

Python 深度解析之

璀璨明珠 pandas 基础篇

前言: 如果说 numpy 是科学计算里底层的砖瓦, 那么 pandas 必定是基于砖瓦的摩天大厦, 功能之强大, 效率之高, 吸引了大批处理数据的人为之疯狂, 使用之人必定爱不释手。Pandas 不仅能处理我们日常工作中的 excel, csv, hdf5, 而且能跟数据库交互, 非常方便。我在接触之后也是喜爱不已, 继上一篇 numpy 之后, 我们今天开始来学习一下 pandas 的用法。

By 浪不沕沙

● Pandas 的简介

Pandas 全称 Python Data Analysis Library, 它是基于 NumPy 的一种工具, 该工具是为了解决数据分析任务而创建的。Pandas 纳入了大量库和一些标准的数据模型, 提供了高效地操作大型数据集所需的工具。最初由 AQR Capital Management 于 2008 年 4 月开发, 并于 2009 年底开源出来, 目前由专注于 Python 数据包开发的 PyData 开发 team 继续开发和维护, 属于 PyData 项目的一部分。Pandas 最初被作为金融数据分析工具而开发出来, 因此, pandas 为时间序列分析提供了很好的支持。Pandas 的名称来自于面板数据 (panel data) 和 python 数据分析 (data analysis)。

◆ Pandas 的对象

- Series: series 是 pandas 中最基本的对象, 他定义了 numpy 的 ndarray 对象的接口 `__array__()`, 因此可以利用 numpy 的数组处理函数直接对 series 对象进行处理。通俗点可以认为是一维数组, 有点类似我们常用的 list。由于比较局限, 日常用的不是很多。我们来看个例子。

```
In [2]: import numpy as np
import pandas as pd

data_list = [i for i in range(5)]
data_series = pd.Series(data_list, index=list('abcde'))
```

```
In [3]: data_series, data_series.index, data_series.values
```

```
Out[3]: (a    0
b    1
c    2
d    3
e    4
dtype: int64,
Index(['a', 'b', 'c', 'd', 'e'], dtype='object'),
array([0, 1, 2, 3, 4], dtype=int64))
```

Series 的 index 和 values 属性分别用来获取索引和对应的值, 如果不指定索引默认是从 0 开始的自然数。

Series 同样支持切片操作, 由于我们指定了索引, 所以我们可以用默认索引的自然数也可以使用指定的索引来切片。尤其要注意的是: **使用默认的自然数切片不包括结尾, 但是使用指定的索引切片则包含结尾的值。**

```
In [4]: data_series[1:4], data_series['b':'e']
```

```
Out[4]: (b    1
         c    2
         d    3
         dtype: int64, b    1
         c    2
         d    3
         e    4
         dtype: int64)
```

Series 同样支持数组切片。

```
In [5]: data_series[[2,3,4]], data_series[['c','d','e']]
```

```
Out[5]: (c    2
         d    3
         e    4
         dtype: int64, c    2
         d    3
         e    4
         dtype: int64)
```

Series 同时还具有字典的一些属性, 比如 `items`, 把索引和对应值组合, 返回一个可迭代对象。

```
In [8]: list(data_series.items())
```

```
Out[8]: [('a', 0), ('b', 1), ('c', 2), ('d', 3), ('e', 4)]
```

```
In [9]: for item in data_series.items():
         print(item)
```

```
('a', 0)
('b', 1)
('c', 2)
('d', 3)
('e', 4)
```

Series 可以实现操作符运算, 在计算时按照指定索引对齐的原则, 缺失的部分默认为 NaN。

```
In [11]: data_series2 = pd.Series(data_list, index=list('bcdef'))
         data_series, data_series2, data_series+data_series2]
```

```
a    0
b    1
c    2
d    3
e    4
dtype: int64 b    0
c    1
d    2
e    3
f    4
dtype: int64 a    NaN
b    1.0
c    3.0
d    5.0
e    7.0
f    NaN
dtype: float64
```

● 2、DataFrame

Dataframe 是 pandas 中用的最频繁的对象, 可以理解为二维 series, 在我们工作中比如 excel 的表, 关系型数据库的表, csv 文件, 读取之后都是打 dataframe 对象。我们着重讲一下 dataframe。

与 numpy 的二维数组不同, pandas 支持复合索引, 来读取一个我自己随便写的一个 excel, 针对表格来讲。

	A	B	C	D	E	F	G
1			大小	厚度	包邮	有无赠送品	发货日期
2	淘宝	ruby	5	5	是	是	2017/12/17
3		prel	6	6	否	否	2017/12/18
4		c	7	7	是	是	2017/12/19
5		java	5	5	否	否	2017/12/20
6		python	6	6	是	是	2017/12/21
7	京东	ruby	7	7	否	否	2017/12/22
8		prel	5	5	是	是	2017/12/23
9		c	6	6	否	是	2017/12/24
10		java	7	7	是	是	2017/12/25
11		python	6	6	否	是	2017/12/26

用 pandas 读取一下, 指定 index 的列和日期。并给 columns 和 index 命名。

```
da = pd.read_excel(r'C:\Users\wqq\Desktop\dataframe.xlsx',
                  sheetname='Sheet1', index_col=[0, 1], parse_dates=['发货日期'])
da.columns.name='属性'
da.index.names=['平台', '商品']
da
```

	属性	大小	厚度	包邮	有无赠送品	发货日期
平台	商品					
淘宝	ruby	5	5	是	是	2017-12-17
	prel	6	6	否	否	2017-12-18
	c	7	7	是	是	2017-12-19
	java	5	5	否	否	2017-12-20
	python	6	6	是	是	2017-12-21
京东	ruby	7	7	否	否	2017-12-22
	prel	5	5	是	是	2017-12-23
	c	6	6	否	是	2017-12-24
	java	7	7	是	是	2017-12-25
	python	6	6	否	是	2017-12-26

暂时还没发现有符合列索引的用法。我们来看一下 index 和 columns 的属性。这里 index 是一个复合索引, 而 columns 是单一的列名。

```
: da.index
: MultiIndex(levels=[['京东', '淘宝'], ['c', 'java', 'perl', 'python', 'ruby']],
              labels=[[1, 1, 1, 1, 1, 0, 0, 0, 0, 0], [4, 2, 0, 1, 3, 4, 2, 0, 1, 3]],
              names=['平台', '商品'])
```

Levels 表示行索引的名称, labels 可以认为是 levels 的索引, 这里的 labels 并没有按照顺序排列, 这是一个很值得探讨的问题。

```
: da.columns
: Index(['大小', '厚度', '包邮', '有无赠送品', '发货日期'], dtype='object', name='属性')
```

columns 里分别是名称、名称的类型、名称的名字。

■ DataFrame 的创建

除了上述的读取 excel, 或者其它文件, 我们还有使用其它方法来创建, 比如: 从 numpy 的数组, 或者字典。

```
arr = np.random.randint(1, 10, (4, 5))
danp = pd.DataFrame(arr, index=list('abcd'), columns=list('ABCDE'))
danp
```

	A	B	C	D	E
a	6	3	3	3	4
b	1	6	7	8	7
c	4	8	4	1	8
d	9	9	3	3	4

同样我们还可以用字典来创建:

```
dics = {'A': [1, 2, 3, 4, 5], 'B': [1, 2, 3, 4, 5], 'C': [1, 2, 3, 4, 5], 'D': [1, 2, 3, 4, 5], 'E': [1, 2, 3, 4, 5]}
dadic = pd.DataFrame(dics, index = list('qwert'))
dadac
```

	A	B	C	D	E
q	1	1	1	1	1
w	2	2	2	2	2
e	3	3	3	3	3
r	4	4	4	4	4
t	5	5	5	5	5

这里默认字典的键为 columns, 我们还有更灵活的用法, 那就是 from_系列的方法, numpy 里也是有的。我们来看看 pandas 的 from 系列。

from_dic(), orient 参数指定字典的键是 index 还是 columns。经本人测试字典的嵌套不能超过两层, 不然多了也是没有效果的。

我们来展示下效果。

```
dics2 = {"a": [1, 2, 3], "b": [2, 3, 4]}
dics3 = {"a": {"A": 1, "B": 4}, "b": {"B": 3, "A": 100}}
dadic2 = pd.DataFrame.from_dict(dics2, orient='index')
dadic3 = pd.DataFrame.from_dict(dics2, orient='columns')
dadic4 = pd.DataFrame.from_dict(dics3, orient="index")
dadic5 = pd.DataFrame.from_dict(dics3, orient="columns")
```

dadic2				dadic3				dadic4				dadic5			
				a	b					A	B				
0	1	2		0	1	2						a	b		
a	1	2	3	1	2	3		a		1	4	A	1	100	
b	2	3	4	2	3	4		b	100	3		B	4	3	

Dataframe 除了 from 系列还有 to 系列, 比较常用的是 to_excel, to_csv, to_excel, to_json, to_dict. 方法非常多, 这里说下格式转换用的 to_dict.

To_dict 可以把一个 dataframe 对象转化为字典列表, 列表字典, 或者嵌套字典。

	a	b
A	1	100
B	4	3

```
dadic5.to_dict(orient='records')
```

```
[{'a': 1, 'b': 100}, {'a': 4, 'b': 3}]
```

```
dadic5.to_dict(orient='list')
```

```
{'a': [1, 4], 'b': [100, 3]}
```

```
dadic5.to_dict(orient='dict')
```

```
{'a': {'A': 1, 'B': 4}, 'b': {'A': 100, 'B': 3}}
```

■ Dataframe 的 index 对象

Index 也就是 Dataframe 的行索引, 比较特殊的就是有一个复合索引的出现, 不然真是没有什么好说的。

Index 对象里有一系列的 get 方法。

```
da.index
```

```
MultiIndex(levels=[['京东', '淘宝'], ['c', 'java', 'perl', 'python', 'ruby']],
            labels=[[1, 1, 1, 1, 1, 0, 0, 0, 0, 0], [4, 2, 0, 1, 3, 4, 2, 0, 1, 3]],
            names=['平台', '商品'])
```

```
da.index.levels[0].get_loc('淘宝')
```

```
1
```

```
da.columns
```

```
Index(['大小', '厚度', '包邮', '有无赠送品', '发货日期'], dtype='object', name='属性')
```

```
da.columns.get_loc('厚度')
```

```
1
```

get_loc 返回对应标签的索引值。

```
da.index.levels[1].get_indexer(['java', 'python'])
```

```
array([1, 3], dtype=int32)
```

```
da.columns.get_indexer(['包邮', '重量', '是否白送'])
```

```
array([ 2, -1, -1], dtype=int32)
```

get_indexer 批量查询对应索引, 如果不存在则返回-1.

下面看看比较复杂的 multiindex 对象。

- 1、从元组创建, 比如创建 0 级索引为 “A” 和 “B”, 1 级索引为 ‘a’, ‘b’, ‘c’ 的 dataframe, 我们可以这么写。

```
index_list = [('A', 'a'), ('A', 'b'), ('A', 'c'), ('B', 'a'), ('B', 'b'), ('B', 'c')]
ind = pd.Index(index_list, name=['0级', '1级'])
```

```
MultiIndex(levels=[['A', 'B'], ['a', 'b', 'c']],
            labels=[[0, 0, 0, 1, 1, 1], [0, 1, 2, 0, 1, 2]],
            names=['0级', '1级'])
```

```
arr1 = np.random.randint(1, 10, (6, 2))
pd.DataFrame(arr1, index=ind, columns=['r', 'y'])
```

		r	y
0级	1级		
A	a	9	2
	b	6	2
	c	2	8
B	a	1	4
	b	1	4
	c	5	6

2、使用 multiindex 的 from_arrays 创建

```
list1 = ['A', 'A', 'B', 'B']
list2 = ['c', 'd', 'c', 'd']
pd.MultiIndex.from_arrays([list1, list2], names=['x', 'y'])

MultiIndex(levels=[['A', 'B'], ['c', 'd']],
            labels=[[0, 0, 1, 1], [0, 1, 0, 1]],
            names=['x', 'y'])
```

可以在创建 Dataframe 的时候传递给 index 参数即可。

3、使用 multiindex 的 from_product 创建

这种方法是最实用的，我们只需要知道每一级的索引包含哪几个，不用我们去自己组合，from_product 会自动使用笛卡尔乘积帮我们组合出所有的复合索引。来看看效果。

```
list3 = list('你好吗')
list4 = list('are')
pd.MultiIndex.from_product([list3, list4], names = ['q', 'w'])

MultiIndex(levels=[['你', '吗', '好'], ['a', 'e', 'r']],
            labels=[[0, 0, 0, 2, 2, 2, 1, 1, 1], [0, 2, 1, 0, 2, 1, 0, 2, 1]],
            names=['q', 'w'])
```

```
ind1 = pd.MultiIndex.from_product([list3, list4], names = ['q', 'w'])
arr = np.random.choice(list('我不是太好'), (9, 4))
pd.DataFrame(arr, index = ind1, columns=list('不好拉倒'))
```

看看我们生成的 Dataframe:

	q	w
你	a	好 我 不 好
	r	是 好 不 我
	e	不 是 我 是
好	a	我 好 是 好
	r	是 好 不 不
	e	好 好 不 我
吗	a	是 不 太 不
	r	我 我 我 我
	e	是 太 是 太

■ Dataframe 的切片

Dataframe 同样有切片功能，而且方法还很多。我们以 excel 京东淘宝数据的例子来逐个演示。

使用列索引来切: `dataframe[‘列名’]`, 切多列的话可以同时以一个 list 传入。

```
da[['大小', '厚度']]
```

	属性	大小	厚度
平台	商品		
淘宝	ruby	5	5
	prel	6	6
	c	7	7
	java	5	5
	python	6	6
京东	ruby	7	7
	prel	5	5
	c	6	6
	java	7	7
	python	6	6


```
da['包邮']
```

平台	商品	是
淘宝	ruby	是
	prel	否
	c	是
	java	否
	python	是
京东	ruby	否
	prel	是
	c	否
	java	是
	python	否

Name: 包邮, dtype: object

使用行索引的切片:

行索引的切法非常多, 比较常用的方法有 `header`, `tail`, `iloc`, `loc`。

`header(int)`: 相当于 mysql 里的 `limit` 选取前 `int` 行。`Tail` 同理, 从最后边往前切。

`iloc`: 以 `integer` 参数来切, `i` 我一直当成是整数来理解的, 可能不对, 但是方便记忆。

`loc`: 以指定的索引名称来切。

下面看下三种切片的效果:

```
da.head(5)
```

	属性	大小	厚度	包邮	有无赠送品	发货日期
平台	商品					
淘宝	ruby	5	5	是	是	2017-12-17
	prel	6	6	否	否	2017-12-18
	c	7	7	是	是	2017-12-19
	java	5	5	否	否	2017-12-20
	python	6	6	是	是	2017-12-21

`tail(int)` 就不多说了, 取最后 `int` 行。

下面重点讲下 `iloc` 和 `loc`。


```
: da.iloc[1:3]
```

```
:
      属性 大小 厚度 包邮 有无赠送品 发货日期
平台 商品
淘宝 prel    6    6  否         否 2017-12-18
      c    7    7  是         是 2017-12-19
```

我们可以按照 list 的切片方法输入, 直接选取 1, 2 两行, 这里也不是包含结尾的。同样可以用 list 的方式传值, 获取不连续的行。

```
: da.iloc[[1, 3, 5]]
```

```
:
      属性 大小 厚度 包邮 有无赠送品 发货日期
平台 商品
淘宝 prel    6    6  否         否 2017-12-18
      java    5    5  否         否 2017-12-20
京东 ruby    7    7  否         否 2017-12-22
```

iloc 不光能切行, 还能同时切割列, 也是用 integer 的参数。

```
da.iloc[1:3, 1:4]
```

```
      属性 厚度 包邮 有无赠送品
平台 商品
淘宝 prel    6  否         否
      c    7  是         是
```

支持两个序列的不连续切割, 我们来看下。

```
da.iloc[[1, 3, 4], [1, 3, 4]]
```

```
      属性 厚度 有无赠送品 发货日期
平台 商品
淘宝 prel    6         否 2017-12-18
      java    5         否 2017-12-20
      python  6         是 2017-12-21
```

再看看 0 轴连续, 1 轴不连续的切法:

```
|: da.iloc[1:4, [1, 3, 4]]
```

```
|:
```

	属性	厚度	有无赠送品	发货日期
平台	商品			
淘宝	prel	6	否	2017-12-18
	c	7	是	2017-12-19
	java	5	否	2017-12-20

iloc 是我们在使用 pandas 经常会用到的方法, 推荐重点掌握, 让我们现在看看 loc.loc 的用法跟 iloc 类似, 只不过是以我们指定的标签索引来切割。

```
: da.loc['淘宝', ['厚度', '发货日期']]
```

```
:
```

属性	厚度	发货日期
商品		
ruby	5	2017-12-17
prel	6	2017-12-18
c	7	2017-12-19
java	5	2017-12-20
python	6	2017-12-21

看看连续切割, 这里因为是复合型的行索引, 如果想按照二级行索引切割, 会麻烦一点。提供两种切法。

```
da.loc['淘宝', '厚度': '发货日期'].loc['ruby': 'java', :]
```

属性	厚度	包邮	有无赠送品	发货日期
商品				
ruby	5	是	是	2017-12-17
prel	6	否	否	2017-12-18
c	7	是	是	2017-12-19
java	5	否	否	2017-12-20

另外一种可以使用 slice 的方法。

```
da.loc[pd.IndexSlice[:, 'python'], :]
```

	属性	包邮	厚度	发货日期	大小	有无赠送品
平台	商品					
京东	python	否	6	2017-12-26	6	是
天猫	python	否	6	2017-12-26	6	是
淘宝	python	是	6	2017-12-21	6	是

这里尤其要说一下的就是, 如果你的 labels 里不是按照升序排列的, 你不能使用这个切片方法。反正耽误了我一个小时才找到这个原因, 吐血。后来只能是在对 da 的 labels 排序完才切割出来。图中天猫那个数据是我在调试的时候加上去的, 不应该出现在数据中。

```
: da = da.sort_index().sort_index(axis=1)
da.index
: MultiIndex(levels=[['京东', '天猫', '淘宝'], ['c', 'java', 'perl', 'python', 'ruby']],
              labels=[[0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2], [0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4]],
              names=['平台', '商品'])
```

同 numpy 一样, pandas 的 dataframe 也支持筛选。

```
da[da.厚度 == 7]
```

	属性	大小	厚度	包邮	有无赠送品	发货日期
平台	商品					
淘宝	c	7	7	是	是	2017-12-19
京东	ruby	7	7	否	否	2017-12-22
	java	7	7	是	是	2017-12-25

Pandas 还有个 query 的查询方法。用来简化筛选条件多的时候。下面展示一下多条件筛选:

```
da[(da.厚度>5) & (da.包邮=='是')]
```

	属性	大小	厚度	包邮	有无赠送品	发货日期
平台	商品					
淘宝	c	7	7	是	是	2017-12-19
	python	6	6	是	是	2017-12-21
京东	java	7	7	是	是	2017-12-25

如果是使用 query 的方法, 我们可以这么写:

```
da.query("厚度>5 and 包邮 == '是'")
```

	属性	大小	厚度	包邮	有无赠送品	发货日期
平台	商品					
淘宝	c	7	7	是	是	2017-12-19
	python	6	6	是	是	2017-12-21
京东	java	7	7	是	是	2017-12-25

除此之外, pandas 还提供了和 mysql 类似的设置变量查询。在使用中变量名只需要在前边加上@符号即可。

```
hd = 5
by = '是'
da.query("厚度>@hd and 包邮 ==@by")
```

	属性	大小	厚度	包邮	有无赠送品	发货日期
平台	商品					
淘宝	c	7	7	是	是	2017-12-19
	python	6	6	是	是	2017-12-21
京东	java	7	7	是	是	2017-12-25

切片的操作差不多就这么多了, 当然还有个 ix 的用法没有说, ix 可以混用整数和标签切片, 容易造成混淆, 官方已经不建议使用了。有兴趣的可以去看下。这里要说下, 使用整数连续切片的时候是不包含下标, 而标签切片则会包含下标。

● 3、Panel

Panel 可以想象为三维数组, 实际上用的不是特别的多。我们可以写个例子看看。

Panel 的作用就是用来存放 Dataframe 的。

```
arr3 = np.random.randint(1, 10, 27).reshape(3, 3, -1)
arr3
```

```
array([[[9, 5, 4],
        [8, 2, 5],
        [7, 5, 7]],

       [[1, 9, 4],
        [1, 2, 9],
        [1, 1, 8]],

       [[3, 5, 8],
        [5, 1, 9],
        [7, 1, 4]]])
```

先随机生成一个 3*3*3 的数组。

```
pd.Panel(arr3)
```

```
<class 'pandas.core.panel.Panel'>
Dimensions: 3 (items) x 3 (major_axis) x 3 (minor_axis)
Items axis: 0 to 2
Major_axis axis: 0 to 2
Minor_axis axis: 0 to 2
```

Panel 并不能直接显示出来, 下边这些乱七八糟的说的就是这个容器里存了 3 个主题, 这个主题的大小是 3*3. 又说了轴是 [0, 1, 2]。我们可以通过切片的方式来查看。

```
pd.Panel(arr3)[0]
```

	0	1	2
0	9	5	4
1	8	2	5
2	7	5	7

```
type(pd.Panel(arr3)[0])
```

```
pandas.core.frame.DataFrame
```

切片完成之后, 其本质是一个 DataFrame 对象。还没有发现如何给它三个轴都添加上索引, 有兴趣的同学可以研究下。

小结: 到这里, pandas 的基础篇也就讲完了, 东西还是比较琐碎的, 不成什么逻辑, 如果想熟练掌握还是需要多写, 纵观全篇, 其实还是有很多方法和 numpy 一样的, 如果我们在学习过程中善于发现和总结, 相同的知识点能够放在一起类比着学习, 我相信你的速度不会慢。下一期我们开始讲 pandas 的分组, 挂表, 透视等相对高级一点的东西。有想学习 python 的同学可以加群:

QQ 群: 518980304, 如果您发现本篇有什么错误, 可能会误导学习的同学, 欢迎批评指正。QQ 号: 383750993.