerky牛肉干 34 牛

19 Minced chicken鸡肉泥 44 鸡

20 pulled pork手撕肉 3 猪

21 sirloin牛里脊肉 25 牛

22 mutton shashlik烤羊肉串 4 羊

23 Chicken salad鸡肉 39 鸡

24 Minced chicken鸡肉泥 20 鸡

25 bacon火腿 5 猪

26 sausage红肠 38 猪

27 pulled pork手撕肉 28 猪

28 sirloin牛里脊肉 12 牛

29 beef jerky牛肉干 21 牛

30 mutton shashlik烤羊肉串 13 羊

31 Chicken salad鸡肉 13 鸡

32 Minced chicken鸡肉泥 26 鸡

33 dried squids鱿鱼干 43 鱿鱼

34 dried fish鱼干 14 鱼

35 sausage红肠 25 猪

36 bacon火腿 19 猪

37 sausage红肠 24 猪

38 bacon火腿 57 猪

39 Chicken salad鸡肉 25 鸡

40 dried squids鱿鱼干 38 鱿鱼

41 dried fish鱼干 18 鱼

42 mutton shashlik烤羊肉串 54 羊

43 beef jerky牛肉干 24 牛

44 sirloin牛里脊肉 15 牛

45 mutton shashlik烤羊肉串 17 羊

46 Chicken salad鸡肉 61 鸡

47 Minced chicken鸡肉泥 29 鸡

添加amount=weight\*price项

product=['bacon火腿','sausage红肠','pulled pork手撕肉','sirloin牛里脊肉','beef jerky牛肉干','mutton shashlik烤羊肉串','Chicken salad鸡肉','Minced chicken鸡肉泥','dried squids鱿鱼干','dried fish鱼干']

price=[1,1.5,2,2.3,2.6,3.6,2,2.6,8,9.9]

get\_Data={k:p\*i for i,k in zip(list(data\_1['weight(g)'].values),range(len(data\_1['weight(g)']))) for m,p in zip(product,price) if m==data\_1['food'][k]}

data\_1['key']=np.arange(48)

data\_1['anount']=data\_1['key'].map(get\_Data)

data\_1.reindex(columns=['food','weight(g)', 'animal', 'anount','key' ])

Out[108]:

food weight(g) animal anount key

0 bacon火腿 17 猪 17.0 0

1 sausage红肠 38 猪 57.0 1

2 pulled pork手撕肉 3 猪 6.0 2

3 sirloin牛里脊肉 12 牛 27.6 3

4 beef jerky牛肉干 31 牛 80.6 4

5 mutton shashlik烤羊肉串 16 羊 57.6 5

6 Chicken salad鸡肉 17 鸡 34.0 6

7 Minced chicken鸡肉泥 14 鸡 36.4 7

8 dried squids鱿鱼干 12 鱿鱼 96.0 8

9 dried fish鱼干 7 鱼 69.3 9

10 sausage红肠 25 猪 37.5 10

11 bacon火腿 30 猪 30.0 11

12 sausage红肠 12 猪 18.0 12

13 bacon火腿 32 猪 32.0 13

14 Chicken salad鸡肉 10 鸡 20.0 14

15 dried squids鱿鱼干 42 鱿鱼 336.0 15

16 dried fish鱼干 8 鱼 79.2 16

17 mutton shashlik烤羊肉串 13 羊 46.8 17

18 beef jerky牛肉干 34 牛 88.4 18

19 Minced chicken鸡肉泥 44 鸡 114.4 19

20 pulled pork手撕肉 3 猪 6.0 20

21 sirloin牛里脊肉 25 牛 57.5 21

22 mutton shashlik烤羊肉串 4 羊 14.4 22

23 Chicken salad鸡肉 39 鸡 78.0 23

24 Minced chicken鸡肉泥 20 鸡 52.0 24

25 bacon火腿 5 猪 5.0 25

26 sausage红肠 38 猪 57.0 26

27 pulled pork手撕肉 28 猪 56.0 27

28 sirloin牛里脊肉 12 牛 27.6 28

29 beef jerky牛肉干 21 牛 54.6 29

30 mutton shashlik烤羊肉串 13 羊 46.8 30

31 Chicken salad鸡肉 13 鸡 26.0 31

32 Minced chicken鸡肉泥 26 鸡 67.6 32

33 dried squids鱿鱼干 43 鱿鱼 344.0 33

34 dried fish鱼干 14 鱼 138.6 34

35 sausage红肠 25 猪 37.5 35

36 bacon火腿 19 猪 19.0 36

37 sausage红肠 24 猪 36.0 37

38 bacon火腿 57 猪 57.0 38

39 Chicken salad鸡肉 25 鸡 50.0 39

40 dried squids鱿鱼干 38 鱿鱼 304.0 40

41 dried fish鱼干 18 鱼 178.2 41

42 mutton shashlik烤羊肉串 54 羊 194.4 42

43 beef jerky牛肉干 24 牛 62.4 43

44 sirloin牛里脊肉 15 牛 34.5 44

45 mutton shashlik烤羊肉串 17 羊 61.2 45

46 Chicken salad鸡肉 61 鸡 122.0 46

47 Minced chicken鸡肉泥 29 鸡 75.4 47

灵活的替换值方法：

前面我们讲过fillna替换空值，还有notnull也可以实现空值替换。这里我们继续推荐一种另外的替换方法-replace法。这种方法不仅简单而且还十分灵活。

#把那些标记为空值的数据替换成pandas能够理解的Na值

data=pd.Series([1.,-999.,2.,999.,-1000.,3.])

data

Out[115]:

0 1.0

1 -999.0

2 2.0

3 999.0

4 -1000.0

5 3.0

dtype: float64

#假如-999是空值标记

data.replace(-999,np.nan)

Out[117]:

0 1.0

1 NaN

2 2.0

3 999.0

4 -1000.0

5 3.0

dtype: float64

#也可以一次替换多值

data.replace([-999,-1000],np.nan)

Out[119]:

0 1.0

1 NaN

2 2.0

3 999.0

4 NaN

5 3.0

dtype: float64

data\_1=[23,'invalue',34]

data\_1=pd.Series(data\_1)

data\_1

Out[124]:

0 23

1 invalue

2 34

dtype: object

data\_1.replace('invalue',2)

Out[122]:

0 23

1 2

2 34

dtype: int64

data\_1=pd.Series([1.,-999.,2.,-999.,-1000.,3.])

data\_1

Out[129]:

0 1.0

1 -999.0

2 2.0

3 -999.0

4 -1000.0

5 3.0

dtype: float64

data\_1.replace([-999,-1000],[np.nan,0])

Out[126]:

0 1.0

1 NaN

2 2.0

3 NaN

4 0.0

5 3.0

dtype: float64

#也可以表示成字典形式

data\_1.replace({-999:np.nan,-1000:0})

Out[130]:

0 1.0

1 NaN

2 2.0

3 NaN

4 0.0

5 3.0

dtype: float64

**重命名轴索引**

data=pd.DataFrame(np.arange(12).reshape(3,4),index=['a','b','c'],columns=['c1','c2','c3','c4'])

#Map可以通过字典把要处理的列（键值）映射到对应的值，也可以通过函数把相应的列映射到对应的值

#通过str.upper函数map把行索引列映射到大写行索引列。

data.index=data.index.map(str.upper)

data

Out[137]:

c1 c2 c3 c4

A 0 1 2 3

B 4 5 6 7

C 8 9 10 11

#可以用rename对轴索引进行修改

data.rename(index=str.title,columns=str.upper)

Out[5]:

C1 C2 C3 C4

A 0 1 2 3

B 4 5 6 7

C 8 9 10 11

#rename通过字典的帮助可实现对部分标签的更新

data.rename(index={'a':'Sun','b':'Zhang','c':'Wang'},columns={'c2':'CNr.'})

data.rename(index={'b':'Zhang','c':'Wang'},inplace=True)

data

Out[11]:

c1 c2 c3 c4

a 0 1 2 3

Zhang 4 5 6 7

Wang 8 9 10 11

#通过inplace=True，可以改变原数据

数据的离散和面元划分

有些时候我们需要把连续数据离散化或拆分成面元。面元指的是一段按组拆分后的数据。面元化就是把数据进行分组。

例： ages=[20,22,25,27,26,31,78,45,20,19,67,34,56,66,59,43]

#面元化的实现

bin\_nr=[min(ages),35,46,57,63,max(ages)]

bin\_nr

Out[15]: [19, 35, 46, 57, 63, 78]

cut\_1=pd.cut(ages,bin\_nr)

cut\_1

Out[17]:

[(19, 35], (19, 35], (19, 35], (19, 35], (19, 35], ..., (19, 35], (46, 57], (63, 78], (57, 63], (35, 46]]

Length: 16

Categories (5, interval[int64]): [(19, 35] < (35, 46] < (46, 57] < (57, 63] < (63, 78]]

#对数据分组后的编号是：

ages

Out[28]: [20, 22, 25, 27, 26, 31, 78, 45, 20, 19, 67, 34, 56, 66, 59, 43]

cut\_1.codes

Out[24]: array([ 0, 0, 0, 0, 0, 0, 4, 1, 0, -1, 4, 0, 2, 4, 3, 1], dtype=int8)

我们发现分组编号中有-1值，这是‘19’这个值造成的，因为我们的分组组限不包括‘19’这个值。解决办法有两个：

* 改变bin\_nr

bin\_nr=[18,min(ages),35,46,57,63,max(ages)]

cut\_1=pd.cut(ages,bin\_nr)

cut\_1

Out[35]:

[(19, 35], (19, 35], (19, 35], (19, 35], (19, 35], ..., (19, 35], (46, 57], (63, 78], (57, 63], (35, 46]]

Length: 16

Categories (6, interval[int64]): [(18, 19] < (19, 35] < (35, 46] < (46, 57] < (57, 63] < (63, 78]]

cut\_1.codes

Out[38]: array([1, 1, 1, 1, 1, 1, 5, 2, 1, 0, 5, 1, 3, 5, 4, 2], dtype=int8)

* 修改参数：

bin\_nr=[min(ages),35,46,57,63,max(ages)]

pd.cut(ages,bin\_nr,include\_lowest=True)

Out[41]:

[(18.999, 35.0], (18.999, 35.0], (18.999, 35.0], (18.999, 35.0], (18.999, 35.0], ..., (18.999, 35.0], (46.0, 57.0], (63.0, 78.0], (57.0, 63.0], (35.0, 46.0]]

Length: 16

Categories (5, interval[float64]): [(18.999, 35.0] < (35.0, 46.0] < (46.0, 57.0] < (57.0, 63.0] < (63.0, 78.0]]

cut2=pd.cut(ages,bin\_nr,include\_lowest=True)

cut2.codes

Out[43]: array([0, 0, 0, 0, 0, 0, 4, 1, 0, 0, 4, 0, 2, 4, 3, 1], dtype=int8)

ages

Out[45]: [20, 22, 25, 27, 26, 31, 78, 45, 20, 19, 67, 34, 56, 66, 59, 43]

#显示分组组限：

cut\_1.categories

Out[31]:

IntervalIndex([(19, 35], (35, 46], (46, 57], (57, 63], (63, 78]]

closed='right',

dtype='interval[int64]')

#统计各组频数

pd.value\_counts(cut2)

Out[55]:

(18.999, 35.0] 9

(63.0, 78.0] 3

(35.0, 46.0] 2

(57.0, 63.0] 1

(46.0, 57.0] 1

dtype: int64

#right=False可改变组限区间闭区间的位置；

bin\_nr=[min(ages),35,46,57,63,max(ages),79]

pd.cut(ages,bin\_nr,right=False)

Out[50]:

[[19, 35), [19, 35), [19, 35), [19, 35), [19, 35), ..., [19, 35), [46, 57), [63, 78), [57, 63), [35, 46)]

Length: 16

Categories (6, interval[int64]): [[19, 35) < [35, 46) < [46, 57) < [57, 63) < [63, 78) < [78, 79)]

cut4=pd.cut(ages,bin\_nr,right=False)

cut4.codes

Out[52]: array([0, 0, 0, 0, 0, 0, 5, 1, 0, 0, 4, 0, 2, 4, 3, 1], dtype=int8)

#通过labels给面元设置名称

bin\_nr=[min(ages),35,46,57,63,max(ages)]

pd.cut(ages,bin\_nr,include\_lowest=True)

cut2=pd.cut(ages,bin\_nr,include\_lowest=True,labels=Name\_group)

cut2

Out[62]:

[青年, 青年, 青年, 青年, 青年, ..., 青年, 中年, 少老年, 老中年, 壮年]

Length: 16

Categories (5, object): [青年 < 壮年 < 中年 < 老中年 < 少老年]

#如果给bins参数赋整数值，意味着传入的数值是面元数量，计算机会根据整个数据最大值和最小值计算等长面元。

bin\_nr=[min(ages),35,46,57,63,max(ages)]

pd.cut(ages,bin\_nr,include\_lowest=True)

cut2=pd.cut(ages,6,include\_lowest=True)

cut2

Out[64]:

[(18.94, 28.833], (18.94, 28.833], (18.94, 28.833], (18.94, 28.833], (18.94, 28.833], ..., (28.833, 38.667], (48.5, 58.333], (58.333, 68.167], (58.333, 68.167], (38.667, 48.5]]

Length: 16

Categories (6, interval[float64]): [(18.94, 28.833] < (28.833, 38.667] < (38.667, 48.5] < (48.5, 58.333] < (58.333, 68.167] < (68.167, 78.0]]

pd.value\_counts(cut2)

Out[68]:

(18.94, 28.833] 7

(58.333, 68.167] 3

(38.667, 48.5] 2

(28.833, 38.667] 2

(68.167, 78.0] 1

(48.5, 58.333] 1

dtype: int64

#等数据点面元化

data\_random=np.random.randn(1000)

cut\_12=pd.qcut(data\_random,8)

cut\_12

Out[71]:

[(-1.193, -0.735], (-0.043, 0.297], (0.297, 0.628], (-1.193, -0.735], (-0.365, -0.043], ..., (-3.453, -1.193], (-0.365, -0.043], (0.628, 1.122], (-0.735, -0.365], (1.122, 2.735]]

Length: 1000

Categories (8, interval[float64]): [(-3.453, -1.193] < (-1.193, -0.735] < (-0.735, -0.365] < (-0.365, -0.043] < (-0.043, 0.297] < (0.297, 0.628] < (0.628, 1.122] < (1.122, 2.735]]

pd.value\_counts(cut\_12)

Out[72]:

(1.122, 2.735] 125

(0.628, 1.122] 125

(0.297, 0.628] 125

(-0.043, 0.297] 125

(-0.365, -0.043] 125

(-0.735, -0.365] 125

(-1.193, -0.735] 125

(-3.453, -1.193] 125

dtype: int64

#自定义分位数（必须是0-1之间的分位数，包括端点，因为这里是按百分比分位）

cut\_12=pd.qcut(data\_random,[0,0.3,0.6,0.85,1])

cut\_12

Out[80]:

[(-3.453, -0.573], (-0.573, 0.218], (0.218, 0.987], (-3.453, -0.573], (-0.573, 0.218], ..., (-3.453, -0.573], (-0.573, 0.218], (0.218, 0.987], (-3.453, -0.573], (0.987, 2.735]]

Length: 1000

Categories (4, interval[float64]): [(-3.453, -0.573] < (-0.573, 0.218] < (0.218, 0.987] < (0.987, 2.735]]

pd.value\_counts(cut\_12)

Out[81]:

(-0.573, 0.218] 300**(0-0.3)**

(-3.453, -0.573] 300**(0.3-0.6)**

(0.218, 0.987] 250**(0.6-0.85)**

(0.987, 2.735] 150**(0.85-1)**

dtype: int64

**检测和过滤异常值**

Outlier（out来了IMG_256）异常值。如何检测那些值是异常值呢？我们来看一个例子：

import pandas as pd

import numpy as np

np.random.seed(12876)

data=pd.DataFrame(np.random.randn(1000,4))

data.describe()

Out[5]:

0 1 2 3

count 1000.000000 1000.000000 1000.000000 1000.000000

mean -0.032748 0.017600 0.005716 0.052615

std 1.029352 1.013877 0.974347 0.990795

min -3.558590 -3.399723 -3.714028 -3.113867

25% -0.731609 -0.654785 -0.703925 -0.604366

50% -0.079017 0.001919 0.020646 0.041902

75% 0.686783 0.688564 0.685262 0.703313

max 3.040923 2.809213 2.827195 3.389950

#根据各列的最大值和最小值对各列数值进行过滤

c\_1=data[2]

c\_1[np.abs(c\_1)>3]

Out[8]:

382 -3.002094

466 -3.714028

Name: 2, dtype: float64

#对整个数据框中所有>3的值进行过滤

**Step1:**

np.abs(data)>3

Out[43]:

0 1 2 3

0 False False False False

1 False False False False

2 False False False False

3 False False False False

4 False False False False

5 False False False False

6 False False False False

7 False False False False

8 False False False False

9 False False False False

10 False False False False

11 False False False False

12 False False False False

13 False False False False

14 False False False False

15 False False False False

16 False False False False

17 False False False False

18 False False False False

19 False False False False

20 False False False False

21 False False False False

22 False False False False

23 False False False False

24 False False False False

25 False False False False

26 False False False False

27 False False False False

28 False False False False

29 False False False False

.. ... ... ... ...

