导库

In [1]:

```
import pandas as pd
import numpy as np

# 若想要实现多行输出
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "last"  # 默认为'last'，即输出最后一个结果
```

2.1DataFrame的构建

`pd.DataFrame(data,[index],[columns])`

1. 字典类构造DataFrame

（字典中值的取值对象不一样→字典中的键就是dataframe的列名）：

- 数组、列表、元组构成的字典构造DataFrame
- Series构成的字典构造DataFrame
- 字典构成的字典构造DataFrame

2. ndarry数组类构造DataFrame

3. 列表类构造DataFrame

（列表中值的取值对象不一样→列表中每个元素对象就是一行数据）：

- 字典构成的列表构造DataFrame
- Series构成的列表构造DataFrame

数组、列表、元组构成的字典构造DataFrame

In [2]:

```
# 数组、列表、元组序列构成的字典构造DataFrame
data0 = {"a": [1, 2, 3, 4],
         "b": (5, 6, 7, 8),
         "c": np.arange(9, 13)
         }
# 构造DataFrame
pd0 = pd.DataFrame(data0)
```

In [3]:

```
pd0
```

Out[3]:

	a	b	c
0	1	5	9
1	2	6	10
2	3	7	11
3	4	8	12

纯列表的字典构造DataFrame(最常用)

In [4]:

```
data1 = {
    "name": ["张三", "李四", "王五", "小明"],
    "sex": ["female", "female", "male", "male"],
    "year": [2001, 2001, 2003, 2002],
    "city": ["北京", "上海", "广州", "北京"],
    "salary": [8000, 14000, 25000, 8500]
}

pd1 = pd.DataFrame(data1)
pd1
```

Out[4]:

	name	sex	year	city	salary
0	张三	female	2001	北京	8000
1	李四	female	2001	上海	14000
2	王五	male	2003	广州	25000
3	小明	male	2002	北京	8500

2.2查询数据（提取数据）

In [5]:

```
data = pd.read_excel("../data/meal_order_detail.xlsx",
                     sheet_name="meal_order_detail1")
```

In [6]:

```
data.head(3)
```

Out[6]:

	detail_id	order_id	dishes_id	logicprn_name	parent_class_name	dishes_name	itemis_add	counts	amounts	cost	place_order_time	discou
0	2956	417	610062	NaN	NaN	蒜蓉生蚝	0	1	49	NaN	2016-08-01 11:05:36	
1	2958	417	609957	NaN	NaN	蒙古烤羊腿	0	1	48	NaN	2016-08-01 11:07:07	
2	2961	417	609950	NaN	NaN	大蒜苋菜	0	1	30	NaN	2016-08-01 11:07:40	

按列名索引方式读取某一列数据： DataFrame['列名']

参数是一个列名字符串，读取对应的列，返回Series

In [7]:

```
data["dishes_name"]
```

Out[7]:

```
0         蒜蓉生蚝
1        蒙古烤羊腿
2         大蒜苋菜
3        芝麻烤紫菜
4          蒜香包
        ...
2774    白饭/大碗
2775      牛尾汤
2776    意文柠檬汁
2777     金玉良缘
2778     酸辣藕丁
Name: dishes_name, Length: 2779, dtype: object
```

按列名索引方式读取多列数据： DataFrame[['列名1',...,'列名n']]

参数是多个列名字符串组成的列表，读取对应的多列，返回DataFrame

In [8]:

```
data[["dishes_name", "amounts"]]
```

Out[8]:

2779 rows × 2 columns

	dishes_name	amounts
0	蒜蓉生蚝	49
1	蒙古烤羊腿	48
2	大蒜苋菜	30
3	芝麻烤紫菜	25
4	蒜香包	13
...	...	...
2774	白饭/大碗	10
2775	牛尾汤	40
2776	意文柠檬汁	13
2777	金玉良缘	30
2778	酸辣藕丁	33

按位置切片方式读取行数据： DataFrame[索引行位置切片]

切片提取出来的行，不包含切片的终止值行位置索引

In [9]:

```
data[601:605]
```

Out[9]:

	detail_id	order_id	dishes_id	logicprn_name	parent_class_name	dishes_name	itemis_add	counts	amounts	cost	place_order_time	disc
601	5711	673	609999	NaN	NaN	鱼香肉丝拌面	0	1	18	NaN	2016-08-04 20:50:10	
602	5715	673	610000	NaN	NaN	酸辣汤面	0	1	16	NaN	2016-08-04 20:50:21	
603	5628	669	609966	NaN	NaN	芝士烩波士顿龙虾	0	1	175	NaN	2016-08-04 21:12:14	
604	5629	669	609933	NaN	NaN	自制猪肉脯	0	1	48	NaN	2016-08-04 21:14:46	

In [10]:

```
data[10:11]
```

Out[10]:

	detail_id	order_id	dishes_id	logicprn_name	parent_class_name	dishes_name	itemis_add	counts	amounts	cost	place_order_time	disco
10	1910	301	610011	NaN	NaN	白饭/大碗	0	1	10	NaN	2016-08-01 11:31:15	

In [11]:

```
data[data["amounts"] >= 50]
```

Out[11]:

	detail_id	order_id	dishes_id	logicprn_name	parent_class_name	dishes_name	itemis_add	counts	amounts	cost	place_order_time	dis
5	1899	301	610019	NaN	NaN	白斩鸡	0	1	88	NaN	2016-08-01 11:15:57	
6	1902	301	609991	NaN	NaN	香烤牛排	0	1	55	NaN	2016-08-01 11:19:12	
7	1906	301	609983	NaN	NaN	干锅田鸡	0	1	88	NaN	2016-08-01 11:22:21	
11	2916	413	609966	NaN	NaN	芝士烩波士顿龙虾	0	1	175	NaN	2016-08-01 12:42:52	
12	2919	413	609965	NaN	NaN	葱姜炒蟹	0	1	109	NaN	2016-08-01 12:43:21	
...	...	...	...	...	...	...	...	...	...	...	...	
2765	6125	716	609991	NaN	NaN	香烤牛排	0	1	55	NaN	2016-08-10 20:52:09	
2768	6128	716	610022	NaN	NaN	杭椒鸡珍	0	1	58	NaN	2016-08-10 20:53:26	
2769	6135	716	610020	NaN	NaN	泡椒凤爪	0	1	58	NaN	2016-08-10 20:56:10	
2771	6734	774	609941	NaN	NaN	清蒸海鱼	0	1	78	NaN	2016-08-10 21:50:35	
2772	6736	774	609944	NaN	NaN	水煮鱼	0	1	65	NaN	2016-08-10 21:53:17	

837 rows × 19 columns

按索引名称对DataFrame进行切片：

DataFrame.loc(行索引名称/判断条件，列索引名称)

In [12]:

```
data.loc[7:10, "dishes_name"]
```

Out[12]:

```
7        干锅田鸡
8     桂圆枸杞鸽子汤
9      番茄有机花菜
10      白饭/大碗
Name: dishes_name, dtype: object
```

In [13]:

```
data.loc[6:10, ["dishes_name", "amounts"]]
```

Out[13]:

	dishes_name	amounts
6	香烤牛排	55
7	干锅田鸡	88
8	桂圆枸杞鸽子汤	48
9	番茄有机花菜	32
10	白饭/大碗	10

In [14]:

```
data.loc[6:10, "dishes_name":"amounts"]
```

Out[14]:

	dishes_name	itemis_add	counts	amounts
6	香烤牛排	0	1	55
7	干锅田鸡	0	1	88
8	桂圆枸杞鸽子汤	0	1	48
9	番茄有机花菜	0	1	32
10	白饭/大碗	0	1	10

In [15]:

```
data.loc[:, ["dishes_name", "amounts"]]
```

Out[15]:

2779 rows × 2 columns

	dishes_name	amounts
0	蒜蓉生蚝	49
1	蒙古烤羊腿	48
2	大蒜苋菜	30
3	芝麻烤紫菜	25
4	蒜香包	13
...	...	...
2774	白饭/大碗	10
2775	牛尾汤	40
2776	意文柠檬汁	13
2777	金玉良缘	30
2778	酸辣藕丁	33

In [16]:

```
data.loc[data["amounts"] >= 50, ["dishes_id", "dishes_name", "amounts"]]
```

Out[16]:

837 rows × 3 columns

	dishes_id	dishes_name	amounts
5	610019	白斩鸡	88
6	609991	香烤牛排	55
7	609983	干锅田鸡	88
11	609966	芝士烩波士顿龙虾	175
12	609965	葱姜炒蟹	109
...	...	...	...
2765	609991	香烤牛排	55
2768	610022	杭椒鸡珍	58
2769	610020	泡椒凤爪	58
2771	609941	清蒸海鱼	78
2772	609944	水煮鱼	65

In [17]:

```
data.loc[data["detail_id"] == 6750, ["dishes_id", "dishes_name"]]
```

Out[17]:

	dishes_id	dishes_name
2774	610011	白饭/大碗

按索引位置对DataFrame进行切片:

DataFrame.iloc[行索引位置，列索引位置]

In [18]:

```
data.iloc[2, :]
```

Out[18]:

```
detail_id                 2961
order_id                   417
dishes_id                609950
logicprn_name               NaN
parent_class_name           NaN
dishes_name              大蒜苋菜
itemis_add                    0
counts                        1
amounts                      30
cost                        NaN
place_order_time    2016-08-01 11:07:40
discount_amt                NaN
discount_reason             NaN
kick_back                   NaN
add_inprice                   0
add_info                    NaN
bar_code                    NaN
picture_file        caipu/303001.jpg
emp_id                     1442
Name: 2, dtype: object
```

In [19]:

```
data.iloc[:, 2]
```

Out[19]:

```
0       610062
1       609957
2       609950
3       610038
4       610003
        ...
2774    610011
2775    609996
2776    609949
2777    610014
2778    610017
Name: dishes_id, Length: 2779, dtype: int64
```

In [20]:

```
data.iloc[4:10, 0:8]
```

Out[20]:

	detail_id	order_id	dishes_id	logicprn_name	parent_class_name	dishes_name	itemis_add	counts
4	2968	417	610003	NaN	NaN	蒜香包	0	1
5	1899	301	610019	NaN	NaN	白斩鸡	0	1
6	1902	301	609991	NaN	NaN	香烤牛排	0	1
7	1906	301	609983	NaN	NaN	干锅田鸡	0	1
8	1907	301	609981	NaN	NaN	桂圆枸杞鸽子汤	0	1
9	1908	301	610030	NaN	NaN	番茄有机花菜	0	1

In [21]:

```
data.iloc[4:10:2, 0:10:2]
```

Out[21]:

	detail_id	dishes_id	parent_class_name	itemis_add	amounts
4	2968	610003	NaN	0	13
6	1902	609991	NaN	0	55
8	1907	609981	NaN	0	48

In [22]:

```
data.iloc[4:10:2, 3::3]
```

Out[22]:

	logicprn_name	itemis_add	cost	discount_reason	add_info	emp_id
4	NaN	0	NaN	NaN	NaN	1442
6	NaN	0	NaN	NaN	NaN	1095
8	NaN	0	NaN	NaN	NaN	1095

In [23]:

```
data.columns
```

Out[23]:

```
Index(['detail_id', 'order_id', 'dishes_id', 'logicprn_name',
       'parent_class_name', 'dishes_name', 'itemis_add', 'counts', 'amounts',
       'cost', 'place_order_time', 'discount_amt', 'discount_reason',
       'kick_back', 'add_inprice', 'add_info', 'bar_code', 'picture_file',
       'emp_id'],
      dtype='object')
```

In [24]:

```
data.iloc[4:9, 3:]
```

Out[24]:

	logicprn_name	parent_class_name	dishes_name	itemis_add	counts	amounts	cost	place_order_time	discount_amt	discount_reason	kick
4	NaN	NaN	蒜香包	0	1	13	NaN	2016-08-01 11:11:30	NaN	NaN	
5	NaN	NaN	白斩鸡	0	1	88	NaN	2016-08-01 11:15:57	NaN	NaN	
6	NaN	NaN	香烤牛排	0	1	55	NaN	2016-08-01 11:19:12	NaN	NaN	
7	NaN	NaN	干锅田鸡	0	1	88	NaN	2016-08-01 11:22:21	NaN	NaN	
8	NaN	NaN	桂圆枸杞鸽子汤	0	1	48	NaN	2016-08-01 11:22:53	NaN	NaN	

In [25]:

```
data.iloc[4::2, 3::2]
```

Out[25]:

1388 rows × 8 columns

	logicprn_name	dishes_name	counts	cost	discount_amt	kick_back	add_info	picture_file
4	NaN	蒜香包	1	NaN	NaN	NaN	NaN	caipu/503002.jpg
6	NaN	香烤牛排	1	NaN	NaN	NaN	NaN	caipu/201001.jpg
8	NaN	桂圆枸杞鸽子汤	1	NaN	NaN	NaN	NaN	caipu/205001.jpg
10	NaN	白饭/大碗	1	NaN	NaN	NaN	NaN	caipu/601005.jpg
12	NaN	葱姜炒蟹	1	NaN	NaN	NaN	NaN	caipu/102005.jpg
...	...	...	...	...	...	...	...	...
2770	NaN	糖醋蕃茄溜青花	1	NaN	NaN	NaN	NaN	caipu/304005.jpg
2772	NaN	水煮鱼	1	NaN	NaN	NaN	NaN	caipu/103005.jpg
2774	NaN	白饭/大碗	1	NaN	NaN	NaN	NaN	caipu/601005.jpg
2776	NaN	意文柠檬汁	1	NaN	NaN	NaN	NaN	caipu/404005.jpg
2778	NaN	酸辣藕丁	1	NaN	NaN	NaN	NaN	caipu/302006.jpg

In [26]:

```
data.iloc[[3, 8], [-1, -3]]
```

Out[26]:

	emp_id	bar_code
3	1442	NaN
8	1095	NaN

数据抽取小结：

- **数据索引：提取数据的列（按列提取数据）**
- 通过列名或以属性（.列名）的方式→ data[列名]/data.列名
- (dataFrame提取出的一列数据是Series对象)
- **数据切片：提取数据的行（按行提取数据）** →DataFrame对象
- 通过切片的方式提→ data[m:n]:选择索引位置 m-n 行 **（不包括n行）**
- data[满足条件的行索引]
- **按行列要求提取数据**
- DataFrame.loc[行索引名称标签/条件,列索引名称标签] → **(包含终止值)**
- DataFrame.iloc[行索引位置,列索引位置]： 按索引位置读取符合要求的数据 → **(行和列均不包含终止值)**

索引方式	参数	作用
DataFrame[参数]	单个字符串（列名）	显示对应的列（Series结构）
	字符串列表（多个列名组成的列表）	显示列表里所有的列（DataFrame结构）
	索引位置整数切片	满足切片的行（不包含终止值）
	布尔型数组	显示数组元素为True值所对应的行

2.3 DataFrame的增、删、改

In [27]:

```
import numpy as np
import pandas as pd
```

In [28]:

```
pd0 = pd.DataFrame(
    np.arange(12).reshape(3, 4)
    , index=["a", "c", "b"]
    , columns=["A", "B", "C", "D"]
)
pd0
```

Out[28]:

	A	B	C	D
a	0	1	2	3
c	4	5	6	7
b	8	9	10	11

In [29]:

```
pd0.loc["a", "D"]
```

Out[29]:

```
3
```

In [30]:

```
pd0.iloc[0:2, 1:3]
```

Out[30]:

	B	C
a	1	2
c	5	6

In [31]:

```
pd0["D"].shape
```

Out[31]:

```
(3,)
```

In [32]:

```
type(pd0["D"])
```

Out[32]:

```
pandas.core.series.Series
```

In [33]:

```
pd0[["D"]].shape
```

Out[33]:

```
(3, 1)
```

In [34]:

```
type(pd0[["D"]])
```

Out[34]:

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

In [35]:

```
pd0[["D"]]
```

Out[35]:

	D
a	3
c	7
b	11

In [36]:

```
# 和numpy一样 进行转置   行与列进行转置
pdt = pd0.T
pdt
```

Out[36]:

	a	c	b
A	0	4	8
B	1	5	9
C	2	6	10
D	3	7	11

In [37]:

```
pd0.loc[pd0["A"] == 4, "B"]  # 行索引为条件判断语句
```

Out[37]:

```
c    5
Name: B, dtype: int32
```

In [38]:

```
pd0
```

Out[38]:

	A	B	C	D
a	0	1	2	3
c	4	5	6	7
b	8	9	10	11

修改数据

找到指定数据的位置，进行赋值（即提取数据，然后赋值）

In [39]:

```
pd0["D"]
```

Out[39]:

```
a     3
c     7
b    11
Name: D, dtype: int32
```

In [40]:

```
pd0["D"] = [11, 22, 33]
pd0
```

Out[40]:

	A	B	C	D
a	0	1	2	11
c	4	5	6	22
b	8	9	10	33

In [41]:

```
pd0["D"] = 66
pd0
```

Out[41]:

	A	B	C	D
a	0	1	2	66
c	4	5	6	66
b	8	9	10	66

In [42]:

```
pd0[0:1] = 999
pd0
```

Out[42]:

	A	B	C	D
a	999	999	999	999
c	4	5	6	66
b	8	9	10	66

In [43]:

```
pd0[0:1] = [99, 88, 77, 66]
pd0
```

Out[43]:

	A	B	C	D
a	99	88	77	66
c	4	5	6	66
b	8	9	10	66

In [44]:

```
s0 = pd0["D"].copy()
s0[0] = 88888888
s0
```

Out[44]:

```
a    88888888
c          66
b          66
Name: D, dtype: int64
```

In [45]:

```
pd0
```

Out[45]:

	A	B	C	D
a	99	88	77	66
c	4	5	6	66
b	8	9	10	66

新增一列数据

data['新列名']，设置一个新的列索引名称，进行赋值

In [46]:

```
pd0
```

Out[46]:

	A	B	C	D
a	99	88	77	66
c	4	5	6	66
b	8	9	10	66

In [47]:

```
pd0["R"] = [77, 6667, 666677]
pd0
```

Out[47]:

	A	B	C	D	R
a	99	88	77	66	77
c	4	5	6	66	6667
b	8	9	10	66	666677

In [48]:

```
pd0["A"] = 2
pd0
```

Out[48]:

	A	B	C	D	R
a	2	88	77	66	77
c	2	5	6	66	6667
b	2	9	10	66	666677

In [49]:

```
pd0
```

Out[49]:

	A	B	C	D	R
a	2	88	77	66	77
c	2	5	6	66	6667
b	2	9	10	66	666677

新增一行数据

data.loc[行索引名称]，设置一个新的行索引名称，进行赋值

In [50]:

```
pd0.loc[1] = [2, 22, 222, 22, 90]
```

In [51]:

```
pd0
```

Out[51]:

	A	B	C	D	R
a	2	88	77	66	77
c	2	5	6	66	6667
b	2	9	10	66	666677
1	2	22	222	22	90

删除数据

pd对象.drop('索引名/列名', axis) →axis=0，删除行；axis=1，删除列

In [52]:

```
pd0
```

Out[52]:

	A	B	C	D	R
a	2	88	77	66	77
c	2	5	6	66	6667
b	2	9	10	66	666677
1	2	22	222	22	90

In [53]:

```
# 不是在原数据进行修改，返回的是结果视图
pd0.drop("B", axis=1)# axis="columns"
```

Out[53]:

	A	C	D	R
a	2	77	66	77
c	2	6	66	6667
b	2	10	66	666677
1	2	222	22	90

In [54]:

```
pd0.drop("b", axis=0)# axis="index"
```

Out[54]:

	A	B	C	D	R
a	2	88	77	66	77
c	2	5	6	66	6667
1	2	22	222	22	90

In [55]:

```
pd0.drop("c", axis=0)
```

Out[55]:

	A	B	C	D	R
a	2	88	77	66	77
b	2	9	10	66	666677
1	2	22	222	22	90

In [56]:

```
pd0.drop(["A", "C"], axis="columns")
```

Out[56]:

	B	D	R
a	88	66	77
c	5	66	6667
b	9	66	666677
1	22	22	90

In [57]:

```
pd0.drop(["a", "b"], axis=0)
```

Out[57]:

	A	B	C	D	R
c	2	5	6	66	6667
1	2	22	222	22	90

修改数据小结：

- **修改数据：找到指定数据的位置(数据抽取的方法)，进行赋值**
- **新增一列数据：设置一个新列索引，进行赋值** data['新列名']=值
- **删除数据：drop('索引名/列名'，axis='index'/'columns')**

dropna(axis=0, how='any', thresh=None, subset=None, inplace=False)

- axis： 轴向 默认0或'index'，表示按行删除；1或'columns'，表示按列删除。
- how： 筛选方式;默认 any :该行/列只要有一个以上的空值，就删除该行/列； all :全部都为空值，就删除该行/列。
- thresh： 非空元素最低数量;int型，默认为None。如果该行/列中，非空元素数量小于这个值，就删除该行/列。
- inplace： 是否原地替换。布尔值，默认为False。如果为True，则在原DataFrame上进行操作，返回值为None。

2.4 统计分析方法

1-描述性统计分析

常用的统计分析指标有计数、求和、求均值、方差、标准差等

- pd对象.describe() 描述性统计为：个数、均值、标准差、
  最小值、25%分位值、50%分位值、75%分位值，以及最大值
- pd对象.value_counts() ： 统计每一类数据出现的次数
- pd对象.astype('category').describe() → category 类型的描述性统计为：
  非空值个数，类别个数，频次最多的类别，频次最多的次数

In [58]:

```
data = pd.read_excel("../data/meal_order_detail.xlsx",
                     sheet_name="meal_order_detail1")
```

In [59]:

```
data.head(3)
```

Out[59]:

	detail_id	order_id	dishes_id	logicprn_name	parent_class_name	dishes_name	itemis_add	counts	amounts	cost	place_order_time	disco
0	2956	417	610062	NaN	NaN	蒜蓉生蚝	0	1	49	NaN	2016-08-01 11:05:36	
1	2958	417	609957	NaN	NaN	蒙古烤羊腿	0	1	48	NaN	2016-08-01 11:07:07	
2	2961	417	609950	NaN	NaN	大蒜苋菜	0	1	30	NaN	2016-08-01 11:07:40	

In [60]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2779 entries, 0 to 2778
Data columns (total 19 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   detail_id          2779 non-null   int64
 1   order_id           2779 non-null   int64
 2   dishes_id          2779 non-null   int64
 3   logicprn_name      0 non-null      float64
 4   parent_class_name  0 non-null      float64
 5   dishes_name        2779 non-null   object
 6   itemis_add         2779 non-null   int64
 7   counts             2779 non-null   int64
 8   amounts            2779 non-null   int64
 9   cost               0 non-null      float64
 10  place_order_time   2779 non-null   datetime64[ns]
 11  discount_amt       0 non-null      float64
 12  discount_reason    0 non-null      float64
 13  kick_back          0 non-null      float64
 14  add_inprice        2779 non-null   int64
 15  add_info           0 non-null      float64
 16  bar_code           0 non-null      float64
 17  picture_file       2779 non-null   object
 18  emp_id             2779 non-null   int64
dtypes: datetime64[ns](1), float64(8), int64(8), object(2)
memory usage: 412.6+ KB
```

删掉所有值都为空的列

In [61]:

```
# 方法1
data = data.drop(
    [
        "logicprn_name",
        "parent_class_name",
        "cost",
        "discount_amt",
        "discount_reason",
        "kick_back",
        "add_info",
        "bar_code",
    ],
    axis="columns"
)
```

In [62]:

```
# 方法2
data = data.dropna(how="all", axis=1)
```

In [63]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2779 entries, 0 to 2778
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype         
---  ------            --------------  -----         
 0   detail_id         2779 non-null   int64         
 1   order_id          2779 non-null   int64         
 2   dishes_id         2779 non-null   int64         
 3   dishes_name       2779 non-null   object        
 4   itemis_add        2779 non-null   int64         
 5   counts            2779 non-null   int64         
 6   amounts           2779 non-null   int64         
 7   place_order_time  2779 non-null   datetime64[ns]
 8   add_inprice       2779 non-null   int64         
 9   picture_file      2779 non-null   object        
 10  emp_id            2779 non-null   int64         
dtypes: datetime64[ns](1), int64(8), object(2)
memory usage: 238.9+ KB
```

In [64]:

```
data.head(2)
```

Out[64]:

	detail_id	order_id	dishes_id	dishes_name	itemis_add	counts	amounts	place_order_time	add_inprice	picture_file	emp_id
0	2956	417	610062	蒜蓉生蚝	0	1	49	2016-08-01 11:05:36	0	caipu/104001.jpg	1442
1	2958	417	609957	蒙古烤羊腿	0	1	48	2016-08-01 11:07:07	0	caipu/202003.jpg	1442

In [65]:

```
data["amounts"]
```

Out[65]:

```
0       49
1       48
2       30
3       25
4       13
        ..
2774    10
2775    40
2776    13
2777    30
2778    33
Name: amounts, Length: 2779, dtype: int64
```

In [66]:

```
data["amounts"].describe()  # 单独提取一列是Series对象，一维结构
```

Out[66]:

```
count    2779.000000
mean       45.337172
std        36.808550
min         1.000000
25%        25.000000
50%        35.000000
75%        56.000000
max       178.000000
Name: amounts, dtype: float64
```

In [67]:

```
data[["amounts", "counts"]]  # 提取多列是DataFrame对象，二维结构
```

Out[67]:

2779 rows × 2 columns

	amounts	counts
0	49	1
1	48	1
2	30	1
3	25	1
4	13	1
...	...	...
2774	10	1
2775	40	1
2776	13	1
2777	30	1
2778	33	1

In [68]:

```
data[["amounts", "counts"]].describe()
```

Out[68]:

	amounts	counts
count	2779.000000	2779.000000
mean	45.337172	1.111191
std	36.808550	0.625428
min	1.000000	1.000000
25%	25.000000	1.000000
50%	35.000000	1.000000
75%	56.000000	1.000000
max	178.000000	10.000000

In [69]:

```
data.describe()
```

Out[69]:

	detail_id	order_id	dishes_id	itemis_add	counts	amounts	add_inprice	emp_id
count	2779.000000	2779.000000	2779.000000	2779.0	2779.000000	2779.000000	2779.0	2779.000000
mean	4545.617128	737.658150	609981.577906	0.0	1.111191	45.337172	0.0	1252.321339
std	1710.910552	312.694193	153.691844	0.0	0.625428	36.808550	0.0	170.157476
min	753.000000	137.000000	606057.000000	0.0	1.000000	1.000000	0.0	982.000000
25%	3263.000000	471.000000	609951.500000	0.0	1.000000	25.000000	0.0	1124.000000
50%	4386.000000	669.000000	609980.000000	0.0	1.000000	35.000000	0.0	1187.000000
75%	5758.500000	1026.000000	610019.000000	0.0	1.000000	56.000000	0.0	1402.000000
max	8238.000000	1323.000000	610072.000000	0.0	10.000000	178.000000	0.0	1610.000000

删除掉方差为0的列（方差为0，说明整列数据取值都一样）

In [70]:

```
data = data.drop(["itemis_add", "add_inprice"], axis="columns")
data
```

Out[70]:

2779 rows × 9 columns

	detail_id	order_id	dishes_id	dishes_name	counts	amounts	place_order_time	picture_file	emp_id
0	2956	417	610062	蒜蓉生蚝	1	49	2016-08-01 11:05:36	caipu/104001.jpg	1442
1	2958	417	609957	蒙古烤羊腿	1	48	2016-08-01 11:07:07	caipu/202003.jpg	1442
2	2961	417	609950	大蒜苋菜	1	30	2016-08-01 11:07:40	caipu/303001.jpg	1442
3	2966	417	610038	芝麻烤紫菜	1	25	2016-08-01 11:11:11	caipu/105002.jpg	1442
4	2968	417	610003	蒜香包	1	13	2016-08-01 11:11:30	caipu/503002.jpg	1442
...	...	...	...	...	...	...	...	...	...
2774	6750	774	610011	白饭/大碗	1	10	2016-08-10 21:56:24	caipu/601005.jpg	1138
2775	6742	774	609996	牛尾汤	1	40	2016-08-10 21:56:48	caipu/201006.jpg	1138
2776	6756	774	609949	意文柠檬汁	1	13	2016-08-10 22:01:52	caipu/404005.jpg	1138
2777	6763	774	610014	金玉良缘	1	30	2016-08-10 22:03:58	caipu/302003.jpg	1138
2778	6764	774	610017	酸辣藕丁	1	33	2016-08-10 22:04:30	caipu/302006.jpg	1138

data.value_counts() 统计数据频数（个数）

In [71]:

```
data.columns
```

Out[71]:

```
Index(['detail_id', 'order_id', 'dishes_id', 'dishes_name', 'counts',
       'amounts', 'place_order_time', 'picture_file', 'emp_id'],
      dtype='object')
```

In [72]:

```
data['counts'].value_counts()
```

Out[72]:

```
1     2638
2       79
3       19
4       19
6       12
8        6
5        3
7        2
10       1
Name: counts, dtype: int64
```

In [73]:

```
data["dishes_name"].value_counts()
```

Out[73]:

```
白饭/大碗          92
凉拌菠菜           77
谷稻小庄           72
麻辣小龙虾          65
白饭/小碗          60
                   ..
三丝鳝鱼            2
咖啡奶香面包          2
铁板牛肉            2
冰镇花螺            1
百里香奶油烤紅酒牛肉      1
Name: dishes_name, Length: 145, dtype: int64
```

统计'counts'这一列数据取值有的类别数（有多少个不同取值的数据构成）

In [74]:

```
len(data["dishes_name"].value_counts())
```

Out[74]:

```
145
```

Series.astype('category').describe()

'category'类型的描述性统计为：非空值个数，类别个数，频次最多的类别，频次最多的次数

In [75]:

```
data["dishes_name"].value_counts()
```

Out[75]:

```
白饭/大碗          92
凉拌菠菜           77
谷稻小庄           72
麻辣小龙虾          65
白饭/小碗          60
                ..
三丝鳝鱼            2
咖啡奶香面包          2
铁板牛肉            2
冰镇花螺            1
百里香奶油烤紅酒牛肉     1
Name: dishes_name, Length: 145, dtype: int64
```

In [76]:

```
data["dishes_name"].astype("category").describe()
```

Out[76]:

```
count      2779
unique      145
top       白饭/大碗
freq         92
Name: dishes_name, dtype: object
```

In [77]:

```
data["amounts"].astype("category").describe()
```

Out[77]:

```
count     2779
unique      55
top         35
freq       239
Name: amounts, dtype: int64
```

成绩表的描述性分析

In [78]:

```
gra_data = pd.read_excel("../data/grade.xls", sheet_name="grade")
```

In [79]:

```
gra_data.head()
```

Out[79]:

	学号	平时成绩	期末成绩
0	2017001	80	66
1	2017002	87	67
2	2017003	85	60
3	2017004	80	55
4	2017005	76	44

In [80]:

```
gra_data.columns=['stu_id','Normal','exam']
gra_data.head()
```

Out[80]:

	stu_id	Normal	exam
0	2017001	80	66
1	2017002	87	67
2	2017003	85	60
3	2017004	80	55
4	2017005	76	44

In [81]:

```
# 新增一列总评成绩：按照平时成绩和期末成绩按3：7的比例获得
gra_data["总评成绩"] = gra_data["Normal"] * 0.3 + gra_data["exam"] * 0.7
gra_data
```

Out[81]:

	stu_id	Normal	exam	总评成绩
0	2017001	80	66	70.2
1	2017002	87	67	73.0
2	2017003	85	60	67.5
3	2017004	80	55	62.5
4	2017005	76	44	53.6
5	2017006	90	76	80.2
6	2017007	84	68	72.8
7	2017008	90	83	85.1
8	2017009	88	80	82.4
9	2017010	90	74	78.8
10	2017011	75	57	62.4
11	2017012	70	45	52.5
12	2017013	81	62	67.7
13	2017014	86	83	83.9
14	2017015	87	74	77.9
15	2017016	89	82	84.1
16	2017017	84	60	67.2
17	2017018	80	60	66.0
18	2017019	91	85	86.8
19	2017020	86	61	68.5
20	2017021	91	81	84.0
21	2017022	70	32	43.4
22	2017023	95	81	85.2
23	2017024	88	91	90.1
24	2017025	88	84	85.2
25	2017026	91	80	83.3
26	2017027	91	89	89.6
27	2017028	90	83	85.1
28	2017029	88	68	74.0
29	2017030	74	47	55.1
30	2017031	81	50	59.3
31	2017032	78	48	57.0
32	2017033	81	60	66.3
33	2017034	84	89	87.5
34	2017035	87	79	81.4
35	2017036	82	70	73.6
36	2017037	89	60	68.7
37	2017038	90	64	71.8

In [82]:

```
gra_data.columns
```

Out[82]:

```
Index(['stu_id', 'Normal', 'exam', '总评成绩'], dtype='object')
```

In [83]:

```
gra_data[["Normal", "exam"]].describe()
```

Out[83]:

	Normal	exam
count	38.000000	38.000000
mean	84.657895	68.368421
std	6.112797	14.695874
min	70.000000	32.000000
25%	81.000000	60.000000
50%	86.500000	68.000000
75%	89.750000	81.000000
max	95.000000	91.000000

2-分组分析

`df.groupby(by=['分组列1','分组列2'])['统计列1','统计列2'].agg([('聚合名称',聚合函数)])`

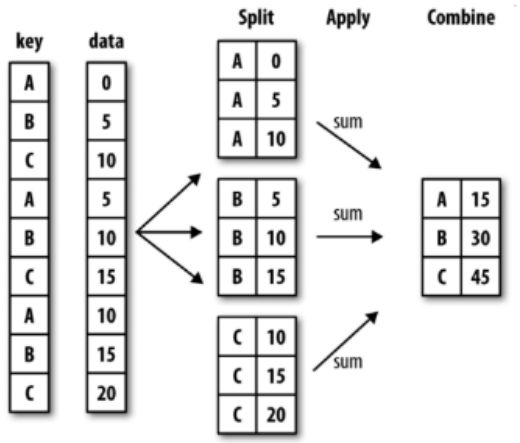
分组分析是指根据分组字段，将分析对象划分成不同的部分，以对比分析各组之间差异性的分析方法。
分组分析常用的统计指标是计数、求和、平均值

In [84]:

```
from IPython.display import Image

Image(
    filename="../data/分组聚合.png",
    width=400,
    height=300,
)
```

Out[84]:



In [85]:

```
df = pd.read_excel("../data/Employee_income.xls", sheet_name="emp_income")
df
```

Out[85]:

	emp_id	sex	age	education	firstjob	region	industry	occupation	salary	subsidy
0	30	男	30	本科	2011-07-20	广州	机械	技术员	5000	500
1	154	男	23	高中	2014-06-23	广州	机械	操作工	2500	1500
2	40	女	28	大专	2011-07-20	广州	机械	文员	3800	200
3	41	女	30	本科	2011-07-20	广州	机械	技术员	4500	500
4	42	男	45	研究生	2000-09-22	广州	机械	主管	7699	1000
5	43	男	37	高中	2000-03-23	广州	商业	店员	3500	600
6	44	女	36	大专	2003-03-24	广州	商业	主管	4500	1000
7	165	女	25	高中	2012-07-21	长沙	机械	文员	2500	500
8	156	男	36	本科	2007-09-22	长沙	机械	技术员	4500	500
9	154	男	38	研究生	2004-08-12	长沙	机械	主管	6500	1000
10	30	男	23	高中	2013-06-23	长沙	机械	操作工	2000	1000
11	154	男	28	大专	2011-07-20	长沙	机械	文员	3000	300
12	40	女	37	高中	2000-03-23	长沙	商业	店员	2500	500
13	165	女	36	大专	2003-03-24	长沙	商业	主管	3500	1000
14	156	男	37	高中	2013-06-23	长沙	商业	店员	2800	1000
15	154	男	36	本科	2011-07-20	长沙	商业	主管	4500	1000
16	46	女	28	高中	2007-03-23	广州	商业	店员	3000	1500
17	47	女	36	本科	2003-03-24	广州	商业	主管	5000	1500

In [86]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18 entries, 0 to 17
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype         
---  ------      --------------  -----         
 0   emp_id      18 non-null     int64         
 1   sex         18 non-null     object        
 2   age         18 non-null     int64         
 3   education   18 non-null     object        
 4   firstjob    18 non-null     datetime64[ns]
 5   region      18 non-null     object        
 6   industry    18 non-null     object        
 7   occupation  18 non-null     object        
 8   salary      18 non-null     int64         
 9   subsidy     18 non-null     int64         
dtypes: datetime64[ns](1), int64(4), object(5)
memory usage: 1.5+ KB
```

In [87]:

```
df.describe()
```

Out[87]:

	emp_id	age	salary	subsidy
count	18.000000	18.000000	18.000000	18.000000
mean	92.277778	32.722222	3961.055556	838.888889
std	60.033134	6.027443	1487.898260	407.487118
min	30.000000	23.000000	2000.000000	200.000000
25%	41.250000	28.000000	2850.000000	500.000000
50%	46.500000	36.000000	3650.000000	1000.000000
75%	154.000000	36.750000	4500.000000	1000.000000
max	165.000000	45.000000	7699.000000	1500.000000

In [88]:

```
df.columns
```

Out[88]:

```
Index(['emp_id', 'sex', 'age', 'education', 'firstjob', 'region', 'industry',
       'occupation', 'salary', 'subsidy'],
      dtype='object')
```

In [89]:

```
df[["occupation"]].value_counts()
```

Out[89]:

```
occupation
主管            6
店员            4
技术员           3
文员            3
操作工           2
dtype: int64
```

In [90]:

```
df[["education", ]].value_counts()
```

Out[90]:

```
education
高中            7
本科            5
大专            4
研究生           2
dtype: int64
```

In [91]:

```
df.columns
```

Out[91]:

```
Index(['emp_id', 'sex', 'age', 'education', 'firstjob', 'region', 'industry',
       'occupation', 'salary', 'subsidy'],
      dtype='object')
```

In [92]:

```
df.iloc[:, 1:]
```

Out[92]:

	sex	age	education	firstjob	region	industry	occupation	salary	subsidy
0	男	30	本科	2011-07-20	广州	机械	技术员	5000	500
1	男	23	高中	2014-06-23	广州	机械	操作工	2500	1500
2	女	28	大专	2011-07-20	广州	机械	文员	3800	200
3	女	30	本科	2011-07-20	广州	机械	技术员	4500	500
4	男	45	研究生	2000-09-22	广州	机械	主管	7699	1000
5	男	37	高中	2000-03-23	广州	商业	店员	3500	600
6	女	36	大专	2003-03-24	广州	商业	主管	4500	1000
7	女	25	高中	2012-07-21	长沙	机械	文员	2500	500
8	男	36	本科	2007-09-22	长沙	机械	技术员	4500	500
9	男	38	研究生	2004-08-12	长沙	机械	主管	6500	1000
10	男	23	高中	2013-06-23	长沙	机械	操作工	2000	1000
11	男	28	大专	2011-07-20	长沙	机械	文员	3000	300
12	女	37	高中	2000-03-23	长沙	商业	店员	2500	500
13	女	36	大专	2003-03-24	长沙	商业	主管	3500	1000
14	男	37	高中	2013-06-23	长沙	商业	店员	2800	1000
15	男	36	本科	2011-07-20	长沙	商业	主管	4500	1000
16	女	28	高中	2007-03-23	广州	商业	店员	3000	1500
17	女	36	本科	2003-03-24	广州	商业	主管	5000	1500

分组

In [93]:

```
df.groupby(by="education")
```

Out[93]:

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000002D844B406D0>
```

In [94]:

```
list(df.groupby(by="education"))
```

Out[94]:

```
[('大专',
     emp_id sex  age education   firstjob region industry occupation  salary  \
  2       40   女   28        大专 2011-07-20     广州       机械         文员    3800
  6       44   女   36        大专 2003-03-24     广州       商业         主管    4500
  11     154   男   28        大专 2011-07-20     长沙       机械         文员    3000
  13     165   女   36        大专 2003-03-24     长沙       商业         主管    3500

     subsidy
  2       200
  6      1000
  11      300
  13     1000  ),
 ('本科',
     emp_id sex  age education   firstjob region industry occupation  salary  \
  0       30   男   30        本科 2011-07-20     广州       机械        技术员    5000
  3       41   女   30        本科 2011-07-20     广州       机械        技术员    4500
  8      156   男   36        本科 2007-09-22     长沙       机械        技术员    4500
  15     154   男   36        本科 2011-07-20     长沙       商业         主管    4500
  17      47   女   36        本科 2003-03-24     广州       商业         主管    5000

     subsidy
  0       500
  3       500
  8       500
  15     1000
  17     1500  ),
 ('研究生',
    emp_id sex  age education   firstjob region industry occupation  salary  \
  4      42   男   45       研究生 2000-09-22     广州       机械         主管    7699
  9     154   男   38       研究生 2004-08-12     长沙       机械         主管    6500

    subsidy
  4     1000
  9     1000  ),
 ('高中',
     emp_id sex  age education   firstjob region industry occupation  salary  \
  1      154   男   23        高中 2014-06-23     广州       机械        操作工    2500
  5       43   男   37        高中 2000-03-23     广州       商业         店员    3500
  7      165   女   25        高中 2012-07-21     长沙       机械         文员    2500
  10      30   男   23        高中 2013-06-23     长沙       机械        操作工    2000
  12      40   女   37        高中 2000-03-23     长沙       商业         店员    2500
  14     156   男   37        高中 2013-06-23     长沙       商业         店员    2800
  16      46   女   28        高中 2007-03-23     广州       商业         店员    3000

     subsidy
  1      1500
  5       600
  7       500
  10     1000
  12      500
  14     1000
  16     1500  )]
```

In [95]:

```
dict(list(df.groupby(by="education")))["大专"]
```

Out[95]:

	emp_id	sex	age	education	firstjob	region	industry	occupation	salary	subsidy
2	40	女	28	大专	2011-07-20	广州	机械	文员	3800	200
6	44	女	36	大专	2003-03-24	广州	商业	主管	4500	1000
11	154	男	28	大专	2011-07-20	长沙	机械	文员	3000	300
13	165	女	36	大专	2003-03-24	长沙	商业	主管	3500	1000

In [96]:

```
# 选取任意数据块
dict(list(df.groupby(by="education")))["本科"]
```

Out[96]:

	emp_id	sex	age	education	firstjob	region	industry	occupation	salary	subsidy
0	30	男	30	本科	2011-07-20	广州	机械	技术员	5000	500
3	41	女	30	本科	2011-07-20	广州	机械	技术员	4500	500
8	156	男	36	本科	2007-09-22	长沙	机械	技术员	4500	500
15	154	男	36	本科	2011-07-20	长沙	商业	主管	4500	1000
17	47	女	36	本科	2003-03-24	广州	商业	主管	5000	1500

聚合

In [97]:

```
df.groupby(by="education")["salary"].agg([np.mean])
```

Out[97]:

	mean
education	
大专	3700.000000
本科	4700.000000
研究生	7099.500000
高中	2685.714286

In [98]:

```
df.groupby(by="education")["salary"].agg([np.mean, np.std, np.max, np.min])
```

Out[98]:

	mean	std	amax	amin
education				
大专	3700.000000	627.162924	4500	3000
本科	4700.000000	273.861279	5000	4500
研究生	7099.500000	847.821031	7699	6500
高中	2685.714286	474.090608	3500	2000

In [99]:

```
df.groupby(by="education")["salary"].agg([("薪资均值", np.mean)])
```

Out[99]:

	薪资均值
education	
大专	3700.000000
本科	4700.000000
研究生	7099.500000
高中	2685.714286

In [100]:

```
def diff_value(arr):
    return arr.max() - arr.min()


age_result = df.groupby(by=["education", "age"])["salary"].agg(
                        [
                         ("薪资人数", np.size),
                         ("薪资平均值", np.mean),
                         ("薪资最高值", np.max),
                         ("薪资最低值", np.min),
                         ("薪资差值", diff_value)
                        ]
)
age_result
```

Out[100]:

		薪资人数	薪资平均值	薪资最高值	薪资最低值	薪资差值
education	age					
大专	28	2	3400.000000	3800	3000	800
	36	2	4000.000000	4500	3500	1000
本科	30	2	4750.000000	5000	4500	500
	36	3	4666.666667	5000	4500	500
研究生	38	1	6500.000000	6500	6500	0
	45	1	7699.000000	7699	7699	0
高中	23	2	2250.000000	2500	2000	500
	25	1	2500.000000	2500	2500	0
	28	1	3000.000000	3000	3000	0
	37	3	2933.333333	3500	2500	1000

In [101]:

```
age_result.columns
```

Out[101]:

```
Index(['薪资人数', '薪资平均值', '薪资最高值', '薪资最低值', '薪资差值'], dtype='object')
```

In [102]:

```
age_result.index
```

Out[102]:

```
MultiIndex([('大专', 28),
            ('大专', 36),
            ('本科', 30),
            ('本科', 36),
            ('研究生', 38),
            ('研究生', 45),
            ('高中', 23),
            ('高中', 25),
            ('高中', 28),
            ('高中', 37)],
           names=['education', 'age'])
```

层级索引

In [103]:

```
age_result.loc[('高中',28), ]
```

Out[103]:

```
薪资人数       1.0
薪资平均值    3000.0
薪资最高值    3000.0
薪资最低值    3000.0
薪资差值       0.0
Name: (高中, 28), dtype: float64
```

In [104]:

```
age_result = df.groupby(by=["education", "age"])["salary"].agg(
    [
        ("人数", np.size),
        ("平均值", np.mean),
        ("最高值", np.max),
        ("最低值", np.min),
        ("差值", diff_value),
    ]
)
age_result
```

Out[104]:

		人数	平均值	最高值	最低值	差值
education	age					
大专	28	2	3400.000000	3800	3000	800
	36	2	4000.000000	4500	3500	1000
本科	30	2	4750.000000	5000	4500	500
	36	3	4666.666667	5000	4500	500
研究生	38	1	6500.000000	6500	6500	0
	45	1	7699.000000	7699	7699	0
高中	23	2	2250.000000	2500	2000	500
	25	1	2500.000000	2500	2500	0
	28	1	3000.000000	3000	3000	0
	37	3	2933.333333	3500	2500	1000

In [105]:

```
# 分组顺序不一样，优先级不一样
age_result = df.groupby(by=["age", "education"])["salary"].agg(
                    [("人数", np.size)
                   , ("平均值", np.mean)
                   , ("最高值", np.max)
                   , ("最低值", np.min)
                    ]
)
age_result
```

Out[105]:

		人数	平均值	最高值	最低值
age	education				
23	高中	2	2250.000000	2500	2000
25	高中	1	2500.000000	2500	2500
28	大专	2	3400.000000	3800	3000
	高中	1	3000.000000	3000	3000
30	本科	2	4750.000000	5000	4500
36	大专	2	4000.000000	4500	3500
	本科	3	4666.666667	5000	4500
37	高中	3	2933.333333	3500	2500
38	研究生	1	6500.000000	6500	6500
45	研究生	1	7699.000000	7699	7699

In [106]:

```
edu_result = df.groupby(by=["education", "occupation"])[["salary", "subsidy"]].agg(
                            [("人数", np.size)
                           , ("平均值", np.mean)
                           , ("最高值", np.max)
                           , ("最低值", np.min)
                            ]
)
edu_result
```

Out[106]:

		salary				subsidy			
		人数	平均值	最高值	最低值	人数	平均值	最高值	最低值
education	occupation								
大专	主管	2	4000.000000	4500	3500	2	1000.0	1000	1000
	文员	2	3400.000000	3800	3000	2	250.0	300	200
本科	主管	2	4750.000000	5000	4500	2	1250.0	1500	1000
	技术员	3	4666.666667	5000	4500	3	500.0	500	500
研究生	主管	2	7099.500000	7699	6500	2	1000.0	1000	1000
高中	店员	4	2950.000000	3500	2500	4	900.0	1500	500
	操作工	2	2250.000000	2500	2000	2	1250.0	1500	1000
	文员	1	2500.000000	2500	2500	1	500.0	500	500

按工种进行分组统计与提成之间的关系

In [107]:

```
def diff_value(arr):
    return arr.max() - arr.min()


df.groupby(by="occupation")["subsidy"].agg(
    [
        ("人数", np.size),
        ("平均提成", np.mean),
        ("最高提成", np.max),
        ("最低提成", np.min),
        ("提成差额", diff_value),
    ]
)
```

Out[107]:

	人数	平均提成	最高提成	最低提成	提成差额
occupation					
主管	6	1083.333333	1500	1000	500
店员	4	900.000000	1500	500	1000
技术员	3	500.000000	500	500	0
操作工	2	1250.000000	1500	1000	500
文员	3	333.333333	500	200	300

3-分布分析

分布分析是指根据分析的目的，将定量数据(**连续型数据**)进行等距或者不等距的分组；
从而研究各组分布规律的一种分析方法，如学生成绩分布、用户年龄分布、收入状况分布等。

在分布分析时，首先用cut()函数确定分布分析中的分层，然后再用groupby()函数实现分组分析

pd.cut(data,bins,right=true,labels=None,include_lowest=False)

- data: 进行划分的一维数组
- bins：取整数值，表示将x划分为多少个等距的区间。取序列值，表示将x划分在指定序列中
- right：分组时是否包含右端点，默认为True（包含）
- labels：分组时是否用自定义标签来代替返回的bins，可选项，默认为NULL。
- include_lowest：分组时是否包含左端点，默认为False（不包含）。

按年龄分布状况[20,30,40,50,60]，分组统计人数、平均月薪、最高月薪和最低月薪

In [108]:

```
df = pd.read_excel("../data/Employee_income.xls", sheet_name="emp_income")
df.head(3)
```

Out[108]:

	emp_id	sex	age	education	firstjob	region	industry	occupation	salary	subsidy
0	30	男	30	本科	2011-07-20	广州	机械	技术员	5000	500
1	154	男	23	高中	2014-06-23	广州	机械	操作工	2500	1500
2	40	女	28	大专	2011-07-20	广州	机械	文员	3800	200

In [109]:

```
df.columns
```

Out[109]:

```
Index(['emp_id', 'sex', 'age', 'education', 'firstjob', 'region', 'industry',
       'occupation', 'salary', 'subsidy'],
      dtype='object')
```

In [110]:

```
df["occupation"].value_counts()
```

Out[110]:

```
主管     6
店员     4
技术员    3
文员     3
操作工    2
Name: occupation, dtype: int64
```

In [111]:

```
df["occupation"].describe()
```

Out[111]:

```
count     18
unique     5
top       主管
freq       6
Name: occupation, dtype: object
```

In [112]:

```
# 年龄分布状况
age_bins = [20, 30, 40, 50]
age_labels = ["青年", "中年", "中老年"]
df["年龄分层"] = pd.cut(df["age"], age_bins, labels=age_labels)
df["年龄分层"]
```

Out[112]:

```
0      青年
1      青年
2      青年
3      青年
4     中老年
5      中年
6      中年
7      青年
8      中年
9      中年
10     青年
11     青年
12     中年
13     中年
14     中年
15     中年
16     青年
17     中年
Name: 年龄分层, dtype: category
Categories (3, object): ['青年' < '中年' < '中老年']
```

In [113]:

```
df
```

Out[113]:

	emp_id	sex	age	education	firstjob	region	industry	occupation	salary	subsidy	年龄分层
0	30	男	30	本科	2011-07-20	广州	机械	技术员	5000	500	青年
1	154	男	23	高中	2014-06-23	广州	机械	操作工	2500	1500	青年
2	40	女	28	大专	2011-07-20	广州	机械	文员	3800	200	青年
3	41	女	30	本科	2011-07-20	广州	机械	技术员	4500	500	青年
4	42	男	45	研究生	2000-09-22	广州	机械	主管	7699	1000	中老年
5	43	男	37	高中	2000-03-23	广州	商业	店员	3500	600	中年
6	44	女	36	大专	2003-03-24	广州	商业	主管	4500	1000	中年
7	165	女	25	高中	2012-07-21	长沙	机械	文员	2500	500	青年
8	156	男	36	本科	2007-09-22	长沙	机械	技术员	4500	500	中年
9	154	男	38	研究生	2004-08-12	长沙	机械	主管	6500	1000	中年
10	30	男	23	高中	2013-06-23	长沙	机械	操作工	2000	1000	青年
11	154	男	28	大专	2011-07-20	长沙	机械	文员	3000	300	青年
12	40	女	37	高中	2000-03-23	长沙	商业	店员	2500	500	中年
13	165	女	36	大专	2003-03-24	长沙	商业	主管	3500	1000	中年
14	156	男	37	高中	2013-06-23	长沙	商业	店员	2800	1000	中年
15	154	男	36	本科	2011-07-20	长沙	商业	主管	4500	1000	中年
16	46	女	28	高中	2007-03-23	广州	商业	店员	3000	1500	青年
17	47	女	36	本科	2003-03-24	广州	商业	主管	5000	1500	中年

In [114]:

```
df["年龄分层2"] = pd.cut(df.age, age_bins)
df
```

Out[114]:

	emp_id	sex	age	education	firstjob	region	industry	occupation	salary	subsidy	年龄分层	年龄分层2
0	30	男	30	本科	2011-07-20	广州	机械	技术员	5000	500	青年	(20, 30]
1	154	男	23	高中	2014-06-23	广州	机械	操作工	2500	1500	青年	(20, 30]
2	40	女	28	大专	2011-07-20	广州	机械	文员	3800	200	青年	(20, 30]
3	41	女	30	本科	2011-07-20	广州	机械	技术员	4500	500	青年	(20, 30]
4	42	男	45	研究生	2000-09-22	广州	机械	主管	7699	1000	中老年	(40, 50]
5	43	男	37	高中	2000-03-23	广州	商业	店员	3500	600	中年	(30, 40]
6	44	女	36	大专	2003-03-24	广州	商业	主管	4500	1000	中年	(30, 40]
7	165	女	25	高中	2012-07-21	长沙	机械	文员	2500	500	青年	(20, 30]
8	156	男	36	本科	2007-09-22	长沙	机械	技术员	4500	500	中年	(30, 40]
9	154	男	38	研究生	2004-08-12	长沙	机械	主管	6500	1000	中年	(30, 40]
10	30	男	23	高中	2013-06-23	长沙	机械	操作工	2000	1000	青年	(20, 30]
11	154	男	28	大专	2011-07-20	长沙	机械	文员	3000	300	青年	(20, 30]
12	40	女	37	高中	2000-03-23	长沙	商业	店员	2500	500	中年	(30, 40]
13	165	女	36	大专	2003-03-24	长沙	商业	主管	3500	1000	中年	(30, 40]
14	156	男	37	高中	2013-06-23	长沙	商业	店员	2800	1000	中年	(30, 40]
15	154	男	36	本科	2011-07-20	长沙	商业	主管	4500	1000	中年	(30, 40]
16	46	女	28	高中	2007-03-23	广州	商业	店员	3000	1500	青年	(20, 30]
17	47	女	36	本科	2003-03-24	广州	商业	主管	5000	1500	中年	(30, 40]

In [115]:

```
df["年龄分层"].value_counts()
```

Out[115]:

```
中年     9
青年     8
中老年    1
Name: 年龄分层, dtype: int64
```

In [116]:

```
# 分组统计人数、平均月薪、最高月薪和最低月薪
aggResult = df.groupby(by=["年龄分层"])["salary"].agg(
                    [("人数", np.size)
                     , ("平均月薪", np.mean)
                     , ("最高月薪", np.max)
                     , ("最低月薪", np.min)
                    ]
)
aggResult
```

Out[116]:

年龄分层	人数	平均月薪	最高月薪	最低月薪
青年	8	3287.500000	5000	2000
中年	9	4144.444444	6500	2500
中老年	1	7699.000000	7699	7699

4-交叉分析

交叉分析通常用于分析两个或两个以上分组变量之间的关系，以交叉表形式进行变量间关系的对比分析；
从数据的不同维度，综合进行分组细分，进一步了解数据的构成、分布特征。交叉分析有数据透视表和交叉表两种：

1. 透视表

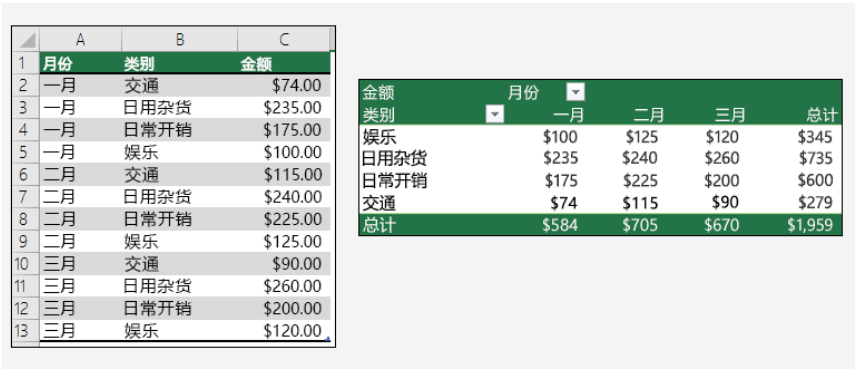
pivot_table() 函数返回值是数据透视表的结果，该函数相当于Excel中的数据透视表功能。

In [117]:

```
from IPython.display import Image

Image(
    filename=r"../data/透视表.png",
    width=600,
    height=600,
)
```

Out[117]:



	A	B	C
1	月份	类别	金额
2	一月	交通	$74.00
3	一月	日用杂货	$235.00
4	一月	日常开销	$175.00
5	一月	娱乐	$100.00
6	二月	交通	$115.00
7	二月	日用杂货	$240.00
8	二月	日常开销	$225.00
9	二月	娱乐	$125.00
10	三月	交通	$90.00
11	三月	日用杂货	$260.00
12	三月	日常开销	$200.00
13	三月	娱乐	$120.00

金额	月份			
类别	一月	二月	三月	总计
娱乐	$100	$125	$120	$345
日用杂货	$235	$240	$260	$735
日常开销	$175	$225	$200	$600
交通	$74	$115	$90	$279
总计	$584	$705	$670	$1,959

In [118]:

```
df = pd.read_excel("../data/透视表.xlsx")
df
```

Out[118]:

	月份	类别	金额
0	一月	交通	74
1	一月	日用杂货	235
2	一月	日常开销	175
3	一月	娱乐	100
4	二月	交通	115
5	二月	日用杂货	240
6	二月	日常开销	225
7	二月	娱乐	125
8	三月	交通	90
9	三月	日用杂货	260
10	三月	日常开销	200
11	三月	娱乐	120

In [119]:

```
df.head(2)
```

Out[119]:

	月份	类别	金额
0	一月	交通	74
1	一月	日用杂货	235

In [120]:

```
df.columns
```

Out[120]:

```
Index(['月份', '类别', '金额'], dtype='object')
```

In [121]:

```
aggResult = df.groupby(by=["月份", "类别"])["金额"].agg(
                    [
                     ("开支", np.sum)
        #           , ('平均月薪',np.mean)
        #           , ('最高月薪',np.max)
        #           , ('最低月薪',np.min)
                     ]
)
aggResult
```

Out[121]:

		开支
月份	**类别**	
一月	**交通**	74
	娱乐	100
	日常开销	175
	日用杂货	235
三月	**交通**	90
	娱乐	120
	日常开销	200
	日用杂货	260
二月	**交通**	115
	娱乐	125
	日常开销	225
	日用杂货	240

透视表 `pd.pivot_table(data,values,index,columns,agg,func,fill_value,margins)`

- `data`： 要应用透视表的数据框。
- `values`： 待聚合的列名，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`： 聚合函数或函数列表，默认为'mean'，可以是任何对groupby有效的函数。
- `fill_value`： 用于替换结果表中的缺失值。
- `margins`： 添加行/列小计和总计，默认为False。

在交叉分析前，先用cut()函数确定交叉分析中的分层，然后再利用pivot_table()函数实现交叉分析。

In [122]:

```
ptResult1 = pd.pivot_table(df
                           ,values=["金额"]
                           ,index=["类别"]
                           ,columns=["月份"]
                           ,aggfunc=[np.sum]
                           ,margins=True
    #                      ,fill_value='无'
)
ptResult1
```

Out[122]:

	sum			
	金额			
月份	**一月**	**三月**	**二月**	**All**
类别				
交通	74	90	115	279
娱乐	100	120	125	345
日常开销	175	200	225	600
日用杂货	235	260	240	735
All	584	670	705	1959

In [123]:

```
df = pd.read_excel("../data/Employee_income.xls", sheet_name="emp_income")
df.head(3)
```

Out[123]:

	emp_id	sex	age	education	firstjob	region	industry	occupation	salary	subsidy
0	30	男	30	本科	2011-07-20	广州	机械	技术员	5000	500
1	154	男	23	高中	2014-06-23	广州	机械	操作工	2500	1500
2	40	女	28	大专	2011-07-20	广州	机械	文员	3800	200

In [124]:

```
age_bins = [20, 30, 40, 50, 60, 70]
age_labels = ["20-29岁", "30-39岁", "40-49岁", "50-59岁", "60-69岁"]
df["年龄分层"] = pd.cut(df.age, age_bins, right=False, labels=age_labels)
df.head(3)
```

Out[124]:

	emp_id	sex	age	education	firstjob	region	industry	occupation	salary	subsidy	年龄分层
0	30	男	30	本科	2011-07-20	广州	机械	技术员	5000	500	30-39岁
1	154	男	23	高中	2014-06-23	广州	机械	操作工	2500	1500	20-29岁
2	40	女	28	大专	2011-07-20	广州	机械	文员	3800	200	20-29岁

(1)对年龄（age）和性别（sex）数据列进行交叉分析，求解不同年龄段中不同性别的平均年龄（人数）。

In [125]:

```
ptResult1 = pd.pivot_table(
    df,
    values=["age"],
    index=["年龄分层"],
    columns=["sex"],
    aggfunc=[np.size],
    margins=True,
    # fill_value="无",
)
ptResult1
```

Out[125]:

	size		
	age		
sex	女	男	All
年龄分层			
20-29岁	3.0	3.0	6
30-39岁	5.0	6.0	11
40-49岁	NaN	1.0	1
All	8.0	10.0	18

In [126]:

```
ptResult1[0:2]
```

Out[126]:

	size		
	age		
sex	女	男	All
年龄分层			
20-29岁	3.0	3.0	6
30-39岁	5.0	6.0	11

In [127]:

```
ptResult1.columns
```

Out[127]:

```
MultiIndex([('size', 'age',   '女'),
            ('size', 'age',   '男'),
            ('size', 'age', 'All')],
           names=[None, None, 'sex'])
```

In [128]:

```
ptResult1[("size", "age", "男")]
```

Out[128]:

```
年龄分层
20-29岁     3.0
30-39岁     6.0
40-49岁     1.0
All        10.0
Name: (size, age, 男), dtype: float64
```

In [129]:

```
ptResult1.loc["30-39岁", ("size", "age", "女")]
```

Out[129]:

```
5.0
```

(2) 对年龄（age）和学历（education）数据列进行交叉分析，求解不同年龄段中不同学历员工的平均月薪。

```
data.pivot_table(values,index,columns,agg,func,fill_value,margins)
```

In [130]:

```
# 分组统计人数、平均月薪、最高月薪和最低月薪
aggResult = df.groupby(by=["年龄分层"])["salary"].agg(
                    {  ("人数", np.size)
                     , ("平均月薪", np.mean)
                     , ("最高月薪", np.max)
                     , ("最低月薪", np.min)
                    }
)
aggResult
```

Out[130]:

年龄分层	最低月薪	平均月薪	人数	最高月薪
20-29岁	2000.0	2800.000000	6	3800.0
30-39岁	2500.0	4254.545455	11	6500.0
40-49岁	7699.0	7699.000000	1	7699.0
50-59岁	NaN	NaN	0	NaN
60-69岁	NaN	NaN	0	NaN

In [131]:

```
ptResult2 = pd.pivot_table(df,
                           values=["salary"],
                           index=["年龄分层"],
                           columns=["education"],
                           aggfunc={
                                   ('薪资均值',np.mean)
                                  ,('薪资最高值',np.max)
                                   }
                          # ,margins=True
                            ,fill_value=0
)
ptResult2
```

Out[131]:

	salary							
	薪资均值				薪资最高值			
education	大专	本科	研究生	高中	大专	本科	研究生	高中
年龄分层								
20-29岁	3400	0	0	2500.000000	3800	0	0	3000
30-39岁	4000	4700	6500	2933.333333	4500	5000	6500	3500
40-49岁	0	0	7699	0.000000	0	0	7699	0

绘制分组条形图：横坐标年龄分层，按纵学历分组绘制最大值

In [132]:

```
type(ptResult2)
```

Out[132]:

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

In [133]:

```
ptResult2.shape
```

Out[133]:

```
(3, 8)
```

In [134]:

```
ptResult2.columns
```

Out[134]:

```
MultiIndex([('salary',  '薪资均值',  '大专'),
            ('salary',  '薪资均值',  '本科'),
            ('salary',  '薪资均值', '研究生'),
            ('salary',  '薪资均值',  '高中'),
            ('salary', '薪资最高值',  '大专'),
            ('salary', '薪资最高值',  '本科'),
            ('salary', '薪资最高值', '研究生'),
            ('salary', '薪资最高值',  '高中')],
           names=[None, None, 'education'])
```

In [135]:

```
ptResult2[[("salary", "薪资均值", "大专")]]
```

Out[135]:

	salary
	薪资均值
education	大专
年龄分层	
20-29岁	3400
30-39岁	4000
40-49岁	0

In [136]:

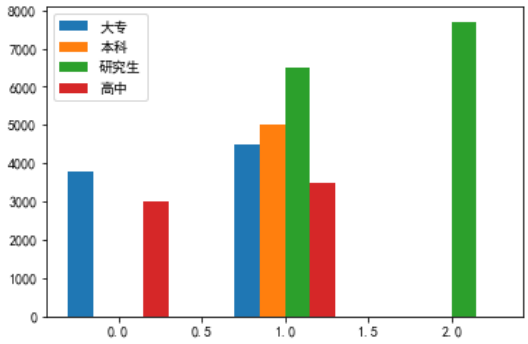
```
import matplotlib.pyplot as plt
plt.rcParams["font.sans-serif"] = ["SimHei"]
plt.rcParams["axes.unicode_minus"] = False  # 显示中文属性设置
```

In [137]:

```
x = np.arange(3)
bar_width = 0.15
plt.figure()
plt.bar(x - 1 * bar_width,
        ptResult2.iloc[:, 4], width=-bar_width, align="edge", label="大专")
plt.bar(x, ptResult2.iloc[:, 5], width=-bar_width, align="edge", label="本科")
plt.bar(x, ptResult2.iloc[:, 6], width=bar_width, align="edge", label="研究生")
plt.bar(x + 1 * bar_width,
        ptResult2.iloc[:, 7], align="edge", width=bar_width, label="高中")
plt.legend()
```

Out[137]:

```
<matplotlib.legend.Legend at 0x2d8467a8b80>
```



(3) 对薪资（salary）和学历（education）数据列进行交叉分析，求解不同薪资段中不同学历员工的平均月薪。

In [138]:

```
salary_bins = [2000, 3000, 4000, 5000, 6000, 7000, 8000]
salary_labels = ["2000-3000元","3000-4000元","4000-5000元",
    "5000-6000元","6000-7000元","7000-8000元",]
df["工资段"] = pd.cut(df['salary'], salary_bins, right=False, labels=salary_labels)

saResult1 = pd.pivot_table(df,
                           values=["salary"],
                           index=["工资段", "年龄分层" ],
                           columns=["education"],
                           aggfunc=[np.mean],
                           margins=True,
                           fill_value="无",
)
saResult1
```

Out[138]:

		mean				
		salary				
	education	大专	本科	研究生	高中	All
工资段	年龄分层					
2000-3000元	20-29岁	无	无	无	2333.333333	2333.333333
	30-39岁	无	无	无	2650.0	2650.0
3000-4000元	20-29岁	3400.0	无	无	3000.0	3266.666667
	30-39岁	3500.0	无	无	3500.0	3500.0
4000-5000元	30-39岁	4500.0	4500.0	无	无	4500.0
5000-6000元	30-39岁	无	5000.0	无	无	5000.0
6000-7000元	30-39岁	无	无	6500.0	无	6500.0
7000-8000元	40-49岁	无	无	7699.0	无	7699.0
All		3700.0	4700.0	7099.5	2685.714286	3961.055556

2. 交叉表

交叉表（Cross-Tabulation，简称crosstab）是一种用于计算分组频率（size）的特殊透视表
pd.crosstab(待分组的行数据,待分组的列数据)

(3)不同性别的学历人数

In [139]:

```
df.head(3)
```

Out[139]:

	emp_id	sex	age	education	firstjob	region	industry	occupation	salary	subsidy	年龄分层	工资段
0	30	男	30	本科	2011-07-20	广州	机械	技术员	5000	500	30-39岁	5000-6000元
1	154	男	23	高中	2014-06-23	广州	机械	操作工	2500	1500	20-29岁	2000-3000元
2	40	女	28	大专	2011-07-20	广州	机械	文员	3800	200	20-29岁	3000-4000元

In [140]:

```
# 交叉表
ctResult = pd.crosstab(df["sex"], df["education"], margins=True)
ctResult
```

Out[140]:

education	大专	本科	研究生	高中	All
sex					
女	3	2	0	3	8
男	1	3	2	4	10
All	4	5	2	7	18

In [141]:

```
# 透视表方法
saResult1 = df.pivot_table(
                           values=["emp_id"],
                           index=["sex"],
                           columns=["education"],
                           aggfunc=[np.size],
                         #  margins=True,
                        #   fill_value="无",
)
saResult1
```

Out[141]:

	size			
	emp_id			
education	大专	本科	研究生	高中
sex				
女	3.0	2.0	NaN	3.0
男	1.0	3.0	2.0	4.0

In [142]:

```
# 分组聚合方法
age_result = df.groupby(by=["sex", "education"])["emp_id"].agg(
                        [
                          ("频数", np.size)
                        # ,('平均值',np.mean)
                        # ,('最高值',np.max)
                        # ,('最低值',np.min)
                        # ,('差值',diff_value)
                         ]
)
age_result
```

Out[142]:

		频数
sex	education	
女	大专	3
	本科	2
	高中	3
男	大专	1
	本科	3
	研究生	2
	高中	4

In [143]:

```
# 交叉表方法
ctResult = pd.crosstab(df["年龄分层"], df["education"])
ctResult
```

Out[143]:

education	大专	本科	研究生	高中
年龄分层				
20-29岁	2	0	0	4
30-39岁	2	5	1	3
40-49岁	0	0	1	0

In [144]:

```
# 透视表方法
saResult1 = df.pivot_table(
                           values=["emp_id"],
                           index=["年龄分层"],
                           columns=["education"],
                           aggfunc=[np.size],
                          margins=True,
                          fill_value=0,
)
saResult1
```

Out[144]:

	size				
	emp_id				
education	大专	本科	研究生	高中	All
年龄分层					
20-29岁	2	0	0	4	6
30-39岁	2	5	1	3	11
40-49岁	0	0	1	0	1
All	4	5	2	7	18

In [145]:

```
# 分组聚合方法
age_result = df.groupby(by=["年龄分层", "education"])["emp_id"].agg(
                        [
                            ("频数", np.size)
                        # ,('平均值',np.mean)
                        # ,('最高值',np.max)
                        # ,( '最低值',np.min)
                        # ,('差值',diff_value)
                        ]
)
age_result
```

Out[145]:

		频数
年龄分层	education	
20-29岁	大专	2.0
	本科	NaN
	研究生	NaN
	高中	4.0
30-39岁	大专	2.0
	本科	5.0
	研究生	1.0
	高中	3.0
40-49岁	大专	NaN
	本科	NaN
	研究生	1.0
	高中	NaN
50-59岁	大专	NaN
	本科	NaN
	研究生	NaN
	高中	NaN
60-69岁	大专	NaN
	本科	NaN
	研究生	NaN
	高中	NaN

5-结构分析

结构分析是在分组和交叉的基础上，计算各组成部分所占的比例，进而分析总体的内部特征的一种分析方法。
重点在于了解各部分占总体的比例,例如:求公司中不同学历员工所占的比例，产品在市场的占有率、股权结构等

在结构分析时，先利用pivot_table()函数进行数据透视表分析，
然后，指定axis参数对数据透视表按行或列进行计算聚合函数(add,sub,multiply,div;sum,mean,var,sd)
（**当axis=0时按列计算，axis=1时按行计算**）。

对年龄（age）和学历（education）数据列进行结构分析，不同年龄分层下各种学历的占比

In [146]:

```
# 年龄分布状况
age_bins = [20, 30, 40, 50, 60, 70]
age_labels = ["20-29岁", "30-39岁", "40-49岁", "50-59岁", "60-69岁"]
df["年龄分层"] = pd.cut(df.age, age_bins, right=False, labels=age_labels)
#透视表
ptResult = pd.pivot_table(df,
                          values=["age"],
                          index=["年龄分层"],
                          columns=["education"],
                          aggfunc=[np.size]
                         ,margins=True
                          ,fill_value=0
)
ptResult
```

Out[146]:

	size				
	age				
education	大专	本科	研究生	高中	All
年龄分层					
20-29岁	2	0	0	4	6
30-39岁	2	5	1	3	11
40-49岁	0	0	1	0	1
All	4	5	2	7	18

In [147]:

```
ptResult.index
```

Out[147]:

```
Index(['20-29岁', '30-39岁', '40-49岁', 'All'], dtype='object', name='年龄分层')
```

In [148]:

```
ptResult.columns
```

Out[148]:

```
MultiIndex([('size', 'age',  '大专'),
            ('size', 'age',  '本科'),
            ('size', 'age', '研究生'),
            ('size', 'age',  '高中'),
            ('size', 'age', 'All')],
           names=[None, None, 'education'])
```

In [149]:

```
ptResult.sum(axis=0)  # 对每一列求和
```

Out[149]:

```
           education
size  age  大专            8
           本科           10
           研究生           4
           高中           14
           All          36
dtype: int64
```

In [150]:

```
# 按行求占比(每一列特征除以除以每一行的总和)
a = ptResult.div(ptResult.loc['All',], axis=1)
a
# div的第一个参数是除法的分子，意思是按行把数据除以该列的总和。即得到某一个年龄分层下，学历占比。
```

Out[150]:

	size				
	age				
education	大专	本科	研究生	高中	All
年龄分层					
20-29岁	0.5	0.0	0.0	0.571429	0.333333
30-39岁	0.5	1.0	0.5	0.428571	0.611111
40-49岁	0.0	0.0	0.5	0.000000	0.055556
All	1.0	1.0	1.0	1.000000	1.000000

In [151]:

```
# 按行求占比(每一列特征除以除以每一行的总和)
a = ptResult.div(ptResult.sum(axis=0), axis=1)
a
# div的第一个参数是除法的分子，意思是按行把数据除以该列的总和。即得到某一个年龄分层下，学历占比。
```

Out[151]:

	size				
	age				
education	大专	本科	研究生	高中	All
年龄分层					
20-29岁	0.25	0.0	0.00	0.285714	0.166667
30-39岁	0.25	0.5	0.25	0.214286	0.305556
40-49岁	0.00	0.0	0.25	0.000000	0.027778
All	0.50	0.5	0.50	0.500000	0.500000

绘制饼图

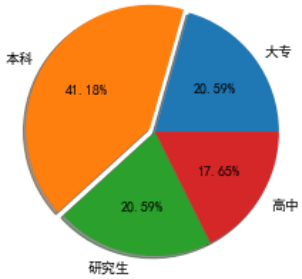
In [152]:

```
import matplotlib.pyplot as plt

plt.rcParams["font.sans-serif"] = ["SimHei"]
plt.rcParams["axes.unicode_minus"] = False  # 显示中文属性设置
```

In [153]:

```
plt.pie(
    a.iloc[-3, 0:4],
    labels=["大专", "本科", "研究生", "高中"],
    autopct="%.2f%%",
    shadow=True,
    explode=(0, 0.05, 0, 0),
)
plt.show()
```



In [154]:

```
# 按列求占比(每一行样本除以除以每一列的总和)
ptResult.div(ptResult.sum(axis=0), axis=1)
```

Out[154]:

	size				
	age				
education	大专	本科	研究生	高中	All
年龄分层					
20-29岁	0.25	0.0	0.00	0.285714	0.166667
30-39岁	0.25	0.5	0.25	0.214286	0.305556
40-49岁	0.00	0.0	0.25	0.000000	0.027778
All	0.50	0.5	0.50	0.500000	0.500000

对年龄（age）和性别（sex）数据列进行结构分析，不同年龄分层下男女占比

In [155]:

```
ptResult1 = df.pivot_table(
                          values=["age"]
                        , index=["年龄分层"]
                        , columns=["sex"]
                        , aggfunc=[np.size]
)
ptResult1
```

Out[155]:

	size	
	age	
sex	女	男
年龄分层		
20-29岁	3.0	3.0
30-39岁	5.0	6.0
40-49岁	NaN	1.0

In [156]:

```
ptResult1.sum(axis="index")
```

Out[156]:

```
          sex
size  age 女       8.0
          男      10.0
dtype: float64
```

In [157]:

```
ptResult1.sum(axis=1)
```

Out[157]:

```
年龄分层
20-29岁     6.0
30-39岁    11.0
40-49岁     1.0
dtype: float64
```

In [158]:

```
ptResult1.div(ptResult1.sum(axis=1), axis=0)
```

Out[158]:

	size	
	age	
sex	女	男
年龄分层		
20-29岁	0.500000	0.500000
30-39岁	0.454545	0.545455
40-49岁	NaN	1.000000

6-相关分析

相关分析（Correlation Analysis）用于研究现象之间是否存在某种依存关系，
并探讨具有依存关系的现象的相关方向以及相关程度,是研究随机变量之间相关关系的一种统计方法。
线性相关关系主要采用皮尔逊（Pearson）相关系数r来度量连续变量之间线性相关强度:
r>0，线性正相关；
r<0，线性负相关；
r=0，表示两个变量之间不存在线性关系，但并不代表两个变量之间不存在任何关系

In [159]:

```
%%html
<style>
table {float:left}
</style>
```

相关系数|r|取值范围	相关程度
0⩽|r|<0.3	低度相关
0.3⩽|r|<0.8	中度相关
0.8⩽|r|⩽1	高度相关

相关分析函数包括DataFrame.corr()和Series.corr(other):
计算相关系数的corr()函数只会对数据框中的数据列进行计算

- 如果由数据框调用corr()函数，那么将会计算列与列之间的相似度。
- 如果由序列调用corr()方法，那么只是该序列与传入的序列之间的相关度。函数返回值如下。
- DataFrame调用：返回DataFrame。
- Series调用：返回一个数值型数据，大小为相关度。

In [160]:

```
# 计算age和salary的相关系数
corrResult1 = df["age"].corr(df["salary"])
corrResult1
```

Out[160]:

0.6781676305144909

In [161]:

```
df.head(3)
```

Out[161]:

	emp_id	sex	age	education	firstjob	region	industry	occupation	salary	subsidy	年龄分层	工资段
0	30	男	30	本科	2011-07-20	广州	机械	技术员	5000	500	30-39岁	5000-6000元
1	154	男	23	高中	2014-06-23	广州	机械	操作工	2500	1500	20-29岁	2000-3000元
2	40	女	28	大专	2011-07-20	广州	机械	文员	3800	200	20-29岁	3000-4000元

In [162]:

```
# 计算age和salary、subsidy的相关系数
corrResult2 = df[["age", "salary", "subsidy"]].corr()
corrResult2
```

Out[162]:

	age	salary	subsidy
age	1.000000	0.678168	0.062137
salary	0.678168	1.000000	0.067629
subsidy	0.062137	0.067629	1.000000

In [163]:

```
# 返回一个相关系数矩阵
df.corr()
```

Out[163]:

	emp_id	age	salary	subsidy
emp_id	1.000000	-0.029849	-0.143666	0.043537
age	-0.029849	1.000000	0.678168	0.062137
salary	-0.143666	0.678168	1.000000	0.067629
subsidy	0.043537	0.062137	0.067629	1.000000

In [164]:

```
gra_data = pd.read_excel("../data/grade.xls", sheet_name="grade")
```

In [165]:

```
gra_data.head()
```

Out[165]:

	学号	平时成绩	期末成绩
0	2017001	80	66
1	2017002	87	67
2	2017003	85	60
3	2017004	80	55
4	2017005	76	44

In [166]:

```
gra_data.columns
```

Out[166]:

```
Index(['学号', '平时成绩', '期末成绩'], dtype='object')
```

In [167]:

```
gra_data.columns=['stu_id','Normal','exam']
gra_data
```

Out[167]:

	stu_id	Normal	exam
0	2017001	80	66
1	2017002	87	67
2	2017003	85	60
3	2017004	80	55
4	2017005	76	44
5	2017006	90	76
6	2017007	84	68
7	2017008	90	83
8	2017009	88	80
9	2017010	90	74
10	2017011	75	57
11	2017012	70	45
12	2017013	81	62
13	2017014	86	83
14	2017015	87	74
15	2017016	89	82
16	2017017	84	60
17	2017018	80	60
18	2017019	91	85
19	2017020	86	61
20	2017021	91	81
21	2017022	70	32
22	2017023	95	81
23	2017024	88	91
24	2017025	88	84
25	2017026	91	80
26	2017027	91	89
27	2017028	90	83
28	2017029	88	68
29	2017030	74	47
30	2017031	81	50
31	2017032	78	48
32	2017033	81	60
33	2017034	84	89
34	2017035	87	79
35	2017036	82	70
36	2017037	89	60
37	2017038	90	64

In [168]:

```
gra_data["总评"] = gra_data["Normal"] * 0.3 + gra_data["exam"] * 0.7
gra_data.head()
```

Out[168]:

	stu_id	Normal	exam	总评
0	2017001	80	66	70.2
1	2017002	87	67	73.0
2	2017003	85	60	67.5
3	2017004	80	55	62.5
4	2017005	76	44	53.6

In [169]:

```
corrResult1 = gra_data["stu_id"].corr(gra_data.exam)
corrResult1
```

Out[169]:

0.07893829895807297

In [170]:

```
corrResult2 = gra_data[["Normal", "exam","总评"]].corr()
corrResult2
```

Out[170]:

	Normal	exam	总评
Normal	1.000000	0.823087	0.869842
exam	0.823087	1.000000	0.996125
总评	0.869842	0.996125	1.000000

In [171]:

```
gra_data[["stu_id", "Normal", "exam", "总评"]].corr()
```

Out[171]:

	stu_id	Normal	exam	总评
stu_id	1.000000	0.117168	0.078938	0.086715
Normal	0.117168	1.000000	0.823087	0.869842
exam	0.078938	0.823087	1.000000	0.996125
总评	0.086715	0.869842	0.996125	1.000000

小结:

1. 描述性统计分析

- pd对象.describe() 描述性统计为：个数、均值、标准差、
  最小值、25%分位值、50%分位值、75%分位值，以及最大值
- pd对象.value_counts()：统计每一类数据出现的次数

2. 分组分析（类别型离散数据分组分析）

- data.groupby(by=['分组列1','分组列2'])[['统计列1','统计列2']].agg({

  ('聚合列名',聚合函数)
  })

3. 分布分析(连续型数据分组分析):
   先用cut()函数确定分布分析中的分层，再用groupby()分组分析得出结果

- pd.cut(data,bins,labels)

4. 交叉分析（透视表）

- ptResult = pd.pivot_table(data #data:完整的DataFram

  ,values=['金额'] #values：待统计列的列名
  ,index=['类别'] #index:交叉表行索引取值所在列名
  ,columns=['月份'] #columns:交叉表列索引取值所在列名
  ,aggfunc=[np.sum] #aggfunc：统计方法的函数名
  , margins=True #margins：是否添加行/列小计和总计
  ,fill_value='无' #fill_value：缺失值显示方式
)

5. 结构分析:
   先利用pivot_table()函数进行数据透视表分析;
   指定axis参数对数据透视表按行或列进行计算（axis=0：按列计算，axis=1：按行计算）

- 按列求占比(元素在列中的占比)：ptResult.div(ptResult.sum(axis=0), axis=1)
- 按行求占比(元素在行中的占比)：ptResult.div(ptResult.sum(axis=1), axis=0)

6. 相关分析：(系数矩阵中值越大，相关性越强，相关系数为1，则说明两列值相同)

- dataFrame[['列名1', '列名2','列名3']].corr()：返回多列之间的相关系数矩阵

In [ ]: