pandas入门

1.pandas的数据结构介绍

1.1 Series

Series 的构建,Series 类似一位数组的对象，它是由一组数据和一组与之相关的数据标签组成

In [1]:

```
import pandas as pd
obj = pd.Series([4,7,-5,3])
obj
```

Out[1]:

```
0    4
1    7
2   -5
3    3
dtype: int64
```

In [2]:

```
obj.values
```

Out[2]:

```
array([ 4,  7, -5,  3], dtype=int64)
```

In [3]:

```
obj.index
```

Out[3]:

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

In [4]:

```
obj2 = pd.Series([4,5,7,3],index = ['a','b','c','d'])
obj2
```

Out[4]:

```
a    4
b    5
c    7
d    3
dtype: int64
```

In [5]:

```
import pandas as pd
obj = pd.Series([4,7,-5,3])
obj
```

Out[5]:

```
0    4
1    7
2   -5
3    3
dtype: int64
```

In [6]:

```
obj2.index
```

Out[6]:

```
Index(['a', 'b', 'c', 'd'], dtype='object')
```

Series 的索引

与普通的numpy数组相比，可以通过索引的方式选择series中的单个或者一组数值

In [7]:

```
obj2['a']
```

Out[7]:

```
4
```

In [8]:

```
obj2['a'] = 5
obj2['a']
```

Out[8]:

```
5
```

In [9]:

```
obj2[['a','b','c']]
```

Out[9]:

```
a    5
b    5
c    7
dtype: int64
```

Series的运算

In [10]:

```
obj2[obj2>5]
```

Out[10]:

```
c    7
dtype: int64
```

In [11]:

```
obj2*2
```

Out[11]:

```
a    10
b    10
c    14
d     6
dtype: int64
```

In [12]:

```
import numpy as np
np.exp(obj2)
```

Out[12]:

```
a     148.413159
b     148.413159
c    1096.633158
d      20.085537
dtype: float64
```

obj可以看作是一个定长的有序字典，因为它是索引导数据值的一个映射

In [13]:

```
’b’ in obj2
```

Out[13]:

```
True
```

In [14]:

```
’e’ in obj2
```

Out[14]:

```
False
```

如果数据被放在一个python的字典中，可以通过这个字典来创建series

In [15]:

```
sdata = {’first’:1,’second’:2,’third’:3}
obj3 = pd.Series(sdata)
obj3
```

Out[15]:

```
first     1
second    2
third     3
dtype: int64
```

索引可以设定,并且挑选出符合索引的数值（nan表示缺失）

In [16]:

```
states = [’hha’,’second’]
obj4 = pd.Series(sdata,index = states)
obj4
```

Out[16]:

```
hha       NaN
second    2.0
dtype: float64
```

检测缺失数字

In [17]:

```
pd.isnull(obj4)
```

Out[17]:

```
hha        True
second    False
dtype: bool
```

In [18]:

```
pd.notnull(obj4)
```

Out[18]:

```
hha       False
second     True
dtype: bool
```

In [19]:

```
obj4.isnull()
```

Out[19]:

```
hha        True
second    False
dtype: bool
```

Series最重要的一个功能是：他在算数运算中会自动对齐不同索引的数据

In [20]:

```
obj3
```

Out[20]:

```
first     1
second    2
third     3
dtype: int64
```

In [21]:

```
obj4
```

Out[21]:

```
hha       NaN
second    2.0
dtype: float64
```

In [22]:

```
obj3+obj4
```

Out[22]:

```
first     NaN
hha       NaN
second    4.0
third     NaN
dtype: float64
```

Series对象本身及其都有一个name属性

In [23]:

```
obj4.name = 'population'
obj4.index.name = 'state'
obj4
```

Out[23]:

```
state
hha       NaN
second    2.0
Name: population, dtype: float64
```

In [24]:

```
obj4.index.name
```

Out[24]:

```
'state'
```

In [25]:

```
obj4['hha']
```

Out[25]:

```
nan
```

1.2.DataFrame

dataframe 是一个表格的数据结构，它含有一组有序的列，每列可以是不同的值类型。 可以被看作是由series组成的字典

dataframe的构建（最常用的是直接传入一个等长列表或者numpy数组组成的字典）

In [26]:

```
from pandas import DataFrame
data ={'state':['ohio','ohio','ohio','nevada','nevada'],
       'year':[2000,2001,2002,2001,2002],
       'pop':[1.5,1.7,3.6,2.4,2.9]
      }
frame = DataFrame(data)
frame
```

Out[26]:

	state	year	pop
0	ohio	2000	1.5
1	ohio	2001	1.7
2	ohio	2002	3.6
3	nevada	2001	2.4
4	nevada	2002	2.9

可以指定列序列，并且改变所有的行

In [27]:

```
DataFrame(data,columns = [’year’,’state’,’pop’])
```

Out[27]:

	year	state	pop
0	2000	ohio	1.5
1	2001	ohio	1.7
2	2002	ohio	3.6
3	2001	nevada	2.4
4	2002	nevada	2.9