970 False False False False

971 False False False False

972 False False False False

973 False False False False

974 False False False False

975 False False False False

976 False False False False

977 False False False False

978 False False False False

979 False False False False

980 False False False False

981 False False False False

982 False False False False

983 False False False False

984 False False False False

985 False False False False

986 False False False False

987 False False False False

988 False False False False

989 False False False False

990 False False False False

991 False False False False

992 False False False False

993 False False False False

994 False False False False

995 False False False False

996 False False False False

997 False False False False

998 False False False False

999 False False False False

[1000 rows x 4 columns]

**Step2:**

#逐列按行，把含有true值的每一行转化成true，把不含的行转成false，返回series

(np.abs(data)>3).any(1)

Out[46]:

0 False

1 False

2 False

3 False

4 False

5 False

6 False

7 False

8 False

9 False

10 False

11 False

12 False

13 False

14 False

15 False

16 False

17 False

18 False

19 False

20 False

21 False

22 False

23 False

24 False

25 False

26 False

27 False

28 False

29 False

970 False

971 False

972 False

973 False

974 False

975 False

976 False

977 False

978 False

979 False

980 False

981 False

982 False

983 False

984 False

985 False

986 False

987 False

988 False

989 False

990 False

991 False

992 False

993 False

994 False

995 False

996 False

997 False

998 False

999 False

Length: 1000, dtype: bool

**Step3:**

#放到索引位置直接索引行

data[(np.abs(data)>3).any(1)]

Out[16]:

0 1 2 3

82 -1.535879 -1.077352 1.293144 3.058112

270 -1.131598 -0.273447 -0.979915 -3.113867

281 0.022546 -0.526429 0.361507 3.021792

283 -0.469536 0.122044 -1.188901 3.389950

327 -0.676235 0.155461 -0.706008 3.026979

340 3.040923 1.046905 -0.375365 -0.572458

358 -3.558590 -1.343718 1.411491 1.001404

382 -1.947519 0.556118 -3.002094 0.701169

466 -1.817789 -0.187283 -3.714028 -0.049148

859 0.906164 -3.399723 -0.360720 0.245022

注意：

1. 参数bool\_only意味着只接受布尔值，如果是空值的话，计算机会检验数据，然后仅仅接收是布尔值的数据。如果是False，非布尔值数据参与比较，且自动被设置为True值参与比较。例如：

TM\_1=TM.astype(object)

TM\_1.loc[0]=3

TM\_1

Out[60]:

0 1 2 3

0 3 3 3 3

1 False False False False

2 False False False False

3 False False False False

4 False False False False

5 False False False False

6 False False False False

7 False False False False

8 False False False False

9 False False False False

10 False False False False

11 False False False False

12 False False False False

13 False False False False

14 False False False False

15 False False False False

16 False False False False

17 False False False False

18 False False False False

19 False False False False

20 False False False False

21 False False False False

22 False False False False

23 False False False False

24 False False False False

25 False False False False

26 False False False False

27 False False False False

28 False False False False

29 False False False False

.. ... ... ... ...

970 False False False False

971 False False False False

972 False False False False

973 False False False False

974 False False False False

975 False False False False

976 False False False False

977 False False False False

978 False False False False

979 False False False False

980 False False False False

981 False False False False

982 False False False False

983 False False False False

984 False False False False

985 False False False False

986 False False False False

987 False False False False

988 False False False False

989 False False False False

990 False False False False

991 False False False False

992 False False False False

993 False False False False

994 False False False False

995 False False False False

996 False False False False

997 False False False False

998 False False False False

999 False False False False

[1000 rows x 4 columns]

TM\_1.any(1,bool\_only=True)

Out[61]:

0 False

1 False

2 False

3 False

4 False

5 False

6 False

7 False

8 False

9 False

10 False

11 False

12 False

13 False

14 False

15 False

16 False

17 False

18 False

19 False

20 False

21 False

22 False

23 False

24 False

25 False

26 False

27 False

28 False

29 False

970 False

971 False

972 False

973 False

974 False

975 False

976 False

977 False

978 False

979 False

980 False

981 False

982 False

983 False

984 False

985 False

986 False

987 False

988 False

989 False

990 False

991 False

992 False

993 False

994 False

995 False

996 False

997 False

998 False

999 False

Length: 1000, dtype: bool

TM\_1.any(1,bool\_only=False)

Out[62]:

0 True

1 False

2 False

3 False

4 False

5 False

6 False

7 False

8 False

9 False

10 False

11 False

12 False

13 False

14 False

15 False

16 False

17 False

18 False

19 False

20 False

21 False

22 False

23 False

24 False

25 False

26 False

27 False

28 False

29 False

970 False

971 False

972 False

973 False

974 False

975 False

976 False

977 False

978 False

979 False

980 False

981 False

982 False

983 False

984 False

985 False

986 False

987 False

988 False

989 False

990 False

991 False

992 False

993 False

994 False

995 False

996 False

997 False

998 False

999 False

Length: 1000, dtype: bool

1. 如果我们这里使用any（0），将会发生错误，原因是返回值仅仅是一个下面的series，把它放到索引位。肯定无法进行索引

(np.abs(data)>3).any(0)

Out[36]:

0 True

1 True

2 True

3 True

dtype: bool

data[(np.abs(data)>3).any(0)]

C:\Users\dongfeng\Anaconda3\lib\site-packages\ipykernel\_launcher.py:1: UserWarning: Boolean Series key will be reindexed to match DataFrame index.

"""Entry point for launching an IPython kernel.

---------------------------------------------------------------------------

IndexingError Traceback (most recent call last)

<ipython-input-37-ebe2c62f7641> in <module>()

----> 1 data[(np.abs(data)>3).any(0)]

~\Anaconda3\lib\site-packages\pandas\core\frame.py in \_\_getitem\_\_(self, key)

1956 if isinstance(key, (Series, np.ndarray, Index, list)):

1957 # either boolean or fancy integer index

-> 1958 return self.\_getitem\_array(key)

1959 elif isinstance(key, DataFrame):

1960 return self.\_getitem\_frame(key)

~\Anaconda3\lib\site-packages\pandas\core\frame.py in \_getitem\_array(self, key)

1996 # check\_bool\_indexer will throw exception if Series key cannot

1997 # be reindexed to match DataFrame rows

-> 1998 key = check\_bool\_indexer(self.index, key)

1999 indexer = key.nonzero()[0]

2000 return self.take(indexer, axis=0, convert=False)

~\Anaconda3\lib\site-packages\pandas\core\indexing.py in check\_bool\_indexer(ax, key)

1937 mask = isnull(result.\_values)

1938 if mask.any():

-> 1939 raise IndexingError('Unalignable boolean Series provided as '

1940 'indexer (index of the boolean Series and of '

1941 'the indexed object do not match')

IndexingError: Unalignable boolean Series provided as indexer (index of the boolean Series and of the indexed object do not match

当然我们可以修改，把其放在列位，但这样的话等于什么也没有做，仅仅选取所有列

data.loc[:,(np.abs(data)>3).any(0)]

Out[40]:

0 1 2 3

0 -0.567046 -0.491171 -0.827547 -0.680408

1 0.206551 -1.456287 0.629825 0.148732

2 -1.130370 -0.056857 1.431013 1.475764

3 -2.125800 0.108440 -1.350079 0.523728

4 -0.736537 -0.846202 0.543960 -1.230080

5 -0.152478 0.087115 -0.304004 0.050398

6 -0.479466 -1.990843 -0.361344 1.096610

7 -0.246875 0.595030 1.469306 -0.438550

8 0.989888 0.174695 0.241461 -0.550926

9 -0.781855 1.660456 1.041865 0.264480

10 1.661492 0.356601 -0.699093 0.869719

11 0.163815 0.246831 -0.250914 -0.694415

12 -1.584598 -1.279383 -1.274184 -0.008277

13 -0.266142 1.401936 0.397230 -0.087856

14 0.016147 0.710451 1.548756 0.292436

15 -0.885661 -0.126581 0.927745 -0.194748

16 -0.018011 0.384556 0.545346 0.447323

17 -1.785552 0.372100 0.014757 -2.023199

18 0.764790 0.900823 -1.039617 1.111810

19 -0.603835 0.951399 -1.166977 2.091614

20 0.227416 0.606907 -0.533333 0.524420

21 0.728552 -0.138527 -1.922079 -2.683756

22 -1.777072 -0.610157 -1.141227 -0.527488

23 -0.393001 1.573780 -0.270537 -0.552588

24 0.142684 -0.237119 -2.089615 0.982052

25 0.048209 -0.858588 1.238078 0.605156

26 -0.736681 0.402256 -0.247741 -0.549385

27 -0.183541 -0.654161 0.539174 -0.597491

28 0.641096 -1.708915 0.660372 0.510122

29 -0.174187 -0.809638 0.032858 -1.431953

.. ... ... ... ...

970 -0.868857 2.001313 0.852164 0.589976

971 -0.557336 0.318217 -0.546586 1.061976

972 -0.884070 -0.641405 -0.210885 0.049715

973 0.785917 0.212696 1.193491 1.007289

974 0.921965 -0.478275 -1.417855 0.911799

975 0.131626 0.050366 0.637130 0.267212

976 -1.425079 0.923022 -0.208123 -0.697405

977 1.723595 -0.022576 -0.954601 -0.374849

978 -2.049331 1.803200 -0.894918 0.955312

979 0.082268 2.073896 0.072605 0.912109

980 -0.166275 -0.463344 1.510438 -0.478526

981 0.438580 0.540105 1.023839 0.441328

982 -0.260641 1.145658 0.062698 1.655832

983 -0.176418 -0.032755 0.906118 2.012923

984 -0.870494 0.038007 -0.371591 -0.262396

985 -0.683679 -0.182418 0.683105 -0.852608

986 -0.641252 1.387388 1.770705 -1.003002

987 -2.111872 -0.438135 0.669196 -0.642335

988 1.279690 -0.179545 -1.728947 0.837643

989 -0.378165 0.507085 -0.106542 0.511993

990 0.741188 -0.681142 -0.192551 -0.712643

991 1.045758 0.238560 -0.713991 -0.463895

992 -1.052059 -0.942410 -0.062352 0.267082

993 -0.184833 1.474266 2.345032 1.189559

994 -0.245563 -0.513430 -0.998651 -0.858027

995 0.745093 0.478441 1.731140 0.848215

996 -0.445829 -0.022585 -0.621001 -1.348158

997 -1.479772 0.688092 -1.209812 -1.247304

998 -0.260688 0.373783 -0.384794 1.073226

999 -0.395423 0.955142 0.525140 1.178641

#把所有数据限制到-3到3的范围内。

data[np.abs(data)>3]=np.sign(data)\*3#左边实现绝对值大于3的值的选取，即设置了x>3或x<-3的选区。右边是符号与左边选区数值对应，绝对值数值为3的数据。通过=赋值到选区的相应位置。

count\_frame=pd.DataFrame([pd.value\_counts(np.abs(data[0])>3),pd.value\_counts(np.abs(data[1])>3),pd.value\_counts(np.abs(data[2])>3),pd.value\_counts(np.abs(data[3])>3)])

count\_frame

Out[77]:

False

0 1000

1 1000

2 1000

3 1000

**排列与随机采样**

通过permutation函数可以实现对数据框或者Series的随机排列。

asr=np.random.randint(-6,6,15).reshape(5,3)

asr\_fra=pd.DataFrame(asr)

asr\_fra

Out[17]:

0 1 2

0 -2 1 0

1 -2 4 3

2 -5 1 4

3 0 4 1

4 4 4 -1

np.random.permutation(asr\_fra)

Out[18]:

array([[-5, 1, 4],

[-2, 4, 3],

[ 4, 4, -1],

[ 0, 4, 1],

[-2, 1, 0]])

#其实质仅仅是对第一列数据进行排列，其他列随着第一列移动

#下面对Series进行随机排列

ser1=pd.Series([3,2,7,9,0,2,1,9,2,2,3,12,17])

np.random.permutation(ser1)

Out[22]: array([ 9, 7, 3, 0, 2, 3, 17, 2, 12, 2, 1, 9, 2], dtype=int64)

#我们也可以通过产生一个随机排列数组，对数组进行排列

as\_frame=pd.DataFrame(np.random.randint(-12,20,42).reshape(7,6))

as\_frame

Out[26]:

0 1 2 3 4 5

0 -5 13 10 14 -2 5

1 -10 13 -9 10 -4 -7

2 6 11 0 -2 -8 9

3 0 -11 -8 -2 -12 16

4 3 -12 -5 18 8 3

5 0 3 4 4 5 10

6 7 0 6 -7 3 -10

pattern=np.random.permutation(6)

pattern

Out[29]: array([5, 4, 3, 0, 1, 2])

as\_frame.take(pattern,0)

Out[28]:

0 1 2 3 4 5

5 0 3 4 4 5 10

4 3 -12 -5 18 8 3

3 0 -11 -8 -2 -12 16

0 -5 13 10 14 -2 5

1 -10 13 -9 10 -4 -7

2 6 11 0 -2 -8 9

注意这里返回的结果是as\_frame的一个子数据框，这是因为pattern的结果是从从0到5的一个排列，数据框的行索引是按照这个pattern的这个结果进行排列。

这里也可以按列索引进行排列

as\_frame.take(pattern,1)

Out[31]:

5 4 3 0 1 2

0 5 -2 14 -5 13 10

1 -7 -4 10 -10 13 -9

2 9 -8 -2 6 11 0

3 16 -12 -2 0 -11 -8

4 3 8 18 3 -12 -5

5 10 5 4 0 3 4

6 -10 3 -7 7 0 6

as\_frame.take(np.random.permutation([4,3,0,1]),1)#可以按照索引子集的排列进行列的重排

Out[33]:

3 4 0 1

0 14 -2 -5 13

1 10 -4 -10 13

2 -2 -8 6 11

3 -2 -12 0 -11

4 18 8 3 -12

5 4 5 0 3

6 -7 3 7 0

as\_frame.take(np.random.permutation(as\_frame.shape[1])[:4],1)

Out[37]:

0 2 4 5

0 -5 10 -2 5

1 -10 -9 -4 -7

2 6 0 -8 9

3 0 -8 -12 16

4 3 -5 8 3

5 0 4 5 10

6 7 6 3 -10

通过random.randint和take函数就行随机抽样

# 五，绘图和可视化

Matplotlib入门，pandas下的绘图函数，画布，画布分割等等

Matplotlib入门

调用Matplotlib API 程序包

1，创建画布

import matplotlib as plt

import matplotlib.pyplot as plt

figure\_1=plt.figure()

(或者详细定义画布

figure\_1=plt.figure(1,(4,6),dpi=400,facecolor='green',edgecolor='yellow',frameon=True)）

figure\_1

Out[43]: <matplotlib.figure.Figure at 0x8ea7987320>

1. 分割画布

aapic\_1=figure\_1.add\_subplot(2,2,1)

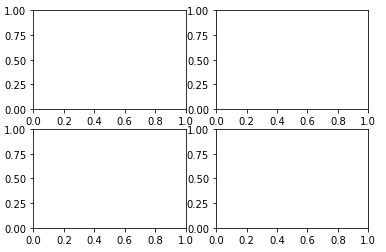
aapic\_2=figure\_1.add\_subplot(2,2,2)

aapic\_3=figure\_1.add\_subplot(2,2,3)

aapic\_4=figure\_1.add\_subplot(2,2,4)

