# Pandas数据分析

# 第3章 Pandas数据分析

目录
CONTENTE

# 第3章 Pandas数据分析

Python

# 3.4 数据帧的列操作和行操作

# 3.4.1 列操作

## （1）列选择

- 选择一列：df[列索引]
- 选择离散多列：df[[列索引数组]]
- 选择连续多列：df[df.columns[start:end:step]]

## [示例] 列选择

```
import pandas as pd

d = {'name':['小明','小花','小兰','小胜'],
     'gender':['男','女','女','男'],
     'age':[20,22,19,23],
     'clazz': ['1班','1班','2班','1班']}
df = pd.DataFrame(d)
print(df)

print(f'选择name列:\n{df['name'] }')

print(f'选择name,age列:\n{df[['name','age']] }')

print(f'选择前2列:\n{df[df.columns[:2]]}')
```

```
选择name列:
0    小明
1    小花
2    小兰
3    小胜
Name: name, dtype: object

选择name,age列:
  name  age
0   小明   20
1   小花   22
2   小兰   19
3   小胜   23

选择前2列:
  name gender
0   小明      男
1   小花      女
2   小兰      女
3   小胜      男
```

## （2）列添加

df[新列索引]= 新列值

**注意：**

- 若新列值未指定index，则从第0行开始往下添加新值；
- 若指定index，则按照index添加新值；
- 若添加的是series列，当值不足会自动添加NaN；
- 若添加的是列表、元组，当值不足会报错。

# [示例] 列的添加和修改

```
import pandas as pd

d = {'name':['小明','小花','小兰','小胜'],
     'gender':['男','女','女','男'],
     'age':[20,22,19,23],
     'clazz': ['1班','1班','2班','1班']}
df = pd.DataFrame(d)
print(df)

#df['city'] = ['北京','上海','深圳']
#print(df)

df['city'] = pd.Series(['北京','上海','深圳'],\
                       index=[0,1,3])
print(df)
```

```
  name gender  age clazz
0   小明      男   20    1班
1   小花      女   22    1班
2   小兰      女   19    2班
3   小胜      男   23    1班
```

```
  name gender  age clazz city
0   小明      男   20    1班   北京
1   小花      女   22    1班   上海
2   小兰      女   19    2班   深圳
3   小胜      男   23    1班  NaN
```

```
  name gender  age clazz city
0   小明      男   20    1班   北京
1   小花      女   22    1班   上海
2   小兰      女   19    2班  NaN
3   小胜      男   23    1班   深圳
```

## （3）列修改

df[原列索引] = 新值

注意事项与列添加一致

## [示例] 列修改

```
import pandas as pd
import numpy as np

d = {'name':['小明','小花','小兰','小胜'],
     'gender':['男','女','女','男'],
     'age':[20,22,19,23],
     'clazz': ['1班','1班','2班','1班']}
df = pd.DataFrame(d)

df['age'] = pd.Series([18,20,19,22])
print(df)

df['age'] = df['age'] + 1
print(df)
```

```
  name gender  age clazz
0   小明      男   18    1班
1   小花      女   20    1班
2   小兰      女   19    2班
3   小胜      男   22    1班
```

```
  name gender  age clazz
0   小明      男   19    1班
1   小花      女   21    1班
2   小兰      女   20    2班
3   小胜      男   23    1班
```

## （4）列删除

- 列删除指令，在原数据帧上删除指定的列，无返回值

  del df[列索引]

- 列删除函数，在原数据帧上删除指定列，并返回被删除的列

  df.pop(列索引)

- 列删除函数，默认在副本上删除指定列，并返回删除列之后的副本

  df.drop(columns=列索引)

## [示例] del命令删除列

```
import pandas as pd

d = {'name':['小明','小花','小兰','小胜'],
     'gender':['男','女','女','男'],
     'age':[20,22,19,23],
     'clazz': ['1班','1班','2班','1班']}
df = pd.DataFrame(d)
print(df)


del df['clazz']
print('删除clazz列后：')
print(df)
```

```
  name gender  age clazz
0   小明      男   20    1班
1   小花      女   22    1班
2   小兰      女   19    2班
3   小胜      男   23    1班

删除clazz列后：
  name gender  age
0   小明      男   20
1   小花      女   22
2   小兰      女   19
3   小胜      男   23
```

## [示例] pop函数删除列

```
import pandas as pd
d = {'name':['小明','小花','小兰','小胜'],
     'gender':['男','女','女','男'],
     'age':[20,22,19,23],
     'clazz': ['1班','1班','2班','1班']}
df = pd.DataFrame(d)
print(df)

t = df.pop('clazz')
print('弹出clazz列后：')
print(df)

print('弹出的列是：')
print(t)
```

```
  name gender  age clazz
0   小明      男   20    1班
1   小花      女   22    1班
2   小兰      女   19    2班
3   小胜      男   23    1班

弹出clazz列后：
  name gender  age
0   小明      男   20
1   小花      女   22
2   小兰      女   19
3   小胜      男   23

弹出的列是：
0    1班
1    1班
2    2班
3    1班
Name: clazz, dtype: object
```

## [示例] drop函数删除列

```
import pandas as pd
d = {'name':['小明','小花','小兰','小胜'],
     'gender':['男','女','女','男'],
     'age':[20,22,19,23],
     'clazz': ['1班','1班','2班','1班']}
df = pd.DataFrame(d)

df.drop(columns='clazz')
print(df)  # 默认drop后返回新对象，原df未变

df = df.drop(columns='clazz')
print(df)  # 将新对象赋值给df，df才变

df.drop(columns='age', inplace=True)
print(df)  # 修改inplace参数，原地drop
```

```
  name gender  age clazz
0   小明      男   20    1班
1   小花      女   22    1班
2   小兰      女   19    2班
3   小胜      男   23    1班
```

```
  name gender  age
0   小明      男   20
1   小花      女   22
2   小兰      女   19
3   小胜      男   23
```

```
  name gender
0   小明      男
1   小花      女
2   小兰      女
3   小胜      男
```

# 3.4.2 行操作

## （1）行选择

- 使用行索引选择行

  df.loc[行索引]、 df.loc[行索引列表]、df.loc[行索引切片]

- 使用行序号（0~n-1）选择行

  df.iloc[行序号] 、 df.iloc[行序号列表]、df.iloc[行序号切片]

- 使用行索引和行序号切片选择行

  df[行索引切片]、 df[行序号切片]

## [示例] loc的用法

```
import pandas as pd
d = {'name':['小明','小花','小兰','小胜'],
     'gender':['男','女','女','男'],
     'age':[20,22,19,23],
     'clazz': ['1班','1班','2班','1班']}

df = pd.DataFrame(d, \
     index=['first','second','third','fourth'])
print(df)

print('选择second行：')
print( df.loc['second'] )

print('选择first和third行：')
print( df.loc[['first','third']])
print( df.loc['first':'third':2]) #结果与上句相同
```

```
        name gender  age clazz
first     小明      男   20    1班
second    小花      女   22    1班
third     小兰      女   19    2班
fourth    小胜      男   23    1班

选择second行：
name      小花
gender     女
age       22
clazz     1班
Name: second, dtype: object

选择first和third行：
       name gender  age clazz
first    小明      男   20    1班
third    小兰      女   19    2班
```

