



大数据，成就未来



# pandas统计分析基础

2018/9/30

# 目录

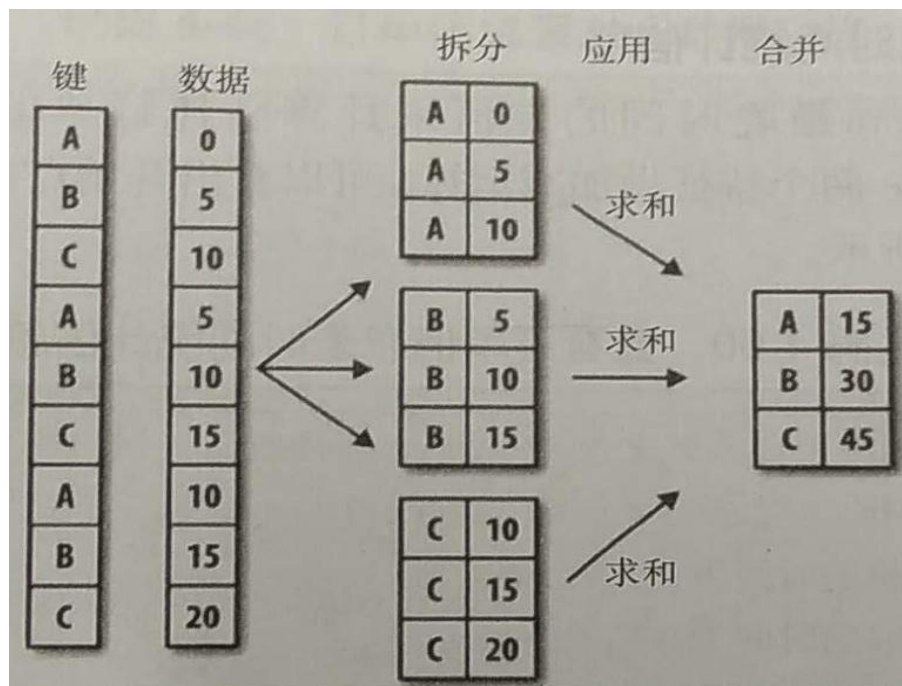
---



# 使用分组聚合进行组内计算

## 任务描述

- 依据某个或者某几个字段对数据集进行分组，并对各组应用一个函数，无论是聚合还是转换，都是数据分析的常用操作。
- pandas提供了一个灵活高效的groupby方法，配合agg方法和apply方法，能够实现分组聚合等操作。
- 分组聚合的原理如图



# 使用groupby方法拆分数据

## groupby方法的参数及其说明

➤ 该方法提供的是分组聚合步骤中的**拆分功能**，能根据索引或字段对数据进行分组。其常用参数与使用格式如下。

```
DataFrame.groupby(by=None, axis=0, level=None, as_index=True, sort=True, group_keys=True, squeeze=False, **kwargs)
```

参数名称	说明
by	接收list，string，mapping或generator。用于确定进行分组的依据。无默认。 <a href="#">特殊说明</a>
axis	接收int。表示操作的轴向，默认对列进行操作。默认为0。
level	接收int或者索引名。代表标签所在级别。默认为None。
as_index	接收boolean。表示聚合后的聚合标签是否以DataFrame索引形式输出。默认为True。
sort	接收boolean。表示是否对分组依据分组标签进行排序。默认为True。
group_keys	接收boolean。表示是否显示分组标签的名称。默认为True。
squeeze	接收boolean。表示是否在允许的情况下对返回数据进行降维。默认为False。

# 使用groupby方法拆分数据

---

## groupby方法的参数及其说明——by参数的特别说明

- 如果传入的是一个函数则对索引进行计算并分组。
- 如果传入的是一个字典或者Series则字典或者Series的值用来做分组依据。
- 如果传入一个NumPy数组则数据的元素作为分组依据。
- 如果传入的是字符串或者字符串列表则使用这些字符串所代表的字段作为分组依据。

# 使用groupby方法拆分数据

## 对菜品订单详情表依据订单编号分组

```
import pandas as pd
import numpy as np
from sqlalchemy import create_engine
engine = create_engine('mysql+pymysql://root:1234@127.0.0.1:\3306/testdb?charset=utf8')
detail = pd.read_sql_table('meal_order_detail1',con = engine)
detailGroup = detail[['order_id','counts','amounts']].groupby(by = 'order_id')
print('分组后的订单详情表为：',detailGroup)
```

```
In [5]: print('分组后的订单详情表为：',detailGroup)
分组后的订单详情表为： <pandas.core.groupby.groupby.DataFrameGroupBy object at 0x0000000009F70E80>
```

# 使用groupby方法拆分数据

## GroupBy对象常用的描述性统计方法

- 实际上分组后的数据对象GroupBy类似Series与DataFrame，是pandas提供的一种对象。
- GroupBy对象常用的描述性统计方法如下。

方法名称	说明	方法名称	说明
count	计算分组的数目，包括缺失值。	cumcount	对每个分组中组员的进行标记，0至n-1。
head	返回每组的前n个值。	size	返回每组的大小。
max	返回每组最大值。	min	返回每组最小值。
mean	返回每组的均值。	std	返回每组的标准差。
median	返回每组的中位数。	sum	返回每组的和。

# 使用groupby方法拆分数据

对菜品订单详情表分组后统计每一组的均值、标准差、中位数等

```
print('订单详情表分组后前5组每组的均值为 : \n', detailGroup.mean().head())
```

```
print('订单详情表分组后前5组每组的标准差为 : \n', detailGroup.std().head())
```

```
print('订单详情表分组后前5组每组的大小为 : ', '\n', detailGroup.size().head())
```

订单详情表分组后前5组每组的均值为：

	counts	amounts
order_id		
1002	1.0000	32.000
1003	1.2500	30.125
1004	1.0625	43.875
1008	1.0000	63.000
1011	1.0000	57.700

订单详情表分组后前5组每组的标准差为：

	counts	amounts
order_id		
1002	0.00000	16.000000
1003	0.46291	21.383822
1004	0.25000	31.195886
1008	0.00000	64.880660
1011	0.00000	50.077828

订单详情表分组后前5组每组的大小为：

	order_id
1002	7
1003	8
1004	16
1008	5
1011	10
	dtype: int64



# 使用agg方法聚合数据

## agg和aggregate函数参数及其说明

- agg , aggregate方法都支持对每个分组应用某函数，包括Python内置函数或自定义函数。同时这两个方法能够也能够直接对DataFrame进行函数应用操作。
- 在正常使用过程中，agg函数和aggregate函数对DataFrame对象操作时功能几乎完全相同，因此只需要掌握其中一个函数即可。它们的参数说明如下表。

*DataFrame.agg(func, axis=0, \*args, \*\*kwargs)*

*DataFrame.aggregate(func, axis=0, \*args, \*\*kwargs)*

参数名称	说明
func	接收list、dict、function。表示应用于每行 / 每列的函数。无默认。
axis	接收0或1。代表操作的轴向。默认为0。

# 使用agg方法聚合数据

## agg方法求统计量（对DataFrame数据）

- 可以使用agg方法一次求出当前数据中所有菜品销量和售价的总和与均值，如  
`detail[['counts','amounts']].agg([np.sum,np.mean])`
- 对于某个字段希望只做求均值操作，而对另一个字段则希望只做求和操作，可以使用字典的方式，将两个字段名分别作为key，然后将NumPy库的求和与求均值的函数分别作为value，如  
`detail.agg({'counts':np.sum,'amounts':np.mean})`
- 在某些时候还希望求出某个字段的多个统计量，某些字段则只要求一个统计量，此时只需要将字典对应key的value变为列表，列表元素为多个目标的统计量即可，如  
`detail.agg({'counts':np.sum,'amounts':[np.mean,np.sum]})`

# 使用agg方法聚合数据

## agg方法求统计量（对DataFrame数据）

`detail[['counts','amounts']].agg([np.sum,np.mean])`

订单详情表的菜品销量与售价的和与均值为：

	counts	amounts
sum	3087.000000	125833.000000
mean	1.111231	45.296256

`detail.agg({'counts':np.sum,'amounts':np.mean})`

订单详情表的菜品销量总和与售价的均值为：

counts	3087.000000
amounts	45.296256
dtype:	float64

`detail.agg({'counts':np.sum,'amounts':[np.mean,np.sum]})`

菜品订单详情表的菜品销量总和与售价的总和与均值为：

	counts	amounts
mean	NaN	45.296256
sum	3087.0	125833.000000

# 使用agg方法聚合数据

## agg方法与自定义的函数（对DataFrame数据）

- 在agg方法可传入读者自定义的函数。

**##自定义函数求两倍的和**

**def DoubleSum(data):**

**s = data.sum()\*2**

**return s**

**print('菜品订单详情表的菜品销量两倍总和为：','\n', detail.agg({'counts':DoubleSum},axis = 0))**

```
....: print('菜品订单详情表的菜品销量两倍总和为：','\n',
....:       detail.agg({'counts':DoubleSum},axis = 0))
菜品订单详情表的菜品销量两倍总和为：
counts      6174.0
dtype: float64
```

# 使用agg方法聚合数据

## agg方法与自定义的函数（对DataFrame数据）

- 使用自定义函数需要注意的是NumPy库中的函数np.mean，np.median，np.prod，np.sum，np.std，np.var能够在agg中直接使用，但是在自定义函数中使用NumPy库中的这些函数，如果计算的时候是单个序列则会无法得出想要的结果，如果是多列数据同时计算则不会出现这种问题。

##自定义函数求两倍的和

def DoubleSum1(data):

    s = np.sum(data)\*2

    return s

```
In [13]: print('订单详情表的菜品销量两倍总和为: \n',
...:         detail.agg({'counts':DoubleSum1},axis = 0).head())
订单详情表的菜品销量两倍总和为:
   counts
0      2.0
1      2.0
2      2.0
3      2.0
4      2.0
```



# 使用agg方法聚合数据

## agg方法与自定义的函数（对DataFrame数据）

- 使用自定义函数需要注意的是NumPy库中的函数np.mean，np.median，np.prod，np.sum，np.std，np.var能够在agg中直接使用，但是在自定义函数中使用NumPy库中的这些函数，如果计算的时候是单个序列则会无法得出想要的结果，如果是多列数据同时计算则不会出现这种问题。

##自定义函数求两倍的和

def DoubleSum1(data):

    s = np.sum(data)\*2

    return s

```
In [13]: print('订单详情表的菜品销量两倍总和为: \n',  
....:         detail.agg({'counts':DoubleSum1},axis = 0).head())  
订单详情表的菜品销量两倍总和为:
```

```
counts  
0      2.0  
1      2.0
```



```
In [16]: print('订单详情表的菜品销量与售价的和的两倍为: \n',  
....:         detail[['counts','amounts']].agg(DoubleSum1))  
订单详情表的菜品销量与售价的和的两倍为:
```

```
counts      6174.0  
amounts    251666.0  
dtype: float64
```





# 使用agg方法聚合数据

## agg方法与自定义的函数（对GroupBy数据）

- 使用agg方法能够实现对每一个字段每一组使用相同的函数。

```
print('订单详情表分组后前3组每组的均值为：\n', detailGroup.agg(np.mean).head(3))
```

```
print('订单详情表分组后前3组每组的标准差为：\n', detailGroup.agg(np.std).head(3))
```

```
In [19]: print('订单详情表分组后前3组每组的均值为：\n',  
....:         detailGroup.agg(np.mean).head(3))  
订单详情表分组后前3组每组的均值为：
```

	counts	amounts
order_id		
1002	1.0000	32.000
1003	1.2500	30.125
1004	1.0625	43.875

```
In [20]: print('订单详情表分组后前3组每组的标准差为：\n',  
....:         detailGroup.agg(np.std).head(3))  
订单详情表分组后前3组每组的标准差为：
```

	counts	amounts
order_id		
1002	0.00000	16.000000
1003	0.46291	21.383822
1004	0.25000	31.195886

# 使用agg方法聚合数据

## agg方法与自定义的函数（对GroupBy数据）

- 如果需要对不同的字段应用不同的函数，则可以和DataFrame中使用agg方法相同。

```
print('订单详情分组前3组每组菜品总数和售价均值为：\n',  
      detailGroup.agg({'counts':np.sum, 'amounts':np.mean}).head(3))
```

```
In [21]: print('订单详情分组前3组每组菜品总数和售价均值为：\n',  
...:         detailGroup.agg({'counts':np.sum, 'amounts':np.mean}).head(3))  
订单详情分组前3组每组菜品总数和售价均值为：  


| order_id | counts | amounts |
|----------|--------|---------|
| 1002     | 7.0    | 32.000  |
| 1003     | 10.0   | 30.125  |
| 1004     | 17.0   | 43.875  |


```



## 使用apply方法聚合数据

- apply方法类似agg方法，能够将函数应用于每一列。**不同之处**在于apply方法传入的函数只能够作用于整个DataFrame或者Series，而无法像agg一样能够对不同字段，应用不同函数获取不同结果。
- *DataFrame.apply(func, axis=0, broadcast=False, raw=False, reduce=None, args=(), \*\*kws)*

参数名称	说明
func	接收functions。表示应用于每行 / 列的函数。无默认。
axis	接收0或1。代表操作的轴向。默认为0。
broadcast	接收boolearn。表示是否进行广播。默认为False。
raw	接收boolearn。表示是否直接将ndarray对象传递给函数。默认为False。
reduce	接收boolearn或者None。表示返回值的格式。默认None。

## 使用apply方法聚合数据

- apply方法类似agg方法，能够将函数应用于每一列。**不同之处**在于apply方法传入的函数只能够作用于整个DataFrame或者Series，而无法像agg一样能够对不同字段，应用不同函数获取不同结果。

`print('订单详情表的菜品销量与售价的均值为：\n', detail[['counts','amounts']].apply(np.mean))`

```
In [22]: print('订单详情表的菜品销量与售价的均值为：\n',
...:         detail[['counts','amounts']].apply(np.mean))
订单详情表的菜品销量与售价的均值为：
counts      1.111231
amounts     45.296256
dtype: float64
```

## 使用apply方法聚合数据

- 使用apply方法对GroupBy对象进行聚合操作其方法和agg方法也相同，只是使用agg方法能够实现对不同的字段进行应用不同的函数，而apply则不行。

```
print('订单详情表分组后前3组每组的均值为：','\n', detailGroup.apply(np.mean).head(3))
```

```
print('订单详情表分组后前3组每组的标准差为：','\n', detailGroup.apply(np.std).head(3))
```

```
In [23]: print('订单详情表分组后前3组每组的均值为：','\n', detailGroup.apply(np.mean).head(3))
...: print('订单详情表分组后前3组每组的标准差为：','\n', detailGroup.apply(np.std).head(3))
```

订单详情表分组后前3组每组的均值为：

	order_id	counts	amounts
order_id			
1002	1.431572e+26	1.0000	32.000
1003	1.253875e+30	1.2500	30.125
1004	6.275628e+61	1.0625	43.875

订单详情表分组后前3组每组的标准差为：

	counts	amounts
order_id		
1002	0.000000	14.813122
1003	0.433013	20.002734
1004	0.242061	30.205287

## 使用transform方法聚合数据

- transform方法能够对整个DataFrame的所有元素进行操作。且transform方法只有一个参数“func”，表示对DataFrame操作的函数。

`print('订单详情表的菜品销量与售价的两倍为：\n',`

`detail[['counts','amounts']].transform(lambda x:x*2).head(4))`

```
In [25]: print('订单详情表的菜品销量与售价的两倍为：\n',  
...:         detail[['counts','amounts']].transform(lambda x:x*2).head(4))
```

订单详情表的菜品销量与售价的两倍为：

	counts	amounts
0	2.0	98.0
1	2.0	96.0
2	2.0	60.0
3	2.0	50.0

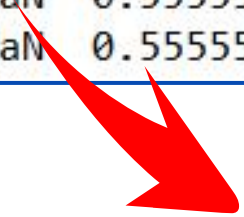
## 使用transform方法聚合数据

- 同时transform方法还能够对DataFrame分组后的对象GroupBy进行操作，可以实现组内离差标准化等操作。

`print('订单详情表分组后实现组内离差标准化后的前五行为：\n',`

`detailGroup.transform(lambda x:(x.mean()-x.min())/(x.max()-x.min())).head())`

```
In [26]: print('订单详情表分组后实现组内离差标准化后的前五行为：\n',
....:         detailGroup.transform(lambda x:(x.mean()-x.min())/(x.max()-x.min())).head())
__main__:2: RuntimeWarning: invalid value encountered in double_scalars
订单详情表分组后实现组内离差标准化后的前五行为：
   counts  amounts
0      NaN  0.555556
1      NaN  0.555556
2      NaN  0.555556
3      NaN  0.555556
4      NaN  0.555556
```



若在计算离差标准化的时候结果中有NaN，这是由于根据离差标准化公式，最大值和最小值相同的情况下分母是0。而分母为0的数在Python中表示为NaN。

# 任务实现

---

## 1.按照时间对菜品订单详情表进行拆分

- 通过分组聚合的方式能够将每天的数据放在一个组内，从而可以方便地对每一个组的内容进行分析。

```
import pandas as pd
import numpy as np
from sqlalchemy import create_engine
engine = create_engine('mysql+pymysql://root:1234@127.0.0.1:3306/testdb?charset=utf8')
detail = pd.read_sql_table('meal_order_detail1',con = engine)
detail['place_order_time'] = pd.to_datetime(detail['place_order_time'])
detail['date'] = [i.date() for i in detail['place_order_time']]
detailGroup = detail[['date','counts','amounts']].groupby(by='date')
print('订单详情表前5组每组的数目为：\n',detailGroup.size().head())
```

# 任务实现

## 1.按照时间对菜品订单详情表进行拆分

- 通过分组聚合的方式能够将每天的数据放在一个组内，从而可以方便地对每一个组的内容进行分析。

```
import pandas as pd
```

```
import numpy as np
```

```
from sqlalchemy import create_engine
```

```
engine = create_engine('mysql+pymysql://root:1234@127.0.0.1:3306/testdb?charset=utf8')
```

```
detail = pd.read_sql_table('meal_order_detail1',con = engine)
```

```
detail['place_order_time'] = pd.to_datetime(detail['place_order_time'])
```

```
detail['date'] = [i.date() for i in detail['place_order_time']]
```

```
detailGroup = detail[['date','counts','amounts']].groupby(by='date')
```

```
print('订单详情表前5组每组的数目为：\n',detailGroup.size().head())
```

订单详情表前5组每组的数目为：

date	
2016-08-01	217
2016-08-02	138
2016-08-03	157
2016-08-04	144
2016-08-05	193
dtype:	int64



# 任务实现

## 2.使用agg方法计算单日菜品销售的平均单价和售价中位数

- 对已经拆分完成的订单详情进行聚合，可以得出每组的销售均值和售价中位数等信息。

```
dayMean = detailGroup.agg({'amounts':np.mean})
```

```
print('订单详情表前五组每日菜品均价为：\n',dayMean.head())
```

```
dayMedian = detailGroup.agg({'amounts':np.median})
```

```
print('订单详情表前五组每日菜品售价中位数为：\n',dayMedian.head())
```

订单详情表前五组每日菜品均价为：

	amounts
date	
2016-08-01	43.161290
2016-08-02	44.384058
2016-08-03	43.885350
2016-08-04	52.423611
2016-08-05	44.927461

订单详情表前五组每日菜品售价中位数为：

	amounts
date	
2016-08-01	33.0
2016-08-02	35.0
2016-08-03	38.0
2016-08-04	39.0
2016-08-05	37.0



# 任务实现

## 2.使用agg方法计算单日菜品销售的平均单价和售价中位数

- 对已经拆分完成的订单详情进行聚合，可以得出每组的销售均值和售价中位数等信息。

```
print('订单详情表前五组每日菜品售价均价和中位数为：\n',  
      detailGroup['amounts'].agg([np.mean,np.median]).head())
```

```
In [34]: print('订单详情表前五组每日菜品售价均价和中位数为：\n',  
      ...:      detailGroup['amounts'].agg([np.mean,np.median]).head())  
订单详情表前五组每日菜品售价均价和中位数为：  
              mean  median  
date  
2016-08-01  43.161290    33.0  
2016-08-02  44.384058    35.0  
2016-08-03  43.885350    38.0  
2016-08-04  52.423611    39.0  
2016-08-05  44.927461    37.0
```

# 任务实现

## 2.使用agg方法计算单日菜品销售的平均单价和售价中位数

- 对已经拆分完成的订单详情进行聚合，可以得出每组的销售均值和售价中位数等信息。

```
print('订单详情表前五组每日菜品售价均价和中位数为：\n',  
      detailGroup.agg({'amounts':[np.mean,np.median]}).head())
```

```
In [35]: print('订单详情表前五组每日菜品售价均价和中位数为：\n',  
      ...:      detailGroup.agg({'amounts':[np.mean,np.median]}).head())  
订单详情表前五组每日菜品售价均价和中位数为：  
      amounts  
      mean median  
date  
2016-08-01  43.161290    33.0  
2016-08-02  44.384058    35.0  
2016-08-03  43.885350    38.0  
2016-08-04  52.423611    39.0  
2016-08-05  44.927461    37.0
```

# 任务实现

## 3.使用apply方法统计单日菜品销售数目

- 除了可以对售价进行计算外，还可以计算单日总共销售的菜品数目。

```
daySaleSum = detailGroup.apply(np.sum)['counts']
```

```
print('订单详情表前五组每日菜品售出数目为：\n',daySaleSum.head())
```

```
In [38]: daySaleSum = detailGroup.apply(np.sum)['counts']
...: print('订单详情表前五组每日菜品售出数目为：\n',daySaleSum.head())
订单详情表前五组每日菜品售出数目为：
  date
2016-08-01    233.0
2016-08-02    151.0
2016-08-03    192.0
2016-08-04    169.0
2016-08-05    224.0
Name: counts, dtype: float64
```

# 任务实现

## 3.使用apply方法统计单日菜品销售数目

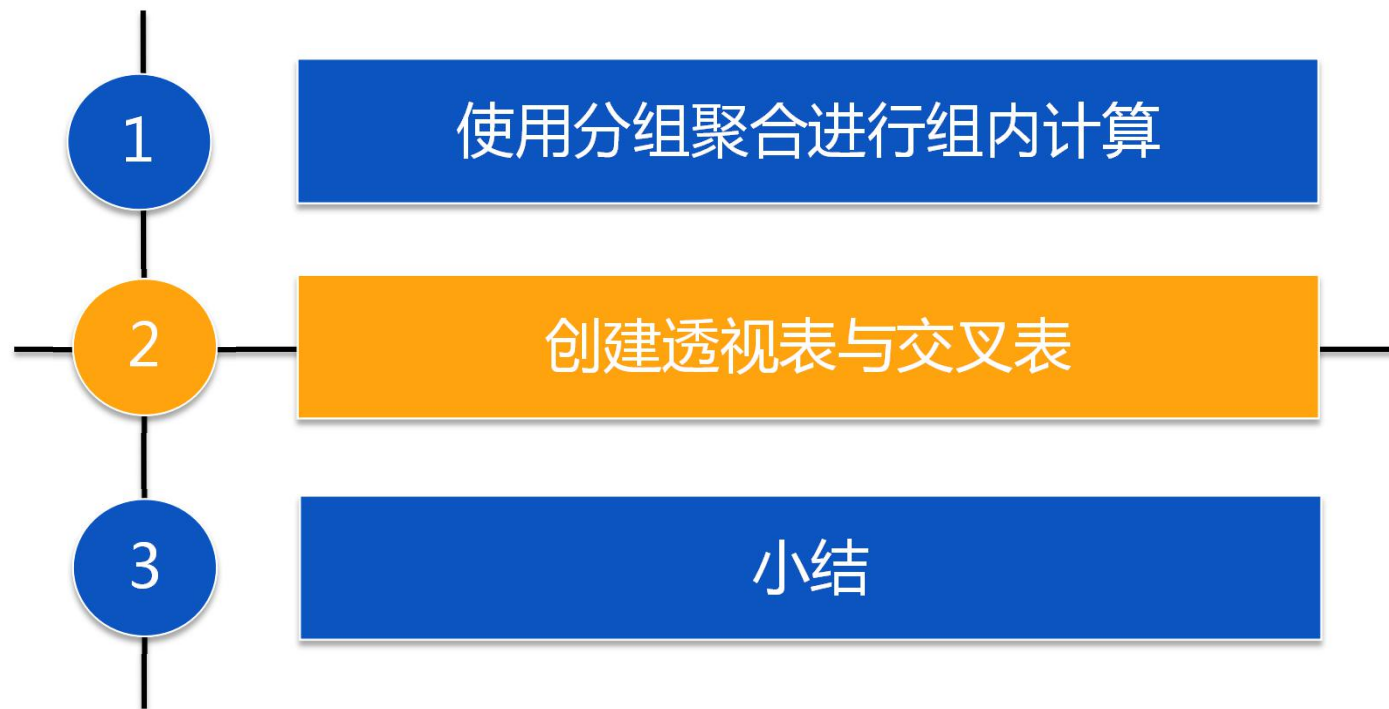
- 除了可以对售价进行计算外，还可以计算单日总共销售的菜品数目。

```
print('订单详情表前五组每日菜品售出数目为：\n', detailGroup['counts'].apply(np.sum).head())
```

```
In [37]: print('订单详情表前五组每日菜品售出数目为：\n', detailGroup['counts'].apply(np.sum).head())
订单详情表前五组每日菜品售出数目为：
date
2016-08-01    233.0
2016-08-02    151.0
2016-08-03    192.0
2016-08-04    169.0
2016-08-05    224.0
Name: counts, dtype: float64
```

# 目录

---



# 创建透视表与交叉表

---

## 任务描述

- 数据透视表是数据分析中常见的工具之一，根据一个或多个键值对数据进行聚合，根据行或者列的分组将数据划分到各个区域。
- 在pandas中，除了可以使用groupby对数据分组聚合实现透视功能外，还提供了更为简单的方法。
- 本小节以T餐饮企业的菜品订单数据为例制作透视表和交叉表，分析不同菜品的销售和金额之间的关系。

## 任务分析

- ( 1 ) 使用pivot\_table函数制作菜品日销售透视表
- ( 2 ) 使用crosstab函数制作菜品销售的交叉表

# 使用povit\_table函数创建透视表

## pivot\_table函数常用参数及其说明

➤ 利用pivot\_table函数可以实现透视表，pivot\_table()函数的常用参数及其使用格式如下。

```
pands.pivot_table(data, values=None, index=None, columns=None, aggfunc='mean', fill_value=None, margins=False, dropna=True, margins_name='All')
```

参数名称	说明
data	接收DataFrame。表示创建表的数据。无默认。
values	接收字符串。用于指定想要聚合的数据字段名，默认使用全部数据。默认为None。
index	接收string或list。表示行分组键。默认为None。
columns	接收string或list。表示列分组键。默认为None。
aggfunc	接收functions。表示聚合函数。默认为mean。
margins	接收boolearn。表示汇总（Total）功能的开关，设为True后结果集中会出现名为“ALL”的行和列。默认为True。
dropna	接收boolearn。表示是否删掉全为NaN的列。默认为False。



# 使用pivot\_table函数创建透视表

## 使用订单号作为透视表索引制作透视表

```
import pandas as pd
import numpy as np
from sqlalchemy import create_engine
engine = create_engine('mysql+pymysql://root:1234@\127.0.0.1:3306/testdb?charset=utf8')
detail = pd.read_sql_table('meal_order_detail1', con = engine)
detailPivot = pd.pivot_table(detail[['order_id', 'counts', 'amounts']], index = 'order_id')
print('以order_id作为分组键创建的订单透视表为：\n', detailPivot.head())
```

以order\_id作为分组键创建的订单透视表为：

order_id	amounts	counts
1002	32.000	1.0000
1003	30.125	1.2500
1004	43.875	1.0625
1008	63.000	1.0000
1011	57.700	1.0000



# 使用pivot\_table函数创建透视表

## pivot\_table函数主要的参数调节

- 在不特殊指定聚合函数aggfunc时，会默认使用numpy.mean进行聚合运算，numpy.mean会自动过滤掉非数值类型数据。

```
In [42]: detailPivot = pd.pivot_table(detail[['order_id', 'counts', 'amounts', 'dishes_name']],  
....:      index = 'order_id')  
....: print('以order_id作为分组键创建的订单透视表为: \n', detailPivot.head())
```

以order\_id作为分组键创建的订单透视表为：

order_id	amounts	counts
1002	32.000	1.0000
1003	30.125	1.2500
1004	43.875	1.0625
1008	63.000	1.0000
1011	57.700	1.0000

# 使用pivot\_table函数创建透视表

## pivot\_table函数主要的参数调节

可以通过指定aggfunc参数修改聚合函数。

```
detailPivot1 = pd.pivot_table(detail[['order_id','counts','amounts']],
```

```
    index = 'order_id',aggfunc = np.sum)
```

```
print('以order_id作为分组键创建的订单销量与售价总和透视表为：\n', detailPivot1.head())
```

```
In [44]: detailPivot1 = pd.pivot_table(detail[['order_id','counts','amounts']],
....:     index = 'order_id',aggfunc = np.sum)
....: print('以order_id作为分组键创建的订单销量与售价总和透视表为：\n',
....:     detailPivot1.head())
```

以order\_id作为分组键创建的订单销量与售价总和透视表为：

order_id	amounts	counts
1002	224.0	7.0
1003	241.0	10.0
1004	702.0	17.0
1008	315.0	5.0
1011	577.0	10.0

# 使用pivot\_table函数创建透视表

## pivot\_table函数主要的参数调节

- 和groupby方法分组的时候相同，pivot\_table函数在创建透视表的时候分组键index可以有多个。

```
detailPivot2 = pd.pivot_table(detail[['order_id','dishes_name','counts','amounts']],  
                                index = ['order_id','dishes_name'],    aggfunc = np.sum)  
print('以order_id和dishes_name作为分组键创建的订单销量与售价总和透视表为：\n',  
      detailPivot2.head())
```

以order\_id和dishes\_name作为分组键创建的订单销量与售价总和透视表为：

order_id	dishes_name	amounts	counts
1002	凉拌菠菜	27.0	1.0
	南瓜枸杞小饼干	19.0	1.0
	焖猪手	58.0	1.0
	独家薄荷鲜虾牛肉卷	45.0	1.0
	白胡椒胡萝卜羊肉汤	35.0	1.0

# 使用pivot\_table函数创建透视表

## pivot\_table函数主要的参数调节

- 和groupby方法分组的时候相同，pivot\_table函数在创建透视表的时候分组键index可以有多个。

```
detailPivot2 = pd.pivot_table(detail[:],  
                                index = ['order_id','dishes_name'],    aggfunc = np.sum)  
print('以order_id和dishes_name作为分组键创建的订单销量与售价总和透视表')  
detailPivot2.head()
```

以order\_id和dishes\_name作为分组键创建的订单销量与售价总和透视表为：

order_id	dishes_name	amounts	counts
1002	凉拌菠菜	27.0	1.0
	南瓜枸杞小饼干	19.0	1.0
	焖猪手	58.0	1.0
	独家薄荷鲜虾牛肉卷	45.0	1.0
	白胡椒胡萝卜羊肉汤	35.0	1.0

```
In [12]: detail.dtypes  
Out[12]:  
detail_id      object  
order_id       object  
dishes_id      object  
logicprn_name  object  
parent_class_name  object  
dishes_name    object  
itemis_add     object  
counts         float64  
amounts        float64  
cost           object  
place_order_time  datetime64[ns]  
discount_amt     object  
discount_reason  object  
kick_back       object  
add_inprice     object  
add_info        object  
bar_code        object  
picture_file    object  
emp_id          object  
dtype: object
```

# 使用pivot\_table函数创建透视表

## pivot\_table函数主要的参数调节

- 通过设置columns参数可以指定列分组

```
detailPivot2 = pd.pivot_table(detail[['order_id','dishes_name','counts','amounts']],  
                               index = 'order_id', columns = 'dishes_name', aggfunc = np.sum)  
print('以order_id和dishes_name作为行列分组键创建的透视表前5行4列为：\n', detailPivot2.iloc[:5,:4])
```

```
以order_id和dishes_name作为行列分组键创建的透视表前5行4列为：  
amounts  
dishes_name  42度海之蓝  北冰洋汽水  38度剑南春  50度古井贡酒  
order_id  
1002         NaN      NaN      NaN      NaN  
1003         NaN      NaN      NaN      NaN  
1004         NaN      NaN      NaN      NaN  
1008         NaN      NaN      NaN      NaN  
1011        99.0      NaN      NaN      NaN
```



# 使用pivot\_table函数创建透视表

## pivot\_table函数主要的参数调节

- 当全部数据列数很多时，若只想要显示某列，可以通过指定values参数来实现。

```
detailPivot4 = pd.pivot_table(detail[['order_id','dishes_name','counts','amounts']],  
                               index = 'order_id', values = 'counts', aggfunc = np.sum)
```

```
print('以order_id作为行分组键counts作为值创建的透视表前5行为：\n',detailPivot4.head())
```

```
In [14]: detailPivot4 = pd.pivot_table(detail[['order_id','dishes_name','counts','amounts']],  
....:      index = 'order_id', values = 'counts', aggfunc = np.sum)  
....: print('以order_id作为行分组键counts作为值创建的透视表前5行为：\n',detailPivot4.head())
```

以order\_id作为行分组键counts作为值创建的透视表前5行为：

order_id	counts
1002	7.0
1003	10.0
1004	17.0
1008	5.0
1011	10.0

# 使用pivot\_table函数创建透视表

## pivot\_table函数主要的参数调节

- 当某些数据不存在时，会自动填充NaN，因此可以指定fill\_value参数，表示当存在缺失值时，以指定数值进行填充。

```
detailPivot5 = pd.pivot_table(detail[['order_id','dishes_name','counts','amounts']],  
                                index = 'order_id', columns = 'dishes_name', aggfunc = np.sum,fill_value = 0)  
print('空值填0后以order_id和dishes_name为行列分组键创建透视表前5行4列为：\n',  
      detailPivot5.iloc[:5,:4])
```

空值填0后以order\_id和dishes\_name为行列分组键创建透视表前5行4列为：

	amounts			
dishes_name	42度海之蓝	北冰洋汽水	38度剑南春	50度古井贡酒
order_id				
1002	0	0	0	0
1003	0	0	0	0
1004	0	0	0	0
1008	0	0	0	0
1011	99	0	0	0

# 使用pivot\_table函数创建透视表

## pivot\_table函数主要的参数调节

- 可以更改margins参数，查看汇总数据。

```
detailPivot6 = pd.pivot_table(detail[['order_id','dishes_name','counts','amounts']],  
                                index = 'order_id',columns = 'dishes_name',aggfunc = np.sum,fill_value = 0,  
                                margins = True)  
print('添加margins后以order_id和dishes_name为分组键的透视表前5行后4列为：\n',  
      detailPivot6.iloc[:5,-4:])
```

添加margins后以order\_id和dishes\_name为分组键的透视表前5行后4列为：

	counts			
dishes_name	黄油曲奇饼干	黄花菜炒木耳	黑米恋上葡萄	All
order_id				
1002	0	0	0	7.0
1003	0	0	0	10.0
1004	0	1	0	17.0
1008	0	0	0	5.0
1011	0	0	0	10.0



# 使用crosstab函数创建交叉表

---

## crosstab函数

- 交叉表是一种特殊的透视表，主要用于计算分组频率。利用pandas提供的crosstab函数可以制作交叉表，crosstab函数的常用参数和使用格式如下。
- *pandas.crosstab(index, columns, values=None, rownames=None, colnames=None, aggfunc=None, margins=False, dropna=True, normalize=False)*

# 使用crosstab函数创建交叉表

## crosstab的常用参数及其说明

参数名称	说明
index	接收string或list。表示行索引键。无默认。
columns	接收string或list。表示列索引键。无默认。
values	接收array。表示聚合数据。默认为None。
aggfunc	接收function。表示聚合函数。默认为None。
rownames	表示行分组键名。无默认。
colnames	表示列分组键名。无默认。
dropna	接收boolearn。表示是否删掉全为NaN的。默认为False。
margins	接收boolearn。默认为True。汇总（Total）功能的开关，设为True后结果集中会出现名为“ALL”的行和列。
normalize	接收boolearn。表示是否对值进行标准化。默认为False。

# 使用crosstab函数创建交叉表

## crosstab函数

- 由于交叉表是透视表的一种，其参数基本保持一致，不同之处在于crosstab函数中的index，columns，values填入的都是对应的从Dataframe中取出的某一列。

```
detailCross = pd.crosstab(index=detail['order_id'],columns=detail['dishes_name'],  
                           values = detail['counts'], aggfunc = np.sum)  
print('以order_id和dishes_name为分组键counts为值的透视表前5行5列为：\n',  
      detailCross.iloc[:5,:5])
```

以order\_id和dishes\_name为分组键counts为值的透视表前5行5列为：

dishes_name	42度海之蓝	北冰洋汽水	38度剑南春	50度古井贡酒	52度泸州老窖
order_id					
1002	NaN	NaN	NaN	NaN	NaN
1003	NaN	NaN	NaN	NaN	NaN
1004	NaN	NaN	NaN	NaN	NaN
1008	NaN	NaN	NaN	NaN	NaN
1011	1.0	NaN	NaN	NaN	NaN

# 任务实现

## 1.创建单日菜品成交总额与总数均价透视表

- 单日菜品的成交总额以及菜品成交总数对于营业状况分析、利润分析具有重要作用，使用透视表能够轻松实现这一操作。

```
import pandas as pd
import numpy as np
from sqlalchemy import create_engine
engine = create_engine('mysql+pymysql://root:1234@127.0.0.1:3306/testdb?charset=utf8')
detail = pd.read_sql_table('meal_order_detail1',con = engine)
detail['place_order_time'] = pd.to_datetime(detail['place_order_time'])
detail['date'] = [i.date() for i in detail['place_order_time']]
PivotDetail = pd.pivot_table(detail[['date','dishes_name','counts','amounts']],
    index = 'date',aggfunc = np.sum, margins = True)
print('订单详情表单日菜品成交总额与总数透视表前5行5列为：\n', PivotDetail.head())
```

# 任务实现

## 1.创建单日菜品成交总额与总数均价透视表

- 单日菜品的成交总额以及菜品成交总数对于营业状况分析、利润分析具有重要作用，使用透视表能够轻松实现这一操作。

订单详情表单日菜品成交总额与总数透视表前5行5列为：

	amounts	counts
date		
2016-08-01	9366.0	233.0
2016-08-02	6125.0	151.0
2016-08-03	6890.0	192.0
2016-08-04	7549.0	169.0
2016-08-05	8671.0	224.0

# 任务实现

## 2.创建单个菜品单日成交总额透视表

- 单个菜品单日的成交总额对后续菜品价格调整，以及菜品种类调整具有十分重要的作用，使用透视表和交叉表能够轻松实现。

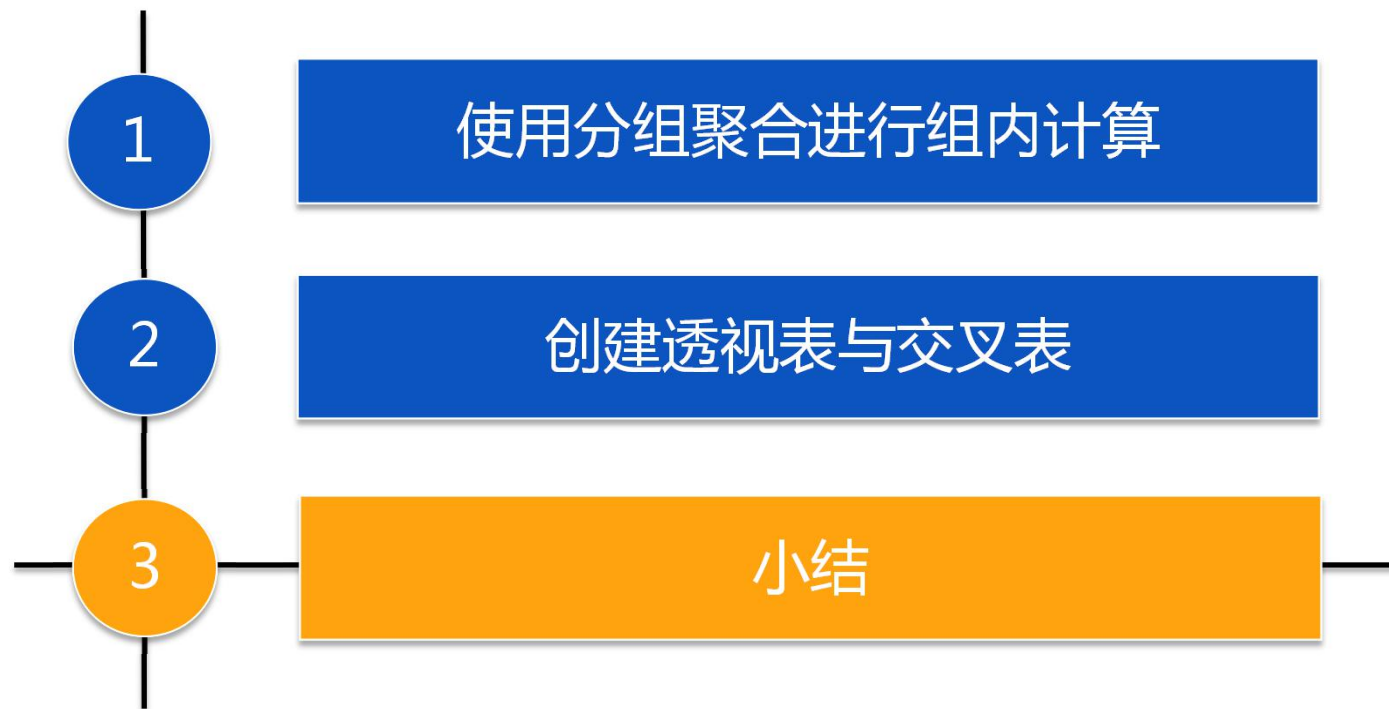
```
CrossDetail = pd.crosstab( index=detail['date'], columns=detail['dishes_name'],
                           values = detail['amounts'], aggfunc = np.sum, margins = True)
print('订单详情表单日单个菜品成交总额交叉表后5行5列为：\n', CrossDetail.iloc[-5:,-5:])
```

订单详情表单日单个菜品成交总额交叉表后5行5列为：

dishes_name	黄尾袋鼠西拉子红葡萄酒	黄油曲奇饼干	黄花菜炒木耳	黑米恋上葡萄	All
date					
2016-08-07	230.0	32.0	105.0	99.0	31306.0
2016-08-08	46.0	NaN	NaN	33.0	6532.0
2016-08-09	138.0	NaN	35.0	99.0	7155.0
2016-08-10	46.0	NaN	70.0	33.0	10231.0
All	736.0	80.0	525.0	561.0	125833.0

# 目录

---





# 小结

---

本章以餐饮数据为例

- 介绍了数据库数据，csv数据，Excel数据三种常用的数据读取与写入方式。
- 阐述了DataFrame的常用属性，方法与描述性统计相关内容。
- 介绍了时间数据的转换，信息提取与算术运算。
- 剖析了分组聚合方法groupby的原理，用法和三种聚合方法。
- 展现了透视表与交叉表的制作方法。

通过本章的学习，读者能够对pandas库有一个整体了解并能够利用pandas库进行基础的统计。



大数据，成就未来

# Thank you!