3，show 出你的画布

plt.show()



import matplotlib.pyplot as plt

import numpy as np

figure\_1=plt.figure()

aapic\_1=figure\_1.add\_subplot(2,2,1)

aapic\_2=figure\_1.add\_subplot(2,2,2)

aapic\_3=figure\_1.add\_subplot(2,2,3)

plt.plot(np.random.randn(50).cumsum(),'k--')

Out[7]: [<matplotlib.lines.Line2D at 0xdd16defd0>]

#直接用“plot”绘图，k--代表黑色虚线图

aapic\_2.hist(np.random.randn(100),bins=20,color='k',alpha=0.3)

Out[9]:

(array([ 1., 1., 1., 5., 3., 7., 8., 14., 11., 6., 11.,

6., 13., 5., 2., 3., 0., 1., 0., 2.]),

array([-2.64526298, -2.37172471, -2.09818644, -1.82464817, -1.5511099 ,

-1.27757163, -1.00403336, -0.73049509, -0.45695681, -0.18341854,

0.09011973, 0.363658 , 0.63719627, 0.91073454, 1.18427281,

1.45781108, 1.73134935, 2.00488762, 2.27842589, 2.55196416,

2.82550243]),

<a list of 20 Patch objects>)

aapic\_1.scatter(np.arange(30),np.arange(30)+3\*np.random.randn(30))

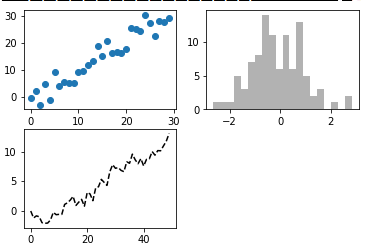
Out[10]: <matplotlib.collections.PathCollection at 0xdd172c390>

#alpha代表透明度，bins代表柱状体个数

plt.show

Out[12]: <function matplotlib.pyplot.show>

plt.show()



这里还有一个简单且更为方便的画图方法；它可以创建一个新的figure（画布）并返回一个含有以创建的subplot对象的numpy数组。

pic,axes=plt.subplots(2,3)

plt.show(0)



饼图：

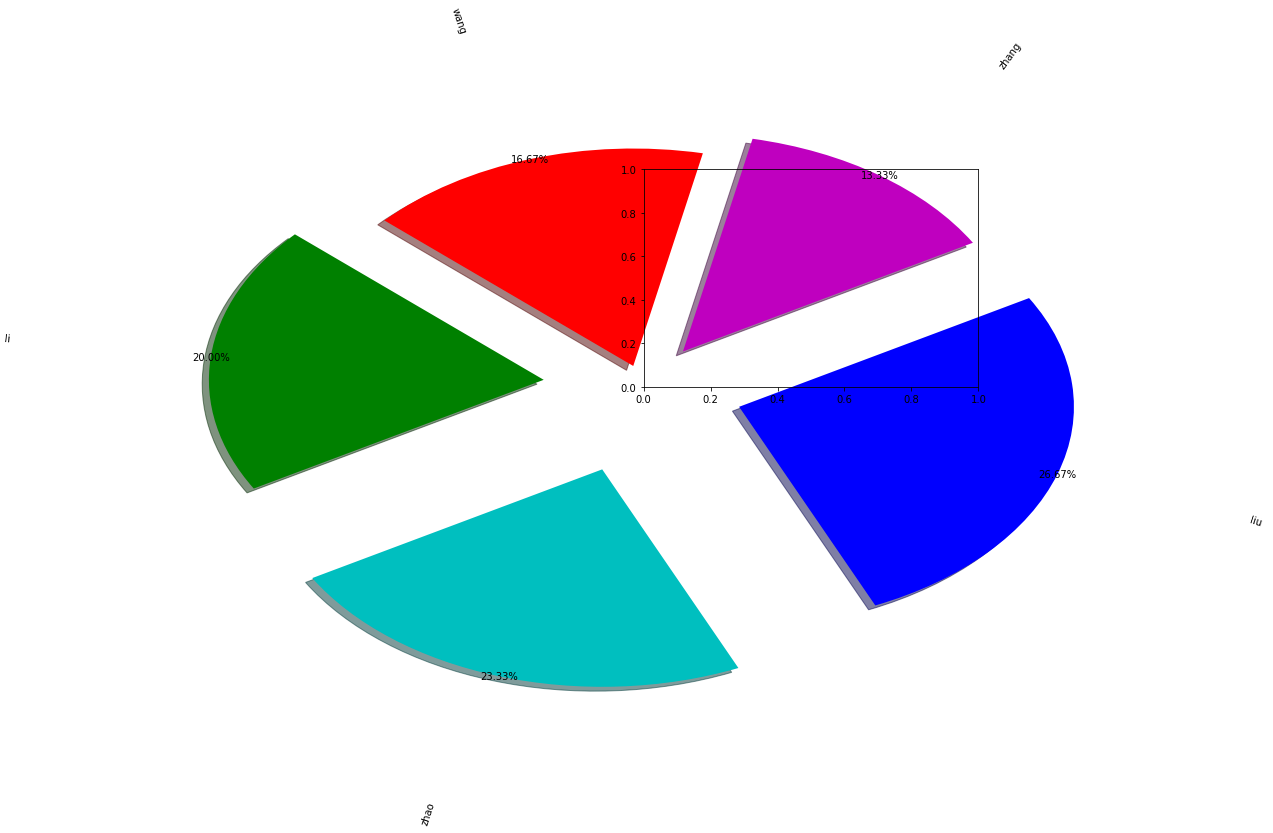
import numpy as np

import matplotlib.pyplot as plt

Asx,sd=plt.subplots(1,1)

sd.pie(np.arange(4,9),explode=[0.2,0.1,0.3,0.4,0.3],labels=['zhang','wang','li','zhao','liu'],colors=['m','r','g','c','b'],autopct='%.2f%%',pctdistance=1,shadow=True,labeldistance=1.6,startangle=30,radius=1,frame=True,rotatelabels=True)

Plt.show()

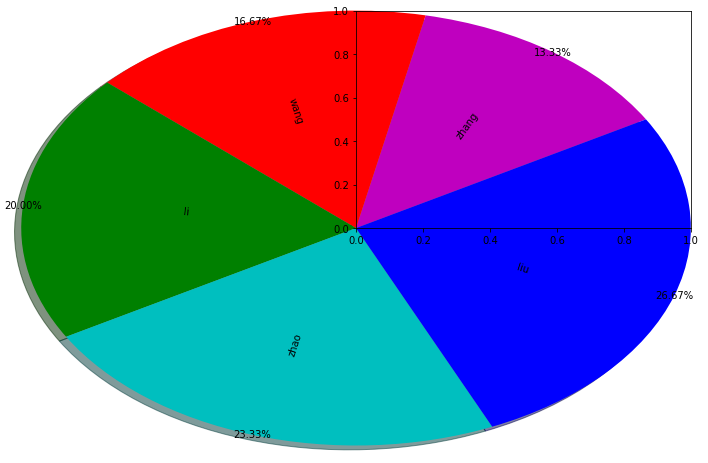


Asx,sd=plt.subplots(1,1)

sd.pie(np.arange(4,9),explode=[0,0,0,0,0],labels=['zhang','wang','li','zhao','liu'],colors=['m','r','g','c','b'],autopct='%.2f%%',pctdistance=1,shadow=True,labeldistance=0.5,startangle=30,radius=1,frame=True,rotatelabels=True)

plt.show()

#labeldistance<1,图例将会在饼图内

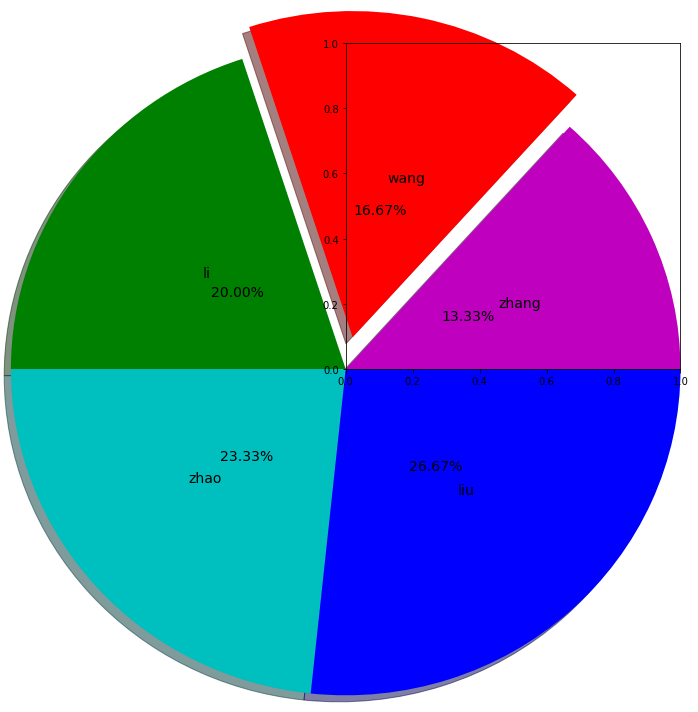


#通过figsize=（6,6）把饼图设置成圆的，然后通过textprops设置字体，通过labeldistance设置标签离圆心距离。通过autopct设置每个部分总总体的百分数，通过pctdistance设置百分数例圆心距离，通过explode设置一个部分的强调。

Asx,sd=plt.subplots(1,1,figsize=(6,6))

sd.pie(np.arange(4,9),explode=[0,0.1,0,0,0],labels=['zhang','wang','li','zhao','liu'],colors=['m','r','g','c','b'],autopct='%.2f%%',pctdistance=0.4,shadow=True,labeldistance=0.5,wedgeprops=None,textprops={'fontsize':14},startangle=0,radius=1,frame=True,rotatelabels=None)

plt.show()



调整Subplot周围的间距：

默认情况下，子图外围是有一定边距的，并且各子图之间上下左右都有一定间距。

间距与子图大小有关，子图像宽，则横向间距小，图像高，则纵向间距小。

除了默认，我们可以选择自己确定间距：

plt.subplots\_adjust(left=None,bottom=None,right=None,top=None,wspace=None,hspace=None)

#wspace和hspace用于控制宽度和高度的百分比，通过调整这两个参数，我们的子图间的上下左右间距会发生变化

例子：

import numpy as np

fig, axes=plt.subplots(2, 2, sharex=True, sharey=True)

for i in range(2):

for j in range(2):

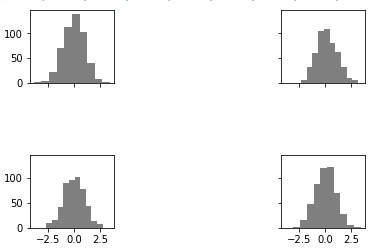
axes[i,j].hist(np.random.randn(500),color='k',alpha=0.5)

plt.subplots\_adjust(wspace=2,hspace=1)

plt.show(0)

<matplotlib.figure.Figure at 0x17becd9908>

<matplotlib.figure.Figure at 0x17befbfda0>



#shareX 和sharey是共X轴和Y轴。调整wspace 盒hspace可以得到不同的子图间距。

for i in range(2):

for j in range(2):

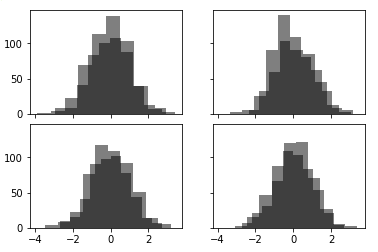
axes[i,j].hist(np.random.randn(500),color='k',alpha=0.5)

plt.subplots\_adjust(wspace=0.2,hspace=0.1)

plt.show(0)

<matplotlib.figure.Figure at 0x17becd9908>

<matplotlib.figure.Figure at 0x17befbfda0>



问：上面图像的X轴和Y轴分别指的是什么？？？？

颜色和线型以及标记

指定图形函数线的颜色，我们可以通过一个指令实现，例：

Pic\_11.plot(X,Y,’g--’) 等价于 ax.plot(x,y,linestyle=’--’,color=’g’)

更多颜色可使用指定其RGB值的形式。对于线性图，我们可以给数据点加上标记(marker)，使人更容易发现那些是数据点。

线型和marker表格：

'-' solid line style

'--' dashed line style

'-.' dash-dot line style

':' dotted line style

'.' point marker

',' pixel marker

'o' circle marker

'v' triangle\_down marker

'^' triangle\_up marker

'<' triangle\_left marker

'>' triangle\_right marker

'1' tri\_down marker

'2' tri\_up marker

'3' tri\_left marker

'4' tri\_right marker

's' square marker

'p' pentagon marker

'\*' star marker

'h' hexagon1 marker

'H' hexagon2 marker

'+' plus marker

'x' x marker

'D' diamond marker

'd' thin\_diamond marker

'|' vline marker

'\_' hline marker

颜色表格：

‘b’ blue

‘g’ green

‘r’ red

‘c’ cyan

‘m’ magenta

‘y’ yellow

‘k’ black

‘w’ white

**美丽温馨的例子：**

marker\_1=['.', ',', 'o', 'v', '^', '<', '>', '1', '2', '3', '4','s','p','\*','h','H','+','x','D','d','|', '\_']

asd,fid\_1=plt.subplots(len(marker\_1),dpi=180,figsize=(6,6\*len(marker\_1)))

for i in range(len(marker\_1)):

N = 50 # 点的个数

x = np.random.rand(N) \* 2 # 随机产生50个0~2之间的x坐标

y = np.random.rand(N) \* 2 # 随机产生50个0~2之间的y坐标

colors = np.random.rand(N) # 随机产生50个0~1之间的颜色值

area = np.pi \* (15 \* np.random.rand(N))\*\*2 # 点的半径范围:0~15

# 画散点图

fid\_1[i].scatter(x, y, s=area, c=colors, alpha=0.5, marker=marker\_1[i])

fid\_1[i].set\_xlabel(marker\_1[i])

plt.show()

如果不涉及子图的话，无需先设置画布。可直接画出图形：

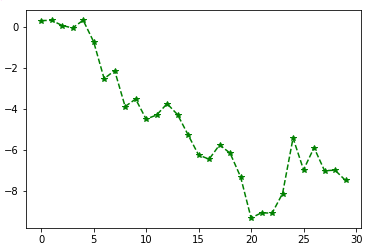
import numpy as np

import matplotlib.pyplot as plt

plt.plot(np.random.randn(30).cumsum(),'g\*--')

Out[3]: [<matplotlib.lines.Line2D at 0xb407f4dd30>]

plt.show()



plt.plot(np.random.randn(30).cumsum(),'g\*--') 等价于

Plt.plot(np.random.randn(30).cumsum(),color=’g’,linestyle=’dashed’,marker=’\*’)

在线性图中，那些非数据点都是根据两个数据点连线插值的，我们可以修改这种插值方式，这里用到drawstyle选项修改：

data=np.array([1,2,3.2,2.3,4.6,7.5,2,3,6.5,7.8,9],dtype=float)

np.unique(data)

Out[9]: array([ 1. , 2. , 2.3, 3. , 3.2, 4.6, 6.5, 7.5, 7.8, 9. ])

data

Out[10]: array([ 1. , 2. , 3.2, 2.3, 4.6, 7.5, 2. , 3. , 6.5, 7.8, 9. ])

data\_uique=np.unique(data)

plt.plot(data\_uique,'o-.',label='line\_point')

Out[28]: [<matplotlib.lines.Line2D at 0xb40947c908>]

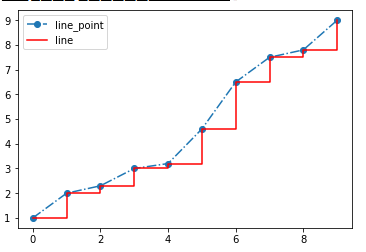
plt.plot(data\_uique,'r-',drawstyle='steps-post',label='line')

Out[29]: [<matplotlib.lines.Line2D at 0xb40947cf98>]

plt.legend(loc='best')

Out[30]: <matplotlib.legend.Legend at 0xb4083ca358>

plt.show()



legend函数介绍：

在画一些曲线图时，常常会出现多条曲线同时画在一张图上面，这时候就需要对不同的曲线进行不同的标注，以使读者能够清晰地知道每条曲线代表的含义。当你画很少的几条曲线时，这时画图命令中自动产生的legend能够基本满足你的需要，此时，你不需要做什么；但当你将很多个曲线画在一张图上时，自动产生的legend矩形框往往会覆盖住已经画出来的曲线，很不美观，这时你就需要写专门的代码对legend的位置进行精确的控制，而不能再依靠系统帮你自动控制了。

