10 数据聚合与分组操作 Data Aggregation and Group Operations

In [94]:
```
import pandas as pd
import numpy as np
```

10.1 Groupby机制 How to Think About Group Operations

In [9]:
```
df = pd.DataFrame({'key1': ['a', 'a', 'b', 'b', 'a'],
                   'key2': ['one', 'two', 'one', 'two', 'one'],
                   'data1': np.random.randn(5),
                   'data2': np.random.randn(5)})
df
#   key1    key2    data1   data2
# 0 a   one 0.846229    -0.583607
# 1 a   two 0.404077    -1.502867
# 2 b   one 0.190257    -0.698330
# 3 b   two 0.742905    0.356835
# 4 a   one -0.053838   -0.467249

#假设想要根据key1标签计算data1列的均值，有多种方法可以实现。
#其中一种是访问data1并使用key1列（它是一个series）调用groupby方法:
#grouped = df['data'].groupby(df['key'])
#group.mean()

grouped = df['data1'].groupby(df['key1'])                    #使用data1数据，gr
grouped
grouped.mean()
# key1
# a   -0.065231
# b   -0.699313
# Name: data1, dtype: float64
```

Out[9]:
```
key1
a   -0.275317
b   -0.193459
Name: data1, dtype: float64
```

In [12]:
```
#将多个数组作为列表传入：
means = df['data1'].groupby([df['key1'], df['key2']]).mean()
means
# key1  key2
# a     one      0.061026
#       two     -0.948005
# b     one      0.006377
#       two     -0.393294
# Name: data1, dtype: float64

#使用两个键对数据进行分组，结果series现在拥有一个包含唯一键对的多层索引：
#即把means的显示，按更好看的方式排列了：
means.unstack()
# key2  one two
# key1
# a 0.061026    -0.948005
# b 0.006377    -0.393294
```

Out[12]:
```
key1  key2
a     one      0.061026
      two     -0.948005
b     one      0.006377
      two     -0.393294
Name: data1, dtype: float64
```

In [16]:
```
#在这个例子中，分组键都是Series，尽管分组键，也可以是正确长度的任何数组：

states = np.array(['Ohio', 'California', 'California', 'Ohio', 'Ohio'])
years = np.array([2005, 2005, 2006, 2005, 2006])
df['data1'].groupby([states, years]).mean()
# California  2005   -0.948005
#             2006    0.006377
# Ohio        2005   -0.535722
#             2006    0.800203
# Name: data1, dtype: float64
```

Out[16]:
```
California  2005   -0.948005
            2006    0.006377
Ohio        2005   -0.535722
            2006    0.800203
Name: data1, dtype: float64
```

In [19]:
```
#分组信息作为想要继续处理的数据，通常包含在同一个Data frame中，这种情况下，可以传递列

df.groupby('key1').mean()
# data1 data2
# key1
# a -0.275317   0.338717
# b -0.193459   0.328065

df.groupby(['key1', 'key2']).mean()
# data1 data2
# key1  key2
# a one 0.061026    -0.317894
# two   -0.948005   1.651938
# b one 0.006377    1.494016
# two   -0.393294   -0.837886
```

```
C:\Users\miran\AppData\Local\Temp\ipykernel_14572\4100888075.py:3: FutureWarn
ing: The default value of numeric_only in DataFrameGroupBy.mean is deprecate
d. In a future version, numeric_only will default to False. Either specify nu
meric_only or select only columns which should be valid for the function.
  df.groupby('key1').mean()
```

Out[19]:

		data1	data2
key1	**key2**		
a	**one**	0.061026	-0.317894
	two	-0.948005	1.651938
b	**one**	0.006377	1.494016
	two	-0.393294	-0.837886

In [20]:
```
#通用group by的方法是size，size返回一个包含组大小信息的series：
.size表示每个堆里有多少个元素
df.groupby(['key1', 'key2']).size()
# key1  key2
# a     one     2
#       two     1
# b     one     1
#       two     1
# dtype: int64
```

Out[20]:
```
key1  key2
a     one     2
      two     1
b     one     1
      two     1
dtype: int64
```

10.1.1 遍历各分组 Iterating over Groups

In [30]:
```
#group by对象支持迭代，会生成一个包含组名和数据块的2维元组序列。

for name, group in df.groupby('key1'):
   print(name)
   print(group)
# a
#   key1 key2     data1     data2
# 0    a  one -0.678150 -1.172726
# 1    a  two -0.948005  1.651938
# 4    a  one  0.800203  0.536938
# b
#   key1 key2     data1     data2
# 2    b  one  0.006377  1.494016
# 3    b  two -0.393294 -0.837886

for (k1, k2), group in df.groupby(['key1', 'key2']):
    print((k1, k2))
    print((group))
# ('a', 'one')
#   key1 key2     data1     data2
# 0    a  one -0.678150 -1.172726
# 4    a  one  0.800203  0.536938
# ('a', 'two')
#   key1 key2     data1     data2
# 1    a  two -0.948005  1.651938
# ('b', 'one')
#   key1 key2     data1     data2
# 2    b  one  0.006377  1.494016
# ('b', 'two')
#   key1 key2     data1     data2
# 3    b  two -0.393294 -0.837886
```

```
a
  key1 key2     data1     data2
0    a  one -0.678150 -1.172726
1    a  two -0.948005  1.651938
4    a  one  0.800203  0.536938
b
  key1 key2     data1     data2
2    b  one  0.006377  1.494016
3    b  two -0.393294 -0.837886
('a', 'one')
  key1 key2     data1     data2
0    a  one -0.678150 -1.172726
4    a  one  0.800203  0.536938
('a', 'two')
  key1 key2     data1     data2
1    a  two -0.948005  1.651938
('b', 'one')
  key1 key2     data1     data2
2    b  one  0.006377  1.494016
('b', 'two')
  key1 key2     data1     data2
3    b  two -0.393294 -0.837886
```

In [35]:
```
#可以选择在任何一块数据上进行操作。比如计算出数据块的字典：
pieces = dict(list(df.groupby('key1')))
pieces['b']
# key1  key2    data1   data2
# 2 b   one 0.006377    1.494016
# 3 b   two -0.393294   -0.837886
pieces['a']
#   key1    key2    data1   data2
# 0 a   one -0.678150   -1.172726
# 1 a   two -0.948005   1.651938
# 4 a   one 0.800203    0.536938
```

Out[35]:

	key1	key2	data1	data2
0	a	one	-0.678150	-1.172726
1	a	two	-0.948005	1.651938
4	a	one	0.800203	0.536938

In [42]:
```
#groupby在axis = 0 轴向上分组，也可以在其他轴向上进行分组。
#eg.根据dtype堆示例df的列进行分组。即float64排一起，object排一起:
df.dtypes
# key1      object
# key2      object
# data1    float64
# data2    float64
# dtype: object

grouped = df.groupby(df.dtypes, axis = 1)
for dtype, group in grouped:
    print(dtype)
    print(group)
# float64
#       data1     data2
# 0 -0.678150 -1.172726
# 1 -0.948005  1.651938
# 2  0.006377  1.494016
# 3 -0.393294 -0.837886
# 4  0.800203  0.536938
# object
#   key1 key2
# 0    a  one
# 1    a  two
# 2    b  one
# 3    b  two
# 4    a  one
```

```
float64
      data1     data2
0 -0.678150 -1.172726
1 -0.948005  1.651938
2  0.006377  1.494016
3 -0.393294 -0.837886
4  0.800203  0.536938
object
  key1 key2
0    a  one
1    a  two
2    b  one
3    b  two
4    a  one
```

