Pandas数据分析

第3章 Pandas数据分析

目录
CONTENTE

3.7 统计函数

3.8 分组与聚合

3.9 透视表与交叉表

第3章 Pandas数据分析

Python

3.7 统计函数

3.7.1 描述性统计

Pandas中重要的描述性统计函数如表所示。

函数	描述
count()	每一列(axis=0，默认)或每一行(axis=1)非NaN数的数量
sum()	每一列(axis=0，默认)或每一行(axis=1)数值之和
mean()	每一列(axis=0，默认)或每一行(axis=1)数值的平均值
median()	每一列(axis=0，默认)或每一行(axis=1)数值的中位数
min()	每一列(axis=0，默认)或每一行(axis=1)数值中的最小值
max()	每一列(axis=0，默认)或每一行(axis=1)数值中的最大值
prod()	每一列(axis=0，默认)或每一行(axis=1)数值的乘积
cumsum()	每一列(axis=0，默认)或每一行(axis=1)数值的累和
cumprod()	每一列(axis=0，默认)或每一行(axis=1)数值的累积
describe	每一列(axis=0，默认)或每一行(axis=1)数值统计信息摘要
mode()	每一列(axis=0，默认)或每一行(axis=1)出现次数最多的数
std()	每一列(axis=0，默认)或每一行(axis=1)数值的标准差

[示例] 非NaN值数量count

```
import pandas as pd
import numpy as np

d = {'a':[1,2,3,4],
     'b':[5,6,np.nan,8],
     'c':[9,10,11,12]}
df = pd.DataFrame(d)
print('df=\n', df, sep='')

print(f'每列非NaN值计数：\n{df.count()}')
print(f'每行非NaN值计数：\n{df.count(axis=1)}')
```

```
df=
   a    b     c
0  1  5.0     9
1  2  6.0    10
2  3  NaN    11
3  4  8.0    12

每列非NaN值计数:
a    4
b    3
c    4
dtype: int64

每行非NaN值计数:
0    3
1    3
2    2
3    3
dtype: int64
```

[示例] 数据的平均值mean、和sum、中位数median（NaN不纳入计算），加参数axis=1则计算行数据。

```
import pandas as pd
import numpy as np

d = {'a':[1,2,3,4],
     'b':[5,6,np.nan,8],
     'c':[9,10,11,12]}
df = pd.DataFrame(d)
print('df=\n', df, sep='')

print(f'每列平均值：\n{df.mean()}')
print(f'每列数据之和：\n{df.sum()}')
print(f'每列数据的中位数：\n{df.median()}')
```

```
df=
   a    b   c
0  1  5.0   9
1  2  6.0  10
2  3  NaN  11
3  4  8.0  12
```

每列平均值:
```
a     2.500000
b     6.333333
c    10.500000
dtype: float64
```

每列数据之和:
```
a    10.0
b    19.0
c    42.0
dtype: float64
```

每列数据的中位数:
```
a     2.5
b     6.0
c    10.5
dtype: float64
```

[示例] 最小值min和最大值max

```
import pandas as pd
import numpy as np

d = {'a':[1,2,3,4],
     'b':[5,6,np.nan,8],
     'c':[9,10,11,12]}
df = pd.DataFrame(d)
print('df=\n', df, sep='')

print('每列最小值:\n', df.min(), sep='')
print('每行最大值:\n', df.max(axis=1), sep='')
```

```
df=
   a    b   c
0  1  5.0   9
1  2  6.0  10
2  3  NaN  11
3  4  8.0  12

每列最小值:
a    1.0
b    5.0
c    9.0
dtype: float64

每行最大值:
0     9.0
1    10.0
2    11.0
3    12.0
dtype: float64
```

[示例] 乘积prod和累积cumprod

```
import pandas as pd
import numpy as np

d = {'a':[1,2,3,4],
     'b':[5,6,np.nan,8],
     'c':[9,10,11,12]}
df = pd.DataFrame(d)
print('df=\n', df, sep='')

print(f'每一列的乘积：\n{df.prod()}')
print(f'每一列的累积：\n{df.cumprod()}')
```

```
df=
   a    b   c
0  1  5.0   9
1  2  6.0  10
2  3  NaN  11
3  4  8.0  12

每一列的乘积:
a       24.0
b      240.0
c    11880.0
dtype: float64

每一列的累积:
    a      b      c
0   1    5.0      9
1   2   30.0     90
2   6    NaN    990
3  24  240.0  11880
```

[示例] 数字类型数据的统计信息摘要describe

```
import pandas as pd
import numpy as np
d = {'a':[1,2,3,4],
     'b':[5,6,np.nan,8],
     'c':[9,10,11,12]}
df = pd.DataFrame(d)
print('df=\n', df, sep='')
print(f'df的统计信息：\n{df.describe().round(2)}')
```

```
df=
   a    b   c
0  1  5.0   9
1  2  6.0  10
2  3  NaN  11
3  4  8.0  12

df的统计信息：
              a       b      c
count   4.00  3.00   4.00
mean    2.50  6.33  10.50
std     1.29  1.53   1.29
min     1.00  5.00   9.00
25%     1.75  5.50   9.75
50%     2.50  6.00  10.50
75%     3.25  7.00  11.25
max     4.00  8.00  12.00
```

数字类型的数据的统计信息，包括count，mean，std，min，max以及百分位数。默认情况下，百分位数分三档：25%，50%，75%（可设置percentiles参数进行调整）。

百分位数：

百分位数是统计学中用来表示数据分布位置的一种指标，它表示在一个有序数据集中，某个百分比的数据点小于或等于该值。例如，第75百分位数（P75）表示数据集中有75%的数据小于或等于这个值。

百分位数的应用：

1. 描述数据分布（如 P25、P50、P75 描述四分位数）
2. 异常值检测（如 P99 可以识别极端值）
3. 成绩排名（如高考分数排名）
4. 金融风控（如 P95 用于评估风险）

[示例] 百分位数进行高考分数排名

```
import pandas as pd
import numpy as np

# 模拟1000名考生的高考分数（范围：500~700）
np.random.seed(42)
scores = np.random.randint(500, 700, size=1000)
df = pd.DataFrame({"考生ID": range(1, 1001), "分数": scores})
print(df.head())

pc = df["分数"].quantile([0.25, 0.5, 0.75, 0.9]).round(1)  # 返回一个Series
print("高考分数百分位数：")
print(pc)

def get_percentile_range(score, percentiles):
    if score >= pc[0.90]:
        return "前10% (P90+)"
```

[示例] 默认只计算数值型特征的统计量

```
import pandas as pd
import numpy as np

d = pd.date_range('20220115', periods=3)
df = pd.DataFrame({'date': d,
                   'numeric': [1, 2, 3],
                   'objects': ['a', 'b', 'c']
                  })

print(df)
print(df.describe())
```

```
        date  numeric objects
0 2022-01-15        1       a
1 2022-01-16        2       b
2 2022-01-17        3       c
```

```
       numeric
count      3.0
mean       2.0
std        1.0
min        1.0
25%        1.5
50%        2.0
75%        2.5
max        3.0
```

[示例] 字符串类型的数据统计信息摘要

```
import pandas as pd
import numpy as np

d = pd.date_range('20220115', periods=3)
df = pd.DataFrame({'date': d,
                   'numeric': [1, 2, 3],
                   'objects': ['a', 'a', 'c']
                  })
print(df)
print(df.describe(include='O'))
```

```
        date  numeric objects
0 2022-01-15        1       a
1 2022-01-16        2       a
2 2022-01-17        3       c
```

```
        objects
count         3
unique        2
top           a
freq          2
```

字符串类型的数据的统计信息，包括count，unique，top，和freq。

- count：计数，这一组数据中包含数据的个数
- unique：表示有多少种不同的值
- top：数据中出现次数最高的值
- freq：出现次数最高的那个值（top）的出现频率

[示例] 时间数据统计信息摘要

```
import pandas as pd
import numpy as np

d = pd.date_range('20220115', periods=3)
df = pd.DataFrame({'date': d,
                   'numeric': [1, 2, 3],
                   'objects': ['a', 'a', 'c']
                  })
print(df)
print(df.describe(include=np.datetime64))
```

```
        date  numeric objects
0 2022-01-15        1       a
1 2022-01-16        2       a
2 2022-01-17        3       c

                                date
count                              3
unique                             3
top        2022-01-15 00:00:00
freq                               1
first      2022-01-15 00:00:00
last       2022-01-17 00:00:00
```

- 时间数据还包括first和last信息

[示例] 每列出现次数最多的数据mode

```
import pandas as pd

df=pd.DataFrame({"A":[14,4,5,4,1],
                 "B":[5,2,54,3,2],
                 "C":[20,20,7,3,8],
                 "D":[14,3,6,2,6]})

print('df=\n', df, sep='')
print(f'每列出现最多的数：\n{df.mode()}')
```

df=

	A	B	C	D
0	14	5	20	14
1	4	2	20	3
2	5	54	7	6
3	4	3	3	2
4	1	2	8	6

每列出现最多的数：

	A	B	C	D
0	4	2	20	6

[示例]每列出现次数最多的数据有多个

```
import pandas as pd

df=pd.DataFrame({"A":[14,4,5,4,5],
                 "B":[5,2,54,3,2],
                 "C":[20,20,7,3,8],
                 "D":[14,3,6,2,6]})

print('df=\n', df, sep='')
print('df.mode=\n', df.mode(),sep='')
```

```
df=
    A   B   C   D
0  14   5  20  14
1   4   2  20   3
2   5  54   7   6
3   4   3   3   2
4   5   2   8   6
df.mode=
   A    B     C    D
0  4  2.0  20.0  6.0
1  5  NaN   NaN  NaN
```

将第一列的最后一个数据从1改为5，则第一列的4和5都是出现次数最多的数，mode返回2行的DataFrame，其他列没有相同最高频数，则填充NaN。

4.7.2 变化率

变化率使用pct_change()函数求解，Series和DatFrames都可以通过pct_change()函数将每个元素与其前一个元素进行比较，并计算变化百分比。

默认情况下，pct_change()对列进行操作; 如果想应用到行上，那么可使用axis=1参数。

[示例] Series的百分比变化率

```
import pandas as pd

s = pd.Series([2,5,6,11,7])

print(s)
print('--------------------')
print (s.pct_change())
```

```
0     2
1     5
2     6
3    11
4     7
dtype: int64
--------------------
0         NaN
1    1.500000
2    0.200000
3    0.833333
4   -0.363636
dtype: float64
```

解析:

自己跟自己没有变化，为NaN

(5-2)/2= 1.500000

(6-5)/5 = 0.200000

(11-6)/6 = 0.833333

(7-11)/11 = -0.363636

[示例] DataFrame在0轴和1轴方向上的的百分比变化率

```
import pandas as pd
d = {'a':[1,2,3,4],
     'b':[5,6,7,8],
     'c':[9,10,11,12]}
df = pd.DataFrame(d)
print(df)
print('--------------------------------')
print(df.pct_change())
print('--------------------------------')
print(df.pct_change(axis=1))
```

```
   a  b   c
0  1  5   9
1  2  6  10
2  3  7  11
3  4  8  12
--------------------------------
          a         b         c
0       NaN       NaN       NaN
1  1.000000  0.200000  0.111111
2  0.500000  0.166667  0.100000
3  0.333333  0.142857  0.090909
--------------------------------
    a         b         c
0 NaN  4.000000  0.800000
1 NaN  2.000000  0.666667
2 NaN  1.333333  0.571429
3 NaN  1.000000  0.500000
```

以1轴方向上的第一行为例：

自己跟自己没有变化，为NaN

(5-1)/1 = 4.000000

(9-5)/5 = 0.800000

[应用实例] 分析某股票每日价格波动百分比
应用价值：快速识别单日异常波动（如 >±5%）

```
import pandas as pd


# 示例数据：苹果股票收盘价（美元）
stock_data = pd.DataFrame({
    'Date': pd.date_range('2023-01-01', periods=5),
    'AAPL_Close': [142.53, 125.86, 143.96, 147.07, 149.80]
})
# 计算每日收益率
stock_data['Daily_Return'] =
stock_data['AAPL_Close'].pct_change(). .fillna(0).apply(lambda x: f"{x:.2%}")
print(stock_data)
```

	Date	AAPL_Close	Daily_Return
0	2023-01-01	142.53	nan%
1	2023-01-02	125.86	-11.70%
2	2023-01-03	143.96	14.38%
3	2023-01-04	147.07	2.16%
4	2023-01-05	149.80	1.86%

4.7.3 协方差

Panda使用**cov函数**求解两个Series或DataFrame的列之间的协方差。如果数据对象中出现NaN数据，将被自动排除。

标准差和**方差**一般是用来描述一维数据的，但现实生活中常常会遇到含有多维数据的数据集，面对这样的数据集，可以按照每一维独立的计算其方差，也可通过**协方差**度量各个维度之间的关系，比如，如衡量国民收入和居民储蓄存款、身高和体重、高中成绩和高考成绩等变量间的线性相关关系

协方差的值如果为正值，则说明两者是正相关的，结果为负值就说明负相关的，如果为0，也是就是统计上说的“相互独立”。

[示例] 用cov求协方差矩阵

```
from pandas import DataFrame
from numpy import array
A = DataFrame(array([[1,5,6],[4,3,9],\
                     [4,2,9],[4,7,2]]))
print(A)
print('--------------------------')
print(A.cov())
```

```
   0  1  2
0  1  5  6
1  4  3  9
2  4  2  9
3  4  7  2
--------------------------
          0         1         2
0  2.250000 -0.750000  0.500000
1 -0.750000  4.916667 -7.166667
2  0.500000 -7.166667 11.000000
```

$$\frac{1}{N-1}\sum_{i=1}^{N}(x_i-\overline{x})^2$$

第1列均值：3.25
第2列均值：4.25
第3列均值：6.5

以第1行第1个数为例：
((1-3.25)²+(4-3.25)² +(4-3.25)²+ +(4-3.25)²) /3
=2.25

以第2行第2个数为例：
((5-4.25)²+(3-4.25)² +(2-4.25)²+(7-4.25)²)/3
=4.916667

$$\frac{1}{N-1}\sum_{i=1}^{N}(x_i-\overline{x})\times(y_i-\overline{y})$$

第1行第2个数（第1列和第2列的协方差）：
((1-3.25)*(5-4.25)+(4-3.25)*(3-4.25)+
(4-3.25)*(2-4.25)+(4-3.25)*(7-4.25))/3 = -0.75

第1行第3个数（第1列和第3列的协方差）：
((1-3.25)*(6-6.5)+(4-3.25)*(9-6.5)+
(4-3.25)*(9-6.5)+(4-3.25)*(2-6.5))/3=0.5

4.7.4 相关性

相关性显示了任何两个数值（系列）之间的线性关系。

DataFrame.corr(method='pearson', min_periods=1)

- method：可选值为’ pearson’ 、’ kendall’ 和’ spearman’ （统计学三大相关系数）。
  - pearson：Pearson相关系数来衡量正态分布的两个线性数据集是否线性相关（同增同降）。**皮尔森相关系数等于两个变量协方差除以两个变量的标准差之积。**
  - kendall：反映分类变量相关性的指标，即针对无序序列的相关系数，非正态分布的数据。
  - spearman：非线性的，非正态分布的数据的相关系数。
- min_periods：样本最少的数据量（至少要有1个）。