比如：

plt.legend(loc='upper center', bbox\_to\_anchor=(0.6,0.95),ncol=3,fancybox=True,shadow=True)

Ncol=表示我们的图例（legend）里的线的标识可以排成三列

Loc=标识图例的位置

bbox\_to\_anchor=图例的精确位置，上面bbox\_to\_anchor被赋予的二元组中，第一个数值用于控制legend的左右移动，值越大越向右边移动，第二个数值用于控制legend的上下移动，值越大，越向上移动。

刻度与标签

图像的刻度与标签都是通过一些方法来实现的，这里有几个方法大家在绘图中经常用到：

Xlim（X值范围）、xticks（X轴刻度值）和xticklabels（X轴刻度标签）

其使用方式有以下两种:

1. 调用不带参数值，返回当前参数值，即是现在正用的参数值。
2. 调用时带参数值，使用该参数值。

例：

table\_1=plt.figure()

ax=table\_1.add\_subplot(1,1,1)

ax.plot(np.random.randn(1000).cumsum())

Out[36]: [<matplotlib.lines.Line2D at 0xb409514550>]

ax.plot(np.random.randn(100000).cumsum())

Out[37]: [<matplotlib.lines.Line2D at 0xb40948f780>]

ticks\_1=ax.set\_xticks([0,25000,50000,75000,100000])

scale\_name=ax.set\_xticklabels(['step1','step2','step3','step4','step5'],rotation=45,fontsize=12)

ax.set\_title('Python-03 Practice')

Out[40]: Text(0.5,1,'Python-03 Practice')

ax.set\_xlabel('Steps')

Out[41]: Text(0.5,0,'Steps')

#下面我们添加图例

ax.plot(np.random.randn(100000).cumsum(),color='m',linestyle='-',label='solid')

Out[44]: [<matplotlib.lines.Line2D at 0xb409852748>]

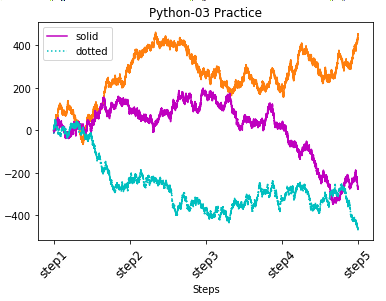
ax.plot(np.random.randn(100000).cumsum(),color='c',linestyle=':',label='dotted')

Out[45]: [<matplotlib.lines.Line2D at 0xb4094f52b0>]

ax.legend(loc='best')#plt.legend也是可以的

Out[46]: <matplotlib.legend.Legend at 0xb409852ac8>

plt.show()



加注解

用text、arrow和annotate等函数进行添加注解，text可以将文本加到图标的指定坐标。

fig,subpic=plt.subplots(1,1)

subpic.plot([2,77,90,2.3,4,5,6,45,34,67,35,66,34,23,76],[34,32,35,43,34,23,45,56,44,57,56,33,55,66,54],'b\*-')

Out[7]: [<matplotlib.lines.Line2D at 0x2eae706be0>]

ticks\_1=subpic.set\_xticks([0,25,50,75,100])

scale\_name=subpic.set\_xticklabels(['jenuary','february','march','april','may'],rotation=40,fontsize=12)

subpic.set\_title('Python-03 Practice\_1',fontsize=16)

Out[11]: Text(0.5,1,'Python-03 Practice\_1')

subpic.set\_xlabel('Weight')

Out[13]: Text(0.5,0,'Weight')

subpic.text(90,35,'key point one',fontsize=10)

Out[15]: Text(90,35,'key point one')

keymenge=[(67,57,'key point two'),(23,66,'key point two')]

for x,y,label in keymenge:

subpic.text(x,y,label,fontsize=12)

subpic.set\_xlim([0,100])

Out[21]: (0, 100)

subpic.set\_ylim([0,100])

Out[22]: (0, 100)

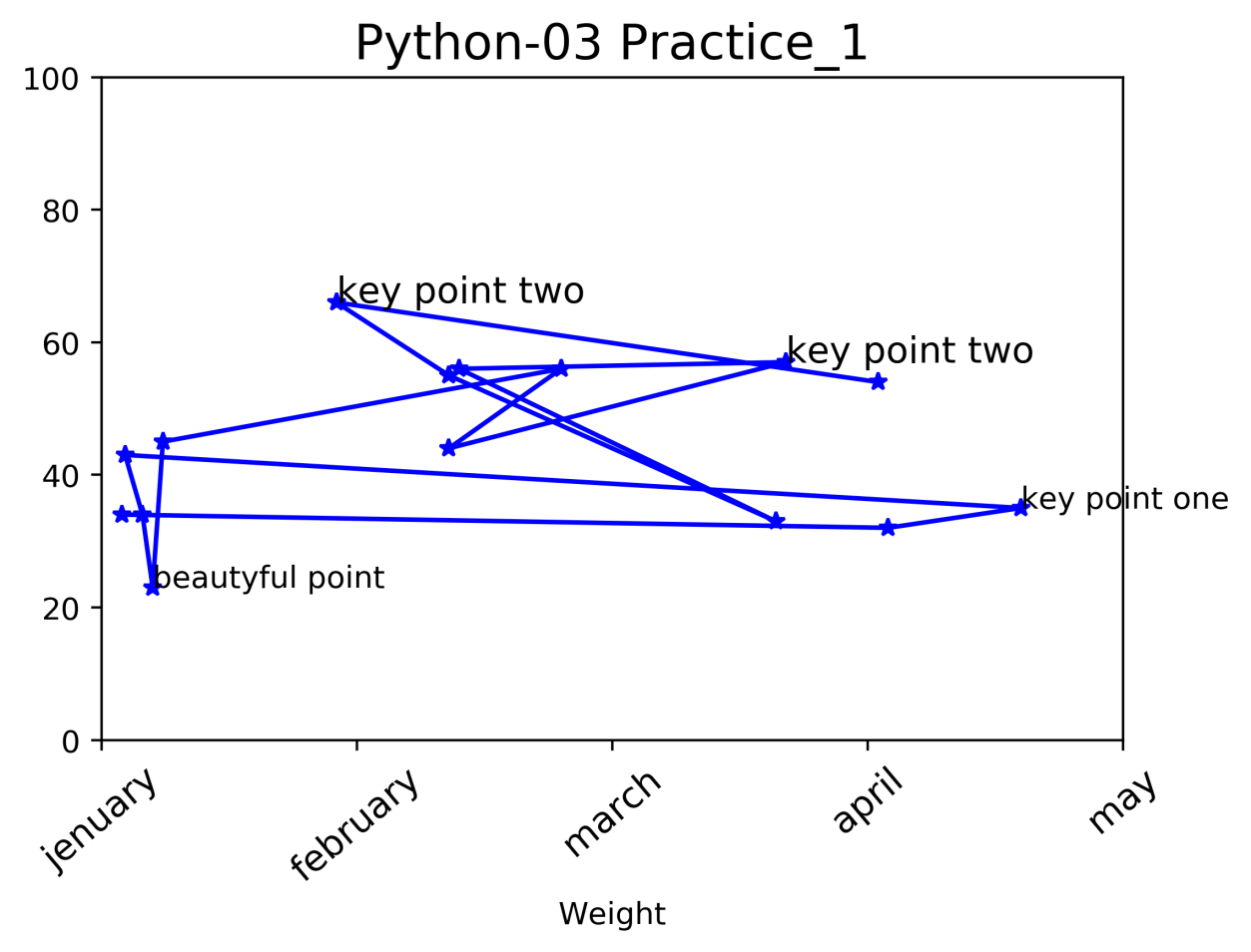
subpic.annotate('beautyful point',xy=(5,23),xytext=(5,23))

Out[18]: Text(5,23,'beautyful point')

plt.savefig('Desktop\python\_01.svg')

plt.savefig('Desktop\python\_01.png',dpi=400,bbox\_inches='tight')

plt.show()



Matplotlib.pyplot的画图方式与R语言十分类似，繁琐是他们共同的特点。与Matplotlib.pyplot不同，pandas在作图上不仅方法简单，而且可以完成各种各样的作图工作。

图像的保存

在上一章，我们已经使用了savefig这个函数来存储图像。事实上，Savefig还可以作为对象figure的方法存储画布上的图像，例如：

import matplotlib.pyplot as plt

pic=plt.figure()

pic\_1=pic.add\_subplot(1,1,1)

import numpy as np

pic\_1.plot(np.random.randn(50).cumsum(),'k--')

Out[6]: [<matplotlib.lines.Line2D at 0xa22b23f0b8>]

pic.savefig('Desktop\dong\_123.pdf',dpi=300,bbox\_inches='tight')

Savefig的主要参数如下：

fname: 表示绝对或者相对文件路径的字符串，文件具体格式由后缀来决定，譬如.pdf,.png格式等。

dpi: 图像分辨率，默认100，（每英寸点数）

Facecolor,edgecolor: 背景色，默认为“w”白色

Format: 显示设置文件格式，png，jpeg，pdf等等，但不要与fname里的文件格式发生冲突。

Bbox\_inches，常用值是tight，可剪除图表周围的空白部分。

Pandas作图

线性图

Series 和 DataFrame 都有一个用于生成各类图表的plot方法，默认状态。他只生成线性图。

Series生成线性图，索引（index）直接被绘制成X轴。当然我们也可以关闭使用index绘制X轴。Use\_index=False

X轴的刻度和界限可以用xsticks和xlim选项来进行调节，ysticks和ylim可以调节Y轴

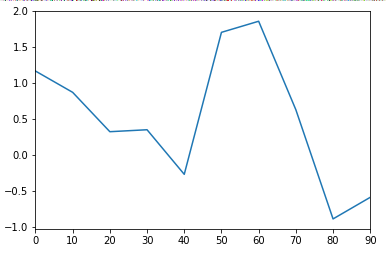
例子

ser=pd.Series(np.random.randn(10).cumsum(),index=np.arange(0,100,10))

ser.plot()

Out[18]: <matplotlib.axes.\_subplots.AxesSubplot at 0xf3e54e0240>

plt.show()



DataFrame的plot方法会在一个子图中为各列数据绘制一条线。自动创建的图例的标签与列索引相同。

例：

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

Frame\_01=pd.DataFrame([[2,4,8,16,32,64],[3,6,12,24,48,96],[5,10,15,20,25,30],[1,3,5,7,9,11],[16,8,4,2,1,0.5]],index=range(0,100,20),columns=['A','B','C','D','E','F'])

pic\_2,subplot\_object=plt.subplots(1,1)

Frame\_01.plot(kind='line',ax=subplot\_object,subplots=False,layout=False,logx=True,xlim=[0,100])

C:\Users\dongfeng\Anaconda3\lib\site-packages\matplotlib\axes\\_base.py:2923: UserWarning: Attempted to set non-positive xlimits for log-scale axis; invalid limits will be ignored.

'Attempted to set non-positive xlimits for log-scale axis; '

Out[3]: <matplotlib.axes.\_subplots.AxesSubplot at 0x5441e71cf8>

plt.show()

Frame\_01

Out[5]:

A B C D E F

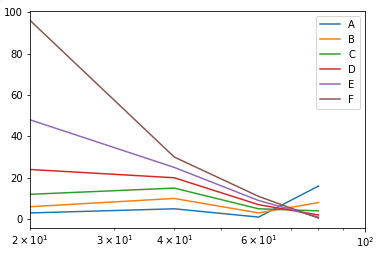
0 2 4 8 16 32 64.0

20 3 6 12 24 48 96.0

40 5 10 15 20 25 30.0

60 1 3 5 7 9 11.0

80 16 8 4 2 1 0.5



Series.plot 方法的常用参数汇总：

|  |  |
| --- | --- |
| label | 用于设置图例的标签 |
| ax | 确定要被绘制的matplotlib subplot对象。如果没有设置，则使用当前matplotlib subplot |
| style | 设置传给matplotlib的风格字符串（'g\*--'） |
| alpha | 图表的填充不透明度（数值为0到1之间的数） |
| kind | 各种图形样式line, bar, barh, kde, density, scatter |
| logy | 在Y轴上使用对数标尺 |
| use\_index | 将对象（Series and DataFrame）的索引用作**刻度标签** |
| rot | 旋转度数（0到360） |
| xticks | 用作X轴刻度的值 |
| yticks | 用作y轴刻度的值 |
| xlim | x的值域 |
| ylim | y的值域 |
| grid | 设置是否显示轴网格线 |

专用于DataFrame的plot的参数

|  |  |
| --- | --- |
| Subplots | 将依据数据框中的每个列绘制的图分别放置到单个的subplot（子画框）里 |
| sharex | Subplots=true时，设定是否共享X的刻度和值域 |
| sharey | Subplots=true时，设定是否共享y的刻度和值域 |
| figsize | 元组，用来表示图像大小（宽，高） |
| title | 设置图像标题 |
| Legend | 设定是否添加一个subplot图例 |
| sort\_columns | 设定是否以字母表中字母先后排列顺序绘制各列。 |

柱状图

柱状图分为水平柱状图和垂直柱状图。当kind=’bar’生成垂直柱状图；kind=’barh’生成水平柱状图。

1. Series生成柱状图

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

figure\_1,get\_information\_of\_pict=plt.subplots(2,1)

datas=pd.Series(np.random.rand(20),index=list('qwertyuiopasdfghjklz'))

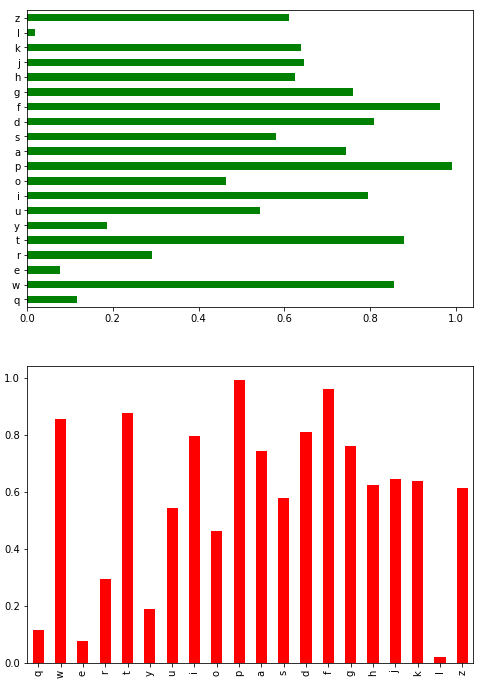
datas.plot(kind='barh',ax=get\_information\_of\_pict[0],figsize=(8,12),color='g')

Out[4]: <matplotlib.axes.\_subplots.AxesSubplot at 0x5be13ea6d8>

datas.plot(kind='bar',ax=get\_information\_of\_pict[1],figsize=(8,12),color='r')

Out[5]: <matplotlib.axes.\_subplots.AxesSubplot at 0x5be18c41d0>

plt.show()



1. 数据框生成柱状图

import numpy as np

import pandas as pd

Framedata\_1=pd.DataFrame(np.arange(16).reshape(4,4)\*\*(1/2)+np.arange(16).reshape(4,4)\*3+6,index=['spring','sommer','autumn','winter'],columns=['Benz','BMW','Porsche','VW'])

Out[11]:

|  | **Benz** | **BMW** | **Porsche** | **VW** |
| --- | --- | --- | --- | --- |
| **spring** | 6.000000 | 10.000000 | 13.414214 | 16.732051 |
| **sommer** | 20.000000 | 23.236068 | 26.449490 | 29.645751 |
| **autumn** | 32.828427 | 36.000000 | 39.162278 | 42.316625 |
| **winter** | 45.464102 | 48.605551 | 51.741657 | 54.872983 |

import numpy as np

import pandas as pd

plt.rcParams['font.sans-serif']=['SimHei']

plt.rcParams['axes.unicode\_minus'] = False

Framedata\_1=pd.DataFrame(np.arange(16).reshape(4,4)\*\*(1/2)+np.arange(16).reshape(4,4)\*3+6,index=['spring','sommer','autumn','winter'],columns=['Benz','BMW','Porsche','VW'])