10.1.2 选择一列或所有列的子集 Selecting a Column or Subset of Columns

In [50]:
```
#将从data frame创建的groupby对象，用列名称或列名称数组进行索引时，会产生用于聚合的列

df.groupby('key1')['data1']
df['data1'].groupby(df['key1'])

df['data1'].groupby(df['key1'])
df[['data2']].groupby(df['key1'])

#对于大型数据集，只需要聚合少部分列。比如要计算data2列均值，以data frame的形式
df.groupby(['key1', 'key2'])[['data2']].mean()
# data2
# key1  key2
# a one -0.317894
# two   1.651938
# b one 1.494016
# two   -0.837886
```

Out[50]:

		data2
key1	**key2**	
a	**one**	-0.317894
	two	1.651938
b	**one**	1.494016
	two	-0.837886

In [52]:
```
#如果传递的是列表或数组，则此索引操作返回的对象是分组的data frame。
#如果只有单个列名作为标量传递，则为分组的series：

s_grouped = df.groupby(['key1','key2'])['data2']
s_grouped
s_grouped.mean()
# key1  key2
# a     one    -0.317894
#       two     1.651938
# b     one     1.494016
#       two    -0.837886
# Name: data2, dtype: float64
```

Out[52]:
```
key1  key2
a     one    -0.317894
      two     1.651938
b     one     1.494016
      two    -0.837886
Name: data2, dtype: float64
```

10.1.3 使用字典和series分组 Grouping with Dictionaries and Series

In [62]:
```
#分组信息可能会以非数组形式存在。参考data frame示例:

people = pd.DataFrame(np.random.randn(5, 5),
                      columns = ['a', 'b', 'c', 'd', 'e'],
                      index = ['Joe', 'Steve', 'Wes', 'Jim', 'Travis'])
people.iloc[2:3, [1, 2]] = np.nan                                                    #
people
# a b    c    d    e
# Joe    1.722658    0.285687    -1.050275    -0.588675    1.061261
# Steve 0.295678    1.910377    -0.552415    0.739731    0.197875
# Wes    0.307568    NaN NaN -0.636794    -1.325021
# Jim    -0.445643    -1.660168    -0.759542    2.103658    0.888480
# Travis    -0.843297    -0.709512    0.537105    1.074780    2.425960

#假设拥有各列分组对应关系，想把各列按组累加:
mapping = {'a': 'red', 'b': 'red', 'c': 'blue',
           'd': 'blue', 'e': 'red', 'f': 'orange'}

#现在根据这个字典构造传给groupby的数组，也可以直接传字典（多写了键'f'用于表名未用的分
by_column = people.groupby(mapping, axis = 1)
by_column.sum()
# blue  red
# Joe    -1.474296    1.582528
# Steve 0.688258    1.429820
# Wes    0.911392    -0.870552
# Jim    -2.155480    0.015388
# Travis    0.197836    0.771081

#Series有相同功能，可以视为固定大小的映射:
map_series = pd.Series(mapping)
map_series
# a       red
# b       red
# c      blue
# d      blue
# e       red
# f    orange
# dtype: object
people.groupby(map_series, axis = 1).count()
#   blue    red
# Joe    2    3
# Steve 2    3
# Wes    1    2
# Jim    2    3
# Travis    2    3
```

Out[62]:

	blue	red
Joe	2	3
Steve	2	3
Wes	1	2
Jim	2	3
Travis	2	3

10.1.4 使用函数分组 Grouping with Functions

In [64]:
```
#与使用字典或seires分组相比，使用函数是定义分组关系更通用的方式。
#作为分组键的函数，将会按照每个索引值调用一次，同时返回值会被用作分组名称。

people.groupby(len).sum()
# a b    c    d    e
# 3 -1.156605   -0.719340   0.257547    -0.083353   0.951709
# 5 -1.107768   0.036274    0.742980    -1.515057   -0.612376
# 6 -1.087238   0.619480    -0.191011   0.776209    -0.075077

#将函数与数组、字典或Series进行混合不困难，所有对象会在内部转换为数组：
key_list = ['one', 'one', 'one', 'two', 'two']
people.groupby([len, key_list]).min()

# a b    c    d    e
# 3 one -0.972724   -0.319155   -0.646630   0.625116    0.090274
# two   0.320395    -0.400184   0.904177    -2.007915   0.351746
# 5 one -1.107768   0.036274    0.742980    -1.515057   -0.612376
# 6 two -1.087238   0.619480    -0.191011   0.776209    -0.075077
```

Out[64]:

		a	b	c	d	e
3	**one**	-0.972724	-0.319155	-0.646630	0.625116	0.090274
	two	0.320395	-0.400184	0.904177	-2.007915	0.351746
5	**one**	-1.107768	0.036274	0.742980	-1.515057	-0.612376
6	**two**	-1.087238	0.619480	-0.191011	0.776209	-0.075077

10.1.5 根据索引层级分组 Grouping by Index Levels

In [67]:
```
# 分层索引能够在轴索引的某个层级上进行聚合:
columns = pd.MultiIndex.from_arrays([['US', 'US', 'US', 'JP', 'JP'],
                                     [1, 3, 5, 1, 1]],
                                    names = ['cty', 'tenor'])
hier_df = pd.DataFrame(np.random.randn(4,5), columns = columns)
hier_df
# cty   US  JP
# tenor 1   3   5   1   1
# 0 -0.799725   0.750988    -0.423918   0.216781    -0.087295
# 1 1.440389    1.384280    -0.668466   -1.085713   -0.202299
# 2 1.408834    -0.055834   0.593940    -1.247511   -0.694462
# 3 -0.041520   -1.494427   -0.955077   0.406244    -0.242701

#根据层级分组时，将层级数值或层级名称传递给level关键字:
hier_df.groupby(level = 'cty', axis = 1).count()
# cty   JP  US
# 0 2   3
# 1 2   3
# 2 2   3
# 3 2   3
```

Out[67]:

cty	JP	US
0	2	3
1	2	3
2	2	3
3	2	3

10.2 数据聚合 Data Aggregation

In [68]:
```
#聚合是指所有根据数组产生标量值的数据转换过程。
#之前例子已经使用了一些聚合操作，包括mean, count, min, sum等。

#优化groupby方法:
#count：分组中的非NA值数量
#sum：非NA值的累和
#mean：非NA值的均值
#median：非NA值的算数中位数
#std, var：无偏的(n-1分母)标准差和方差
#min, max：非NA值的最小值、最大值
#prod：非NA值的乘积
#first, last：非NA值的第一个和最后一个值
```

In [74]:
```
#可以使用自行制定的聚合，再调用已经在分组对象上定义好的方法。
df
#   key1    key2    data1   data2
# 0 a   one -0.678150   -1.172726
# 1 a   two -0.948005   1.651938
# 2 b   one 0.006377    1.494016
# 3 b   two -0.393294   -0.837886
# 4 a   one 0.800203    0.536938
grouped = df.groupby('key1')
grouped['data1'].quantile(0.9)
# key1
# a    0.504533
# b   -0.033590
# Name: data1, dtype: float64

#要使用自己的聚合函数，需要将函数传递给aggregate或agg方法:
def peak_to_peak(arr):
    return arr.max() - arr.min()
grouped.agg(peak_to_peak)
#   data1   data2
# key1
# a 1.748209    2.824664
# b 0.399671    2.331901

#.describe()也有效，虽然其并不是聚合函数
grouped.describe()
#   data1   data2
# count mean    std min 25% 50% 75% max count   mean    std min 25% 50% 75% ma
# key1
# a 3.0 -0.275317   0.94115 -0.948005   -0.813078   -0.678150   0.061026    0.
# b 2.0 -0.193459   0.28261 -0.393294   -0.293376   -0.193459   -0.093541   0.
```

```
C:\Users\miran\AppData\Local\Temp\ipykernel_14572\3792748813.py:19: FutureWar
ning: ['key2'] did not aggregate successfully. If any error is raised this wi
ll raise in a future version of pandas. Drop these columns/ops to avoid this
warning.
  grouped.agg(peak_to_peak)
```

Out[74]:

	data1								
	count	mean	std	min	25%	50%	75%	max	count
key1									
a	3.0	-0.275317	0.94115	-0.948005	-0.813078	-0.678150	0.061026	0.800203	3.0
b	2.0	-0.193459	0.28261	-0.393294	-0.293376	-0.193459	-0.093541	0.006377	2.0

10.2.1 逐列及多函数应用 Column-Wise and Multiple Function Application

In [96]:
```
tips = pd.read_csv('C:/Users/miran/lpthw/tips.csv')

tips['tip_pct'] = tips['tip'] / tips['total_bill']
tips[:6]
# total_bill    tip smoker  day time    size    tip_pct
# 0 16.99   1.01    No  Sun Dinner  2   0.059447
# 1 10.34   1.66    No  Sun Dinner  3   0.160542
# 2 21.01   3.50    No  Sun Dinner  3   0.166587
# 3 23.68   3.31    No  Sun Dinner  2   0.139780
# 4 24.59   3.61    No  Sun Dinner  4   0.146808
# 5 25.29   4.71    No  Sun Dinner  4   0.186240

#对series或data frame所有列进行聚合就是使用aggregate和所需函数，或者是调用mean或st
grouped = tips.groupby(['day', 'smoker'])
grouped_pct = grouped['tip_pct']
grouped_pct.agg('mean')
# day   smoker
# Fri   No        0.151650
#       Yes       0.174783
# Sat   No        0.158048
#       Yes       0.147906
# Sun   No        0.160113
#       Yes       0.187250
# Thur  No        0.160298
#       Yes       0.163863
# Name: tip_pct, dtype: float64

#如果传递的是函数或者函数名的列表，会获得一个列名是这些函数名的data frame：
grouped_pct.agg(['mean', 'std', peak_to_peak])
# mean  std peak_to_peak
# day   smoker
# Fri   No  0.151650    0.028123    0.067349
# Yes   0.174783    0.051293    0.159925
# Sat   No  0.158048    0.039767    0.235193
# Yes   0.147906    0.061375    0.290095
# Sun   No  0.160113    0.042347    0.193226
# Yes   0.187250    0.154134    0.644685
# Thur  No  0.160298    0.038774    0.193350
# Yes   0.163863    0.039389    0.151240

#如果传递的是(name, function)元组的列表，每个元组的第一个元素将作为data frame的列名
#给元组命名：
grouped_pct.agg([('foo', 'mean'), ('bar', np.std)])
#   foo bar
# day   smoker
# Fri   No  0.151650    0.028123
# Yes   0.174783    0.051293
# Sat   No  0.158048    0.039767
# Yes   0.147906    0.061375
# Sun   No  0.160113    0.042347
# Yes   0.187250    0.154134
# Thur  No  0.160298    0.038774
# Yes   0.163863    0.039389
```

Out[96]:

		foo	bar
day	smoker		
Fri	No	0.151650	0.028123
	Yes	0.174783	0.051293
Sat	No	0.158048	0.039767
	Yes	0.147906	0.061375
Sun	No	0.160113	0.042347
	Yes	0.187250	0.154134
Thur	No	0.160298	0.038774
	Yes	0.163863	0.039389

In [99]:
```
#在Data frame中，有更多的选项，可以指定应用到所有列上的函数列表或每一列上要应用的不同
#假设想要计算tip_pct列，和total_bill列的3个相同的统计值：

functions = ['count', 'mean', 'max']
result = grouped['tip_pct', 'total_bill'].agg(functions)
result
# tip_pct   total_bill
# count mean    max count   mean    max
# day   smoker
# Fri   No  4   0.151650    0.187735    4   18.420000   22.75
# Yes   15  0.174783    0.263480    15  16.813333   40.17
# Sat   No  45  0.158048    0.291990    45  19.661778   48.33
# Yes   42  0.147906    0.325733    42  21.276667   50.81
# Sun   No  57  0.160113    0.252672    57  20.506667   48.17
# Yes   19  0.187250    0.710345    19  24.120000   45.35
# Thur  No  45  0.160298    0.266312    45  17.113111   41.19
# Yes   17  0.163863    0.241255    17  19.190588   43.11
```

```
C:\Users\miran\AppData\Local\Temp\ipykernel_14572\1743732889.py:5: FutureWarn
ing: Indexing with multiple keys (implicitly converted to a tuple of keys) wi
ll be deprecated, use a list instead.
  result = grouped['tip_pct', 'total_bill'].agg(functions)
```

Out[99]:

		tip_pct			total_bill		
		count	mean	max	count	mean	max
day	smoker						
Fri	No	4	0.151650	0.187735	4	18.420000	22.75
	Yes	15	0.174783	0.263480	15	16.813333	40.17
Sat	No	45	0.158048	0.291990	45	19.661778	48.33
	Yes	42	0.147906	0.325733	42	21.276667	50.81
Sun	No	57	0.160113	0.252672	57	20.506667	48.17
	Yes	19	0.187250	0.710345	19	24.120000	45.35
Thur	No	45	0.160298	0.266312	45	17.113111	41.19
	Yes	17	0.163863	0.241255	17	19.190588	43.11

In [106]:
```
#产生的data frame拥有分层列，与分别聚合每一列，再以列名作为keys参数使用concat将结果

result['tip_pct']
#   count    mean     max
# day   smoker
# Fri   No  4   0.151650    0.187735
# Yes   15  0.174783    0.263480
# Sat   No  45  0.158048    0.291990
# Yes   42  0.147906    0.325733
# Sun   No  57  0.160113    0.252672
# Yes   19  0.187250    0.710345
# Thur  No  45  0.160298    0.266312
# Yes   17  0.163863    0.241255

#可以传递具有自定义名称的元组列表：
ftuples = [('Durchschnitt', 'mean'), ('Abweichung', np.var)]
grouped['tip_pct', 'total_bill'].agg(ftuples)
# tip_pct   total_bill
# Durchschnitt  Abweichung  Durchschnitt    Abweichung
# day   smoker
# Fri   No  0.151650    0.000791    18.420000   25.596333
# Yes   0.174783    0.002631    16.813333   82.562438
# Sat   No  0.158048    0.001581    19.661778   79.908965
# Yes   0.147906    0.003767    21.276667   101.387535
# Sun   No  0.160113    0.001793    20.506667   66.099980
# Yes   0.187250    0.023757    24.120000   109.046044
# Thur  No  0.160298    0.001503    17.113111   59.625081
# Yes   0.163863    0.001551    19.190588   69.808518

#假设想要将不同的函数应用到一个或多个列上，需要将含有列名与函数对应关系的字典传递给ag
grouped.agg({'tip': np.max, 'size': 'sum'})
#   tip size
# day   smoker
# Fri   No  3.50    9
# Yes   4.73    31
# Sat   No  9.00    115
# Yes   10.00   104
# Sun   No  6.00    167
# Yes   6.50    49
# Thur  No  6.70    112
# Yes   5.00    40

grouped.agg({'tip_pct' : ['min','max','mean','std'],
             'size' : 'sum'})
#   tip_pct size
# min   max mean    std sum
# day   smoker
# Fri   No  0.120385    0.187735    0.151650    0.028123    9
# Yes   0.103555    0.263480    0.174783    0.051293    31
# Sat   No  0.056797    0.291990    0.158048    0.039767    115
# Yes   0.035638    0.325733    0.147906    0.061375    104
# Sun   No  0.059447    0.252672    0.160113    0.042347    167
# Yes   0.065660    0.710345    0.187250    0.154134    49
# Thur  No  0.072961    0.266312    0.160298    0.038774    112
```

```
# Yes    0.090014    0.241255    0.163863    0.039389    40
```

```
C:\Users\miran\AppData\Local\Temp\ipykernel_14572\706861479.py:17: FutureWarn
ing: Indexing with multiple keys (implicitly converted to a tuple of keys) wi
ll be deprecated, use a list instead.
  grouped['tip_pct', 'total_bill'].agg(ftuples)
```

Out[106]:

		tip_pct				size
		min	max	mean	std	sum
day	smoker					
Fri	No	0.120385	0.187735	0.151650	0.028123	9
	Yes	0.103555	0.263480	0.174783	0.051293	31
Sat	No	0.056797	0.291990	0.158048	0.039767	115
	Yes	0.035638	0.325733	0.147906	0.061375	104
Sun	No	0.059447	0.252672	0.160113	0.042347	167
	Yes	0.065660	0.710345	0.187250	0.154134	49
Thur	No	0.072961	0.266312	0.160298	0.038774	112
	Yes	0.090014	0.241255	0.163863	0.039389	40

10.2.2 返回不含行索引的聚合数据 Returning Aggregated Data Without Row Indexes

In [107]:
```
#在前面的例子中，聚合数据返回时带有索引，有时1层，由唯一的分组键联合形成。
#因为不是所有的情况下都需要索引，所以大多数情况下，可以通过向group by传递as_index =

tips.groupby(['day', 'smoker'], as_index = False).mean()
#   day smoker  total_bill  tip size    tip_pct
# 0 Fri No  18.420000   2.812500    2.250000    0.151650
# 1 Fri Yes 16.813333   2.714000    2.066667    0.174783
# 2 Sat No  19.661778   3.102889    2.555556    0.158048
# 3 Sat Yes 21.276667   2.875476    2.476190    0.147906
# 4 Sun No  20.506667   3.167895    2.929825    0.160113
# 5 Sun Yes 24.120000   3.516842    2.578947    0.187250
# 6 Thur    No  17.113111   2.673778    2.488889    0.160298
# 7 Thur    Yes 19.190588   3.030000    2.352941    0.163863

#通过在结果上调用reset_index也可以获得同样的结果。使用as_index = False可以避免一些i
```

```
C:\Users\miran\AppData\Local\Temp\ipykernel_14572\3960698187.py:4: FutureWarn
ing: The default value of numeric_only in DataFrameGroupBy.mean is deprecate
d. In a future version, numeric_only will default to False. Either specify nu
meric_only or select only columns which should be valid for the function.
  tips.groupby(['day', 'smoker'], as_index = False).mean()
```

Out[107]:

	day	smoker	total_bill	tip	size	tip_pct
0	Fri	No	18.420000	2.812500	2.250000	0.151650
1	Fri	Yes	16.813333	2.714000	2.066667	0.174783
2	Sat	No	19.661778	3.102889	2.555556	0.158048
3	Sat	Yes	21.276667	2.875476	2.476190	0.147906
4	Sun	No	20.506667	3.167895	2.929825	0.160113
5	Sun	Yes	24.120000	3.516842	2.578947	0.187250
6	Thur	No	17.113111	2.673778	2.488889	0.160298
7	Thur	Yes	19.190588	3.030000	2.352941	0.163863

10.3 应用：通用拆分- 应用- 联合 Apply: General split-apply-combine

In [110]:
```
#group by方法最常见的目的是apply(应用)，apply将对象拆分成多块，然后在每一块上调用传
#split, apply, combine

#比如想要按组选出消费百分比（tip-pct），最高的五组。
#首先，写一个在特定列中，选出最大值所在行的函数：
def top(df, n = 5, column = 'tip_pct'):
    return df.sort_values(by = column)[-n:]
top(tips, n = 6)
# total_bill    tip smoker  day time    size    tip_pct
# 109   14.31   4.00    Yes Sat Dinner  2   0.279525
# 183   23.17   6.50    Yes Sun Dinner  4   0.280535
# 232   11.61   3.39    No  Sat Dinner  2   0.291990
# 67    3.07    1.00    Yes Sat Dinner  1   0.325733
# 178   9.60    4.00    Yes Sun Dinner  2   0.416667
# 172   7.25    5.15    Yes Sun Dinner  2   0.710345
```

Out[110]:

	total_bill	tip	smoker	day	time	size	tip_pct
109	14.31	4.00	Yes	Sat	Dinner	2	0.279525
183	23.17	6.50	Yes	Sun	Dinner	4	0.280535
232	11.61	3.39	No	Sat	Dinner	2	0.291990
67	3.07	1.00	Yes	Sat	Dinner	1	0.325733
178	9.60	4.00	Yes	Sun	Dinner	2	0.416667
172	7.25	5.15	Yes	Sun	Dinner	2	0.710345

In [112]:
```
#按照smoker分组，调用apply，得到以下结果：

tips.groupby('smoker').apply(top)
# total_bill    tip smoker  day time    size    tip_pct
# smoker
# No    88  24.71   5.85    No  Thur    Lunch   2   0.236746
# 185   20.69   5.00    No  Sun Dinner  5   0.241663
# 51    10.29   2.60    No  Sun Dinner  2   0.252672
# 149   7.51    2.00    No  Thur    Lunch   2   0.266312
# 232   11.61   3.39    No  Sat Dinner  2   0.291990
# Yes   109 14.31   4.00    Yes Sat Dinner  2   0.279525
# 183   23.17   6.50    Yes Sun Dinner  4   0.280535
# 67    3.07    1.00    Yes Sat Dinner  1   0.325733
# 178   9.60    4.00    Yes Sun Dinner  2   0.416667
# 172   7.25    5.15    Yes Sun Dinner  2   0.710345
```

Out[112]:

		total_bill	tip	smoker	day	time	size	tip_pct
smoker								
No	**88**	24.71	5.85	No	Thur	Lunch	2	0.236746
	185	20.69	5.00	No	Sun	Dinner	5	0.241663
	51	10.29	2.60	No	Sun	Dinner	2	0.252672
	149	7.51	2.00	No	Thur	Lunch	2	0.266312
	232	11.61	3.39	No	Sat	Dinner	2	0.291990
Yes	**109**	14.31	4.00	Yes	Sat	Dinner	2	0.279525
	183	23.17	6.50	Yes	Sun	Dinner	4	0.280535
	67	3.07	1.00	Yes	Sat	Dinner	1	0.325733
	178	9.60	4.00	Yes	Sun	Dinner	2	0.416667
	172	7.25	5.15	Yes	Sun	Dinner	2	0.710345

In [115]:
```
#top函数在data frame的每一行分组，使用pandas，concat将函数结果粘贴在一起，并使用分组
#结果包含一个分层索引，该分层索引的内部层级包含原data frame的索引值：

tips.groupby(['smoker', 'day']).apply(top, n = 1, column = 'total_bill')
```

Out[115]:

			total_bill	tip	smoker	day	time	size	tip_pct
smoker	**day**								
No	**Fri**	**94**	22.75	3.25	No	Fri	Dinner	2	0.142857
	Sat	**212**	48.33	9.00	No	Sat	Dinner	4	0.186220
	Sun	**156**	48.17	5.00	No	Sun	Dinner	6	0.103799
	Thur	**142**	41.19	5.00	No	Thur	Lunch	5	0.121389
Yes	**Fri**	**95**	40.17	4.73	Yes	Fri	Dinner	4	0.117750
	Sat	**170**	50.81	10.00	Yes	Sat	Dinner	3	0.196812
	Sun	**182**	45.35	3.50	Yes	Sun	Dinner	3	0.077178
	Thur	**197**	43.11	5.00	Yes	Thur	Lunch	4	0.115982

In [121]:
```
#groupby对象上调用describe方法:
#在groupby对象内部，调用describe方法:

result = tips.groupby('smoker')['tip_pct'].describe()
result
# count mean    std min 25% 50% 75% max
# smoker
# No    151.0   0.159328    0.039910    0.056797    0.136906    0.155625    0.
# Yes   93.0    0.163196    0.085119    0.035638    0.106771    0.153846    0.
result.unstack('smoker')
#  smoker
# count  No        151.000000
#        Yes        93.000000
# mean   No          0.159328
#        Yes         0.163196
# std    No          0.039910
#        Yes         0.085119
# min    No          0.056797
#        Yes         0.035638
# 25%    No          0.136906
#        Yes         0.106771
# 50%    No          0.155625
#        Yes         0.153846
# 75%    No          0.185014
#        Yes         0.195059
# max    No          0.291990
#        Yes         0.710345
# dtype: float64

f = lambda x: x.describe()
grouped.apply(f)
# total_bill    tip size    tip_pct
# day   smoker
# Fri   No  count   4.000000    4.000000    4.00    4.000000
# mean  18.420000   2.812500    2.25    0.151650
# std   5.059282    0.898494    0.50    0.028123
# min   12.460000   1.500000    2.00    0.120385
# 25%   15.100000   2.625000    2.00    0.137239
# ...   ... ... ... ... ... ...
# Thur  Yes min 10.340000   2.000000    2.00    0.090014
# 25%   13.510000   2.000000    2.00    0.148038
# 50%   16.470000   2.560000    2.00    0.153846
# 75%   19.810000   4.000000    2.00    0.194837
# max   43.110000   5.000000    4.00    0.241255
# 64 rows × 4 columns
```

Out[121]:

			total_bill	tip	size	tip_pct
day	**smoker**					
Fri	**No**	**count**	4.000000	4.000000	4.00	4.000000
		mean	18.420000	2.812500	2.25	0.151650
		std	5.059282	0.898494	0.50	0.028123
		min	12.460000	1.500000	2.00	0.120385
		25%	15.100000	2.625000	2.00	0.137239
...	**...**	**...**	...	...	...	...
Thur	**Yes**	**min**	10.340000	2.000000	2.00	0.090014
		25%	13.510000	2.000000	2.00	0.148038
		50%	16.470000	2.560000	2.00	0.153846
		75%	19.810000	4.000000	2.00	0.194837
		max	43.110000	5.000000	4.00	0.241255

64 rows × 4 columns

10.3.1 压缩分组键 Suppressing the Group Keys

In [122]:
```
#在之前的例子中，可以看到所得到的对象，具有分组键形成的分层索引，以及每个原始对象的索
#通过向groupby传递group_keys = False禁用功能:
tips.groupby('smoker', group_keys = False).apply(top)
#    total_bill  tip smoker  day time    size    tip_pct
# 88     24.71   5.85     No  Thur    Lunch   2   0.236746
# 185    20.69   5.00     No  Sun Dinner  5   0.241663
# 51     10.29   2.60     No  Sun Dinner  2   0.252672
# 149    7.51    2.00     No  Thur    Lunch   2   0.266312
# 232    11.61   3.39     No  Sat Dinner  2   0.291990
# 109    14.31   4.00     Yes Sat Dinner  2   0.279525
# 183    23.17   6.50     Yes Sun Dinner  4   0.280535
# 67     3.07    1.00     Yes Sat Dinner  1   0.325733
# 178    9.60    4.00     Yes Sun Dinner  2   0.416667
# 172    7.25    5.15     Yes Sun Dinner  2   0.710345
```

Out[122]:

	total_bill	tip	smoker	day	time	size	tip_pct
88	24.71	5.85	No	Thur	Lunch	2	0.236746
185	20.69	5.00	No	Sun	Dinner	5	0.241663
51	10.29	2.60	No	Sun	Dinner	2	0.252672
149	7.51	2.00	No	Thur	Lunch	2	0.266312
232	11.61	3.39	No	Sat	Dinner	2	0.291990
109	14.31	4.00	Yes	Sat	Dinner	2	0.279525
183	23.17	6.50	Yes	Sun	Dinner	4	0.280535
67	3.07	1.00	Yes	Sat	Dinner	1	0.325733
178	9.60	4.00	Yes	Sun	Dinner	2	0.416667
172	7.25	5.15	Yes	Sun	Dinner	2	0.710345

10.3.2 分位数与桶分析 Quantile and Bucket Analysis

In [133]:
```
#pandas有一些工具，cut和qcut，用于将数据按照箱位或样本分位数，进行分桶。
#与groupby一起使用这些函数，可以对数据集更方便地进行分桶或分位分析。
#考虑一个简单的随机数据集和使用cut的等长桶分类：

frame = pd.DataFrame({'data1': np.random.randn(1000),
                      'data2': np.random.randn(1000)})
frame
# data1 data2
# 0 1.602979    0.084592
# 1 -1.469225   1.383114
# 2 0.572765    -0.279215
# 3 1.201605    -0.570384
# 4 -0.315840   0.111993
# ...   ... ...
# 995   -0.803695   -0.181080
# 996   -0.818455   -1.138788
# 997   0.554210    -0.659000
# 998   0.310772    0.139069
# 999   -0.585090   -0.462089
# 1000 rows × 2 columns

quartiles = pd.cut(frame.data1, 4)
quartiles[:10]
# 0     (-0.148, 1.415]
# 1      (1.415, 2.977]
# 2    (-1.711, -0.148]
# 3     (-0.148, 1.415]
# 4     (-0.148, 1.415]
# 5    (-1.711, -0.148]
# 6    (-1.711, -0.148]
# 7    (-1.711, -0.148]
# 8     (-0.148, 1.415]
# 9    (-1.711, -0.148]
# Name: data1, dtype: category
# Categories (4, interval[float64, right]): [(-3.28, -1.711] < (-1.711, -0.148

#cut返回的categorical对象，可以直接传递给groupby，可以计算出data2列的统计值集合：
def get_status(group):
    return {'min': group.min(), 'max': group.max(),
            'count': group.count(), 'mean': group.mean()}
grouped = frame.data2.groupby(quartiles)
grouped.apply(get_status).unstack()
#   min max count   mean
# data1
# (-3.335, -1.683]  -2.270345   1.706985    46.0    0.162840
# (-1.683, -0.0386] -3.010468   2.594667    428.0   -0.038289
# (-0.0386, 1.606]  -3.029556   2.542962    464.0   -0.062568
# (1.606, 3.251]    -1.629679   2.416200    62.0    0.146844
```

Out[133]:

data1	min	max	count	mean
(-3.848, -2.044]	-1.676549	1.564576	17.0	-0.224431
(-2.044, -0.247]	-2.810099	3.231770	385.0	0.025145
(-0.247, 1.55]	-2.330297	3.273541	542.0	0.051042
(1.55, 3.347]	-1.721874	1.943851	56.0	0.080124

In [136]:

```
#等长桶，为了根据样本分位数计算出等大小的桶，则需要使用qcut。将传递labels = False来

#返回分位数数值
grouping = pd.qcut(frame.data1, 10, labels = False)
grouped = frame.data2.groupby(grouping)
grouped.apply(get_stats).unstack()
# min   max count   mean
# data1
# 0 -2.195265   1.771138    100.0   -0.077749
# 1 -2.583276   2.251311    100.0   0.034486
# 2 -2.265753   2.283414    100.0   -0.040770
# 3 -2.810099   3.231770    100.0   0.153009
# 4 -2.271441   2.251273    100.0   0.066903
# 5 -1.986069   2.689219    100.0   0.084137
# 6 -2.295374   3.273541    100.0   0.007942
# 7 -2.223820   2.593281    100.0   0.064324
# 8 -2.330297   2.458295    100.0   0.126238
# 9 -1.721874   2.137621    100.0   -0.038346
```

Out[136]:

data1	min	max	count	mean
0	-2.195265	1.771138	100.0	-0.077749
1	-2.583276	2.251311	100.0	0.034486
2	-2.265753	2.283414	100.0	-0.040770
3	-2.810099	3.231770	100.0	0.153009
4	-2.271441	2.251273	100.0	0.066903
5	-1.986069	2.689219	100.0	0.084137
6	-2.295374	3.273541	100.0	0.007942
7	-2.223820	2.593281	100.0	0.064324
8	-2.330297	2.458295	100.0	0.126238
9	-1.721874	2.137621	100.0	-0.038346

10.3.3 示例：使用制定分组值填充缺失值 Example: Filling Missing Values with Group-Specific Values

In [143]:
```
#在清除缺失值，有时会使用dropna来去除缺失值，可能想使用修正值或来自其他数据的值，来输
#fillna是一个可以使用的正确工具，这里使用平均值/中位数来填充na值：

s = pd.Series(np.random.randn(6))
s[::2] = np.nan
s
# 0         NaN
# 1    0.844902
# 2         NaN
# 3    2.151037
# 4         NaN
# 5   -0.061953
# dtype: float64
s.fillna(s.mean())
# 0   -0.393170
# 1    1.014675
# 2   -0.393170
# 3    0.550999
# 4   -0.393170
# 5   -2.745183
# dtype: float64
s.fillna(s.median())
# 0    0.210979
# 1   -1.052561
# 2    0.210979
# 3    0.465911
# 4    0.210979
# 5    0.210979
# dtype: float64
```

Out[143]:
```
0    0.210979
1   -1.052561
2    0.210979
3    0.465911
4    0.210979
5    0.210979
dtype: float64
```

In [157]:
```
#假设需要填充值按组来变换。
#一个方法是对数据分组后使用Apply和一个在每个数据块上都调用fillna的函数。

states = ['Ohio', 'New York', 'Vermont', 'Florida',
          'Oregon', 'Nevada', 'California', 'Idaho']
group_key = ['East'] * 4 + ['West'] * 4
group_key
# ['East', 'East', 'East', 'East', 'West', 'West', 'West', 'West']

data = pd.Series(np.random.randn(8), index = states)
data
# Ohio          1.596840
# New York     -0.120325
# Vermont      -0.689562
# Florida      -0.133668
# Oregon       -0.192804
# Nevada        1.458002
# California    1.447963
# Idaho        -0.853654
# dtype: float64

#现在将一些值设置为缺失值:
data[['Vermont', 'Nevada', 'Idaho']] = np.nan

data
# Ohio         -0.706360
# New York     -1.579717
# Vermont            NaN
# Florida      -0.141346
# Oregon       -0.827373
# Nevada             NaN
# California    0.706036
# Idaho              NaN
# dtype: float64

data.groupby(group_key).mean()
# East   -0.781024
# West    0.480829
# dtype: float64

#使用分组的平均值来填充na值:
fill_mean = lambda g: g.fillna(g.mean())
data.groupby(group_key).apply(fill_mean)
# Ohio          0.227815
# New York     -0.734865
# Vermont      -0.401797
# Florida      -0.698342
# Oregon       -0.155629
# Nevada        0.134802
# California    0.425232
# Idaho         0.134802
# dtype: float64

#在另一种情况下，可能已经在代码中为每个分组预定义了填充值。
#由于每个分组都有一个内置name属性。

fill_values = {'East': 0.5, 'West': -1}
```

```
fill_func = lambda g: g.fillna(fill_values[g.name])
data.groupby(group_key).apply(fill_func)
# Ohio          0.692459
# New York     -1.384951
# Vermont       0.500000
# Florida       1.265729
# Oregon        1.074760
# Nevada       -1.000000
# California    2.355515
# Idaho        -1.000000
# dtype: float64
```

```
C:\Users\miran\AppData\Local\Temp\ipykernel_14572\2414783060.py:43: FutureWar
ning: Not prepending group keys to the result index of transform-like apply.
In the future, the group keys will be included in the index, regardless of wh
ether the applied function returns a like-indexed object.
To preserve the previous behavior, use

        >>> .groupby(..., group_keys=False)

To adopt the future behavior and silence this warning, use

        >>> .groupby(..., group_keys=True)
  data.groupby(group_key).apply(fill_mean)
C:\Users\miran\AppData\Local\Temp\ipykernel_14572\2414783060.py:59: FutureWar
ning: Not prepending group keys to the result index of transform-like apply.
In the future, the group keys will be included in the index, regardless of wh
ether the applied function returns a like-indexed object.
To preserve the previous behavior, use

        >>> .groupby(..., group_keys=False)

To adopt the future behavior and silence this warning, use

        >>> .groupby(..., group_keys=True)
  data.groupby(group_key).apply(fill_func)
```

Out[157]:

```
Ohio         -0.082448
New York      0.138421
Vermont       0.500000
Florida      -1.548313
Oregon       -1.248471
Nevada       -1.000000
California    0.524015
Idaho        -1.000000
dtype: float64
```

10.3.4 示例：随机采样与排列 Example: Random Sampling and Permutation

In [174]:
```
#如从大数据集中，抽取随机样本以，用于蒙特卡罗模拟目的或某些其他程序。
#很多方法执行抽取，使用series的sample方法。

#红桃，黑桃，梅花，方块
suits = ['H', 'S', 'C', 'D']
card_val = (list(range(1, 11)) + [10] * 3) * 4
card_val
base_names = ['A'] + list(range(2, 11)) + ['J', 'k', 'Q']
base_names
cards = []
for suit in ['H', 'S', 'C', 'D']:
    cards.extend(str(num) + suit for num in base_names)
deck = pd.Series(card_val, index = cards)

# deck[:13]
# AH      1
# 2H      2
# 3H      3
# 4H      4
# 5H      5
# 6H      6
# 7H      7
# 8H      8
# 9H      9
# 10H    10
# JH     10
# kH     10
# QH     10
# dtype: int64

# card_val
# [1,
#  2,
#  3,
#  4,
#  5,
#  6,
#  7,
#  8,
#  9,
#  10,
#  10,
#  10,
#  10,
#  *4]

#base_names
#['A', 2, 3, 4, 5, 6, 7, 8, 9, 10, 'J', 'k', 'Q']
```