[示例] pearson相关性

$$\frac{\sum_{i=1}^{N}(x_i-\overline{x})\times(y_i-\overline{y})}{\sqrt{\sum_{i=1}^{N}(x_i-\overline{x})^2\times\sum_{i=1}^{N}(y_i-\overline{y})^2}}$$

```
import pandas as pd
import numpy as np

df = pd.DataFrame([[1,5,6],[4,3,9],[4,2,9],[4,7,2]],\
                  columns=['a', 'b', 'c'])
print(df)
print('----------------------------')
print('df中列相关性:')
print(df.corr())



print('----------------------------')
a = df['a']
b = df['b']
print('列a与列b相关性:',a.cov(b)/(a.std()*b.std()))
print('列a与列b相关性:',a.corr(b))
```

```
   a  b  c
0  1  5  6
1  4  3  9
2  4  2  9
3  4  7  2
----------------------------
df中列相关性:
          a         b         c
a  1.000000 -0.225494  0.100504
b -0.225494  1.000000 -0.974508
c  0.100504 -0.974508  1.000000
----------------------------
列a与列b相关性: -0.2254938040084865
列a与列b相关性: -0.2254938040084865
```

[应用实例] 学习事件相关性

```
import pandas as pd

df = pd.read_excel('e:/cov_corr.xlsx')
print(df)
cov_matrix = df.cov()
print("\n协方差矩阵:")
print(cov_matrix.round(2))

corr_matrix = df.corr()
print("\n相关系数矩阵:")
print(corr_matrix.round(2))
```

	数学成绩	物理成绩	缺勤天数	课外活动时间
数学成绩	75.44	43.51	-2.90	4.63
物理成绩	43.51	53.63	-2.18	7.60
缺勤天数	-2.90	-2.18	1.19	-0.45
课外活动时间	4.63	7.60	-0.45	17.06

	数学成绩	物理成绩	缺勤天数	课外活动时间
数学成绩	1.00	0.68	-0.31	0.13
物理成绩	0.68	1.00	-0.27	0.25
缺勤天数	-0.31	-0.27	1.00	-0.10
课外活动时间	0.13	0.25	-0.10	1.00

数学成绩 vs 物理成绩：协方差: 44.15 (正) 相关系数: 0.75 (强正相关)，数理成绩间存在合理强度的正相关
数学成绩 vs 缺勤天数：协方差: -3.54 (负) 相关系数: -0.30 (中等负相关)，缺勤较多与成绩稍低相关
课外活动时间 vs数学成绩 ： 协方差: 4.63 (正) 相关系数: 0.13 (若正相关)，课外活动与数学成绩关联很小
思考：协方差值受单位影响(如缺勤天数的值较小)，相关系数标准化后更易比较相关强度

4.7.5 数据排名

为Series或Dataframe中的每个元素生成排名（序号），使用rank函数实现：

```
rank(axis=0,
        method: str = 'average',
        numeric_only: Union[bool, NoneType] = None,    # 仅排序数值
        na_option: str =  ‘keep’ ,  # NaN数据的排序方法
        ascending: bool = True,      # False则是由大到小排序
        pct: bool = False)              #是否以百分比形式显示返回的排名
```

【示例】为数据系列生成排名。
设置method=‘first’,
则相同的数据，索引靠前的排名也靠前。

```
import pandas as pd


s = pd.Series([17,18,20,16,17,19,21])
print(s)
print('--------------')
t = s.rank(method='first')   # 排名序号为float
t = t.map(int)   # 可通过map转为int
print(t)
```

```
0    17
1    18
2    20
3    16
4    17
5    19
6    21
dtype: int64
--------------
0    2
1    4
2    6
3    1
4    3
5    5
6    7
dtype: int64
```

【示例】为数据系列生成排名。
若method='min',
则相同的数据，排名取最小排名。

```
import pandas as pd


s = pd.Series([17,18,20,16,17,19,21])
print(s)
print('---------------')
t = s.rank(method='min') # 排名序号为float
t = t.astype(int) # 也可用astype函数转为int
print(t)
```

```
0    17
1    18
2    20
3    16
4    17
5    19
6    21
dtype: int64
---------------
0    2
1    4
2    6
3    1
4    2
5    5
6    7
dtype: int32
```

【示例】为数据系列生成排名。
若method=‘max’,
则相同的数据，排名取最大排名。

```
import pandas as pd


s = pd.Series([17,18,20,16,17,19,21])
print(s)
print('--------------')
t = s.rank(method='max').astype(int)
print(t)
```

```
0    17
1    18
2    20
3    16
4    17
5    19
6    21
dtype: int64
--------------
0    3
1    4
2    6
3    1
4    3
5    5
6    7
dtype: int32
```

【示例】为数据系列生成排名。
若method='average',
则相同的数据，取first排名的平均值。

```
import pandas as pd


s = pd.Series([17,18,20,16,17,19,21])
print(s)
print('--------------')
t = s.rank(method='average')
print(t)
```

```
0    17
1    18
2    20
3    16
4    17
5    19
6    21
dtype: int64
--------------
0    2.5
1    4.0
2    6.0
3    1.0
4    2.5
5    5.0
6    7.0
dtype: float64
```

[示例] DataFrame的rank排名

默认mothod='average'，axis=0

```
import pandas as pd
df = pd.DataFrame({'b':[5,7,-3,2],
        'a':[2,1,0,1], 'c':[5,5,8,-3]})
print(df)
print('--------------------')
print(df.rank())
print('--------------------')
print(df.rank(axis=1))
```

```
   b  a  c
0  5  2  5
1  7  1  5
2 -3  0  8
3  2  1 -3
--------------------
     b    a    c
0  3.0  4.0  2.5
1  4.0  2.5  2.5
2  1.0  1.0  4.0
3  2.0  2.5  1.0
--------------------
     b    a    c
0  2.5  1.0  2.5
1  3.0  1.0  2.0
2  1.0  2.0  3.0
3  3.0  2.0  1.0
```

[应用实例] 学生排名

场景：

某班级有 10 名学生，已知他们的数学成绩（含并列分数），需要：

① 按成绩从高到低排名（用于奖学金评定）

② 处理重复分数的排名（避免争议）

③ 分析成绩的百分位数（定位学生水平）

[应用实例] 学生排名

```
import pandas as pd
data = { "姓名":['王谦','曹童','一磊','子鸿','湘崎',\
                 '文杰', '唐中','龙涛','自强','佳欣'],
         "数学成绩": [85, 92, 78, 92, 70, \
                      85, 88, 78, 90, 82]
}
df = pd.DataFrame(data)
print(df)


df["排名"] = df["数学成绩"].rank(ascending=Fals
print(df.sort_values("排名"))

df["百分位数"] = df["数学成绩"].rank(pct=True).r
print(df.sort_values("百分位数", ascending=False
```

超过95%的学生

	姓名	数学成绩	排名	百分位数
1	曹童	92	1	0.95
3	子鸿	92	1	0.95
8	自强	90	3	0.80
6	唐中	88	4	0.70
0	王谦	85	5	0.55
5	文杰	85	5	0.55
9	佳欣	82	7	0.40
2	一磊	78	8	0.25
7	龙涛	78	8	0.25
4	湘崎	70	10	0.10

第3章 Pandas数据分析

Python

3.8 分组聚合

4.8.1 分组

groupby 是 Pandas 中最强大且最常用的功能之一，它允许我们对数据集进行分组，然后对每个分组应用聚合函数。

df.groupby(columns)

参数columns是分组键，可以是**列名**或**列名组成的列表**。

[示例] 基础用法

```
import pandas as pd

sales = pd.DataFrame({
    '地区': ['北京', '上海', '广州', '北京', '上海', '广州'],
    '销售员': ['张三', '李四', '王五', '赵六', '钱七', '孙八'],
    '销售额': [5000, 8000, 6000, 4500, 7000, 5500],
    '订单数': [5, 8, 6, 4, 7, 5]
})

print(sales)
```

	地区	销售员	销售额	订单数
0	北京	张三	5000	5
1	上海	李四	8000	8
2	广州	王五	6000	6
3	北京	赵六	4500	4
4	上海	钱七	7000	7
5	广州	孙八	5500	5

[示例] 基础用法

```
# 按地区分组
grouped = sales.groupby('地区')

# 查看分组结果
for name, group in grouped:
    print(f"\n地区: {name}")
    print(group)
```

Pandas 的 groupby 默认对中文按 Unicode 编码顺序排序：

上 (U+4E0A) → 北 (U+5317) → 广 (U+5E7F)

```
地区: 上海
   地区  销售员   销售额  订单数
1  上海   李四  8000    8
4  上海   钱七  7000    7

地区: 北京
   地区  销售员   销售额  订单数
0  北京   张三  5000    5
3  北京   赵六  4500    4

地区: 广州
   地区  销售员   销售额  订单数
2  广州   王五  6000    6
5  广州   孙八  5500    5
```

[示例] 基础用法

```
# 计算各地区平均销售额
avg_sales = grouped['销售额'].mean()
print("各地区平均销售额:\n", avg_sales)

# 计算各地区销售总额和总订单数
sum_stats = grouped.agg({
    '销售额': 'sum',
    '订单数': 'sum'
})
print("\n各地区总销售额和总订单数:\n", sum_stats)

print(grouped['销售额'].agg(['sum', 'mean', 'count']))
```

```
各地区平均销售额:
地区
上海    7500.0
北京    4750.0
广州    5750.0
Name: 销售额, dtype: float64
```

地区	销售额	订单数
上海	15000	15
北京	9500	9
广州	11500	11

地区	sum	mean	count
上海	15000	7500.0	2
北京	9500	4750.0	2
广州	11500	5750.0	2

[示例] 基础用法

```
# 按地区和销售员多列分组计算平均销售额
multi_grouped = sales.groupby(['地区', '销售员'])
avg_sales_multi = multi_grouped['销售额'].mean()
print("\n各地区-销售员平均销售额:")
print(avg_sales_multi)
```

```
各地区-销售员平均销售额:
地区  销售员
上海  李四     8000.0
      钱七     7000.0
北京  张三     5000.0
      赵六     4500.0
广州  孙八     5500.0
      王五     6000.0
Name: 销售额, dtype: float64
```

[综合实例1] 小型电商数据集分组统计

（1）有一个小型电商数据集，包含产品类别信息：

```
import pandas as pd
data = {
    '订单ID': [1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008],
    '产品类别': ['电子', '服装', '电子', '食品', '家居', '服装', '食品', '电子'],
    '销售额': [1200, 350, 899, 45, 230, 420, 68, 1500]
}
df = pd.DataFrame(data)
print("原始数据集:")
print(df)
```

原始数据集：

	订单ID	产品类别	销售额
0	1001	电子	1200
1	1002	服装	350
2	1003	电子	899
3	1004	食品	45
4	1005	家居	230
5	1006	服装	420
6	1007	食品	68
7	1008	电子	1500

（2）获取所有唯一产品类别

```
unique_categories = df['产品类别'].unique()
print("\n所有唯一产品类别:", unique_categories)
```

```
所有唯一产品类别: ['电子' '服装' '食品' '家居']
```

（3）统计每个类别的出现次数

```
category_counts = df['产品类别'].value_counts()
print("\n每个类别的出现次数:")
print(category_counts)
```

```
每个类别的出现次数:
产品类别
电子    3
服装    2
食品    2
家居    1
Name: count, dtype: int64
```

（4）计算每个类别的总销售额

```
category_sales = df.groupby('产品类别')['销售额'].sum()
print("\n每个类别的总销售额:")
print(category_sales)
```

```
每个类别的总销售额:
产品类别
家居      230
服装      770
电子     3599
食品      113
Name: 销售额, dtype: int64
```

[综合实例2]　小费数据集的分组比较

（1）读取seaborn库中自带数据集tips.csv

```
import pandas as pd


data = pd.read_csv('e:/tips.csv')
pd.set_option('display.unicode.ambiguous_as_wide', True)
pd.set_option('display.unicode.east_asian_width', True)
print(data.head())  # 打印前5行
```

```
   total_bill   tip     sex    smoker  day    time      size
0       16.99   1.01  Female     No    Sun    Dinner     2
1       10.34   1.66    Male     No    Sun    Dinner     3
2       21.01   3.50    Male     No    Sun    Dinner     3
3       23.68   3.31    Male     No    Sun    Dinner     2
4       24.59   3.61  Female     No    Sun    Dinner     4
```

(2) 将列名改为对应中文

```
data.rename(columns={'total_bill':'消费总额', 'tip':'小费',\
                     'sex':'性别', 'smoker':'是否吸烟', 'day':'星期',\
                     'time':'时间', 'size':'人数'}, inplace=True)
print(data.head())
```

	消费总额	小费	性别	是否吸烟	星期	时间	人数
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

（3）将时间数据改为中文

```
data['就餐时间'] =  data['就餐时间'].map(lambda x:'晚餐' if x=='Dinner' else '中餐')
print(data.head())
data.to_csv('e:/tips2.csv')
```

	消费总额	小费	性别	是否吸烟	星期	就餐时间	就餐人数
0	16.99	1.01	Female	No	Sun	晚餐	2
1	10.34	1.66	Male	No	Sun	晚餐	3
2	21.01	3.50	Male	No	Sun	晚餐	3
3	23.68	3.31	Male	No	Sun	晚餐	2
4	24.59	3.61	Female	No	Sun	晚餐	4

(4) 将小费保留1位小数

```
data['小费'] = data['小费'].apply(lambda x : np.round(x,1))
print(data.head())
```

	消费总额	小费	性别	是否吸烟	星期	时间	人数
0	16.99	1.0	Female	No	Sun	晚餐	2
1	10.34	1.7	Male	No	Sun	晚餐	3
2	21.01	3.5	Male	No	Sun	晚餐	3
3	23.68	3.3	Male	No	Sun	晚餐	2
4	24.59	3.6	Female	No	Sun	晚餐	4

（5）将性别数据改为中文

```
data['性别'] =  data['性别'].map({'Female':'女','Male':'男'})
print(data.head())
```

	消费总额	小费	性别	是否吸烟	星期	时间	人数
0	16.99	1.0	女	No	Sun	晚餐	2
1	10.34	1.7	男	No	Sun	晚餐	3
2	21.01	3.5	男	No	Sun	晚餐	3
3	23.68	3.3	男	No	Sun	晚餐	2
4	24.59	3.6	女	No	Sun	晚餐	4

（6）男、女分组后获得第一行、第3行和最后一行数据

```
thf = data.groupby('性别').first()
th3 = data.groupby('性别').nth(2)  # 从0行开始计数
thl = data.groupby('性别').last()
print(thf,th3,thl,sep='\n')
```

```
      消费总额  小费 是否吸烟 星期  时间  人数
性别
女      16.99  1.0      No   Sun  晚餐     2
男      10.34  1.7      No   Sun  晚餐     3

      消费总额  小费 是否吸烟 星期  时间  人数
性别
女      35.26  5.0      No   Sun  晚餐     4
男      23.68  3.3      No   Sun  晚餐     2

     消费总额  小费 是否吸烟  星期  时间  人数
性别
女      18.78  3.0      No  Thur  晚餐     2
男      17.82  1.8      No   Sat   晚餐     2
```

（7）男、女最高和最低消费总额

```
mac = data.groupby('性别')['消费总额'].max()
mic = data.groupby('性别')['消费总额'].min()
print(mac,mic,sep='\n')
```

```
性别
女    44.30
男    50.81
Name: 消费总额, dtype: float64

性别
女    3.07
男    7.25
Name: 消费总额, dtype: float64
```

（8）添加'人均消费列'，保留2位小数

```
data['人均消费'] = round(data['消费总额']/data['人数'], 2)
print(data.head())
```

	消费总额	小费	性别	是否吸烟	星期	时间	人数	人均消费
0	16.99	1.0	女	No	Sun	晚餐	2	8.49
1	10.34	1.7	男	No	Sun	晚餐	3	3.45
2	21.01	3.5	男	No	Sun	晚餐	3	7.00
3	23.68	3.3	男	No	Sun	晚餐	2	11.84
4	24.59	3.6	女	No	Sun	晚餐	4	6.15

（9）查询吸烟男性中人均消费大于15的数据项

```
b = data[(data['是否吸烟']=='Yes') & (data['性别']=='男') & (data['人均消费']>15)]
print(b)
```

	消费总额	小费	性别	是否吸烟	星期	时间	人数	人均消费
83	32.68	5.0	男	Yes	Thur	中餐	2	16.34
170	50.81	10.0	男	Yes	Sat	晚餐	3	16.94
173	31.85	3.2	男	Yes	Sun	晚餐	2	15.92
175	32.90	3.1	男	Yes	Sun	晚餐	2	16.45
179	34.63	3.6	男	Yes	Sun	晚餐	2	17.32
182	45.35	3.5	男	Yes	Sun	晚餐	3	15.12
184	40.55	3.0	男	Yes	Sun	晚餐	2	20.27
237	32.83	1.2	男	Yes	Sat	晚餐	2	16.42

(10) 统计男性给小费的平均值

```
g = data.groupby('性别')  # 按性别分组
mg = g.get_group('男')  # 提取男性组
print(mg.head())  # 输出男性组前5行
mt = mg['小费'].mean()  # 统计男性组的小费均值
print('男性小费均值：', round(mt, 2))
```

	消费总额	小费	性别	是否吸烟	星期	时间	人数	人均消费
1	10.34	1.7	男	No	Sun	晚餐	3	3.45
2	21.01	3.5	男	No	Sun	晚餐	3	7.00
3	23.68	3.3	男	No	Sun	晚餐	2	11.84
5	25.29	4.7	男	No	Sun	晚餐	4	6.32
6	8.77	2.0	男	No	Sun	晚餐	2	4.38

男性小费均值： 3.09

（11）比较男性顾客和女性顾客给的平均小费

```
s = data.groupby('性别')['小费'].mean()
print(s)
```

```
性别
女    2.832184
男    3.089809
Name: 小费, dtype: float64
```

结论：男性给的小费更多

(12) 比较每天给的平均小费

```
d = data.groupby('星期')['小费'].mean()
print(d)
```

```
星期
Fri     2.731579
Sat     2.995402
Sun     3.255263
Thur    2.767742
Name: 小费, dtype: float64
```

结论：周日给的小费最多

（13）比较不同就餐时间给的平均小费

```
t = data.groupby('时间')['小费'].mean()
print(t)
```

```
时间
中餐    2.725000
晚餐    3.103409
Name: 小费, dtype: float64
```

结论：晚餐给的小费更多

（14）比较不同就餐时间的人均消费和平均小费（置于列表中）

```
bt = data.groupby('时间')[['人均消费','小费']].mean()
print(bt)
```

时间	人均消费	小费
中餐	7.316176	2.725000
晚餐	8.109205	3.103409

结论：1）晚餐人均消费和小费都比中餐多

2）人均消费越多，小费给得越多

(15）分析‘吸烟’ + ‘性别’组合因素（置于列表中)对慷慨度的影响

```
ss = data.groupby(['是否吸烟','性别'])['小费'].mean()
print(ss)
```

```
是否吸烟  性别
No        女    2.772222
          男    3.111340
Yes       女    2.930303
          男    3.055000
Name: 小费, dtype: float64
```

结论：不吸烟的男性给的小费更多

（16）根据就餐时间分组，查看‘消费总额’的总值和均值

```
import pandas as pd

data = pd.read_csv('e:/tips.csv')
data.rename(columns={'total_bill':'消费总额', 'tip':'小费',\
                     'sex':'性别', 'smoker':'是否吸烟', 'day':'星期',\
                     'time':'就餐时间', 'size':'就餐人数'}, inplace=True)
# agg函数可在groupby之后，对数据做一种或多种聚合操作
bt = data.groupby('时间')['消费总额'].agg(['mean','sum'])
print(bt)
```

```
          mean     sum
时间
中餐  17.168676  1167.47
晚餐  20.797159  3660.30
```

（17）根据就餐时间分组，查看‘消费总额’的总值和‘小费’的总值和均值

```
bt = data.groupby('时间').agg({'消费总额':'sum', '小费':['sum','mean']})
print(bt)
```

	消费总额	小费	
	sum	sum	mean
时间			
中餐	1167.47	185.3	2.725000
晚餐	3660.30	546.2	3.103409

（18）tips数据中小费的金额是美元，将之按7.2的汇率换成人民币。

```
bt = data.groupby('时间')['小费'].mean().apply(lambda x : x*7.2)


print(round(bt,1))
```

```
时间
中餐    19.6
晚餐    22.3
Name: 小费, dtype: float64
```

（19）增加'日平均消费总额'列，按'星期'统计每日消费总额平均值。

transform函数可以很方便地将某个或某些函数处理过程作用在传入数据的每一列上，从而返回与**输入数据形状一致**的运算结果。

```
data['日平均消费总额'] = data.groupby('星期')['消费总额'].transform('mean')
print(data)
```

	消费总额	小费	性别	是否吸烟	星期	时间	人数	人均消费	日平均消费总额
0	16.99	1.0	女	No	Sun	晚餐	2	8.49	21.41
1	10.34	1.7	男	No	Sun	晚餐	3	3.45	21.41
2	21.01	3.5	男	No	Sun	晚餐	3	7.00	21.41
3	23.68	3.3	男	No	Sun	晚餐	2	11.84	21.41
4	24.59	3.6	女	No	Sun	晚餐	4	6.15	21.41

第3章 Pandas数据分析

Python

3.9 透视表与交叉表

第四章 Pandas数据分析

透视表（pivot table）是各种电子表格程序和其他数据分析软件中一种常见的数据汇总工具。它根据一个或多个键对数据进行聚合，并根据行和列上的分组键将数据分配到各个矩形区域中。

在Pandas中，pivot_table()是一种进行分组统计的函数：

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

- data：一个DataFrame数据集。
- values：待聚合的列的名称，默认聚合所有数值列。
- index：用于分组的列名或其他分组键，出现在结果透视表的行。
- columns：用于分组的列名或其他分组键，出现在结果透视表的列
- aggfunc：是聚合函数或函数列表，**默认为‘mean’**，可用其他聚合函数。
- fill_value：用于替换结果表中的缺失值
- dropna是一个boolean值，默认为True，删除缺失值。
- margins_name：一个string，默认为‘ALL’（合计），当参数margins为True时，margins_name是ALL行和列的名字。

[示例] 数据透视表分析中晚餐时间男女性给的平均小费。

```
import pandas as pd

data = pd.read_csv('e:/tips.csv')
data.rename(columns={'total_bill':'消费总额', 'tip':'小费',\
                     'sex':'性别', 'smoker':'是否吸烟', 'day':'星期',\
                     'time':'时间', 'size':'人数'}, inplace=True)

print(data.pivot_table(values='小费',index='性别',columns='时间'))
```

时间	Dinner	Lunch
性别		
Female	3.002115	2.582857
Male	3.144839	2.882121

[示例] 数据透视表分析中晚餐男女顾客的消费总额。

```
import pandas as pd
data = pd.read_csv('e:/tips.csv')
data.rename(columns={'total_bill':'消费总额', 'tip':'小费',\
                     'sex':'性别', 'smoker':'是否吸烟', 'day':'星期',\
                     'time':'就餐时间', 'size':'就餐人数'}, inplace=True)

print(data.pivot_table(values='消费总额',index='就餐时间',columns='性别',\
aggfunc='sum',margins=True,margins_name='合计'))
```

性别	Female	Male	合计
就餐时间			
Dinner	999.08	2661.22	3660.30
Lunch	571.87	595.60	1167.47
合计	1570.95	3256.82	4827.77

4.9.2 交叉表

交叉表是一种**特殊的透视表**，专用于计算**分组频率**，虽然可以用pivot_table()实现，但是pandas.crosstab()函数会更方便，**默认统计个数**（次数）。

```
pd.crosstab(index, columns, margins=False)
```

参数index是**行索引数据**，columns是**列索引数据**。

[示例] 统计男女顾客中晚餐总计有多少人就餐

```
import pandas as pd

data = pd.read_csv('e:/tips.csv')
data.rename(columns={'total_bill':'消费总额', 'tip':'小费',\
                     'sex':'性别', 'smoker':'是否吸烟', 'day':'星期',\
                     'time':'就餐时间', 'size':'就餐人数'}, inplace=True)

print(pd.crosstab(index=data['性别'], columns=data['就餐时间']))
```

```
就餐时间    Dinner  Lunch
性别
Female      52      35
Male        124     33
```

[示例] 统计男女顾客中晚餐总计有多少人就餐

```
import pandas as pd
data = pd.read_csv('e:/tips.csv')
data.rename(columns={'total_bill':'消费总额', 'tip':'小费',\
                     'sex':'性别', 'smoker':'是否吸烟', 'day':'星期',\
                     'time':'时间', 'size':'人数'}, inplace=True)

print(pd.crosstab(index=data['性别'], columns=data['时间'], \
margins=True))
```

```
就餐时间    Dinner  Lunch  All
性别
Female      52      35     87
Male        124     33     157
All         176     68     244
```

END