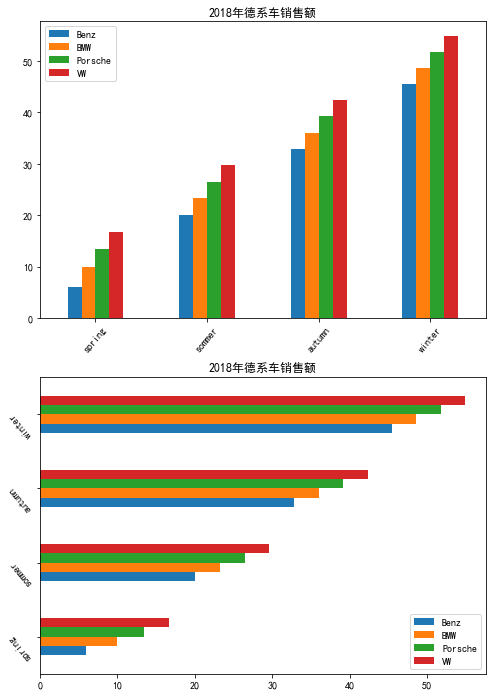
import matplotlib.pyplot as plt

picrange,pic\_inf=plt.subplots(2,1)

Framedata\_1.plot(kind='bar',ax=pic\_inf[0],title='2018年德系车销售额',rot=50,figsize=(8,12))

Framedata\_1.plot(kind='barh',ax=pic\_inf[1],title='2018年德系车销售额',rot=130,figsize=(8,12))

plt.show()



​import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

plt.rcParams['font.sans-serif']=['SimHei']

plt.rcParams['axes.unicode\_minus'] = False

Framedata\_1=pd.DataFrame(np.arange(16).reshape(4,4)\*\*(1/2)+np.arange(16).reshape(4,4)\*3+6,index=['spring','sommer','autumn','winter'],columns=['Benz','BMW','Porsche','VW'])

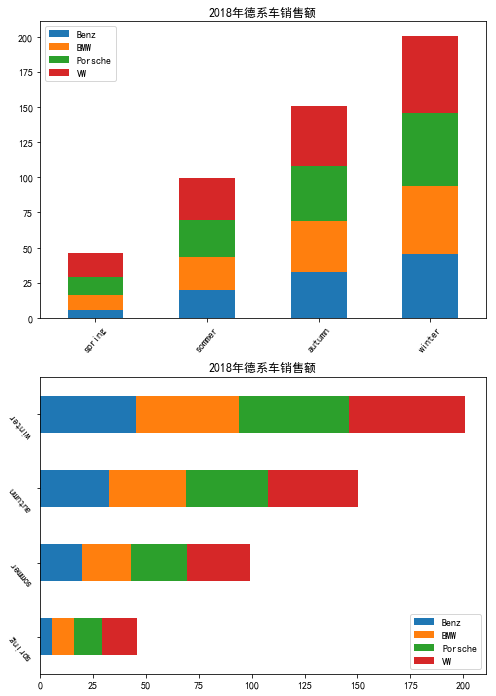
import matplotlib.pyplot as plt

picrange,pic\_inf=plt.subplots(2,1)

Framedata\_1.plot(kind='barh',ax=pic\_inf[1],stacked=True,title='2018年德系车销售额',rot=130,figsize=(8,12))

Framedata\_1.plot(kind='bar',ax=pic\_inf[0],stacked=True,title='2018年德系车销售额',rot=50,figsize=(8,12))

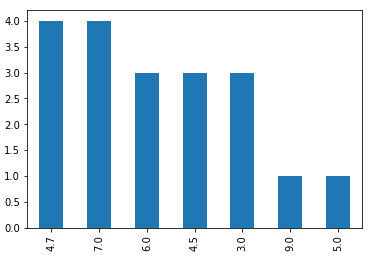
plt.show()



serie\_1=pd.Series([3,4.5,5,3,6,9,4.7,4.7,4.7,4.7,4.5,4.5,6,6,3,7,7,7,7])

serie\_1.value\_counts().plot(kind='bar')

plt.show()



#应用serie\_1.value.counts()在series中寻找数据重复的次数并作为纵坐标，Series中的数据作为横坐标。

直方图与密度图：

首先我们要区分直方图与柱状图。

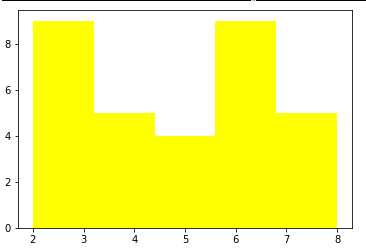
柱状图：

* 柱状图的某一个轴（X或Y轴）可以没有严格的刻度，并且柱的宽度随图形大小，柱的数量等因素的变化而变化，并没有严格的公式来保证，因此没有实际意义，仅仅用来区分类别。通常柱状图是用条形的长度表示各类别对应的实际数据（譬如频数）的大小。
* 柱状图是分开排列
* 主要用于展示分类数据

直方图：

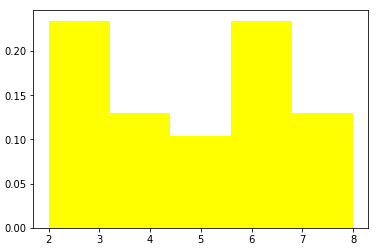
* 直方图通常是用面积表示各组数据大小（例如频数），矩形的高度表示每一组的频数或频率或其他匹配数据，宽度则表示各组的组距，因此其高度与宽度均有意义。
* 由于分组数据具有连续性，直方图的各矩形通常是连续排列。
* 直方图主要用于展示数据型数据。

plt.hist(np.array([1,2,3,4,5,1.2,1.34,1.78,2.1,2.4,2.8,2.9,3.1,3.5,3.7,4.2,4.9,5.6,5.3,5.8,5.9,5,3,6,7,8.5,6.4,7.3,7.8,7.2,5.6,6.6,6.45,6.99,3.45,2.36,5.67,8.13]),bins=5,normed=False,range=(2,8),color='yellow')



plt.hist(np.array([1,2,3,4,5,1.2,1.34,1.78,2.1,2.4,2.8,2.9,3.1,3.5,3.7,4.2,4.9,5.6,5.3,5.8,5.9,5,3,6,7,8.5,6.4,7.3,7.8,7.2,5.6,6.6,6.45,6.99,3.45,2.36,5.67,8.13]),bins=5,normed=True,range=(2,8),color='yellow')

plt.hist(np.array([1,2,3,4,5,1.2,1.34,1.78,2.1,2.4,2.8,2.9,3.1,3.5,3.7,4.2,4.9,5.6,5.3,5.8,5.9,5,3,6,7,8.5,6.4,7.3,7.8,7.2,5.6,6.6,6.45,6.99,3.45,2.36,5.67,8.13]),bins=5,normed=True,range=(2,8),color='yellow')



关于Normed（数据标准化）算法的解释：

sf,axes=plt.subplots()

data= np.array([1,1,2,3,3,3,3,3,4,5.1])

counts= axes.hist(data, normed= True)

counts

Out[9]:

(array([ 0.48780488, 0. , 0.24390244, 0. , 1.2195122 ,

0. , 0. , 0.24390244, 0. , 0.24390244]),

array([ 1. , 1.41, 1.82, 2.23, 2.64, 3.05, 3.46, 3.87, 4.28,

4.69, 5.1 ]),

<a list of 10 Patch objects>)

np.diff(counts[1])

Out[10]: array([ 0.41, 0.41, 0.41, 0.41, 0.41, 0.41, 0.41, 0.41, 0.41, 0.41])

#组距的计算

(counts[0]\*np.diff(counts[1])).sum()

Out[13]: 1.0

#可以看到标准化后的结果和组距相乘然后求和等于1，这才是标准化的实际意义，而不是看标准化的结果是否都必须小于1。

#下面我们确定Python标准化的算法：

sf,axes=plt.subplots()

data= np.array([1,1,2,3,3,3,3,3,4,5.1])

counts\_1= axes.hist(data, normed= False)

counts\_1

Out[14]:

(array([ 2., 0., 1., 0., 5., 0., 0., 1., 0., 1.]),

array([ 1. , 1.41, 1.82, 2.23, 2.64, 3.05, 3.46, 3.87, 4.28,

4.69, 5.1 ]),

<a list of 10 Patch objects>)

probality\_1=counts\_1[0]/np.sum(counts\_1[0]\*np.diff(counts\_1[1]))

probality\_1

Out[16]:

array([ 0.48780488, 0. , 0.24390244, 0. , 1.2195122 ,

0. , 0. , 0.24390244, 0. , 0.24390244])

最终我们得出算法公式为：

**频数矩阵/Sum(频数矩阵\*组距矩阵)**

验证算法：

plt.hist(np.array([1,2,3,4,5,1.2,1.34,1.78,2.1,2.4,2.8,2.9,3.1,3.5,3.7,4.2,4.9,5.6,5.3,5.8,5.9,5,3,6,7,8.5,6.4,7.3,7.8,7.2,5.6,6.6,6.45,6.99,3.45,2.36,5.67,8.13]),bins=5,normed=True,range=(2,8),color='yellow')

Out[17]:

(array([ 0.234375 , 0.13020833, 0.10416667, 0.234375 , 0.13020833]),

array([ 2. , 3.2, 4.4, 5.6, 6.8, 8. ]),

<a list of 5 Patch objects>)

pro1=plt.hist(np.array([1,2,3,4,5,1.2,1.34,1.78,2.1,2.4,2.8,2.9,3.1,3.5,3.7,4.2,4.9,5.6,5.3,5.8,5.9,5,3,6,7,8.5,6.4,7.3,7.8,7.2,5.6,6.6,6.45,6.99,3.45,2.36,5.67,8.13]),bins=5,range=(2,8),color='yellow')

pro1[0]

Out[19]: array([ 9., 5., 4., 9., 5.])

pro\_12=pro1[0]/np.sum(pro1[0]\*np.diff(pro1[1]))

pro\_12

Out[21]: array([ 0.234375 , 0.13020833, 0.10416667, 0.234375 , 0.13020833])

密度图通过kind的KDE关键字来实现的，这里的密度指的是概率密度。它是通过计算观测数据可能产生的概率密度分布而产生的

ser\_1=np.random.normal(0,1,200)

ser\_2=np.random.normal(16,1,200)

pis\_1,sdr\_1=plt.subplots()

A=np.concatenate((ser\_1,ser\_2))

ser\_1=np.random.normal(0,1,200)

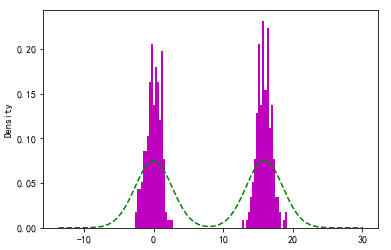
ser\_2=np.random.normal(12,1,200)

value\_1=pd.Series(A)

value\_1.hist(bins=75,color='m',normed=True)

value\_1.plot(kind='kde',style='g--')

plt.show()



**散布图**

散布图必须通过两个数据序列才能绘制而成。

也可以通过一个数据框绘制成散布矩阵,我们先绘制一个复杂的散布矩阵

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

X2=np.random.normal(3,1,1001)

X1=np.arange(1,1002,1)

X3=np.random.randn(1001)

X4=np.random.rand(1001)

X4=np.random.rand(1001)

X5=np.random.beta(2,1,1001)

test\_data=np.column\_stack((np.column\_stack((np.column\_stack((np.column\_stack((X1,X2)),X3)),X4)),X5))

test\_frame=pd.DataFrame(test\_data)

pd.scatter\_matrix(test\_frame,diagonal='kde',color='g',figsize=(10,10))

C:\Users\dongfeng\Anaconda3\lib\site-packages\ipykernel\_launcher.py:1: FutureWarning: pandas.scatter\_matrix is deprecated. Use pandas.plotting.scatter\_matrix instead

"""Entry point for launching an IPython kernel.

Out[7]:

array([[<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342B2D89E8>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x0000003422A85240>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x0000003422A9E2B0>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342AA85860>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342AAD87F0>],

[<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342AAD8828>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342B33FFD0>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342B3860F0>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342B39BDA0>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342B419518>],

[<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342B452898>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342B489828>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342B4C3828>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342B4D4F98>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342B522F98>],

[<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342B5684A8>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342B5A23C8>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342B53A7F0>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342B603A58>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342B63AF28>],

[<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342B67D4A8>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342B6A32B0>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342C6B2518>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342C6EA908>,

<matplotlib.axes.\_subplots.AxesSubplot object at 0x000000342C722828>]], dtype=object)

plt.show()

# 

散布矩阵数据索引图如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| X1X1密度图 | X2X1 | X3X1 | X4X1 | X5X1 |
| X1X2 | X2X2密度图 | X3X2 | X4X2 | X5X2 |
| X1X3 | X2X3 | X3X3密度图 | X4X3 | X5X3 |
| X1X4 | X2X4 | X3X4 | X4X4密度图 | X5X4 |
| X1X5 | X2X5 | X3X5 | X4X5 | X5X5密度图 |

接下来我们通过简单的散布图来验证我们的散布矩阵图：

X1X1密度图：

# 

X3X1 和 X1X3：

fig\_123,axec=plt.subplots(1,2,dpi=140)

for j in range(1):

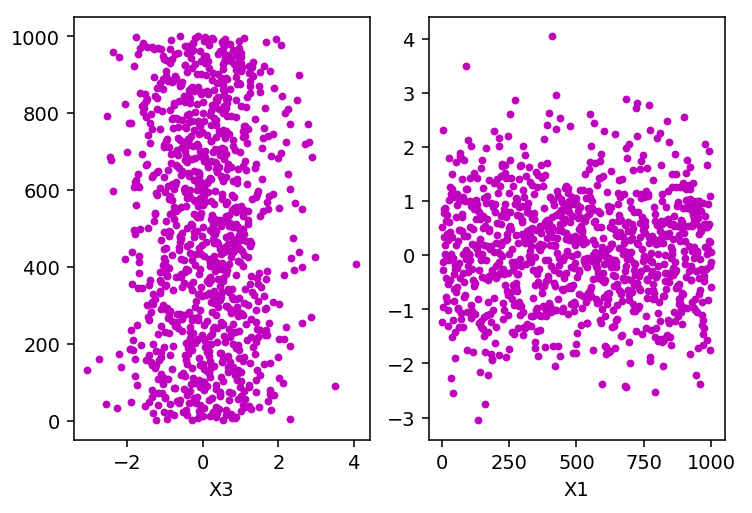
axec[j].scatter(test\_frame[2],test\_frame[0],color='m',marker='.')

axec[j].set\_xlabel('X3')

axec[j+1].scatter(test\_frame[0],test\_frame[2],color='m',marker='.')

axec[j+1].set\_xlabel('X1')

plt.show()



# **数据地图（basemap）**

from mpl\_toolkits.basemap import Basemap

...: import matplotlib.pyplot as plt

...: import pandas as pd

...: import numpy as np

...: from matplotlib import cm

# 绘制基础地图，选择绘制的区域，因为是绘制中国地图，故选取如下经纬度，lat\_0和lon\_0是地图中心的维度和经度。

china\_map=Basemap(projection='stere',lat\_0=34,lon\_0=115,llcrnrlat=28 ,urcrnrlat=42,llcrnrlon=105,urcrnrlon=129,rsphere=(2000,2000),resolution='l',area\_thresh=350)

#参数解释：

#Projection- 地图投影方式，常用的有'ortho'、'merc'、'stere'和’cyl’,’cass’、’lcc’等。

#llcrnrlat- 所需地图域左下角的纬度（度）。

#urcrnrlat- 所需地图域的右上角的纬度（度）。

#llcrnrlon- 所需地图域左下角的经度（度）。

#urcrnrlon- 所需地图域的右上角的经度（度）。

china\_map.drawmapboundary(color=’g’,zorder=0) # 绘制边界

china\_map.fillcontinents(color='y',lake\_color='b',zorder=2) # 填充大陆，发现填充之后无法显示散点图，应该是被覆盖了,因此取消

china\_map.drawstates(color='m',zorder=3) # 绘制省

china\_map.drawcoastlines(color='r',zorder=3) # 绘制海岸线，必须绘制，即使是不靠海也需绘制

china\_map.drawcountries(color='r',zorder=3)

#linewidth 设置线宽

#linestyle 设置线形。默认为 solid，可以是 dash，也可以是 matplotlib 其它选项。

#color 设置颜色。默认为 black(k)。

#antialiased 抗锯齿选项。默认为 True.

china\_map.drawrivers(linewidth=0.5, linestyle='solid', color='#1E90FF',zorder=3)

#zorder 设置图层位置。默认情况下由 Basemap 设置.

china\_map.drawlsmask(land\_color='0 ',ocean\_color='#1E90FF',zorder=1)

#china\_map.drawcountries(color='y') # 绘制国家，不太适合此例子，但需保留

#china\_map.bluemarble()

parallels = np.arange(28.,42.,2.)

china\_map.drawparallels(parallels,labels=[1,0,0,0],fontsize=10,zorder=4) # 绘制纬线

meridians = np.arange(105.,129.,3.)

china\_map.drawmeridians(meridians,labels=[0,0,0,1],fontsize=10,zorder=4) # 绘制经线

data\_1=pd.read\_csv(r'Desktop\dizhenshuju.csv',header=None,delimiter=',').values

lat=data\_1[:,2];lon=data\_1[:,1];Seismic\_grade=data\_1[:,3]

Seismic\_grade\_float=np.array(Seismic\_grade,dtype=np.float64)

class\_1=(Seismic\_grade\_float/np.max(Seismic\_grade\_float))\*5

x,y = china\_map(lon,lat)#地图上的精度维度匹配参数x，y

china\_map.scatter(x,y,s=class\_1,cmap=cm.hsv,c='#8A2BE2',zorder=5) # 使用matplotlib的散点图绘制函数

plt.savefig('Desktop\dong\_1217.pdf',dpi=300,bbox\_inches='tight')

plt.show()

**画布的复杂分割：**

**Subplot2grid**

在网格中创建一个子图。网格是由shape指定的，位于loc指定的位置，横跨各个方向上的rowspan个，colspan个单元格。 loc的索引是基于0的。

matplotlib.pyplot.subplot2grid(shape, loc, rowspan=1, colspan=1, \*\*kwargs)

例子：

def subpicnr\_invisible(fig):

for i, ax in enumerate(fig.axes): #利用list(enumerate(plt.gcf().axes))枚举函数生成元组列表[(画框编号,画框对象)(),...]

ax.text(0.5, 0.5, "ax%d" % (i+1))

plt.figure(0,dpi=150)

pic\_1=plt.subplot2grid((3,3),(0,0),colspan=3)

#画框pic\_1位于（0,0），这里的0意味着0行和0列，colspan=3意味着横跨三列，默认的rowspan=1意味着横跨一行

#ax2 = plt.subplot2grid((3, 3), (1, 0), colspan=2)

pic\_2 = plt.subplot2grid((3, 3), (1, 0), colspan=2)

#画框pic\_2位于（1,0），这里的0意味着1行和0列,（‘1’这一行有0和1两列），colspan=2意味着横跨两列，默认的rowspan=1意味着横跨一行。

pic\_3 = plt.subplot2grid((3,3), (1, 2), rowspan=2)

pic\_4 = plt.subplot2grid((3,3), (2, 0))

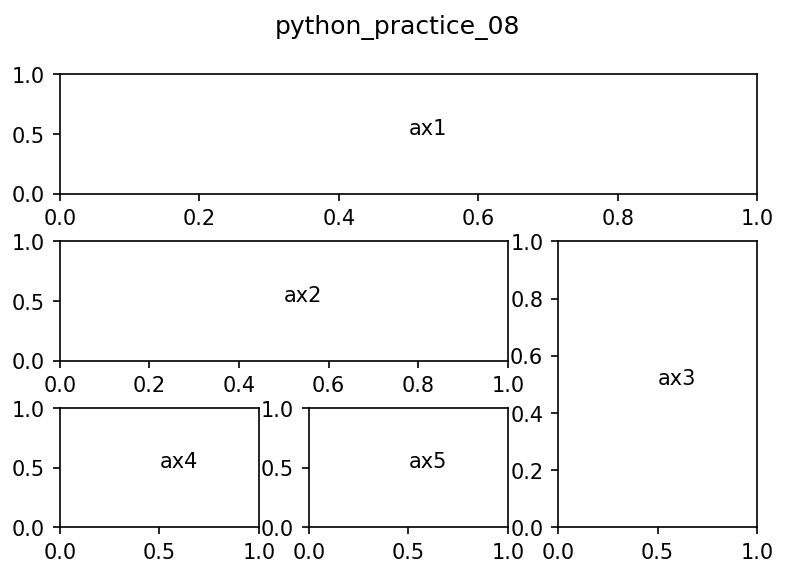
pic\_5 = plt.subplot2grid((3,3), (2, 1))

plt.subplots\_adjust(wspace=0.25,hspace=0.4)

plt.suptitle("python\_practice\_08")

subpicnr\_invisible(plt.gcf())#plt.gcf返回所有画布信息,

plt.show()



例子2

def subpicnr\_invisible(fig):

for i, ax in enumerate(fig.axes): #利用list(enumerate(plt.gcf().axes))枚举函数生成元组列表[(画框编号,画框对象)(),...]

ax.text(0.5, 0.5, "ax%d" % (i+1))

plt.figure(0,dpi=150)

pic\_1=plt.subplot2grid((3,3),(0,0),colspan=2)

#画框pic\_1位于（0,0），这里的0意味着0行和0列，colspan=3意味着横跨三列，默认的rowspan=1意味着横跨一行

pic\_2 = plt.subplot2grid((3, 3), (0, 2), colspan=1)

#pic\_2 = plt.subplot2grid((3, 3), (1, 0), colspan=2)

#画框pic\_2位于（1,0），这里的0意味着1行和0列,（‘1’这一行有0和1两列），colspan=2意味着横跨两列，默认的rowspan=1意味着横跨一行。

pic\_3 = plt.subplot2grid((3,3), (1, 0),rowspan=2,colspan=2)

pic\_4 = plt.subplot2grid((3,3), (1, 2))

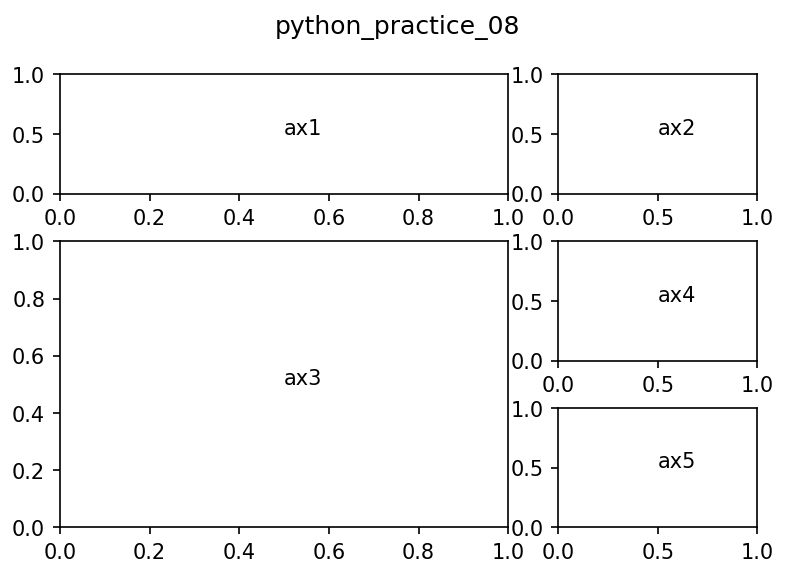
pic\_5 = plt.subplot2grid((3,3), (2, 2))

plt.subplots\_adjust(wspace=0.25,hspace=0.4)

plt.suptitle("python\_practice\_08")

subpicnr\_invisible(plt.gcf())#plt.gcf返回所有画布信息,

plt.show()

**GridSpec：**

例子1：

def subpicnr\_invisible(fig):

for i, ax in enumerate(fig.axes): #利用list(enumerate(plt.gcf().axes))枚举函数生成元组列表[(画框编号,画框对象)(),...]

ax.text(0.5, 0.5, "ax%d" % (i+1))

import matplotlib.gridspec as gridspec

plt.figure(0,dpi=150)

gs = gridspec.GridSpec(3, 3)

pic\_1=plt.subplot(gs[0,:])#按照数组索引理解即可

pic\_2 = plt.subplot(gs[1,:2])

pic\_3 = plt.subplot(gs[1:,-1])

pic\_4 = plt.subplot(gs[2,0])

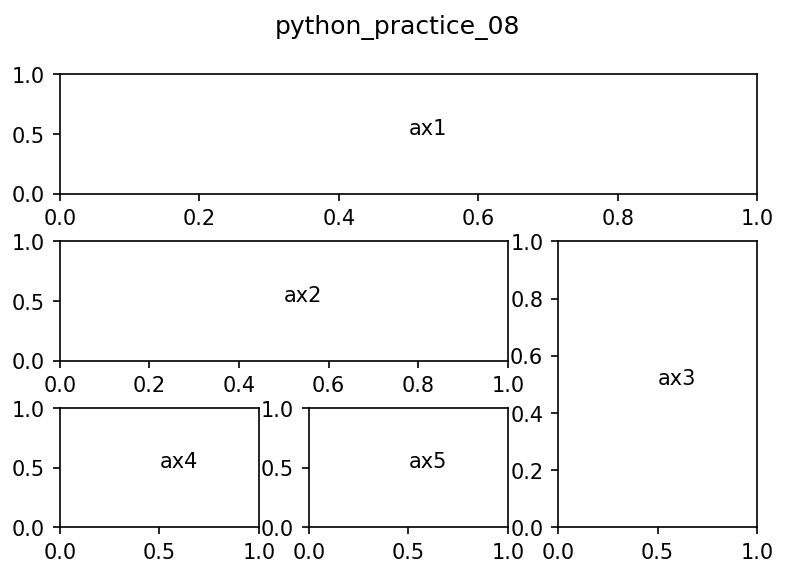
pic\_5 = plt.subplot(gs[2,1])

plt.subplots\_adjust(wspace=0.25,hspace=0.4)

plt.suptitle("python\_practice\_08")

subpicnr\_invisible(plt.gcf())#plt.gcf返回所有画布信息,

plt.show()



例子2：

def subpicnr\_invisible(fig):

for i, ax in enumerate(fig.axes): #利用list(enumerate(plt.gcf().axes))枚举函数生成元组列表[(画框编号,画框对象)(),...]

ax.text(0.5, 0.5, "ax%d" % (i+1))

import matplotlib.gridspec as gridspec

plt.figure(0,dpi=150)

gs = gridspec.GridSpec(3, 3)

pic\_1=plt.subplot(gs[0,:2])#按照数组索引理解即可

pic\_2 = plt.subplot(gs[0,-1])

pic\_3 = plt.subplot(gs[1:,:-1])

pic\_4 = plt.subplot(gs[1,-1])

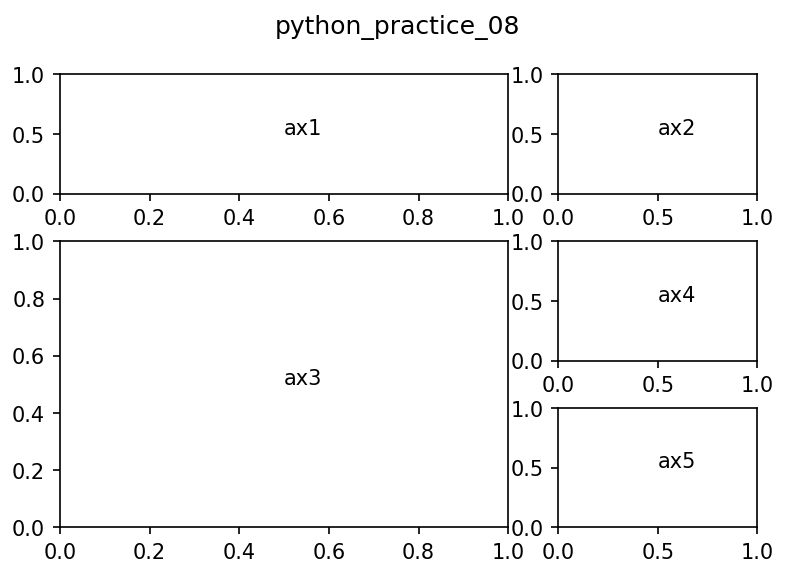
pic\_5 = plt.subplot(gs[2,-1])

plt.subplots\_adjust(wspace=0.25,hspace=0.4)

plt.suptitle("python\_practice\_08")

subpicnr\_invisible(plt.gcf())#plt.gcf返回所有画布信息,

plt.show()



例子：

plt.figure(dpi=100,figsize=(10,10))

gs = gridspec.GridSpec(3, 3)

pic\_1=plt.subplot(gs[0,:2])#按照数组索引理解即可

pic\_2 = plt.subplot(gs[0,-1])

pic\_3 = plt.subplot(gs[1:,:-1])

pic\_4 = plt.subplot(gs[1,-1])

pic\_5 = plt.subplot(gs[2,-1])

plt.subplots\_adjust(wspace=0.25,hspace=0.4)

plt.suptitle("python\_practice\_08",fontsize='20')

n = 730

X = np.linspace(-2\*np.pi,2\*np.pi,n)

Y = np.sin(2\*X)\*X+np.pi

Y\_1=np.sin(2\*X)\*X-np.pi

pic\_1.plot(X,Y,linestyle='-',color='#FFB6C1',alpha=1.00)

pic\_1.plot(X,Y\_1,linestyle='-.',color='#1E90FF',alpha=1.00)

n\_1=1000

X\_2 = np.random.normal(0,1,n\_1)

Y\_2= np.random.normal(0,1,n\_1)

pic\_2.scatter(X\_2,Y\_2,c=np.linspace(0,1,1000),cmap='coolwarm')

n\_2=10

X\_3 = np.arange(n\_2); Y\_3 = (1-X\_3/np.float(n\_2)) \* np.random.uniform(0,0.5,n\_2); Y\_3\_1 = -(1-X\_3/np.float(n\_2)) \* np.random.uniform(0,0.5,n\_2)

pic\_3.bar(X\_3,Y\_3,facecolor='#FFB6C1',width=0.6,align='center')

pic\_3.bar(X\_3,Y\_3\_1,facecolor='#87CEFA',width=0.6,align='center')

for x,y in zip(X\_3,Y\_3):

pic\_3.text(x, y+0.03,'%.2f' % y,ha='center')

for x,y in zip(X\_3,Y\_3\_1):

pic\_3.text(x, y-0.03,'%.2f' % y,ha='center')

f=lambda x,y: (1-x/2+x\*\*5+y\*\*3)\*np.exp(-x\*\*2-y\*\*2)

x\_4 = np.linspace(-3,3,1000)

y\_4 = np.linspace(-3,3,1000)

X\_4,Y\_4 = np.meshgrid(x\_4,y\_4)

pic\_4.contourf(X\_4, Y\_4, f(X\_4,Y\_4), 8, alpha=.75, cmap='coolwarm')

pic\_4.contour(X\_4, Y\_4, f(X\_4,Y\_4), 8, colors='black')