In [176]:
```
#拥有一个长度为52的series，series的索引包含了牌名，series的值可以用于blackjack和其
#从这副牌拿出5张牌可以写成：

def draw(deck, n = 5):
    return deck.sample(n)
draw(deck)
# 2S     2
# 8H     8
# 9H     9
# kC    10
# 5S     5
# dtype: int64
```

Out[176]:
```
kC     10
10D    10
9S      9
8C      8
5H      5
dtype: int64
```

In [181]:
```
#假设想从每个花色中随机抽取两张牌。由于花色是排名的最后两个字符，可以基于这点分组，并

get_suit = lambda card: card[-1]                #last letter is suit
deck.groupby(get_suit).apply(draw, n = 2)
# C  kC    10
#    4C     4
# D  QD    10
#    kD    10
# H  3H     3
#    8H     8
# S  JS    10
#    4S     4
# dtype: int64
```

Out[181]:
```
C  5C     5
   4C     4
D  8D     8
   6D     6
H  kH    10
   6H     6
S  8S     8
   3S     3
dtype: int64
```

In [182]:
```
deck.groupby(get_suit, group_keys = False).apply(draw, n = 2)

# 6C    6
# 3C    3
# 4D    4
# 9D    9
# 9H    9
# 3H    3
# 3S    3
# 8S    8
# dtype: int64
```

Out[182]:
```
6C    6
3C    3
4D    4
9D    9
9H    9
3H    3
3S    3
8S    8
dtype: int64
```

10.3.5 示例：分组加权平均和相关性 Example: Group Weighted Average and Correlation

In [185]:
```
#在groupby的拆分-应用-联合的范式下，data frame的列间操作或两个series间的操作，分组加
#包含分组键和权重值的数据集：

df = pd.DataFrame({'category': ['a', 'a', 'a', 'a',
                                'b', 'b', 'b', 'b'],
                   'data': np.random.randn(8),
                   'weights': np.random.rand(8)})
df
# category  data    weights
# 0 a   0.791799    0.309267
# 1 a   0.743786    0.884674
# 2 a   -0.911523   0.025357
# 3 a   0.262631    0.351922
# 4 b   -0.947891   0.360473
# 5 b   -0.407730   0.997720
# 6 b   -0.679761   0.854525
# 7 b   -0.084914   0.643864
```

Out[185]:

	category	data	weights
0	a	1.019356	0.453931
1	a	-0.384367	0.808556
2	a	-0.096252	0.633160
3	a	0.560670	0.452824
4	b	-0.394675	0.016117
5	b	0.265086	0.981214
6	b	1.842837	0.384724
7	b	-0.675537	0.608134

In [188]:
```
#通过category进行分组加权平均：

grouped = df.groupby('category')
get_wavg = lambda g: np.average(g['data'], weights = g['weights'])
grouped.apply(get_wavg)

# category
# a    0.146852
# b    0.277317
# dtype: float64
```

Out[188]:
```
category
a    0.146852
b    0.277317
dtype: float64
```

In [197]:
```
#标普500(SPX符号)和股票的收盘价:
close_px = pd.read_csv('C:/Users/miran/lpthw/stock_px.csv', parse_dates = True

close_px.info
# <bound method DataFrame.info of                AAPL   MSFT    XOM      SPX
# 2003-01-02    7.40  21.11  29.22   909.03
# 2003-01-03    7.45  21.14  29.24   908.59
# 2003-01-06    7.45  21.52  29.96   929.01
# 2003-01-07    7.43  21.93  28.95   922.93
# 2003-01-08    7.28  21.31  28.83   909.93
# ...            ...    ...    ...      ...
# 2011-10-10  388.81  26.94  76.28  1194.89
# 2011-10-11  400.29  27.00  76.27  1195.54
# 2011-10-12  402.19  26.96  77.16  1207.25
# 2011-10-13  408.43  27.18  76.37  1203.66
# 2011-10-14  422.00  27.27  78.11  1224.58
# [2214 rows x 4 columns]>

close_px[-4:]
# AAPL   MSFT    XOM SPX
# 2011-10-11    400.29  27.00   76.27   1195.54
# 2011-10-12    402.19  26.96   77.16   1207.25
# 2011-10-13    408.43  27.18   76.37   1203.66
# 2011-10-14    422.00  27.27   78.11   1224.58

#计算一个data frame，包含标普指数(SPX)每日收益的年度相关性。
#作为实现的一种方式，首先创建一个计算每列与'SPX'列，成对关联的函数:

spx_corr = lambda x: x.corrwith(x['SPX'])

#使用pct_change计算close-px百分比的变化:
rets = close_px.pct_change().dropna()

#最后按年对百分比变化进行分组，可以使用单行函数从每个行标签中提取每个datetime标签的ye
get_year = lambda x: x.year
by_year = rets.groupby(get_year)
by_year.apply(spx_corr)
# AAPL   MSFT    XOM SPX
# 2003   0.541124    0.745174    0.661265    1.0
# 2004   0.374283    0.588531    0.557742    1.0
# 2005   0.467540    0.562374    0.631010    1.0
# 2006   0.428267    0.406126    0.518514    1.0
# 2007   0.508118    0.658770    0.786264    1.0
# 2008   0.681434    0.804626    0.828303    1.0
# 2009   0.707103    0.654902    0.797921    1.0
# 2010   0.710105    0.730118    0.839057    1.0
# 2011   0.691931    0.800996    0.859975    1.0

#计算苹果和微软的年度相关性:
by_year.apply(lambda g: g['AAPL'].corr(g['MSFT']))
# 2003    0.480868
# 2004    0.259024
# 2005    0.300093
# 2006    0.161735
# 2007    0.417738
# 2008    0.611901
# 2009    0.432738
```

```
# 2010    0.571946
# 2011    0.581987
# dtype: float64
```

Out[197]:
```
2003    0.480868
2004    0.259024
2005    0.300093
2006    0.161735
2007    0.417738
2008    0.611901
2009    0.432738
2010    0.571946
2011    0.581987
dtype: float64
```

10.3.6 示例：逐组线性回归 Example: Group-Wise Linear Regression

In [206]:
```
#只要该函数返回一个pandas对象或标量值，就可以使用groupby执行更复杂的按分组统计分析。
#可以定义一下regress回归函数，使用statsmodels剂量经济学库
#函数对每个数据块执行普通最小二乘OLS回归:

import statsmodels.api as sm
def regress(data, yvar, xvars):
    Y = data[yvar]
    X = data[xvars]
    X['intercept'] = 1.
    result = sm.OLS(Y, X).fit()
    return result.params

by_year.apply(regress, 'AAPL', ['SPX'])

# SPX   intercept
# 2003  1.195406    0.000710
# 2004  1.363463    0.004201
# 2005  1.766415    0.003246
# 2006  1.645496    0.000080
# 2007  1.198761    0.003438
# 2008  0.968016    -0.001110
# 2009  0.879103    0.002954
# 2010  1.052608    0.001261
# 2011  0.806605    0.001514
```

Out[206]:

	SPX	intercept
2003	1.195406	0.000710
2004	1.363463	0.004201
2005	1.766415	0.003246
2006	1.645496	0.000080
2007	1.198761	0.003438
2008	0.968016	-0.001110
2009	0.879103	0.002954
2010	1.052608	0.001261
2011	0.806605	0.001514

10.4 数据透视表与交叉表 Pivot Tables and Cross-Tabulation

In [214]:

```
#data frame拥有一个pivot_table方法，还有一个顶层的pandas.pivot_table函数。
#除了为groupby提供一个方便接口，pivot_table可以添加部分总计，也成作边距。

#在小费数据集，如果想要计算一张在行方向上按day和smoker排列的分组平均值(pivot_table)
tips.pivot_table(index = ['day', 'smoker'])
# size  tip tip_pct total_bill
# day   smoker
# Fri   No  2.250000    2.812500    0.151650    18.420000
# Yes   2.066667    2.714000    0.174783    16.813333
# Sat   No  2.555556    3.102889    0.158048    19.661778
# Yes   2.476190    2.875476    0.147906    21.276667
# Sun   No  2.929825    3.167895    0.160113    20.506667
# Yes   2.578947    3.516842    0.187250    24.120000
# Thur  No  2.488889    2.673778    0.160298    17.113111
# Yes   2.352941    3.030000    0.163863    19.190588

#功能也可以使用groupby时限。
tips.pivot_table(['tip_pct', 'size'], index = ['time', 'day'],
                 columns = 'smoker')
# size  tip_pct
# smoker    No  Yes No  Yes
# time  day
# Dinner    Fri 2.000000    2.222222    0.139622    0.165347
# Sat   2.555556    2.476190    0.158048    0.147906
# Sun   2.929825    2.578947    0.160113    0.187250
# Thur  2.000000    NaN 0.159744    NaN
# Lunch Fri 3.000000    1.833333    0.187735    0.188937
# Thur  2.500000    2.352941    0.160311    0.163863



#通过传递margins = True来扩充标来包含部分总计。
#这会添加all行和列标签，其中相应的值是单层中，所有数据的分组统计值：
tips.pivot_table(['tip_pct', 'size'], index = ['time', 'day'],
                 columns = 'smoker', margins = True)
# size  tip_pct
# smoker    No  Yes All No  Yes All
# time  day
# Dinner    Fri 2.000000    2.222222    2.166667    0.139622    0.165347    0.
# Sat   2.555556    2.476190    2.517241    0.158048    0.147906    0.153152
# Sun   2.929825    2.578947    2.842105    0.160113    0.187250    0.166897
# Thur  2.000000    NaN 2.000000    0.159744    NaN 0.159744
# Lunch Fri 3.000000    1.833333    2.000000    0.187735    0.188937    0.1887
# Thur  2.500000    2.352941    2.459016    0.160311    0.163863    0.161301
# All       2.668874    2.408602    2.569672    0.159328    0.163196    0.1608

#all的值是均值，切该均值是不考虑吸烟者与非吸烟者(all列)，或行分组中任意两级的。
#要使用不同的聚合函数时，将函数传递给aggfunc。例如，'count'或者len将给出一张分组大小

tips.pivot_table('tip_pct', index = ['time', 'smoker'], columns = 'day',
                 aggfunc = len, margins = True)
# day   Fri Sat Sun Thur    All
# time  smoker
# Dinner    No  3.0 45.0    57.0    1.0 106
# Yes   9.0 42.0    19.0    NaN 70
# Lunch No  1.0 NaN NaN 44.0    45
```

```
# Yes    6.0 NaN NaN 17.0    23
# All        19.0    87.0    76.0    62.0    244

#如果产生了空值或者na,可以传递fill_value:
tips.pivot_table('tip_pct', index = ['time', 'size', 'smoker'],
                 columns = 'day', aggfunc = 'mean', fill_value = 0)
# day    Fri Sat Sun Thur
# time   size    smoker
# Dinner    1    No   0.000000    0.137931    0.000000    0.000000
# Yes    0.000000    0.325733    0.000000    0.000000
# 2 No   0.139622    0.162705    0.168859    0.159744
# Yes    0.171297    0.148668    0.207893    0.000000
# 3 No   0.000000    0.154661    0.152663    0.000000
# Yes    0.000000    0.144995    0.152660    0.000000
# 4 No   0.000000    0.150096    0.148143    0.000000
# Yes    0.117750    0.124515    0.193370    0.000000
# 5 No   0.000000    0.000000    0.206928    0.000000
# Yes    0.000000    0.106572    0.065660    0.000000
# 6 No   0.000000    0.000000    0.103799    0.000000
# Lunch 1    No   0.000000    0.000000    0.000000    0.181728
# Yes    0.223776    0.000000    0.000000    0.000000
# 2 No   0.000000    0.000000    0.000000    0.166005
# Yes    0.181969    0.000000    0.000000    0.158843
# 3 No   0.187735    0.000000    0.000000    0.084246
# Yes    0.000000    0.000000    0.000000    0.204952
# 4 No   0.000000    0.000000    0.000000    0.138919
# Yes    0.000000    0.000000    0.000000    0.155410
# 5 No   0.000000    0.000000    0.000000    0.121389
# 6 No   0.000000    0.000000    0.000000    0.173706
```

```
C:\Users\miran\AppData\Local\Temp\ipykernel_14572\3801167867.py:5: FutureWarn
ing: pivot_table dropped a column because it failed to aggregate. This behavi
or is deprecated and will raise in a future version of pandas. Select only th
e columns that can be aggregated.
  tips.pivot_table(index = ['day', 'smoker'])
```

Out[214]:

		day	Fri	Sat	Sun	Thur
time	size	smoker				
Dinner	1	No	0.000000	0.137931	0.000000	0.000000
		Yes	0.000000	0.325733	0.000000	0.000000
	2	No	0.139622	0.162705	0.168859	0.159744
		Yes	0.171297	0.148668	0.207893	0.000000
	3	No	0.000000	0.154661	0.152663	0.000000
		Yes	0.000000	0.144995	0.152660	0.000000
	4	No	0.000000	0.150096	0.148143	0.000000
		Yes	0.117750	0.124515	0.193370	0.000000
	5	No	0.000000	0.000000	0.206928	0.000000
		Yes	0.000000	0.106572	0.065660	0.000000
	6	No	0.000000	0.000000	0.103799	0.000000
Lunch	1	No	0.000000	0.000000	0.000000	0.181728
		Yes	0.223776	0.000000	0.000000	0.000000
	2	No	0.000000	0.000000	0.000000	0.166005
		Yes	0.181969	0.000000	0.000000	0.158843
	3	No	0.187735	0.000000	0.000000	0.084246
		Yes	0.000000	0.000000	0.000000	0.204952
	4	No	0.000000	0.000000	0.000000	0.138919
		Yes	0.000000	0.000000	0.000000	0.155410
	5	No	0.000000	0.000000	0.000000	0.121389
	6	No	0.000000	0.000000	0.000000	0.173706

In [ ]:
```
#pivot table选项:

# values: 需要聚合的列名，默认情况下聚合所有数值型的列
# index: 在结果透视表的行上进行分组的列名或其他分组键
# columns: 在结果透视表的列上进行分组的列名或其他分组键
# aggfunc: 聚合函数或函数列表（'mean'）；可是groupby上下文的任意有效函数
# fill_value: 结果表中替换缺失值的值
# dropna: 替换缺失值的值
# margins: 添加行/列小计和总计
```

10.4.1 交叉表: Cross-Tabulations: Crosstab

In [218]:
```
#交叉表是数据透视表的特殊情况，计算的事分组中的频率。
```

10.5 总结 Conclusion