和series一样，如果传入的列在数据中找不到，就会产生na数值

In [28]:

```
frame2 = DataFrame(data,columns = [’year’,’state’,’pop’,’debt’],index = [’one’,’two’,’three’,’fo
ur’,’five’])
frame2
```

Out[28]:

	year	state	pop	debt
one	2000	ohio	1.5	NaN
two	2001	ohio	1.7	NaN
three	2002	ohio	3.6	NaN
four	2001	nevada	2.4	NaN
five	2002	nevada	2.9	NaN

In [29]:

```
frame2.columns
```

Out[29]:

```
Index([’year’, ’state’, ’pop’, ’debt’], dtype=’object’)
```

可将dataframe的列获取为一个series

In [30]:

```
frame2[’state’]
```

Out[30]:

```
one        ohio
two        ohio
three      ohio
four     nevada
five     nevada
Name: state, dtype: object
```

In [31]:

```
frame2.year
```

Out[31]:

```
one      2000
two      2001
three    2002
four     2001
five     2002
Name: year, dtype: int64
```

通过行数或者标签获得信息

In [32]:

```
frame2.ix['three']
```

```
c:\users\administrator\appdata\local\programs\python\python37\lib\site-packages\ip
ykernel_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[32]:

```
year     2002
state    ohio
pop       3.6
debt      NaN
Name: three, dtype: object
```

列也可以通过赋值的方式进行修改

In [33]:

```
frame2['debt'] = np.arange(5.)
frame2
```

Out[33]:

	year	state	pop	debt
one	2000	ohio	1.5	0.0
two	2001	ohio	1.7	1.0
three	2002	ohio	3.6	2.0
four	2001	nevada	2.4	3.0
five	2002	nevada	2.9	4.0

In [34]:

```
val = pd.Series([-1.2,-1.5,-1.7],index = ['two','four','five'])
frame2['debt'] = val
frame2
```

Out[34]:

	year	state	pop	debt
one	2000	ohio	1.5	NaN
two	2001	ohio	1.7	-1.2
three	2002	ohio	3.6	NaN
four	2001	nevada	2.4	-1.5
five	2002	nevada	2.9	-1.7

为不存在的列赋值会创建一个新列

In [35]:

```
frame2['eastern'] = frame2.state =='ohio'
frame2
```

Out[35]:

	year	state	pop	debt	eastern
one	2000	ohio	1.5	NaN	True
two	2001	ohio	1.7	-1.2	True
three	2002	ohio	3.6	NaN	True
four	2001	nevada	2.4	-1.5	False
five	2002	nevada	2.9	-1.7	False

In [36]:

```
del frame2['pop']
```

In [37]:

```
frame2.columns
```

Out[37]:

```
Index(['year', 'state', 'debt', 'eastern'], dtype='object')
```

dataframe 另外一种构建方式（嵌套字典)

In [38]:

```
pop = {'nevada':{2001:2.4,2002:2.9},
      'ohio':{2000:1.7,2002:3.6}}
```

如果将上述传给dataframe 那么就会被解释为：字典的键值作为列，内层键值作为行索引

In [39]:

```
frame3 = DataFrame(pop)
frame3
```

Out[39]:

	nevada	ohio
2000	NaN	1.7
2001	2.4	NaN
2002	2.9	3.6

dataframe的转置

In [40]:

```
frame3.T
```

Out[40]:

	2000	2001	2002
nevada	NaN	2.4	2.9
ohio	1.7	NaN	3.6

dataframe可以进行索引的显示指定

In [41]:

```
DataFrame(pop, index = [2000,2001,2002,2003])
```

Out[41]:

	nevada	ohio
2000	NaN	1.7
2001	2.4	NaN
2002	2.9	3.6
2003	NaN	NaN

In [42]:

```
frame3[’ohio’]
```

Out[42]:

```
2000    1.7
2001    NaN
2002    3.6
Name: ohio, dtype: float64
```

In [43]:

```
frame3[’ohio’][:1]
```

Out[43]:

```
2000    1.7
Name: ohio, dtype: float64
```

[:-1]表示除去最后一行的其他所有行

In [44]:

```
frame3[’ohio’][:-1]
```

Out[44]:

```
2000    1.7
2001    NaN
Name: ohio, dtype: float64
```

In [45]:

```
pdata = {’ohio’:frame3[’ohio’][:-1],
         ’nevada’:frame3[’nevada’][:2]
        }
DataFrame(pdata)
```

Out[45]:

	ohio	nevada
2000	1.7	NaN
2001	NaN	2.4

可以输入给dataframe构造器的数据

jupyter

如果设置了dataframe的index和columns的name属性，那么这些信息也会被显示出来

In [46]:

```
frame3.index.name = 'year';frame3.columns.name = 'state'
frame3
```

Out[46]:

state	nevada	ohio
year		
2000	NaN	1.7
2001	2.4	NaN
2002	2.9	3.6

2.索引对象

pandas的索引对象负责管理轴标签和其它元数据（比如：轴名称）

In [47]:

```
import numpy as np
obj = pd.Series(np.arange(3),index = ['a','b','c'])
index = obj.index
index
```

Out[47]:

```
Index(['a', 'b', 'c'], dtype='object')
```

In [48]:

```
index[1:]
```

Out[48]:

```
Index(['b', 'c'], dtype='object')
```

index对象不可以进行修改，这个非常重要，保证了index对象在多个数据结构之间安全共享

index[1] = 'a'是错误的

In [49]:

```
index = pd.Index(np.arange(3))
obj2 = pd.Series([1.5,-2.5,0],index = index)
obj2.index is index
```

Out[49]:

```
True
```

jupyter

Index除了长得像数组，Index的功能也类似一个的固定大小的集合

In [50]:

```
frame3
```

Out[50]:

state	nevada	ohio
year		
2000	NaN	1.7
2001	2.4	NaN
2002	2.9	3.6

In [51]:

```
'ohio' in frame3.columns
```

Out[51]:

```
True
```

In [52]:

```
2000 in frame3.index
```

Out[52]:

```
True
```

jupyter

3.基本功能

3.1重新索引，创建一个适应新索引的新对象

In [53]:

```
obj = pd.Series([4.5,7.2,-5.3,3.6],index = ['a','b','c','d'])
obj
```

Out[53]:

```
a    4.5
b    7.2
c   -5.3
d    3.6
dtype: float64
```

In [54]:

```
obj2 = obj.reindex(['a','b','c','d','e'])
obj2
```

Out[54]:

```
a    4.5
b    7.2
c   -5.3
d    3.6
e    NaN
dtype: float64
```

In [55]:

```
obj2 = obj.reindex(['a','b','c','d','e'],fill_value = 0)
obj2
```

Out[55]:

```
a    4.5
b    7.2
c   -5.3
d    3.6
e    0.0
dtype: float64
```

In [56]:

```
obj3 = pd.Series(['blue','purple','yellow'],index = [0,2,4])
obj3.reindex(range(6),method = 'ffill')
```

Out[56]:

```
0      blue
1      blue
2    purple
3    purple
4    yellow
5    yellow
dtype: object
```

jupyter

In [57]:

```
frame = pd.DataFrame (np.arange(9).reshape((3,3)),index = ['a','c','d'],columns = ['ohio','texas','california'])
frame
```

Out[57]:

	ohio	texas	california
a	0	1	2
c	3	4	5
d	6	7	8

In [58]:

```
frame2 = frame.reindex(['a','b','c','d'])
frame2
```

Out[58]:

	ohio	texas	california
a	0.0	1.0	2.0
b	NaN	NaN	NaN
c	3.0	4.0	5.0
d	6.0	7.0	8.0

In [59]:

```
states = ['texas','utah','california']
frame.reindex(columns = states)
```

Out[59]:

	texas	utah	california
a	1	NaN	2
c	4	NaN	5
d	7	NaN	8

In [60]:

```
frame.reindex(index = ['a','b','c','d'],columns = states )
```

Out[60]:

	texas	utah	california
a	1.0	NaN	2.0
b	NaN	NaN	NaN
c	4.0	NaN	5.0
d	7.0	NaN	8.0

jupyter

4.丢弃指定轴上的项

In [61]:

```
obj = pd.Series(np.arange(5.),index = ['a','b','c','d','e'])
obj
```

Out[61]:

```
a     0.0
b     1.0
c     2.0
d     3.0
e     4.0
dtype: float64
```

In [62]:

```
new_obj = obj.drop('c')
new_obj
```

Out[62]:

```
a     0.0
b     1.0
d     3.0
e     4.0
dtype: float64
```

In [63]:

```
obj.drop(['d','c'])
```

Out[63]:

```
a     0.0
b     1.0
e     4.0
dtype: float64
```

In [64]:

```
data = pd.DataFrame(np.arange(16).reshape((4,4)),
                  index = ['ohio','colorado','utah','new work'],
                  columns = ['one','two','three','gour'])
data
```

Out[64]:

	one	two	three	gour
ohio	0	1	2	3
colorado	4	5	6	7
utah	8	9	10	11
new work	12	13	14	15

In [65]:

```
data.drop(['ohio','colorado'])
```

Out[65]:

	one	two	three	gour
utah	8	9	10	11
new work	12	13	14	15

In [66]:

```
data.drop('two',axis = 1)
```

Out[66]:

	one	three	gour
ohio	0	2	3
colorado	4	6	7
utah	8	10	11
new work	12	14	15

In [67]:

```
data.drop(['two','gour'],axis = 1)
```

Out[67]:

	one	three
ohio	0	2
colorado	4	6
utah	8	10
new work	12	14

5.索引、选取和过滤

In [ ]:

In [ ]:

In [ ]: