

一 Pandas.....	1
1 Series.....	1
1.1 Series 的构造.....	1
1.1.1 创建一个空的 Series 对象:	1
1.1.2 通过一个 ndarray 对象传递到 data 参数.....	1
1.1.3 从字典创建一个 Series 对象.....	2
1.1.4 通过标量值创建对象.....	3
1.2 Series 基础操作.....	3
1.2.1 Series 对象元素的查看.....	3
1.2.2 Series 对象的修改.....	5
1.2.3 Series 的数学运算.....	7
2 DataFrame.....	12
2.1 DataFrame 对象的构造.....	12
2.1.1 通过一个字典构建 DataFrame 对象.....	13
2.1.2 通过字典列表.....	13
2.1.3 通过 Series 列表.....	14
2.2 DataFrame 的基本操作.....	14
2.2.1 DataFrame 的增删改查.....	14
① DataFrame 的查找.....	14
② 删除某一列属性.....	15
③ 插入某一列属性.....	17
④ 插入或修改一个新元组.....	18
2.2.2 DataFrame 的算术运算.....	20
3 Pandas 进阶操作.....	21
3.1 文件读写操作.....	21
3.1.1 csv 模文件处理.....	21
3.1.2 Excel 文件处理.....	23
3.2 处理缺失数据.....	25
3.2.1 isna()方法.....	25
3.2.2 notna().....	25
3.2.3 notnull()方法.....	26
3.2.4 dropna()方法.....	28
3.3 Pandas 高阶操作.....	29
3.3.1 连接、合并.....	29
① concat()方法.....	29
② merge()合并函数.....	30
③ join().....	32
3.3.2 apply()函数.....	32
3.3.3 Groupby 分组.....	33
二 Sklearn 学习.....	34

一 Pandas

Pandas 是为 Python 编程语言编写的一个用于数据操作和分析的开源软件库。在 Pandas 库出现之前，Python 主要用于数据迁移和准备，因为 Pandas 高效，灵活，便捷的特点而，使得 Python 在数据分析与处理上变得得心应手，从而广泛应用于金融，统计，数据分析等学术和商业领域，可以说 Pandas 是当下 Python 大热的一个重要推手。

Pandas 是基于 Python Numpy 库开发，并构建了一些更为高级的数据结构 (Series, DataFrame, Panel)。因为在 Pandas 0.20.0 以后的版本中，Panel 遭到官方的摒弃，淘汰于历史尘埃中，故在本章 Pandas 学习中主要介绍 Series 和 DataFrame。环境如下：操作系统，Ubuntu 16.04，Python 版本为 3.6.5，IDE 为 Pycharm，基于 IPython 有着语法高亮，自动补全等特点，Python Shell 使用了 IPython。

1 Series

Series，官方的定义如下：

Series is a one-dimensional labeled array capable of holding any data type (integers, strings, floating point numbers, Python objects, etc.). The axis labels are collectively referred to as the **index**.

简单的说，Series 类似于字典数据类型，他是一个一维的标记数组，每个元素可以储存任意的数据类型，包括整型，浮点型，字符型甚至一个 Python 对象。Series 的标签称为索引，其与 Series 中的数据严格对照。索引的类型任意，可以为类似与列表的数字下标，也可以是一个字符串等等，当索引对应的数据为空时，数据为 NAN。

1.1 Series 的构造

使用 Pandas.Series() 构造一个 Series 对象。查看 Series 构造函数，有如下参数：

```
def __init__(self, data=None, index=None, dtype=None, name=None,
              copy=False, fastpath=False):
```

data 即为传递过来的数据列表，其数据可为任意数据类型，index 对应的是 Series 对象的索引列表，长度与数据列表，相同当 index 为空时 (none)，默认为 data 列表的索引 (0,1,2.....)，dtype 用于指定数据的数据类型，若 dtype 为 none 则根据实际情况推断真实类型，若类型不统一有冲突则为 object 类型。name 参数允许我们给一个 Series 对象起名字，默认为 none，copy 实参为布尔类型，其用于选择是否拷贝 data 数据。fastpath 也为布尔类型，其通常与 copy 结合使用。以下为具体代码：

1.1.1 创建一个空的 Series 对象：

```
In [1]: from pandas import *
In [2]: x = Series()
In [3]: x
Out[3]: Series([], dtype: float64)
In [4]: type(x)
Out[4]: pandas.core.series.Series
```

当 Series 对象为空数据为空时，默认数据类型为 float64.

1.1.2 通过一个 ndarray 对象传递到 data 参数

```

In [1]: import pandas as pd
In [2]: import numpy as np
In [3]: list = np.array(['a','b','c'])
In [4]: s = pd.Series(list)
In [5]: type(s)
Out[5]: pandas.core.series.Series
In [6]: s
Out[6]:
0    a
1    b
2    c
dtype: object

```

发现，当数据列表元素的数据类型为字符串型时，Series 对应的数据类型为 object。除此之外，Series 支持的数据类型还有整数浮点数，布尔类型，datetime 类型等等.....
代码：

```

In [1]: from datetime import *
In [2]: from pandas import *
In [3]: today = datetime.today()
In [4]: yesterday = today - timedelta(days=1)
In [5]: x = Series([today,yesterday])
In [6]: x
Out[6]:
0    2018-05-24 22:05:45.719978
1    2018-05-23 22:05:45.719978
dtype: datetime64[ns]

```

1.1.3 从字典创建一个 Series 对象

Series 与 dict 有着很大的相似，都有着索引(键)，属性值(键值)。所以 Series 支持直接将一个字典对象作为 data 参数的一个实参传递。若构造中没有指定索引，者按照字典顺序取得对应的字典键及其键值分别传递到索引与数据中。若构造时指定了索引，索引中与标签对应的数据的值得到保留，空缺的键值以 NaN(NaN 并非一个数字，其在 pandas 中是数据缺失的一个特殊标志) 填充，最后实际的顺序由索引列表决定。

代码：

未指定索引时：

```

In [1]: import pandas as pd

In [2]: import numpy as np

In [3]: dic = {'a' : 0, 'b' : 1, 'c' : 2}

In [4]: s = pd.Series(dic)

In [5]: s
Out[5]:
a    0
b    1
c    2
dtype: int64

```

指定索引时:

```

In [12]: s = pd.Series(dic,index=['b','c','d','a'])

In [13]: s
Out[13]:
b    1.0
c    2.0
d    NaN
a    0.0
dtype: float64

```

1.1.4 通过标量值创建对象

当 **data** 参数为一个数（也可为其他类型），可通过索引长度（此情况下索引的指定为必须）动态调整 **Series** 长度，此时索引对应的数据为同一的。

代码:

```

In [2]: import pandas as pd

In [3]: import numpy as np

In [4]: s = pd.Series(np.random.randint(3),index=[x for x in range(5)])

In [5]: s
Out[5]:
0    1
1    1
2    1
3    1
4    1
dtype: int64

```

1.2 Series 基础操作

1.2.1 Series 对象元素的查看

① 通过 **head()**和 **tail()**查看 **Series** 对象的前 **n** 个或末尾 **n** 个元素。

代码:

```

In [3]: s = pd.Series("test",list(range(9)))

In [4]: s.head(5)
Out[4]:
0    test
1    test
2    test
3    test
4    test
dtype: object

In [5]: s.tail(5)
Out[5]:
4    test
5    test
6    test
7    test
8    test
dtype: object

```

② 通过指定索引查看

显式得类似字典一般指定索引值，得到数据元素。

代码:

```

In [28]: file
Out[28]:
   ch    big
0  a  1.5986
1  b  0.7536
2  c  0.2211
3  d  0.3212
4  e  3.3606
5  f  2.1616

In [29]: test = pd.Series(list(file["big"]),index=file["ch"])

In [30]: test
Out[30]:
ch
a    1.5986
b    0.7536
c    0.2211
d    0.3212
e    3.3606
f    2.1616
dtype: float64

In [31]: test['c']
Out[31]: 0.2211

```

Series 与 npdarray 类似，支持切片操作及筛选数据等操作。

```

In [50]: test
Out[50]:
ch
a    1.5986
b    0.7536
c    0.2211
d    0.3212
e    3.3606
f    2.1616
dtype: float64

In [51]: test[test>2]#筛选出大于2的数据
Out[51]:
ch
e    3.3606
f    2.1616
dtype: float64

In [52]: test2 = test[-3:]

In [53]: test2
Out[53]:
ch
d    0.3212
e    3.3606
f    2.1616
dtype: float64

```

1.2.2 Series 对象的修改

① 重新修改索引

Series 对象的 `index` 属性可以通过传递 `Series.index` 参数进行修改, 索引的修改不会影响到原始数据。此外还可以使用 `reindex()` 方法指定新索引创建一个新的 Series 对象。这个方法会对新的 Series 对象的内容进行重写, 旧索引的保留, 新的以 `NaN` 补充。


```

In [82]: test
Out[82]:
1    1.5986
2    0.7536
3    0.2211
4    0.3212
5    3.3606
6    2.1616
dtype: float64

In [83]: test5 = test.reindex([random.choice(1) for x in range(6)])

In [84]: test5
Out[84]:
n    NaN
x    NaN
e    NaN
c    NaN
g    NaN
A    NaN
dtype: float64

In [85]: test.index=[random.choice(1) for x in range(6)]

In [86]: test
Out[86]:
B    1.5986
e    0.7536
y    0.2211
S    0.3212
d    3.3606
E    2.1616
dtype: float64

```

② 设置 name 属性

可通过 Series 直接构造时传递 name 参数，也可以通过 Series.name()来修改。

代码：

```

In [1]: import pandas as pd

In [2]: s = pd.Series(['a','b','c'],name='letters')

In [3]: s
Out[3]:
0    a
1    b
2    c
Name: letters, dtype: object

In [4]: s.name = "alphabet"

In [5]: s
Out[5]:
0    a
1    b
2    c
Name: alphabet, dtype: object

```

③ 替换 Series 的数据

Series 对象常通过 replace()函数进行数据替换，与 loc()方法通过索引进行操作不一样，replace()指定指定 value 进行修改。

```

In [12]: s = pd.Series([0, 1, 2, 3, 4])

In [13]: s.replace(0, 5)
Out[13]:
0    5
1    1
2    2
3    3
4    4
dtype: int64

```

若数据中有相同的 **value**，会一并替换。

```

In [14]: s1 = pd.Series(1, index=list(range(4)))

In [15]: s1
Out[15]:
0    1
1    1
2    1
3    1
dtype: int64

In [16]: s1.replace(1, 2)
Out[16]:
0    2
1    2
2    2
3    2
dtype: int64

```

1.2.3 Series 的数学运算

首先 **Series** 对象与列表类似，其内置的函数支持一些基础的数学操作如求最极值，平均值，中间值，大小排序等等。

代码：


```

In [57]: test
Out[57]:
ch
a    1.5986
b    0.7536
c    0.2211
d    0.3212
e    3.3606
f    2.1616
dtype: float64

In [58]: test.max()#max(), min () 求的最大, 最小数据
Out[58]: 3.3606

In [59]: test.mean()#求得平均数
Out[59]: 1.4027833333333333

In [60]: test.median()#计算中间数
Out[60]: 1.1761

In [61]: test.sort_values()##Series的sort函数第索引即数据大小进行排序
Out[61]:
ch
c    0.2211
d    0.3212
b    0.7536
a    1.5986
f    2.1616
e    3.3606
dtype: float64

```

此外，Series 对象之间还支持一些算术运算，包括加减乘除等。

```
In [24]: test
Out[24]:
0    1.5986
1    0.7536
2    0.2211
3    0.3212
4    3.3606
5    2.1616
dtype: float64

In [25]: test2 = test * 2

In [26]: test2
Out[26]:
0    3.1972
1    1.5072
2    0.4422
3    0.6424
4    6.7212
5    4.3232
dtype: float64

In [27]: test + test2
Out[27]:
0    4.7958
1    2.2608
2    0.6633
3    0.9636
4    10.0818
5    6.4848
dtype: float64

In [28]: test / test2
Out[28]:
0    0.5
1    0.5
2    0.5
3    0.5
4    0.5
5    0.5
dtype: float64
```

也可以使用 `add()` 函数对每个是实数型数据同时做加法。

```
In [19]: s = pd.Series(list(range(5)))
```

```
In [20]: s
```

```
Out[20]:
```

```
0    0
```

```
1    1
```

```
2    2
```

```
3    3
```

```
4    4
```

```
dtype: int64
```

```
In [21]: s.add(0.5)
```

```
Out[21]:
```

```
0    0.5
```

```
1    1.5
```

```
2    2.5
```

```
3    3.5
```

```
4    4.5
```

```
dtype: float64
```

对于 `dtypeSeries` 对象索引不一致的数据做算术运算，会产生 `NaN` 做填充。（注意当 `Series` 对象之间包含有字符类型时，加法运算为对应的字符做连接操作，减法除法则不支持，乘法则只支持 `Series` 对象的数乘）。

```
In [54]: test3
```

```
Out[54]:
```

```
0    1
```

```
1    b
```

```
2    c
```

```
3    d
```

```
4    e
```

```
5    0
```

```
dtype: object
```

```
In [55]: test4
```

```
Out[55]:
```

```
0    2
```

```
1   bb
```

```
2   cc
```

```
3   dd
```

```
4   ee
```

```
5    0
```

```
dtype: object
```

```
In [56]: test3 + test4
```

```
Out[56]:
```

```
0    3
```

```
1   bbb
```

```
2   ccc
```

```
3   ddd
```

```
4   eee
```

```
5    0
```

```
dtype: object
```

```
In [57]: test4 - test3
```

```
-----  
TypeError                                Traceback (most recent call last)
```

```

In [29]: test2.index = [x + 1 for x in range(len(test2))]

In [30]: test2
Out[30]:
1    3.1972
2    1.5072
3    0.4422
4    0.6424
5    6.7212
6    4.3232
dtype: float64

In [31]: test
Out[31]:
0    1.5986
1    0.7536
2    0.2211
3    0.3212
4    3.3606
5    2.1616
dtype: float64

In [32]: test2 + test
Out[32]:
0    NaN
1    3.9508
2    1.7283
3    0.7634
4    4.0030
5    8.8828
6    NaN
dtype: float64

```

此外 `describe()` 函数会对 `Series` 对象进行汇总统计。

对于实数型数据(`float64`)，结果的索引将包括计数数目，平均值，标准偏差，最小值，最大值以及更低的 50 和更高的百分位数。默认情况下，较低的百分位是 25，而较高的百分位是 75.50 百分位与中位数相同。

```

In [98]: test
Out[98]:
B    1.5986
e    0.7536
y    0.2211
S    0.3212
d    3.3606
E    2.1616
dtype: float64

In [99]: test.describe()
Out[99]:
count    6.000000
mean     1.402783
std      1.219412
min      0.221100
25%      0.429300
50%      1.176100
75%      2.020850
max      3.360600
dtype: float64

```

对于对象数据 **object**（例如字符串或时间戳），结果的索引将包括 **count**, **unique**, **top** 和 **freq**. **top** 是最常见的价值。频率 **freq** 是最常见的频率。时间戳还包括第一个和最后一个项目。

```
In [100]: test4
Out[100]:
0      2
1     bb
2     cc
3     dd
4     ee
5      0
dtype: object

In [101]: test4.describe()
Out[101]:
count      6
unique      6
top        bb
freq        1
dtype: object
```

2 DataFrame

DataFrame 的官方定义如下：

DataFrame is a 2-dimensional labeled data structure with columns of potentially different types. You can think of it like a spreadsheet or SQL table, or a dict of Series objects. It is generally the most commonly used pandas object. Like Series, DataFrame accepts many different kinds of input.

相较于 **Series**, **DataFrame** 对象是一个一个表格型的二维数据结构，类似数据库中的表和一个 excel 表格。和 **Series** 相同，每个 **DataFrame** 对象都有一组有序的索引列（**Index**），每一行可以是不同的值类型(就像 **Series** 允许存储不同类型的元素一样)，基本上可以 **DataFrame** 看成共享同一个索引 **Index** 的 **Series** 集合。

2.1 DataFrame 对象的构造

DataFrame 的构造方法与 **Series** 类似，只不过可以同时接受多条一维数据源，每一条都会成为单独的一列。**DataFrame** 的构造函数如下：

```
def __init__(self, data=None, index=None, columns=None, dtype=None,
              copy=False):
```

data 实参类似一个字典，作为原始数据的传递参数，字典的键对应的不是 **index** 的名字，而是每一列的 **name** 属性。**Index** 是一个列表，其组成元素为每一行 **DataFrame** 元素的行标签，与此对应，**columns** 对应的是每一列的列标签，即 **name** 属性，当传递的 **data** 为字典且 **columns** 为空时，则将 **data** 的键列表传递给 **columns**。若 **columns** 不为空，则 **columns** 重载(override) **data** 字典的键列表。直接传递 **dtype** 参数的实例很少见，其究竟指代的每一行亦或是每一列的数据类型有待考却，但是任意一个 **DataFrame** 对象可直接访问其 **DataFrame.dtypes** 属性，返回一个 **Series** 对象，其中元素对应了每一二个列属性(column)的数据类型。**copy** 参数与 **Series** 构造中的类似，当 **copy** 为真时，会在在构造函数创建了一个 **DataFrame** 对象的同时会调用该对象的数据(**data**)及其索引副本创建一个副本，副本的改动不影响原始对象。

DataFrame 创建很灵活，其中传递的 `data` 可以是一个 `ndarrays` 或列表的字典，也可以是一个结构化的数组，甚至是一个字典列表。

2.1.1 通过一个字典构建 DataFrame 对象

代码：

```
In [1]: import pandas as pd
In [2]: import random
In [3]: import string
In [4]: s = string.ascii_letters
In [5]: dict1 = {'a':list(range(5)), 'b':[random.choice(s) for i in range(5)]}
In [6]: dict1
Out[6]: {'a': [0, 1, 2, 3, 4], 'b': ['I', 'J', 'G', 'L', 'n']}
In [7]: D = pd.DataFrame(dict1)
In [8]: D
Out[8]:
```

	a	b
0	0	I
1	1	J
2	2	G
3	3	L
4	4	n

显然，在构造中，函数自动将字典的键的列表出阿迪给 `columns` 作为列属性，当 `columns` 不为 `None` 时，又会对字典传递的键的列表进行截取，即已有列属性的内容予以保留，不存在的则以 `NaN` 覆盖，效果如下：

代码：

```
In [14]: H = pd.DataFrame(dict1, columns=['OK', 'a'])
In [15]: H
Out[15]:
```

	OK	a
0	NaN	0
1	NaN	1
2	NaN	2
3	NaN	3
4	NaN	4

2.1.2 通过字典列表

代码：


```

In [1]: import pandas as pd
In [2]: import numpy as np
In [3]: data2 = [{'a': 1, 'b': 2}, {'a': 5, 'b': 10, 'c': 20}]
In [4]: pd.DataFrame(data2)
Out[4]:
   a  b    c
0  1  2  NaN
1  5 10 20.0
In [5]: pd.DataFrame(data2, index=['one', 'two'])
Out[5]:
   a  b    c
one 1  2  NaN
two 5 10 20.0
In [6]: pd.DataFrame(data2, columns=['a', 'c', 'd'])
Out[6]:
   a    c  d
0  1  NaN NaN
1  5 20.0 NaN

```

2.1.3 通过 Series 列表

代码:

```

In [12]: x = pd.Series(['a', 'b', 'c', 'd'], index=[x + 1 for x in range(4)])
In [13]: y = pd.Series([1, 2, 3, 4])
In [14]: mans = pd.DataFrame([x, y], index=['m', 'n'])
In [15]: mans
Out[15]:
   0  1  2  3  4
m NaN a  b  c  d
n 1.0 2  3  4 NaN

```

注意: 当两个 Series 对象的索引不完全相同时, 在最终的 DataFrame 对象中取两者的并集, 空缺部分补 NaN。

2.2 DataFrame 的基本操作

2.2.1 DataFrame 的增删改查

① DataFrame 的查找

通过指定 columns 名称找出某一列或某几列

```

In [189]: mans["ID"]
Out[189]:
0    2016212070
1    2016212071
2    2016212072
3    2016212073
4    2016212074
Name: ID, dtype: int64

In [190]: mans[mans.force>90]
Out[190]:
   ID wujiang intelligence force captain
2  2016212072  马超             69      99      95
3  2016212073  赵云             69     100      84

```


若想找出某一行元组，也可使用 `query()` 函数，其接受一个字符串，字符串的内容即为判别式，例如想找出智力(`intelligence>60`)的武将

```
In [95]: mans
Out[95]:
```

	ID	wujiang	intelligence	force
0	2016212070	关羽	68	88
1	2016212071	张飞	51	81
2	2016212072	马超	69	99
3	2016212073	赵云	69	100
4	2016212074	黄忠	76	87

```
In [96]: mans.query('intelligence>60')
Out[96]:
```

	ID	wujiang	intelligence	force
0	2016212070	关羽	68	88
2	2016212072	马超	69	99
3	2016212073	赵云	69	100
4	2016212074	黄忠	76	87

另外还可以使用 `iloc` 和 `loc` 属性指定索引来查找具体某一编号的元组

```
In [118]: mans.iloc[1]
Out[118]:
```

ID	2016212071
wujiang	张飞
intelligence	51
force	81
captain	95

Name: 1, dtype: object

`loc` 第一个逗号前指定索引，后一个列表指定所要查询的 `columns`，只会切片操作。

```
In [169]: mans.loc[:2016212073,["ID","wujiang","force"]]
Out[169]:
```

	ID	wujiang	force
0	2016212070	关羽	88
1	2016212071	张飞	81
2	2016212072	马超	99
3	2016212073	赵云	100
4	2016212074	黄忠	87

② 删除某一列属性

与列表的删除类似，`DataFrame` 对象同样可以使用 `del` 和 `pop` 函数进行删除，只不过列表删除的对象为列表的一个基本元素，而 `DataFrame` 删除对象为某一列 `column`

```

In [62]: del mans["ID"]

In [63]: mans
Out[63]:
   Name
0  关羽
1  张飞
2  马超
3  赵云
4  黄忠
5  于禁
6  徐晃
7  张辽
8  乐进
9  张郃

In [64]: mans.pop("Name")
Out[64]:
0  关羽
1  张飞
2  马超
3  赵云
4  黄忠
5  于禁
6  徐晃
7  张辽
8  乐进
9  张郃
Name: Name, dtype: object

In [65]: mans
Out[65]:
Empty DataFrame
Columns: []
Index: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

```

使用 `drop()` 函数可以便捷删除行或列

```

In [145]: mans
Out[145]:
   ID wujiang intelligence force captain
0  2016212070    关羽         68      88      94
1  2016212071    张飞         51      81      95
2  2016212072    马超         69      99      95
3  2016212073    赵云         69     100      84
4  2016212074    黄忠         76      87      88
5  2016212075   太史慈         65      91      84

In [146]: mans.drop(columns="captain")
Out[146]:
   ID wujiang intelligence force
0  2016212070    关羽         68      88
1  2016212071    张飞         51      81
2  2016212072    马超         69      99
3  2016212073    赵云         69     100
4  2016212074    黄忠         76      87
5  2016212075   太史慈         65      91

```

③ 插入某一列属性

DataFrame 对象可像字典一样插入新的键值对操作一般增加新的列，只不过插入的 column 为一个列表。

```

In [104]: captain = [ random.randint(80,98) for i in range(5)]

In [105]: mans["captain"] = captain

In [106]: mans
Out[106]:
   ID wujiang intelligence force captain
0  2016212070    关羽         68      88      94
1  2016212071    张飞         51      81      95
2  2016212072    马超         69      99      95
3  2016212073    赵云         69     100      84
4  2016212074    黄忠         76      87      88

```

另外，DataFrame 对象也使用 assign() 函数进行 column 的增加。assign() 函数内传递的是一个**型，可接受任意数量的关键字 ***kwargs*。**kwargs* 分解为 *keyword* 和 *value pairs* 键。*keyword* 为新增属性列的名字，*value pairs* 为对应的属性列表。assign() 可接受任意数目的 *keyword* 和 *value pairs* 对。

```

In [81]: intelligence = [ random.randint(20,80) for i in range(10)]
In [82]: force = [random.randint(80,100) for i in range(10)]
In [83]: mans = pd.read_excel("/home/ubuntu/wujiang.xlsx",header=0)
In [84]: mans.assign(intelligence=intelligence,force=force)
Out[84]:

```

	ID	wujiang	intelligence	force
0	2016212070	关羽	68	88
1	2016212071	张飞	51	81
2	2016212072	马超	69	99
3	2016212073	赵云	69	100
4	2016212074	黄忠	76	87
5	2016212075	于禁	77	86
6	2016212076	徐晃	57	88
7	2016212077	张辽	72	100
8	2016212078	乐进	37	83
9	2016212079	张郃	52	87

assign()内支持匿名函数 lambda:

```

In [90]: mans
Out[90]:

```

	ID	wujiang	intelligence	force
0	2016212070	关羽	68	88
1	2016212071	张飞	51	81
2	2016212072	马超	69	99
3	2016212073	赵云	69	100
4	2016212074	黄忠	76	87

```

In [91]: mans.assign(total = lambda x : x["intelligence"] + x["force"])
Out[91]:

```

	ID	wujiang	intelligence	force	total
0	2016212070	关羽	68	88	156
1	2016212071	张飞	51	81	132
2	2016212072	马超	69	99	168
3	2016212073	赵云	69	100	169
4	2016212074	黄忠	76	87	163

注意: assign()函数的操作结果返回的是我一个新对象, 即旧对象的一个 copy, 实际要保存的话最后另做切片赋值或使用 head()赋值等等。

④ 插入或修改一个新元组

欲插入或修改某一行, 可指定对象的索引 loc[]来进行操作。


```

In [122]: mans
Out[122]:
   ID wujiang  intelligence  force  captain
0  2016212070    关羽         68     88     94
1  2016212071    张飞         51     81     95
2  2016212072    马超         69     99     95
3  2016212073    赵云         69    100     84
4  2016212074    黄忠         76     87     88

In [123]: mans.loc[5] = [2016212075,"太史慈",65,91,84]

In [124]: mans
Out[124]:
   ID wujiang  intelligence  force  captain
0  2016212070    关羽         68     88     94
1  2016212071    张飞         51     81     95
2  2016212072    马超         69     99     95
3  2016212073    赵云         69    100     84
4  2016212074    黄忠         76     87     88
5  2016212075    太史慈         65     91     84

```

此外，也可通过 `append()` 函数添加一个字典参数。

```

In [165]: mans
Out[165]:
   ID wujiang  intelligence  force  captain
0  2016212070    关羽         68     88     94
1  2016212071    张飞         51     81     95
2  2016212072    马超         69     99     95
3  2016212073    赵云         69    100     84
4  2016212074    黄忠         76     87     88

In [166]: x
Out[166]:
{'ID': 2016212075,
 'wujiang': '太史慈',
 'intelligence': 46,
 'force': 92,
 'captain': 70}

In [167]: mans.append(x,ignore_index=True)
Out[167]:
   ID wujiang  intelligence  force  captain
0  2016212070    关羽         68     88     94
1  2016212071    张飞         51     81     95
2  2016212072    马超         69     99     95
3  2016212073    赵云         69    100     84
4  2016212074    黄忠         76     87     88
5  2016212075    太史慈         46     92     70

```

删除某一行元组则使用到 `drop()` 函数。

```
In [146]: mans.drop(columns="captain")
Out[146]:
```

	ID	wujiang	intelligence	force
0	2016212070	关羽	68	88
1	2016212071	张飞	51	81
2	2016212072	马超	69	99
3	2016212073	赵云	69	100
4	2016212074	黄忠	76	87
5	2016212075	太史慈	65	91

```
In [147]: mans.drop([5])
Out[147]:
```

	ID	wujiang	intelligence	force	captain
0	2016212070	关羽	68	88	94
1	2016212071	张飞	51	81	95
2	2016212072	马超	69	99	95
3	2016212073	赵云	69	100	84
4	2016212074	黄忠	76	87	88

2.2.2 DataFrame 的算术运算

与 Series 对象类似，DataFrame 内对象根据其类型支持不同的算术操作。字符型支持数乘，加法，实数型支持加减乘除运算。

```
In [178]: mans + mans
Out[178]:
```

	ID	wujiang	intelligence	force	captain	
0	4032424140	关羽	关羽	136	176	188
1	4032424142	张飞	张飞	102	162	190
2	4032424144	马超	马超	138	198	190
3	4032424146	赵云	赵云	138	200	168
4	4032424148	黄忠	黄忠	152	174	176

```
In [179]: mans2
Out[179]:
```

	ID	intelligence	force	captain
0	2016212070	68	88	94
1	2016212071	51	81	95
2	2016212072	69	99	95
3	2016212073	69	100	84
4	2016212074	76	87	88

```
In [180]: mans2.add(10) - mans2
Out[180]:
```

	ID	intelligence	force	captain
0	10	10	10	10
1	10	10	10	10
2	10	10	10	10
3	10	10	10	10
4	10	10	10	10

```
In [181]: mans2.add(10) * mans2
Out[181]:
```

	ID	intelligence	force	captain
0	4065111131375805600	5304	8624	9776
1	4065111135408229751	3111	7371	9975
2	4065111139440653904	5451	10791	9975
3	4065111143473078059	5451	11000	7896
4	4065111147505502216	6536	8439	8624

3 Pandas 进阶操作

3.1 文件读写操作

3.1.1 csv 模文件处理

Csv(Comma-Separated Values)文件，也称逗号分隔值文件，是一个通常作为表格存储的纯文本文件。其在商业以及科学领域广泛应用。Csv 文件的特点就是数据之间的分割符是逗号。若要将分割符为其他字符的文件转为 csv 文件，只需要一行简单的 sed 命令。

```
`sed -i 's/原有字符/目标字符/' 目标文件`
```

① 读取 csv 文件

Pandas 库在处理 csv 文件上得心应手。pandas.read_csv 其将一个 csv 文件读取为一个 DataFrame 对象。pandas.read_csv()函数有以下几个重要参数：

- filepath_or_buffer : str，这个参数对应目标文件的 path 路径，在引用时最好使用直接路径。
- sep: str，指定分隔符，默认为逗号
- header: int or list of ints。用作列名 columns 的行号，当 csv 文件无指定 columns 时，传递 header=0。

- `index_col` : int or sequence or False, 用作行索引的列编号或者列名, 如果给定一个序列则有多行索引。如果文件不规则, 行尾有分隔符, 则可以设定 `index_col=False` 来是的 pandas 不适用第一列作为行索引。
- `narrow`: int, default None, 需要读取的行数。
例如读取一个这样的文件



	ID	wujiang	force
1	2016212070	关羽	98
2	2016212071	张飞	99
3	2016212072	马超	97
4	2016212073	赵云	96
5	2016212074	黄忠	93
6	2016212075	于禁	80
7	2016212076	徐晃	89
8	2016212077	张辽	91
9	2016212078	乐进	88
10	2016212079	张郃	90

转换为 csv 文件

```
ubuntu@Lenovo:~$ sed -i 's/ \+/,/g' wujiang.csv
ubuntu@Lenovo:~$ cat wujiang.csv
ID,wujiang,force
2016212070,关羽,98
2016212071,张飞,99
2016212072,马超,97
2016212073,赵云,96
2016212074,黄忠,93
2016212075,于禁,80
2016212076,徐晃,89
2016212077,张辽,91
2016212078,乐进,88
2016212079,张郃,90ubuntu@Lenovo:~$ |
```

将使用 `read_csv` 模块读取文件

```
In [11]: file = pd.read_csv("/home/ubuntu/wujiang.csv")
```

```
In [12]: file
```

```
Out[12]:
```

	ID	wujiang	force
0	2016212070	关羽	98
1	2016212071	张飞	99
2	2016212072	马超	97
3	2016212073	赵云	96
4	2016212074	黄忠	93
5	2016212075	于禁	80
6	2016212076	徐晃	89
7	2016212077	张辽	91
8	2016212078	乐进	88
9	2016212079	张郃	90

```
In [13]: type(file)
```

```
Out[13]: pandas.core.frame.DataFrame
```

```
In [14]: |
```

② 写入 csv 文件

`to_csv()`函数负责将数据写出到文本格式。其参数有：`path_or_buf`, `sep`(设置 `sep`(分隔符))、`index` 和 `header`(禁止写出行和列名)、`cols`(需要写出的列,以及列的顺序), `encoding`(编码方式)等等。

```
In [15]: file.to_csv(r"~/mans.csv",header=True,encoding="utf-8")
```

```
ubuntu@Lenovo:~$ cat mans.csv
,ID,wujiang,force
0,2016212070,关羽,98
1,2016212071,张飞,99
2,2016212072,马超,97
3,2016212073,赵云,96
4,2016212074,黄忠,93
5,2016212075,于禁,80
6,2016212076,徐晃,89
7,2016212077,张辽,91
8,2016212078,乐进,88
9,2016212079,张郃,90
```

3.1.2 Excel 文件处理

Excel 文件时 Windows 上最流行的数据处理软件,其 `xlsx` 文档广泛应用于各个领域。

① 读取 excel 文件

`Pandas.read_excel()`模块是用于读取 excel 文件的一种更重要手段,其中主要参数如下:

- `io`: string, 文件的 path 路径
- `sheet_name`: string, int, mixed list of strings/ints, or None, default 0。使用的第几个 sheet 表, 返回多表使用 `sheetname=[0,1]`, 若 `sheetname=None` 是返回全表 注意: int/string 返回的是 `DataFrame`, 而 `none` 和 `list` 返回的是 `dict of DataFrame`。
- `header`: int, list of ints, default 0。指定列名行所在, `header=None` 表示, 数据不含列名。

- `index_col`: int, list of ints, default None。行索引的列编号或者列名。
- `names`: array-like, default None。指定每列的名称。

例如读取一个这样的 excel 文件

	A	B	C	D
1	ID	wujiang	force	
2	2016212070	关羽	98	
3	2016212071	张飞	99	
4	2016212072	马超	97	
5	2016212073	赵云	96	
6	2016212074	黄忠	93	
7	2016212075	于禁	80	
8	2016212076	徐晃	89	
9	2016212077	张辽	91	
10	2016212078	乐进	88	
11	2016212079	张郃	90	
12				

```
In [7]: file1 = pd.read_excel("/home/ubuntu/wujiang.xlsx")
```

```
In [8]: file1
```

```
Out[8]:
```

	ID	wujiang	force
0	2016212070	关羽	98
1	2016212071	张飞	99
2	2016212072	马超	97
3	2016212073	赵云	96
4	2016212074	黄忠	93
5	2016212075	于禁	80
6	2016212076	徐晃	89
7	2016212077	张辽	91
8	2016212078	乐进	88
9	2016212079	张郃	90

```
In [9]: type(file1)
```

```
Out[9]: pandas.core.frame.DataFrame
```

显然读取的结果为一个 DataFrame 对象

② 写出 excel 文件

存储函数为 `pd.DataFrame.to_excel()`，只有 DataFrame 写入 excel,其具体参数如下：

- `excel_writer`: string or ExcelWriter object File path or existing ExcelWriter 文件的直接路径
- `sheet_name`: string, default 'Sheet1'。使用的 sheet 表数。
- `columns`: sequence, 选择输出的列，默认为所有。
- `header`: boolean or list of string, default True。columns 的名字，若无则天 None。
- `encoding`: string, default None。编码方式。
- `index`: boolean, default True，行索引的名称，False 表示不显示行索引。

代码：

```
In [27]: file1.to_excel("/home/ubuntu/mans.xlsx",columns=["wujiang","force"],ind
...: ex=False)
```

1	wujiang	force		
2	关羽	98		
3	张飞	99		
4	马超	97		
5	赵云	96		
6	黄忠	93		
7	于禁	80		
8	徐晃	89		
9	张辽	91		
10	乐进	88		
11	张郃	90		

3.2 处理缺失数据

pandas 中 NA 的主要表现为 np.nan，表示值为空 NaN 或缺失 missing。另外 Python 内建的 None 也会被当做 NA 处理。处理 NA 的方法有四种：dropna , fillna , isnull , notnull 。

3.2.1 isna()方法

isna()方法用于检测缺失值，他通过检测 DataFrame 中每个元素的 bool 值的掩码，指示元素是否不是 NaN，所以其返回结果为一个所有数值为布尔值的 DataFrame 类型对象。

```
In [48]: mans
Out[48]:
```

	ID	wujiang	force
0	2016212070	关羽	98.0
1	2016212071	张飞	99.0
2	2016212072	NaN	97.0
3	2016212073	NaN	NaN
4	2016212074	NaN	NaN
5	2016212075	于禁	NaN
6	2016212076	徐晃	89.0
7	2016212077	张辽	91.0
8	2016212078	乐进	88.0
9	2016212079	张郃	90.0

```
In [49]: t = mans.isna()

In [50]: t
Out[50]:
```

	ID	wujiang	force
0	False	False	False
1	False	False	False
2	False	True	False
3	False	True	True
4	False	True	True
5	False	False	True
6	False	False	False
7	False	False	False
8	False	False	False
9	False	False	False

```
In [51]: type(t)
Out[51]: pandas.core.frame.DataFrame
```

True 表示对应数值为 NaN。

3.2.2 notna()

与 isna()相反，notna()用于检测已存在的真实值(不为 NaN)。


```

In [55]: mans.isna()
Out[55]:
   ID  wujiang  force
0  False    False  False
1  False    False  False
2  False     True  False
3  False     True   True
4  False     True   True
5  False    False   True
6  False    False  False
7  False    False  False
8  False    False  False
9  False    False  False

In [56]: mans.notna()
Out[56]:
   ID  wujiang  force
0  True     True   True
1  True     True   True
2  True    False   True
3  True    False  False
4  True    False  False
5  True     True  False
6  True     True   True
7  True     True   True
8  True     True   True
9  True     True   True

```

3.2.3 notnull()方法

`notnull()`方法与`notna()`有所异同，相同之处在于都是用于检测和处理缺失值，并且返回一个同样规格，数据类型全为布尔型的 **DataFrame** 对象。但是 `notnull()`对缺失值的判定更加宽广，不仅可以选择性地检测 **NaN**(通过设置 `pandas.options.mode.use_inf_as_na = True`)，也可以检测空字符和空的 **numpy** 类型。

```

In [77]: mans
Out[77]:
   ID wujiang  force
0  2016212070  关羽    98.0
1  2016212071  张飞    99.0
2  2016212072   NaN    97.0
3  2016212073   NaN    NaN
4  2016212074   NaN    NaN
5  2016212075  于禁    NaN
6  2016212076  徐晃    89.0
7  2016212077  张辽    91.0
8  2016212078  乐进    88.0
9  2016212079  张郃    90.0

In [78]: mans.notna()
Out[78]:
   ID wujiang  force
0  True     True   True
1  True     True   True
2  True    False   True
3  True     True  False
4  True    False  False
5  True     True  False
6  True     True   True
7  True     True   True
8  True     True   True
9  True     True   True

In [79]: mans.notnull()
Out[79]:
   ID wujiang  force
0  True     True   True
1  True     True   True
2  True    False   True
3  True     True  False
4  True    False  False
5  True     True  False
6  True     True   True
7  True     True   True
8  True     True   True
9  True     True   True

In [80]: pd.options.mode.use_inf_as_na = False

In [81]: mans.notnull()
Out[81]:
   ID wujiang  force
0  True     True   True
1  True     True   True
2  True    False   True
3  True     True  False
4  True    False  False
5  True     True  False
6  True     True   True
7  True     True   True
8  True     True   True
9  True     True   True

```

3.2.4 dropna()方法

① 对于 Series 对象，dropna 返回仅包含非空数据和索引值的 Series 对象。

```
In [17]: x
Out[17]:
0      NaN
1    0.7536
2    0.2211
3    0.3212
4      NaN
5    2.1616
Name: value, dtype: float64

In [18]: x.dropna()
Out[18]:
1    0.7536
2    0.2211
3    0.3212
5    2.1616
Name: value, dtype: float64
```

② 对于 DataFrame 对象，一个 NaN 的出现意味着至少丢掉一行或者一列。为此我们可以对 dropna() 传递额外的参数对缺失项进行特殊处理。

- axis: {0 or 'index', 1 or 'columns'}, default 0。axis 参数接受一个集合，集合可选 0 或 1 或 'index' 或 'columns'，0 或 'index' 表示 drop 行，1 或 'columns' 表示 drop 列。
- how: {'any', 'all'}, default 'any'。How 集合包含 'any' 时，会将包含一个 NaN 项的整行都 drop 删除。'all' 则表示只有所有值为 NaN 时才 drop 所在行列。
- thresh: int, optional。thresh 为可选项，thresh 的数值表示当非 NaN 项数目大于 thresh 数值时保留整行。

```
In [41]: mans
Out[41]:
   ID wujiang  force
0  2016212070   关羽   98.0
1  2016212071   张飞   99.0
2  2016212072   NaN   97.0
3  2016212073   NaN    NaN
4  2016212074   NaN    NaN
5  2016212075   于禁    NaN
6  2016212076   徐晃   89.0
7  2016212077   张辽   91.0
8  2016212078   乐进   88.0
9  2016212079   张郃   90.0

In [42]: mans.dropna(axis=0, how='any')
Out[42]:
   ID wujiang  force
0  2016212070   关羽   98.0
1  2016212071   张飞   99.0
6  2016212076   徐晃   89.0
7  2016212077   张辽   91.0
8  2016212078   乐进   88.0
9  2016212079   张郃   90.0
```



```
In [46]: mans.dropna(axis=0,how='all',thresh=2)
Out[46]:
```

	ID	wujiang	force
0	2016212070	关羽	98.0
1	2016212071	张飞	99.0
2	2016212072	NaN	97.0
5	2016212075	于禁	NaN
6	2016212076	徐晃	89.0
7	2016212077	张辽	91.0
8	2016212078	乐进	88.0
9	2016212079	张郃	90.0

3.3 Pandas 高阶操作

3.3.1 连接、合并

Pandas 提供了多种操作将 Series, DataFrame 对象通过索引以及关系代数逻辑组合在一起。

① concat()方法

concat()函数严格按照轴对 DataFrame 对象进行串联或并联操作。串联,即 concat()中 axis 参数为 0 或'index',即多个 Series 或 DataFrame 对象按照行索引连接,类似 append()函数从一个对象中添加其他对象。并联,即 concat()中参数为 1 或'columns',具体操作类似与数据库中的多表连接,多个 DataFrame 对象按照列并在一起,根据 join 参数又分为内联和外联两种,当 join='inner'时,两个 DataFrame 对象连接得到共有 columns 组成的新 DataFrame 对象,当 join='outer'(即默认情况下),会将生成两个 DataFrame 对象并联的 columns 下的新 DataFrame 对象,具体操作见下:

```
In [97]: s1 = pd.Series(['a', 'b'])
In [98]: s2 = pd.Series(['c', 'd'])
In [99]: pd.concat([s1, s2],axis=0)
Out[99]:
```

0	a
1	b
0	c
1	d

dtype: object

```
In [100]: pd.concat([s1, s2],axis=1)
Out[100]:
```

	0	1
0	a	c
1	b	d

```

In [101]: s3 = pd.Series(['b','c','d'])
In [102]: pd.concat([s1, s3],axis=0)
Out[102]:
0    a
1    b
0    b
1    c
2    d
dtype: object

In [103]: pd.concat([s1, s3],axis=1)
Out[103]:
   0  1
0  a  b
1  b  c
2 NaN d

In [104]: pd.concat([s1, s3],axis=1,join='inner')
Out[104]:
   0  1
0  a  b
1  b  c

```

此外，还有参数 `ignore_index`(默认为 `False`)，当其为 `True` 时，表示连接后重新编排索引，这主要是考虑到 `axis=0` 时，连接中有些索引会重复(如上图)，造成不必要的影响。

```

In [105]: pd.concat([s1, s3],axis=0)
Out[105]:
0    a
1    b
0    b
1    c
2    d
dtype: object

In [106]: pd.concat([s1, s3],axis=0,ignore_index=True)
Out[106]:
0    a
1    b
2    b
3    c
4    d
dtype: object

```

② merge()合并函数

`merge()`函数作为连接操作类似数据库语言外连接操作中的 `join`。其中主要参数如下：

- `left` : DataFrame 左边的 DataFrame 对象
- `right` : DataFrame 右边的 DataFrame 对象
- `how` : {'left', 'right', 'outer', 'inner'}, default 'inner'。 `how` 参数表示数据融合的方式。`outer` 和 `inner` 的区别和上述 `concat()`中 `join` 参数类似，`left` 和数据库中的左链接相似，即保留左边 DataFrame 对象，以此为基础合并新的 DataFrame 对象。`right` 则与类似 SQL 中的右外连接，与 `left` 正好相反。

- **on:label or list.** 用来对齐的那一列的名字，用到这个参数的时候一定要保证左表和右表用来对齐的那一列都有相同的列名。如果 **on=None** 并且不合并索引，则默认为连个 DataFrame 对象 **columns** 的交集。

```
In [110]: left
Out[110]:
  key1 key2  A  B
0  K0   K0  A0 B0
1  K0   K1  A1 B1
2  K1   K0  A2 B2
3  K2   K1  A3 B3

In [111]: right
Out[111]:
  key1 key2  C  D
0  K0   K0  C0 D0
1  K1   K0  C1 D1
2  K1   K0  C2 D2
3  K2   K0  C3 D3

In [112]: pd.merge(left,right,how='left')
Out[112]:
  key1 key2  A  B  C  D
0  K0   K0  A0 B0  C0 D0
1  K0   K1  A1 B1  NaN NaN
2  K1   K0  A2 B2  C1 D1
3  K1   K0  A2 B2  C2 D2
4  K2   K1  A3 B3  NaN NaN

In [113]: pd.merge(left,right,how='right')
Out[113]:
  key1 key2  A  B  C  D
0  K0   K0  A0 B0  C0 D0
1  K1   K0  A2 B2  C1 D1
2  K1   K0  A2 B2  C2 D2
3  K2   K0  NaN NaN  C3 D3

In [114]: pd.merge(left,right,how='inner')
Out[114]:
  key1 key2  A  B  C  D
0  K0   K0  A0 B0  C0 D0
1  K1   K0  A2 B2  C1 D1
2  K1   K0  A2 B2  C2 D2

In [115]: pd.merge(left,right,how='outer')
Out[115]:
  key1 key2  A  B  C  D
0  K0   K0  A0 B0  C0 D0
1  K0   K1  A1 B1  NaN NaN
2  K1   K0  A2 B2  C1 D1
3  K1   K0  A2 B2  C2 D2
4  K2   K1  A3 B3  NaN NaN
5  K2   K0  NaN NaN  C3 D3
```

```
In [116]: pd.merge(left,right,how='outer',on=['key1', 'key2'])
Out[116]:
```

