



pandas



pandas

- pandas是一种Python数据分析的利器，是一个开源的数据分析包，最初是应用于金融数据分析工具而开发出来的，因此pandas为时间序列分析提供了很好的支持。pandas是PyData项目的一部分。
- 官网：<http://pandas.pydata.org/>
- 官方文档：<http://pandas.pydata.org/pandas-docs/stable/>
- <https://www.py pandas.cn/>

pandas安装

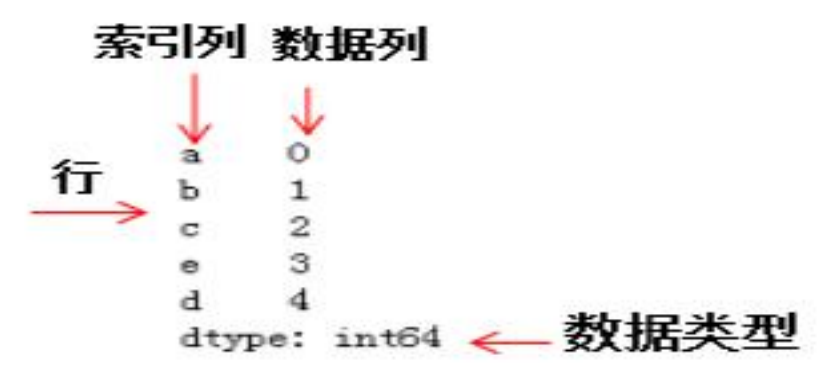
- 安装过程详见: [官方安装文档](#)
- Python版本要求: 2.7、3.4、3.5、3.6
- 依赖Python库: [setuptools](#)、[NumPy](#)、[python-dateutil](#)、[pytz](#)
- 安装方式:
 - Python的Anaconda发行版, 已经安装好pandas库, 不需要另外安装
 - 使用Anaconda界面安装, 选择对应的pandas进行勾选安装即可
 - 使用Anaconda命令安装: `conda install pandas`
 - 使用PyPi安装命令安装: `pip install pandas`

pandas简介

- pandas是专门为处理表格和混杂数据设计的。而NumPy更适合处理统一的数值数组数据。
pandas主要数据结构：Series和DataFrame
- 系列(Series)是能够保存任何类型的数据(整数，字符串，浮点数，Python对象等)的一维标记数组。轴标签统称为索引。
- `import pandas as pd`
- `obj = pd.Series([4, 7, -5, 3])`
- Series的字符串表现形式为：索引在左边，值在右边。由于我们没有为数据指定索引，于是会自动创建一个0到N-1（N为数据的长度）的整数型索引。你可以通过Series 的values和index属性获取其数组表示形式和索引对象：
- `obj2 = pd.Series([4, 7, -5, 3], index=['d', 'b', 'a', 'c'])`

创建Series

series是一种一维数据结构，每一个元素都带有一个索引，与一维数组的含义相似，其中索引可以为数字或字符串。series结构名称：|索引列|数据列



```
import pandas as pd

obj1 = pd.Series([0,1, 2, 3, 4], index=['a', 'b', 'c', 'd','e'])

print(obj1)
```

创建Series

如何从列表，数组，字典构建series

```
mylist = list('abcdefghijklmnopqrstuvwxyz') # 列表
```

```
myarr = np.arange(26) # 数组
```

```
mydict = dict(zip(mylist, myarr)) # 字典
```

构建方法

```
ser1 = pd.Series(mylist)
```

```
ser2 = pd.Series(myarr)
```

```
ser3 = pd.Series(mydict)
```

```
print(ser3.head()) # 打印前5个数据
```

创建Series

创建的Series带有一个可以对各个数据点进行标记的索引

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

数据被存放在一个Python字典中，也可以直接通过这个字典来创建Series

```
sdata = {'Ohio': 35000, 'Texas': 71000, 'Oregon': 16000, 'Utah': 5000}
```

```
pd.Series(sdata)
```

```
states = ['California', 'Ohio', 'Oregon', 'Texas']
```

```
obj4 = pd.Series(sdata, index=states)
```

```
pd.isnull(obj4) #缺失值检测
```

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

```
obj4.isnull()
```

查找Series

创建Series对象并且查询index、values、dtype属性值

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

```
print(obj2.values)
```

```
print(obj2.index)
```

```
print(obj2.dtype)
```

Series值的获取主要有两种方式：

- 1、通过方括号+索引名的方式读取对应索引的数，有可能返回多条数据，
- 2、通过方括号+下标值的方式读取对应下标值的数据，下标值的取值范围为：`[0, len(Series.values))`；另外下标值也可以是负数，表示从右往左获取数据

查找Series

```
obj3 = pd.Series([4, 7, -5, 3,1], index=['d', 'b', 'a', 'c','a'])
```

```
obj3['a']
```

```
obj3[0]
```

```
obj3[-1]
```

Series获取多个值的方式类似NumPy中的ndarray的切片操作，通过方括号+下标值/索引值+冒号(:)的形式来截取series对象中的一部分数据

```
obj3[1:4]
```

```
obj3[1::2]
```

修改Series

修改index索引：通过数组创建Series的时候，如果没有为数据指定索引的话，会自动创建一个从0到N-1的整数索引；当Series对象创建好后，可以通过index修改索引值

```
obj4 = pd.Series([4, 7, -5, 3,1], index=['d', 'b', 'a', 'c','a'])
```

```
obj4.index = ['aa','bb','cc','dd','ee']
```

注：在创建Series对象的时候可以直接修改index属性值，和dtype属性值。

```
obj5 = pd.Series([4, 7, -5, 3,1],dtype=float, index=['d', 'b', 'a', 'c','a'])
```

Series的运算

NumPy中的数组运算，在Series中都保留了，均可以使用，并且Series进行数组运算的时候，索引与值之间的映射关系不会发生改变。

注意：其实在操作Series的时候，基本上可以把Series看成NumPy中的ndarray数组来进行操作。ndarray数组的绝大多数操作都可以应用到Series上。

```
obj6= Series([195, 73], index=["a", "b"])
```

```
print(obj6/ 268)
```

```
obj6+ 100
```

```
obj6* 2
```

```
obj6- 73
```

DataFrame

DataFrame是一个表格型的数据结构，它含有一组有序的列，每列可以是不同的值类型（数值、字符串、布尔值等）。DataFrame既有行索引也有列索引，它可以被看做由Series组成的字典（共用同一个索引）。DataFrame中的数据是以一个或多个二维块存放的（而不是列表、字典或别的一维数据结构）。有关DataFrame内部的技术细节远远超出了本书所讨论的范围。


```
data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada', 'Nevada'],
        'year': [2000, 2001, 2002, 2001, 2002, 2003],
        'pop': [1.5, 1.7, 3.6, 2.4, 2.9, 3.2]}

frame = pd.DataFrame(data)

pd.DataFrame(data, columns=['year', 'state', 'pop'])
```

DataFrame

dataframe是一种二维数据结构，数据以表格形式（与excel类似）存储，有对应的行和列。dataframe结构名称：



The diagram illustrates a DataFrame structure with annotations. A red arrow labeled '索引列' (Index Column) points to the index values (0, 1, 2, 3, 4). A red arrow labeled '行' (Row) points to the first column of data. A red arrow labeled '列名' (Column Name) points to the column headers 'crim' and 'medv'. A red arrow labeled '数据' (Data) points to the numerical values in the data columns.

		crim	medv
0	0.00632	24.0	
1	0.02731	21.6	
2	0.02729	34.7	
3	0.03237	33.4	
4	0.06905	36.2	

DataFrame

DataFrame是一个表格型的数据结构，它含有一组有序的列，每列可以是不同的值类型（数值、字符串、布尔值等）。DataFrame既有行索引也有列索引，它可以被看做由Series组成的字典（共用同一个索引）。DataFrame中的数据是以一个或多个二维块存放的（而不是列表、字典或别的一维数据结构）。有关DataFrame内部的技术细节远远超出了本书所讨论的范围。

```
data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada', 'Nevada'],  
        'year': [2000, 2001, 2002, 2001, 2002, 2003],  
        'pop': [1.5, 1.7, 3.6, 2.4, 2.9, 3.2]}  
  
frame = pd.DataFrame(data)  
  
pd.DataFrame(data, columns=['year', 'state', 'pop'])
```

DataFrame

列在数据中找不到，就会在结果中产生缺失值

```
frame2 = pd.DataFrame(data, columns=['year', 'state', 'pop', 'debt'], index=['one', 'two', 'three',  
'four', 'five', 'six'])
```

```
frame2.columns
```

```
frame2.year, frame2.loc['three']
```

重新索引

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

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

```
obj3 = pd.Series(['blue', 'purple', 'yellow'], index=[0, 2, 4])
```

```
obj3.reindex(range(6), method='ffill')
```

DataFrame

reindex可以修改（行）索引和列

```
frame = pd.DataFrame(np.arange(9).reshape((3, 3)),index=['a', 'c', 'd'],  
columns=['Ohio', 'Texas', 'California'])
```

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

```
data.drop(['Colorado', 'Ohio'])#删除行
```

```
data.drop('a', axis=1)#删除列
```


DataFrame

```
data = pd.DataFrame(np.arange(16).reshape((4, 4)), index=['Ohio', 'Colorado', 'Utah', 'New York'], columns=列['one', 'two', 'three', 'four'])
```

data['two'] = data.two

data[['three', 'one']] 列表形式

data[:2] # 切片

data[data['three'] > 5] # 布尔型数组选取数据

data[data < 5] = 0 # 赋值

data.loc['Colorado', ['two', 'three']] # 选择一行和多列

data.iloc[2, [3, 0, 1]] # 用iloc和整数进行选取

data.iloc[[行1, 2], [列3, 0, 1]]

data.iloc[2]

第二行

'two' 则是可以的，还有多条件
作为条件 筛选出来是3行，而非1列

→ 也可以是列表，第一是5的 第二是1是列
是交叉的

不指定
↑
data[2:][1:2]
相当一行两列

data[1:3][['two', 'one']] 也是交叉部分。
两个括号

索引、选取和过滤 *data.iloc[:, :3] 行列切片*

Series索引 (obj[...]) 的工作方式类似于NumPy数组的索引, 只不过Series的索引值不只是整数。

```
obj = pd.Series(np.arange(4.), index=['a', 'b', 'c', 'd'])
```

```
obj['b']
```

```
obj[2:4]
```

```
obj[['b', 'a', 'd']]
```

```
obj[[1, 3]]
```

```
obj[obj < 2]
```

```
obj['b':'c'] = 5 #切片可以对Series的相应部分进行设置
```

```
obj['b':'c'] #利用标签的切片运算与普通的Python切片运算不同
```

索引函数也适用于一个标签或多个标签的切片

Series索引 (obj[...]) 的工作方式类似于NumPy数组的索引，只不过Series的索引值不只是整数。

取到Utah (比较特殊)
data.loc['Utah', 'two']

data.iloc[:, :3][data.three > 5]

算术运算和数据对齐

Series索引 (obj[...]) 的工作方式类似于NumPy数组的索引，只不过Series的索引值不只是整数。

```
data.loc[:'Utah', 'two']
```

```
data.iloc[:, :3][data.three > 5]
```

```
df1 = pd.DataFrame(np.arange(9.).reshape((3, 3)), columns=list('bcd'), index=['Ohio', 'Texas', 'Colorado'])
```

```
df2 = pd.DataFrame(np.arange(12.).reshape((4, 3)), columns=list('bde'), index=['Utah', 'Ohio', 'Texas', 'Oregon'])
```

它们相加后将会返回一个新的DataFrame，其索引和列为原来那两个DataFrame的并集：

```
df1 + df2
```

有一个为空就为空，

```
df1 - df2
```

算术运算和数据对齐

将函数应用到由各列或行所形成的一维数组上。DataFrame的apply方法即可实现此功能：

```
frame = pd.DataFrame(np.random.randn(4, 3), columns=list('bde'), index=['Utah', 'Ohio', 'Texas', 'Oregon'])
```

```
f = lambda x: x.max() - x.min()
```

```
frame.apply(f)
```

```
frame.apply(f, axis='columns')
```

```
def f1(x):
```

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

```
frame.apply(f1)
```

文件另存为或改为CSV格式 → 通常用的
file name = r'1.csv'
data = pd.read_csv(filename, 'utf-8') # 转CSV文件
显示隐藏，显示文件扩展名，从而将txt改为CSV
从excel
V
转CSV文件

THANKS!