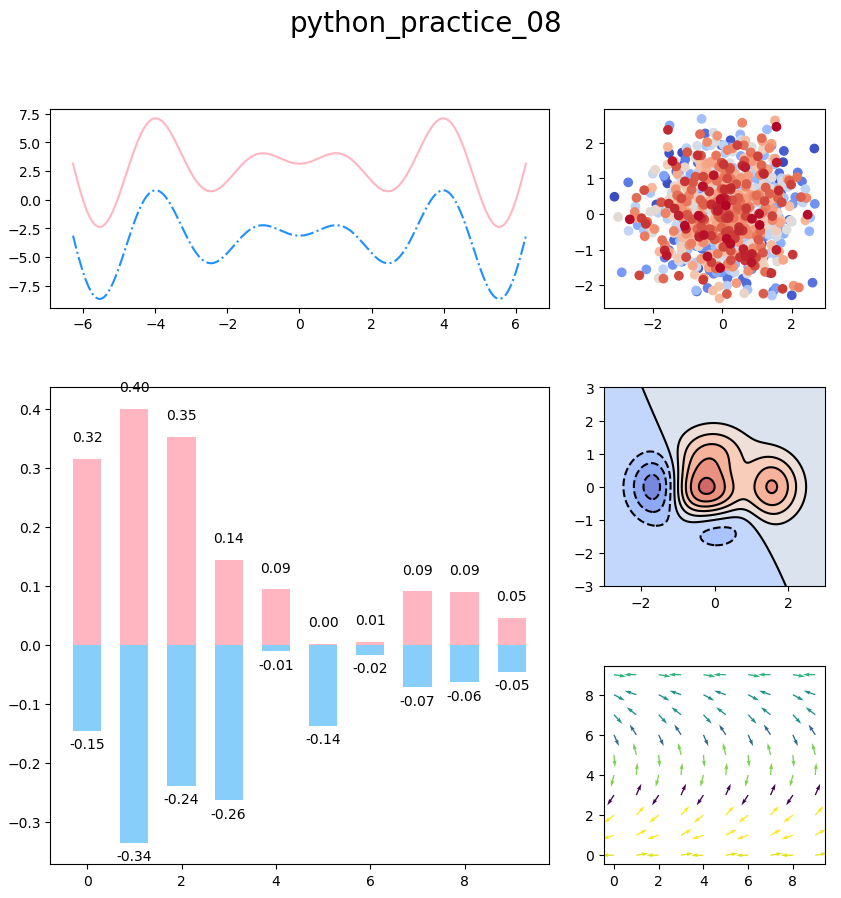
t=np.linspace(-np.pi,np.pi,20)

u,v=np.array([np.cos(theta) for theta in t]),np.array([np.sin(theta) for theta in t])

X\_5,Y\_5= np.mgrid[0:10,0:10]#晶格化

pic\_5.quiver(X\_5,Y\_5,u,v,np.random.randn(10))

plt.show()



# 绘制立体图

**曲面图：**

plot\_surface

plot\_surface(X, Y, Z, \*args, \*\*kwargs)

默认情况下，它将以纯色着色，但它也通过提供\* cmap \*参数来支持颜色映射。

'rstride`和`cstride` kwargs设置了用于对输入数据进行采样以生成图形的步幅。如果传入1k个1k数组，则步幅的默认值将导致绘制100x100的网格。 默认为10。 如果同时提供了stride和count kwargs（rcount、ccount），则引发ValueError。

`rcount`和`ccount` kwargs取代`rstride`和`cstride`作为表面绘图的默认采样方法。这些参数将决定从输入数据中最多取多少个均匀间隔的样本来生成图。 这是默认的采样方法，除非使用“经典”风格。 如果同时指定了步幅和数量（stride and count），将会引发ValueError。

参数：

\* X \*，\* Y \*，\* Z \* 数据值为二维数组

\* rstride \* 数组行步幅（步长）

\* cstride \* 数组列步幅（步长）

\* rcount \* 最多使用行，默认为50

\* ccount \* 最多使用列，默认为50

\*颜色\* 曲面片的颜色

\* cmap \* 曲面片调色板。

\* facecolors \* 单个曲面片的表面色

\* norm \* 一个标准化实例，用于将值映射到颜色

\* vmin \* 映射的最小值

\* vmax \* 映射的最大值

\*shade\* 是否遮蔽表面色

例子：参数化坐标轴下的三维球

from mpl\_toolkits.mplot3d import Axes3D

import numpy as np

import matplotlib.pyplot as plt

u = np.linspace(0,2\*np.pi,1000)

v = np.linspace(0,np.pi,1000)

x=10\*np.outer(np.sin(v),np.cos(u))

y=10\*np.outer(np.sin(v),np.sin(u))

z=10\*np.outer(np.cos(v),np.ones(len(np.cos(v))))

#创建二维数据集X,Y和Z，注意他们的值必须在各矩阵相同位置处一一对应。

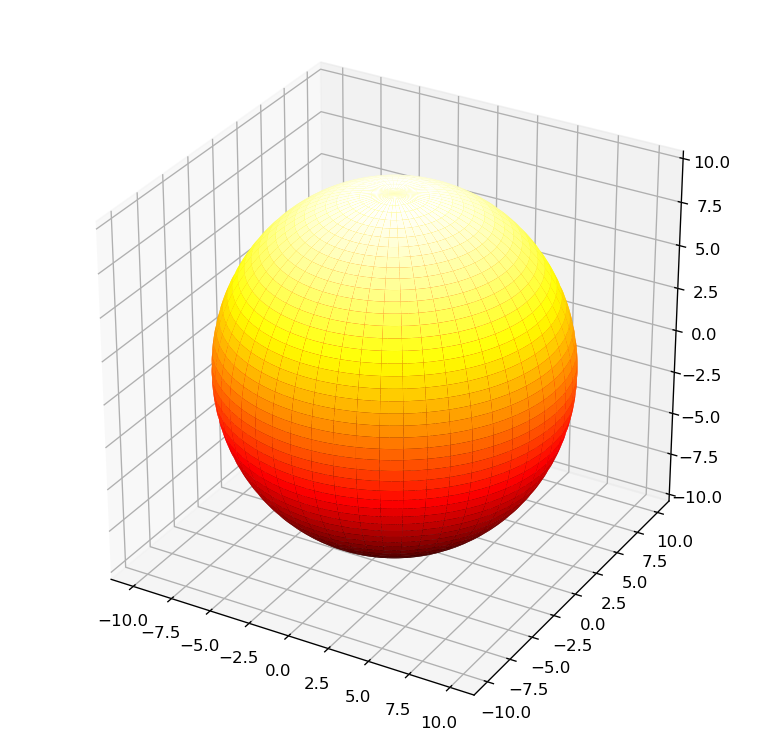
fig=plt.figure(0,figsize=(8,8),dpi=120)

ax=fig.add\_subplot(1,1,1,projection='3d')

#ax.plot\_surface(x,y,z,rcount=1000,ccount=1000,cmap='coolwarm')#太浪费时间了，用下面替换语句

ax.plot\_surface(x,y,z,cmap='hot')

plt.show()



例子：非参数化坐标轴下曲面图

import numpy as np

from mpl\_toolkits.mplot3d import Axes3D

import matplotlib.pyplot as plt

import random

def fun(x, y):

return np.sqrt(x\*\*2 + y\*\*2)

fig = plt.figure(0,dpi=120,figsize=(6,6))

ax = fig.add\_subplot(111, projection='3d')

x = y = np.linspace(-5.0, 5.0, 100)

X, Y = np.meshgrid(x, y)

zs = np.array([fun(x,y) for x,y in zip(np.ravel(X), np.ravel(Y))])

Z = zs.reshape(Y.shape)

ax.plot\_surface(X, Y, Z,cmap='jet')

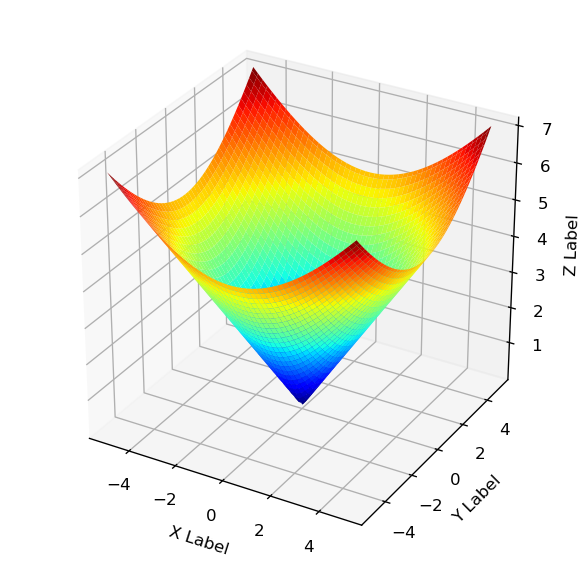
ax.set\_xlabel('X Label')

ax.set\_ylabel('Y Label')

ax.set\_zlabel('Z Label')

#(np.ravel(X)).shape

plt.show()



import numpy as np

from mpl\_toolkits.mplot3d import Axes3D

import matplotlib.pyplot as plt

import random

def fun(x, y):

return np.cos(np.sqrt(x\*\*2 + y\*\*2))

fig = plt.figure(0,dpi=120,figsize=(6,6))

ax = fig.add\_subplot(111, projection='3d')

x = y = np.linspace(-5.0, 5.0, 100)

X, Y = np.meshgrid(x, y)

zs = np.array([fun(x,y) for x,y in zip(np.ravel(X), np.ravel(Y))])

Z = zs.reshape(Y.shape)

ax.plot\_surface(X, Y, Z,cmap='jet')

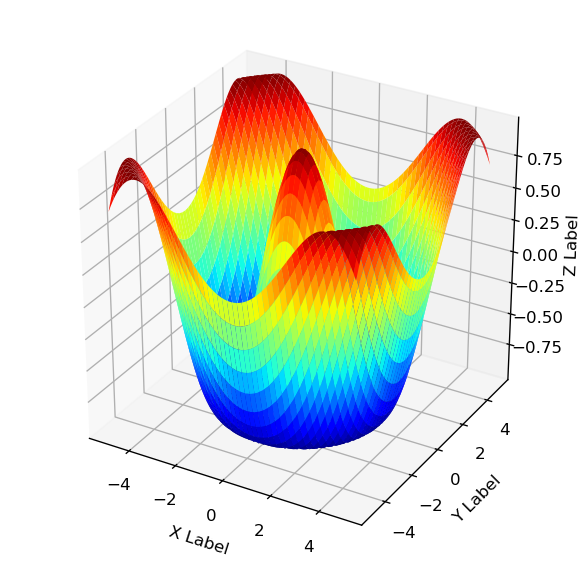
ax.set\_xlabel('X Label')

ax.set\_ylabel('Y Label')

ax.set\_zlabel('Z Label')

#(np.ravel(X)).shape

plt.show()



例子，带色度条的三维图

import numpy as np

from mpl\_toolkits.mplot3d import Axes3D

import matplotlib.pyplot as plt

import random

f=lambda x,y: (1-x/2+x\*\*5+y\*\*3)\*np.exp(-x\*\*2-y\*\*2)

fig = plt.figure(0,dpi=160,figsize=(10,10))

ax = fig.add\_subplot(111, projection='3d')

x = y = np.linspace(-5.0, 5.0, 100)

X, Y = np.meshgrid(x, y)

zs = np.array([f(x,y) for x,y in zip(np.ravel(X), np.ravel(Y))])

Z = zs.reshape(Y.shape)

pic\_3dim=ax.plot\_surface(X, Y, Z,cmap='Spectral\_r')

plt.colorbar(pic\_3dim) #画出色度条

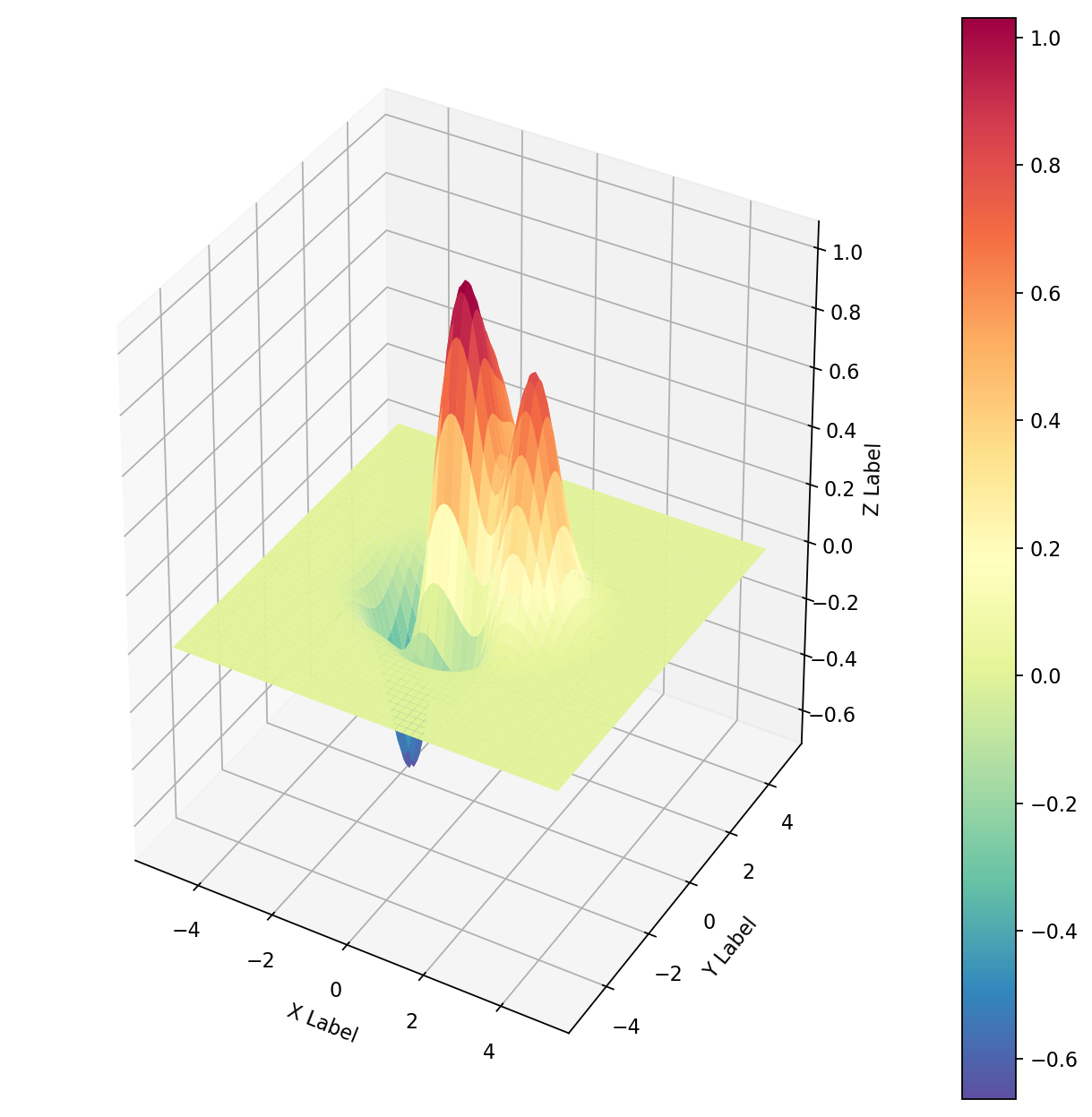
ax.set\_xlabel('X Label')

ax.set\_ylabel('Y Label')

ax.set\_zlabel('Z Label')

#(np.ravel(X)).shape

plt.show()



另外的一种表面图绘制方法：

例如：

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D #绘制3D坐标的函数

import numpy as np

def fun(x,y):

return np.power(x,2)+np.sin(np.power(y,2))\*x

fig1=plt.figure(0,dpi=160,figsize=(8,8))

ax=Axes3D(fig1)#三维化画布并产生三维画框

X=np.arange(-3,3,0.05)

Y=np.arange(-3,3,0.05)

X,Y=np.meshgrid(X,Y)#生成坐标点

Z=fun(X,Y)

plt.title('python-08')

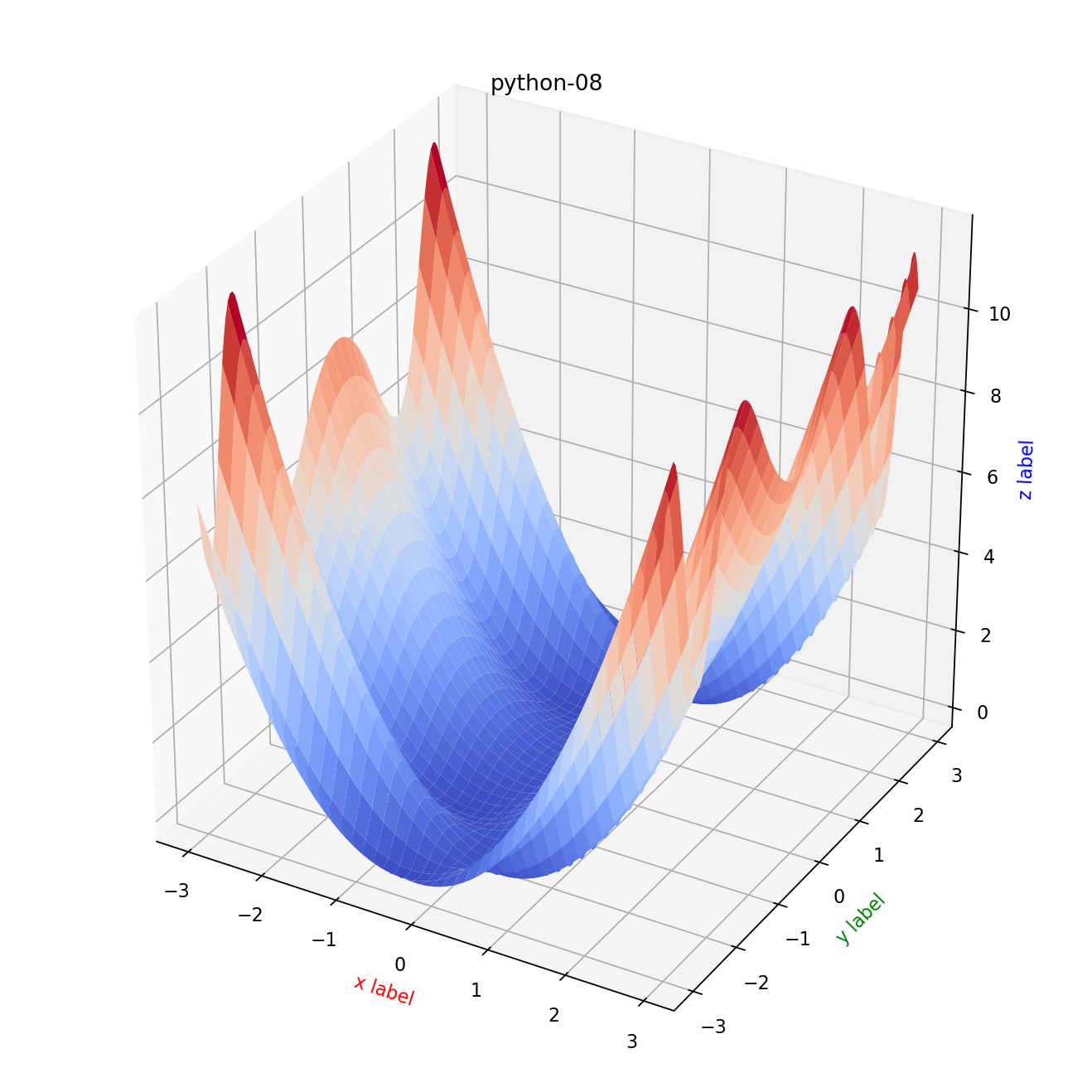
ax.plot\_surface(X, Y, Z,cmap='coolwarm')

ax.set\_xlabel('x label', color='r')

ax.set\_ylabel('y label', color='g')

ax.set\_zlabel('z label', color='b')

plt.show()



# 曲线图

例子：我的葫芦

from mpl\_toolkits.mplot3d import Axes3D

fig = plt.figure(0,dpi=100,figsize=(6,6))

ax = fig.add\_subplot(1,1,1, projection='3d')

theta = np.linspace(-20 \* np.pi, 20 \* np.pi, 1000)

z = np.linspace(0,10,1000);phi=np.linspace(0,2\*np.pi,1000)

r=z\*np.sin(phi)

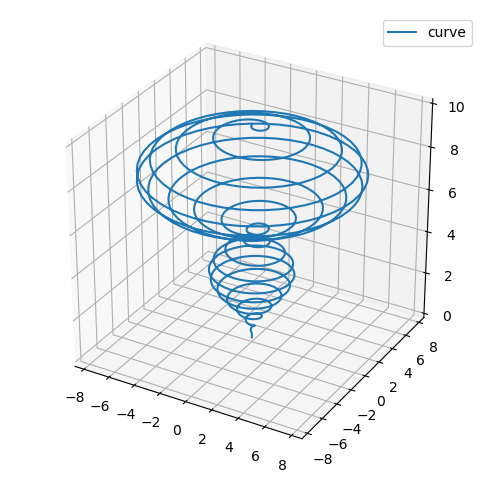
x = r \* np.sin(theta)

y = r \* np.cos(theta)

ax.plot(x, y, z, label='curve')

ax.legend()

plt.show()



# 散点图（见LDA）

**线框图**

例子：

import numpy as np

def fun(x,y):

return np.power(x,2)+np.sin(np.power(y,2))\*x

fig1=plt.figure(0,dpi=160,figsize=(8,8))

ax=Axes3D(fig1)#三维化画布并产生三维画框

X=np.arange(-3,3,0.05)

Y=np.arange(-3,3,0.05)

X,Y=np.meshgrid(X,Y)#生成坐标点

Z=fun(X,Y)

plt.title('python-08')

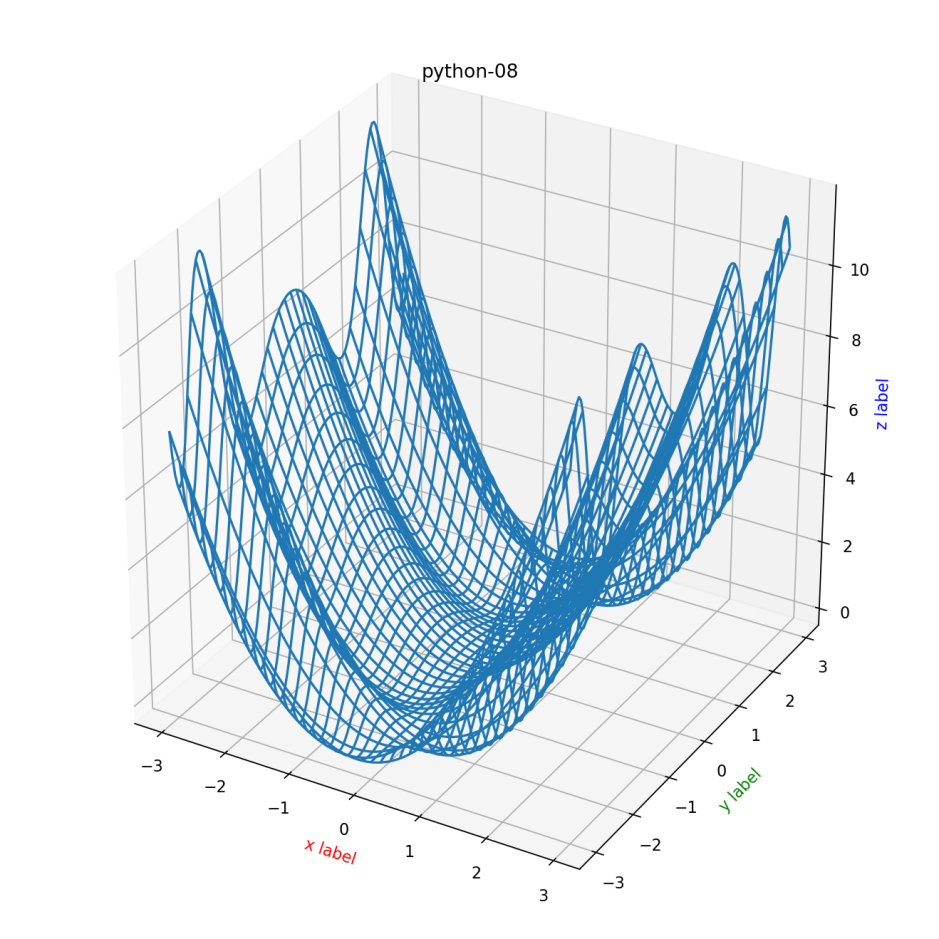
ax.plot\_wireframe(X, Y, Z, rstride=3, cstride=3)#一定要调节成大的扫描步长才有效果

ax.set\_xlabel('x label', color='r')

ax.set\_ylabel('y label', color='g')

ax.set\_zlabel('z label', color='b')

plt.show()



# 等高线图

contour和contourf都是画三维等高线图的，不同点在于contourf会对等高线间的区域进行填充。

参数：

\*X\*, \*Y\*, \*Z\* 数组型数据

\*extend3d\* 是否在3D中扩展等高线图（默认值：False）

\*stride\* \*步幅\*，用于扩展等高线图的步幅（步长）

\*zdir\* 等高线图产生方向: x, y 或 z (default)

\*offset\* 如果赋值，绘制等高线投影到垂直于zdir并且通过偏移量确定位置的平面

例子：

import numpy as np

def fun(x,y):

return (1/(2\*np.pi))\*np.exp(-0.5\*(x\*\*2+y\*\*2))

fig1=plt.figure(0,dpi=160,figsize=(8,8))

ax=Axes3D(fig1)#三维化画布并产生三维画框

X=np.arange(-3,3,0.05)

Y=np.arange(-3,3,0.05)

X,Y=np.meshgrid(X,Y)#生成坐标点

Z=fun(X,Y)

plt.title('python-08')

ax.plot\_surface(X, Y, Z, rstride=3, cstride=3,cmap='jet')

ax.contour(X, Y, Z, zdir='z',offset=0.16,cmap='coolwarm')

cset = ax.contour(X, Y, Z, zdir='x', offset=-3, cmap='coolwarm')

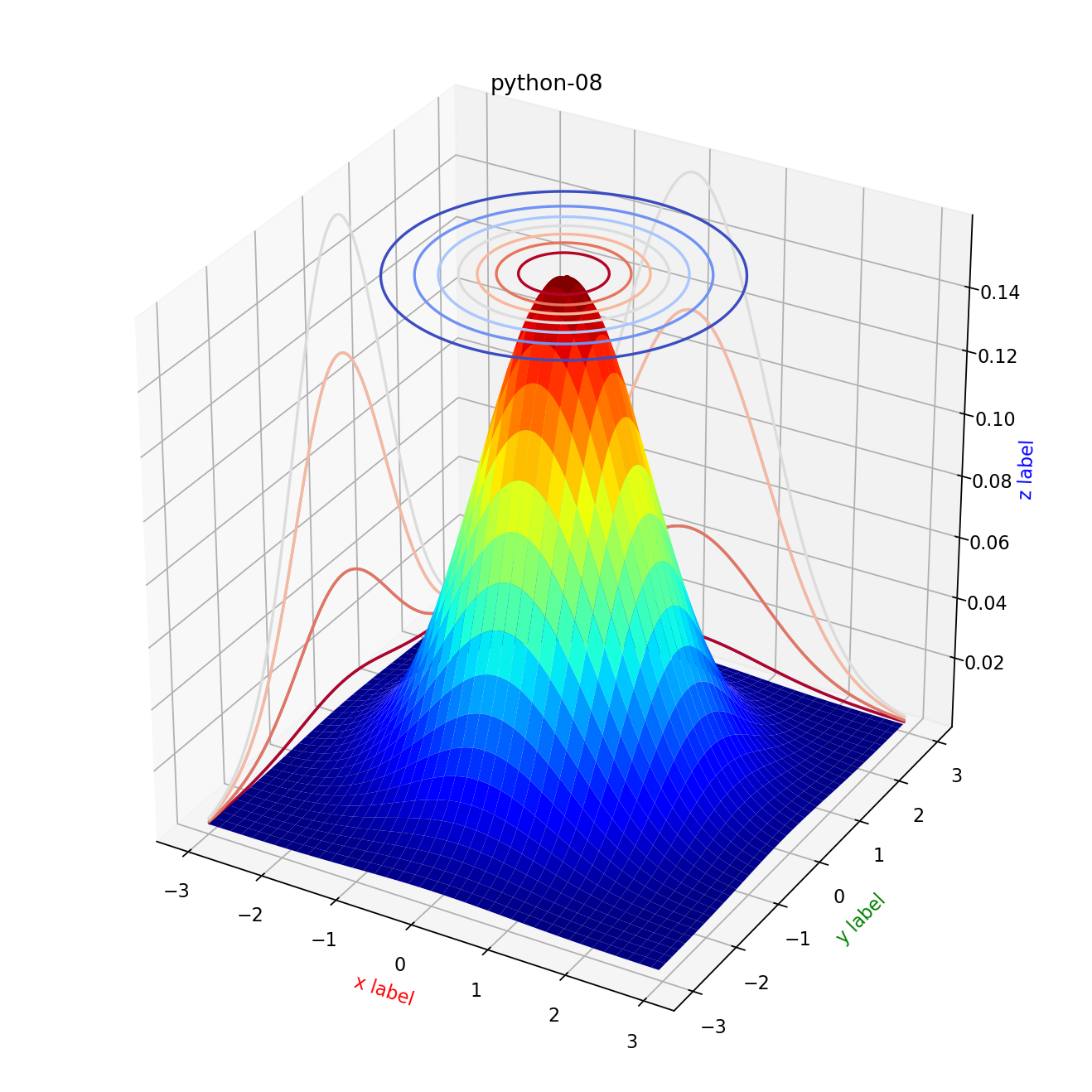
cset = ax.contour(X, Y, Z, zdir='y', offset=3, cmap='coolwarm')

ax.set\_xlabel('x label', color='r')

ax.set\_ylabel('y label', color='g')

ax.set\_zlabel('z label', color='b')

plt.show()



**动态图**

运动中的布朗运动

from matplotlib import pyplot as plt

from matplotlib import animation

import numpy as np

%matplotlib qt5

def randn\_point():

# 产生随机散点图的x和y数据

x=np.random.randn(100)

y=np.random.randn(100)

return x,y

fig,ax1=plt.subplots(1,1,dpi=130,figsize=(8,6))

# 先绘制初始图形

x1,y1=randn\_point()

sca1 = ax1.scatter(x1,y1) # 散点图

def init():

# 构造开始帧函数init

# 改变散点图数据

x1, y1 = randn\_point()

data1 = [[x,y] for x,y in zip(x1,y1)]

sca1.set\_offsets(data1) # 散点图

label = 'timestep {0}'.format(0)

ax1.set\_xlabel(label)

return sca1,ax1 # 注意返回值，我们要更新的就是这些数据

def animate(i):

# 接着，构造自定义动画函数animate，用来更新每一帧上各个x对应的y坐标值，参数表示第i帧

x1, y1 = randn\_point()

x2, y2 = randn\_point()

data1 = [[x,y] for x,y in zip(x1,y1)]

sca1.set\_offsets(data1) # 散点图

label = 'timestep {0}'.format(i)

ax1.set\_xlabel(label)

return sca1,ax1

# 接下来，我们调用FuncAnimation函数生成动画。参数说明：

# fig 进行动画绘制的figure

# func 自定义动画函数，即传入刚定义的函数animate

# frames 动画长度，一次循环包含的帧数

# init\_func 自定义开始帧，即传入刚定义的函数init

# interval 更新频率，以ms计

# blit 选择更新所有点，还是仅更新产生变化的点。应选择True，但mac用户请选择False，否则无法显示动画

ani = animation.FuncAnimation(fig=fig,func=animate,frames=1000,init\_func=init,interval=100,blit=False)

plt.show()

布朗运动过程模拟：

from matplotlib import pyplot as plt

from matplotlib import animation

import numpy as np

%matplotlib qt5

def randn\_point(t):

Data=np.random.randn(100,10001)

brown\_moving=Data.T[0]\*t+sum([Data\_x\*((np.sqrt(2)\*np.sin(i\*np.pi\*t))/(i\*np.pi)) for Data\_x,i in list(zip(Data.T[1:],range(1,10001)))])

angel=np.random.randn(100)\*360

return brown\_moving\*np.cos(angel),brown\_moving\*np.sin(angel)

fig,ax1=plt.subplots(1,1,dpi=130,figsize=(8,6))

# 先绘制初始图形

plt.xlim([-100,100])

#plt.ylim([-50,50])

x1,y1=randn\_point(np.random.randint(100))

sca1 = ax1.scatter(x1,y1)

def init():

# 构造开始帧函数init

# 改变散点图数据

x1, y1 = randn\_point(np.random.randint(100))

data1 = [[x,y] for x,y in zip(x1,y1)]

sca1.set\_offsets(data1) # 散点图

label = 'timestep {0}'.format(0)

ax1.set\_xlabel(label)

return sca1,ax1 # 注意返回值，我们要更新的就是这些数据

def animate(i):

# 接着，构造自定义动画函数animate，用来更新每一帧上各个x对应的y坐标值，参数表示第i帧

x1, y1 = randn\_point(i)

x2, y2 = randn\_point(i)

data1 = [[x,y] for x,y in zip(x1,y1)]

sca1.set\_offsets(data1) # 散点图

label = 'timestep {0}'.format(i)

ax1.set\_xlabel(label)

return sca1,ax1

ani = animation.FuncAnimation(fig=fig,func=animate,frames=100,init\_func=init,interval=1,blit=False)

plt.show()