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

③ join()

join()通过索引与其他 DataFrame 对象连接。其操作较为简答，故不过多说明。

```
In [123]: left
Out[123]:
```

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

```
In [124]: right
Out[124]:
```

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

```
In [125]: left.join(right,lsuffix='_left',rsuffix='_right')
Out[125]:
```

	key1_left	key2_left	A	B	key1_right	key2_right	C	D
0	K0	K0	A0	B0	K0	K0	C0	D0
1	K0	K1	A1	B1	K1	K0	C1	D1
2	K1	K0	A2	B2	K1	K0	C2	D2
3	K2	K1	A3	B3	K2	K0	C3	D3

其中,lsuffix 参数表示对左边的 DataFrame 对象中的重复列名通过添加后缀的方式重命名,rsuffix 则对右边的 DataFrame 对象进行同样的操作。

3.3.2 apply()函数

apply()是 DataFrame 中的一个模块，其根据 axis 参数的值选择性地对 DataFrame 对象中的索引(index)或列(columns)进行函数 func 应用。apply()中主要参数如下

- > func : function 。对行或列进行操作的函数
- > axis: {0 or 'index', 1 or 'columns'}, default 0 。0 or 'index'表示对以索引为轴进行操作，1 or 'columns'表示以列为轴进行函数操作。
- > raw : bool, default False。当 row 为 False 时，将 DataFrame 对象按行或列拆解为 Series 作为参数传递到 func 中。当 row 为 True 时，则以 ndarray 为参数传递给 func。

代码：


```

In [149]: def f(x):
...:     return pd.Series([x.max(),x.min()],index = ['max','min'])
...:
...:

In [150]: rand.apply(f,axis=0)
Out[150]:
   0  1  2  3
max 6  5  5  6
min 2  0  0  0

In [151]: rand.apply(f,axis=0)
Out[151]:
   0  1  2  3
max 6  5  5  6
min 2  0  0  0

In [152]: rand.apply(f,axis=1)
Out[152]:
      max  min
0       5    3
1       6    0
2       6    0
3       2    0
4       6    1

```

此外，func 函数亦可为匿名函数

```

In [153]: rand.apply(lambda x : 2*x ,axis=0)
Out[153]:
   0  1  2  3
0   6  8 10 10
1  12  6  4  0
2  12  6  0  6
3   4  0  0  2
4   8 10  2 12

In [154]: rand.apply(np.sort,axis=1)
Out[154]:
0    [3, 4, 5, 5]
1    [0, 2, 3, 6]
2    [0, 3, 3, 6]
3    [0, 0, 1, 2]
4    [1, 4, 5, 6]
dtype: object

```

此外，类似的还有 `applymap()`，其中的函数 `func` 的参数为 `DataFrame` 对象中的每一个元素，其返回对象为与 `DataFrame` 同规模的一个新的 `DataFrame` 对象。该新对象中的每一个元素为原始对象对应的函数对函数 `func` 的应用结果。

```

In [162]: df = pd.DataFrame([["abcd","bg"],["lookatme","google"]])

In [163]: df.applymap(lambda x : len(str(x)))
Out[163]:
   0  1
0  4  2
1  8  6

```

3.3.3 Groupby 分组

Pandas 中 `groupby` 功能使得我们能轻松的对数据集进行切片，切块等操作。`groupby()` 内的一个重要参数。by 参数形式多样，其决定了分组的根据。如果 by 为一个函数，这个

函数将对对象索引上的每一个值调用。当 **by** 为一个 **label** 标签时，则按照这个标签对数据进行划分。

```
In [172]: x
Out[172]:
```

	A	B	C	D
0	foo	one	1.346315	1.400316
1	bar	one	1.500850	0.925859
2	foo	two	-0.579266	-1.419153
3	bar	three	1.062421	0.131519
4	foo	two	-1.304869	0.315941
5	bar	two	0.181252	-0.484416
6	foo	one	0.525914	0.229039
7	foo	three	1.766541	-1.149076

```
In [173]: x.groupby('A')
Out[173]: <pandas.core.groupby.groupby.DataFrameGroupBy object at 0x7fe6e9423f28>

In [174]: x.groupby('A').sum()
Out[174]:
```

	C	D
A		
bar	2.744523	0.572961
foo	1.754635	-0.622934

```
In [175]: x.groupby(['A','B']).sum()
Out[175]:
```

A	B	C	D
bar	one	1.500850	0.925859
	three	1.062421	0.131519
	two	0.181252	-0.484416
foo	one	1.872229	1.629354
	three	1.766541	-1.149076
	two	-1.884134	-1.103212

Groupby 操作类似于 SQL 语言中的分组，其通过 **by** 这个分组依据进行划分得到一个 **groupby** 对象，最后应用 **sum()**或 **mean()**过滤数据得到一个分组 **DataFrame** 对象。

二 Sklearn 学习

由于前段时间参加互联网+的一个比赛加上临近期末各种报告繁多，Sklearn 模块这块没能完成。由此简要谈谈对这方面的理解，只是稍微看了点资料。Sklearn 是 python 中针对机器学习编写的一个第三方模块，其对于常见的一些机器学习方法进行了封装。所以在实际分析中可以直接调用该模块中的方法进行各种分类，回归以及聚类任务，而不需要用到我们实验二中我们编写的那些繁冗的代码。

在对某个数据集进行分析时，针对庞大的数据，我们首先需要从数据集中提取出有效的数据，一方面是对数据集中一些缺省的数据如何处理，如果直接删除未免太浪费，当然也可以像上面 pandas 学习中针对 **missing data** 的方式根据缺省程度决定是否保留，再一个有些数据集的属性特征太多，如何提取选出对分类影响最大的那部分特征，又要采取什么何时度量单位，算法进行比较从而选出不冗余的属性。做完这些，我们算是成功提炼了一个有效的训练集，接下来按理要进行训练模型，但是考虑到数据集还是庞大，构成的特征矩阵过大，导致计算时间过长。因此在这个特征矩阵熵我们还需要采取一些数据手段进行优化，其中一个思路就是将矩阵的维度递归式的降解，具体的方法我想 sklearn 或者其他数学库中有，我们应该比较选出最合适的方法。然后是训练模型算法的选择了，常有的算法主要分聚类，分类，回归几大类，首先根据训练的任务，到底是监督学习还是无监督学习进行一个初步选择，然后在各大算法之间进行比较，看那个模型的准确率高，泛化性能好，之后对于学习器我们还需要采用手段取强化，提炼。就像集成学习中 **AdaBoost** 算法那样，通过对已生成的学习器包括数据集进行动态调整，一次次地递归，不断纠错，加强模型的泛化能力，

然后或是选择最终的生成器，亦或是生成的各个生成器加权结合，得到强化的模型。这些便是我对于 **sklearn** 那块要解决的问题的一点粗浅看法，我相信还有更优秀的算法以及优化方式，而这些也是我将来要学习的目标。