## [示例] iloc的用法

```
import pandas as pd
d = {'name':['小明','小花','小兰','小胜'],
     'gender':['男','女','女','男'],
     'age':[20,22,19,23],
     'clazz': ['1班','1班','2班','1班']}

df = pd.DataFrame(d,\
     index=['first','second','third','fourth'])
print(df)

print('选择第1行（从0开始）：')
print( df.iloc[1] )

print('选择0，2行：')
print( df.iloc[[0,2]] )

print('选择0:2行：')
print( df.iloc[0:2] )
```

```
        name gender  age clazz
first     小明      男   20    1班
second    小花      女   22    1班
third     小兰      女   19    2班
fourth    小胜      男   23    1班

选择第1行（从0开始）：
name      小花
gender     女
age       22
clazz     1班
Name: second, dtype: object

选择0，2行:
        name gender  age clazz
first     小明      男   20    1班
third     小兰      女   19    2班

选择0:2行:
        name gender  age clazz
first     小明      男   20    1班
second    小花      女   22    1班
```

## [示例] 行索引切片和行位置切片

```
import pandas as pd

d1 = {'name':['小明','小花','小兰','小胜'],
      'gender':['男','女','女','男'],
      'age':[20,22,19,23],
      'clazz': ['1班','1班','2班','1班']}
df1 = pd.DataFrame(d1,\
       index=['first','second','third','fourth'])
```

| | name | gender | age | clazz |
|---|---|---|---|---|
| first | 小明 | 男 | 20 | 1班 |
| second | 小花 | 女 | 22 | 1班 |
| third | 小兰 | 女 | 19 | 2班 |

```
# 用行索引进行切片，返回的是*前闭后闭*的DataFrame
print(df1['first':'third'])
```

| | name | gender | age | clazz |
|---|---|---|---|---|
| second | 小花 | 女 | 22 | 1班 |
| third | 小兰 | 女 | 19 | 2班 |

```
# 用行位置索引进行切片，返回的是*前闭后开*的DataFrame
print(df1[1:3])
```

## （2）行添加

df._append(other, ignore_index=False, verify_integrity=False, sort=False)

- other： 指定要添加的数据，DataFrame 或 Series 对象，或这些对象的列表
  ignore_index： 如果为 True， 则重新进行0~n-1索引
- verify_integrity： 如果为 True ， 则遇到重复索引内容时报错
- sort： 当 sort=False（默认值）时，追加后的数据帧会保持数据帧的列顺序；当 sort=True 时，追加后的数据帧会按列索引的字典序排序。

# [示例] 行添加（添加的行若没有索引名，则用0~N-1作为新的数据帧的行索引名）

```
import pandas as pd
d = {'name':['小明','小花','小兰','小胜'],
     'gender':['男','女','女','男'],
     'age':[20,22,19,23],
     'clazz': ['1班','1班','2班','1班']}

df = pd.DataFrame(d,\
     index=['first','second','third','fourth'])
print(df)

s = pd.Series(['小丽','女','21','3班'], \
      index=['name','gender','age','clazz'])
# 添加无行索引名的s，必须设置ignore_index为True
df1= df._append(s, ignore_index=True)
print('添加Series行后的df1：')
print(df1)
```

```
        name gender  age clazz
first     小明     男   20    1班
second    小花     女   22    1班
third     小兰     女   19    2班
fourth    小胜     男   23    1班
```

添加Series行后的df1：

```
  name gender age clazz
0   小明     男  20    1班
1   小花     女  22    1班
2   小兰     女  19    2班
3   小胜     男  23    1班
4   小丽     女  21    3班
```

## [示例] 行添加（添加的行有name，则用name作为新添加行的索引，原索引不变）

```
import pandas as pd
d = {'name':['小明','小花','小兰','小胜'],
     'gender':['男','女','女','男'],
     'age':[20,22,19,23],
     'clazz': ['1班','1班','2班','1班']}

df = pd.DataFrame(d, \
     index=['first','second','third','fourth'])
print(df)

s = pd.Series(['小丽','女','21','3班'],\
     index=['name','gender','age','clazz'],\
     name='five')
df1= df._append(s)  # s有name，ignore_index才可缺省
print('添加Series行后的df1：')
print(df1)
```

```
        name gender  age clazz
first     小明     男   20    1班
second    小花     女   22    1班
third     小兰     女   19    2班
fourth    小胜     男   23    1班

添加Series行后的df1：
        name gender age clazz
first     小明     男  20    1班
second    小花     女  22    1班
third     小兰     女  19    2班
fourth    小胜     男  23    1班
five      小丽     女  21    3班
```

## [示例] 行添加（设置sort参数为True，添加后数据帧的列索引被排序）

```
import pandas as pd
d = {'name':['小明','小花','小兰','小胜'],
     'gender':['男','女','女','男'],
     'age':[20,22,19,23],
     'clazz': ['1班','1班','2班','1班']}

df = pd.DataFrame(d, \
     index=['first','second','third','fourth'])
print(df)

s = pd.Series([30,'女','小云','3班'],\
      index=['age','gender','name','clazz'],\
      name='five')
df1= df._append(s)
print(df1)
df2= df._append(s,sort=True)
print(df2)
```

| | name | gender | age | clazz |
|---|---|---|---|---|
| first | 小明 | 男 | 20 | 1班 |
| second | 小花 | 女 | 22 | 1班 |
| third | 小兰 | 女 | 19 | 2班 |
| fourth | 小胜 | 男 | 23 | 1班 |

| | name | gender | age | clazz |
|---|---|---|---|---|
| first | 小明 | 男 | 20 | 1班 |
| second | 小花 | 女 | 22 | 1班 |
| third | 小兰 | 女 | 19 | 2班 |
| fourth | 小胜 | 男 | 23 | 1班 |
| five | 小云 | 女 | 30 | 3班 |

| | age | clazz | gender | name |
|---|---|---|---|---|
| first | 20 | 1班 | 男 | 小明 |
| second | 22 | 1班 | 女 | 小花 |
| third | 19 | 2班 | 女 | 小兰 |
| fourth | 23 | 1班 | 男 | 小胜 |
| five | 30 | 3班 | 女 | 小云 |

## [示例] 行添加（若追加的行中存在原数据没有的列，会新增一列，并用nan填充）。

```
import pandas as pd

d = {'name':['小明','小花','小兰','小胜'],
     'gender':['男','女','女','男'],
     'age':[20,22,19,23],
     'clazz': ['1班','1班','2班','1班']}

df = pd.DataFrame(d,\
     index=['first','second','third','fourth'])
print(df)

s = pd.Series(['小丽','女','21','3班',95],\
     index=['name','gender','age','clazz','score'],\
     name='five')
df1= df._append(s)
print('添加Series行后的df1：')
print(df1)
```

```
        name gender  age clazz
first     小明     男   20    1班
second    小花     女   22    1班
third     小兰     女   19    2班
fourth    小胜     男   23    1班

添加Series行后的df1：
        name gender age clazz  score
first     小明     男  20    1班    NaN
second    小花     女  22    1班    NaN
third     小兰     女  19    2班    NaN
fourth    小胜     男  23    1班    NaN
five      小丽     女  21    3班   95.0
```

NaN被定义为浮点型。一列中有不同类型的值时，会自动同一为高类型的值。

## [示例] 行添加（若追加的行数据中缺少原数据某列，也以nan填充）。

```
import pandas as pd
d = {'name':['小明','小花','小兰','小胜'],
     'gender':['男','女','女','男'],
     'age':[20,22,19,23],
     'clazz': ['1班','1班','2班','1班']}

df = pd.DataFrame(d,\
     index=['first','second','third','fourth'])
print(df)

s = pd.Series(['小丽','女','21'],\
      index=['name','gender','age'],\
      name='five')
df1= df._append(s)
print('添加Series行后的df1：')
print(df1)
```

```
        name gender  age clazz
first     小明     男   20    1班
second    小花     女   22    1班
third     小兰     女   19    2班
fourth    小胜     男   23    1班

添加Series行后的df1：
        name gender age clazz
first     小明     男  20    1班
second    小花     女  22    1班
third     小兰     女  19    2班
fourth    小胜     男  23    1班
five      小丽     女  21   NaN
```

[示例] 行添加（在原数据帧后添加另一个数据帧）。

```
import pandas as pd

d1 = {'name':['小明','小花','小兰','小胜'],
      'gender':['男','女','女','男'],
      'age':[20,22,19,23],
      'clazz': ['1班','1班','2班','1班']}
df1 = pd.DataFrame(d1, \
      index=['first','second','third','fourth'])
d2 = {'name':['小丽','小刚'],
      'gender':['女','女'],
      'age':[ 21 ,19],
      'clazz': ['3班','2班']}
df2 = pd.DataFrame(d2, index=['fifth','sixth'])

df3 = df1._append(df2)
print('添加df2后的df1：')
print(df3)
```

添加df2后的df1：

| | name | gender | age | clazz |
|---|---|---|---|---|
| first | 小明 | 男 | 20 | 1班 |
| second | 小花 | 女 | 22 | 1班 |
| third | 小兰 | 女 | 19 | 2班 |
| fourth | 小胜 | 男 | 23 | 1班 |
| fifth | 小丽 | 女 | 21 | 3班 |
| sixth | 小刚 | 女 | 19 | 2班 |

## （3）行删除

DataFrame.drop(labels=None,axis=0, index=None, columns=None, inplace=False)

参数说明：

labels：要删除的行列的名字，多个则用**列表**给定

**axis：默认为0，指删除行**，因此删除columns时要指定axis=1；

index：直接指定要删除的行

columns：直接指定要删除的列

inplace：是否改变原数据帧，默认不改变。

**因此，删除行列有两种方式：**

**1）labels和axis的组合**

**2）index或columns直接指定要删除的行或列**

## [示例] 行和列的删除

```
import pandas as pd
d = {'name':['小明','小花','小兰','小胜'],
     'gender':['男','女','女','男'],
     'age':[20,22,19,23],
     'clazz': ['1班','1班','2班','1班']}

df = pd.DataFrame(d,\
     index=['first','second','third','fourth'])
print(df)

df2 = df.drop(labels='first',axis=0)  # 删除指定行
print('删除行后的df：')
print(df2)

df3 = df.drop(labels='gender',axis=1) # 删除指定列
print('删除列后的df：')
print(df3)
```

```
        name gender  age clazz
first     小明      男   20    1班
second    小花      女   22    1班
third     小兰      女   19    2班
fourth    小胜      男   23    1班

删除行后的df：
        name gender  age clazz
second    小花      女   22    1班
third     小兰      女   19    2班
fourth    小胜      男   23    1班

删除列后的df：
        name  age clazz
first     小明   20    1班
second    小花   22    1班
third     小兰   19    2班
fourth    小胜   23    1班
```

## [示例] 行和列的删除

```
import pandas as pd
d = {'name':['小明','小花','小兰','小胜'],
     'gender':['男','女','女','男'],
     'age':[20,22,19,23],
     'clazz': ['1班','1班','2班','1班']}

df = pd.DataFrame(d,\
     index=['first','second','third','fourth'])
print(df)

df2 = df.drop(index='first') # 删除指定行
print('删除行后的df：')
print(df2)

df3 = df.drop(columns='gender') # 删除指定列
print('删除列后的df：')
print(df3)
```

```
        name gender  age clazz
first     小明      男   20    1班
second    小花      女   22    1班
third     小兰      女   19    2班
fourth    小胜      男   23    1班

删除行后的df：
        name gender  age clazz
second    小花      女   22    1班
third     小兰      女   19    2班
fourth    小胜      男   23    1班

删除列后的df：
        name  age clazz
first     小明   20    1班
second    小花   22    1班
third     小兰   19    2班
fourth    小胜   23    1班
```

# 3.4.3 行列联合操作

可以先选行再选列，也可以先选列再选行。

常用的方式如下两例，利用iloc或loc选行再选列。

## [例] 按索引序号选择数据帧中的数据

```
import pandas as pd
d = {'name':['小明','小花','小兰','小胜'],
     'gender':['男','女','女','男'],
     'age':[20,22,19,23],
     'clazz': ['1班','1班','2班','1班']}
df = pd.DataFrame(d,\
     index=['first','second','third','fourth'])
print(df)

df2 = df.iloc[:2, [0,2]] # 取连续行，离散列
print(df2)

df3 = df.iloc[:2, :2] # 取连续行，连续列
print(df3)

df4 = df.iloc[[1,3],[1,2]] # 取离散行，离散列
print(df4)
```

```
        name  gender  age clazz
first     小明     男    20    1班
second    小花     女    22    1班
third     小兰     女    19    2班
fourth    小胜     男    23    1班

        name  age
first     小明   20
second    小花   22

        name gender
first     小明     男
second    小花     女

        gender  age
second      女    22
fourth      男    23
```

## [例] 按索引名选择数据帧中的数据

```
import pandas as pd
d = {'name':['小明','小花','小兰','小胜'],
     'gender':['男','女','女','男'],
     'age':[20,22,19,23],
     'clazz': ['1班','1班','2班','1班']}
df = pd.DataFrame(d,\
     index=['first','second','third','fourth'])
print(df)
# 取连续行，离散列
df2 = df.loc['second':'fourth',['name','gender']]
print(df2)
# 取连续行，连续列
df3 = df.loc['second':'fourth','name':'age']
print(df3)
# 取离散行，离散列
df4 = df.loc[['second','fourth'],['name','age']]
print(df4)
```

```
        name gender  age clazz
first     小明      男   20    1班
second    小花      女   22    1班
third     小兰      女   19    2班
fourth    小胜      男   23    1班
```

```
        name  gender
second    小花      女
third     小兰      女
fourth    小胜      男
```

```
        name gender  age
second    小花      女   22
third     小兰      女   19
fourth    小胜      男   23
```

```
        name  age
second    小花   22
fourth    小胜   23
```

# 第3章 Pandas数据分析

Python

## 3.5 高级索引

# 3.5.1 重建索引

## 1. reindex函数

根据新索引进行**重排**，并按指定的索引返回对应的内容（副本）。

DataFrame.reindex(labels=None, index=None, columns=None, axis=None, method=None, copy=True, level=None, **fill_value=nan**, limit=None, tolerance=None)

- index：创建后的行索引。
- columns：创建后的列索引。
- fill_value：缺失值（NaN）的替代值。
- limit：最大填充量。
- method：缺失值替代方法：pad或**ffill**（向前填充值）、bfill或backfill（向后填充值）、nearest（从最近的索引值填充）。

## [示例] 重建索引（原来没有的行/列填充NaN）

```
import numpy as np
import pandas as pd

np.random.seed(13)
data = np.random.randint(1,9,size=(4,5))
df = pd.DataFrame(data,index=[2,4,1,3],\
                  columns=['a','c','e','d','b'])
print(df)

print('重建行索引之后:')
df1 = df.reindex(index=[1,2,4,3])
print(df1)

print('重建列索引之后:')
df2 = df.reindex(columns=['a','b','c','d','E'])
print(df2)
```

```
   a  c  e  d  b
2  3  1  3  1  7
4  3  5  2  5  3
1  4  3  5  7  3
3  7  7  6  6  3
```

重建行索引之后:

```
   a  c  e  d  b
1  4  3  5  7  3
2  3  1  3  1  7
4  3  5  2  5  3
3  7  7  6  6  3
```

重建列索引之后:

```
   a  b  c  d   E
2  3  7  1  1 NaN
4  3  3  5  5 NaN
1  4  3  3  7 NaN
3  7  3  7  6 NaN
```

## [示例] 重建索引并填充0值

```
import numpy as np
import pandas as pd

np.random.seed(13)
data = np.random.randint(1,9,size=(4,5))
df = pd.DataFrame(data, index=[2,4,1,3], \
                        columns=['a','c','e','d','b'])
print(df)

print('重建列索引并填充0后:')
df2 = df.reindex(columns=['a','b','c','d','E'],\
                 fill_value=0)
print(df2)
```

```
   a  c  e  d  b
2  3  1  3  1  7
4  3  5  2  5  3
1  4  3  5  7  3
3  7  7  6  6  3
```

重建列索引并填充0后:

```
   a  b  c  d  E
2  3  7  1  1  0
4  3  3  5  5  0
1  4  3  3  7  0
3  7  3  7  6  0
```

## [示例] 重建行索引并前向（根据前一行的值）填充

```
import numpy as np
import pandas as pd

np.random.seed(13)
data = np.random.randint(1,9,size=(5,5))
df = pd.DataFrame(data,index=[2,4,1,3,0],\
                  columns=['a','c','e','d','b'])
print(df)

# 行索引无序，先重建行索引，使之有序
df1 = df.reindex(index=[0,1,2,3,4])
# df1有序，才可以前向填充
print("添加新行，前向填充：")
df3 = df1.reindex(index=[0,1,2,3,4,5,6],\
                  method='ffill')
print(df3)
```

```
   a  c  e  d  b
2  3  1  3  1  7
4  3  5  2  5  3
1  4  3  5  7  3
3  7  7  6  6  3
0  2  4  5  3  1
```

添加新行，前向填充：

```
   a  c  e  d  b
0  2  4  5  3  1
1  4  3  5  7  3
2  3  1  3  1  7
3  7  7  6  6  3
4  3  5  2  5  3
5  3  5  2  5  3
6  3  5  2  5  3
```

## [示例] 重建列索引并后向（根据后一列的值）填充

```
import numpy as np
import pandas as pd

np.random.seed(13)
data = np.random.randint(1,9,size=(4,5))
df = pd.DataFrame(data,index=[2,4,1,3],\
                          columns=[1,3,2,4,5])
print(df)

# 列索引可排序(如整数或浮点数)，才可前向或后向填充
df1 = df.reindex(columns=[1,2,3,4,5])
print('重建列索引之后:')
# 若后面没有列了，则填充NaN
df2 = df1.reindex(columns=[0,1,2,3,4,5,6],\
                  method='bfill')
print(df2)
```

```
   1  3  2  4  5
2  3  1  3  1  7
4  3  5  2  5  3
1  4  3  5  7  3
3  7  7  6  6  3

重建列索引之后:
   0  1  2  3  4  5  6
2  3  3  3  1  1  7 NaN
4  3  3  2  5  5  3 NaN
1  4  4  5  3  7  3 NaN
3  7  7  6  7  6  3 NaN
```

# 3.5.1 重建索引

2. rename函数

rename()方法实现**重命名**行索引或列索引（副本），通常使用原索引与新索引（原列名与新列名）组成的字典作为参数。

DataFrame.rename(mapper=None, index=None, columns=None, axis=None, copy=True, inplace=False, level=None, errors='ignore')

## [示例] 重命名列索引

```
import pandas as pd
import numpy as np
data = {'name':pd.Series(['小明','小花','小兰','小胜']),
        'gender':pd.Series(['男','女','女','男']),
        'age':pd.Series([20,22,19,23]),
        'clazz': pd.Series(['1班','1班','2班','1班'])}
df = pd.DataFrame(data)
print(df)
# 在副本上重命名指定的列索引
print('--------------------------')
print(df.rename(columns={'name':'a','clazz':'b'}))
print('--------------------------')
# 原地重命名所有列索引(修改df的columns成员的值)
df.columns = ['姓名','性别','年龄','班级']
print(df)
```

```
  name gender  age clazz
0   小明      男   20    1班
1   小花      女   22    1班
2   小兰      女   19    2班
3   小胜      男   23    1班
--------------------------
     a gender  age    b
0   小明      男   20   1班
1   小花      女   22   1班
2   小兰      女   19   2班
3   小胜      男   23   1班
--------------------------
  姓名 性别  年龄  班级
0  小明  男   20   1班
1  小花  女   22   1班
2  小兰  女   19   2班
3  小胜  男   23   1班
```

## [示例] 重命名行索引

```
import pandas as pd
import numpy as np
data = {'name':pd.Series(['小明','小花','小兰','小胜']),
        'gender':pd.Series(['男','女','女','男']),
        'age':pd.Series([20,22,19,23]),
        'clazz': pd.Series(['1班','1班','2班','1班'])}
df = pd.DataFrame(data)
print(df)
print('--------------------------')
# 在副本上重命名指定的行索引
print(df.rename(index={0:'*',1:'**'}))
print('--------------------------')
# 原地重命名所有行索引(修改df的index成员的值)
df.index = np.arange(1, df.shape[0]+1)
print(df)
```

```
  name gender  age clazz
0   小明      男   20    1班
1   小花      女   22    1班
2   小兰      女   19    2班
3   小胜      男   23    1班
--------------------------
  name gender  age clazz
*   小明      男   20    1班
**  小花      女   22    1班
2   小兰      女   19    2班
3   小胜      男   23    1班
--------------------------
  name gender  age clazz
1   小明      男   20    1班
2   小花      女   22    1班
3   小兰      女   19    2班
4   小胜      男   23    1班
```

# 3.5.2 重新设置索引

1. set_index方法

将列数据设置为行索引：

DataFrame.set_index(keys, drop=True, _append=False, inplace=False, verify_integrity=False)

- keys：**列索引**或**列索引列表**，表示需要设置为索引的列。
- drop：默认为True，删除用作新索引的列。
- _append：默认为False，是否将列附加到现有索引。
- inplace：默认为False，适当修改DataFrame(不要创建新对象)。
- verify_integrity：默认为false，检查新索引的副本。

## [示例] 设置某列为新的行索引

```
import pandas as pd
import numpy as np
data = {'name':pd.Series(['小明','小花','小兰','小胜']),
        'gender':pd.Series(['男','女','女','男']),
        'age':pd.Series([20,22,19,23]),
        'clazz': pd.Series(['1班','1班','2班','1班'])}
df = pd.DataFrame(data)
print(df)
# 设置单索引
df1 = df.set_index('name')   # name变为行索引
print(df1)
print(df1.loc['小兰'])  # 用新的行索引取行
```

```
  name gender  age clazz
0   小明      男   20    1班
1   小花      女   22    1班
2   小兰      女   19    2班
3   小胜      男   23    1班
       gender  age clazz
name
小明       男    20    1班
小花       女    22    1班
小兰       女    19    2班
小胜       男    23    1班

gender     女
age       19
clazz     2班
Name: 小兰, dtype: object
```

## [示例] 设置层次化索引

```
import pandas as pd
import numpy as np
data = {'name':['小明','小花','小兰','小胜'],
        'gender':['男','女','女','男'],
        'age':[20,22,19,23],
        'clazz':['1班','1班','2班','1班']}
df = pd.DataFrame(data)
print(df)
# 设置层次化索引
df2 = df.set_index(['clazz','name'])
print(df2)
print(df2.loc['1班','小花'])
print(df2.loc['1班'])
```

```
  name gender  age clazz
0   小明      男     20    1班
1   小花      女     22    1班
2   小兰      女     19    2班
3    小胜     男     23    1班

                gender  age
clazz name
1班    小明       男     20
       小花       女     22
2班    小兰       女     19
1班    小胜       男     23

gender     女
age         22
Name: (1班, 小花), dtype: object

          gender  age
name
小明        男     20
小花        女     22
小胜        男     23
```

**层次化索引**是Pandas的一个重要的功能，它可以在一个轴上有多个（两个或两个以上）索引，这就表示着，它能够以低维度形式来表示高维度的数据。

对于DataFrame来说，行和列都能够进行层次化索引。层次化索引通过二维数组进行设置，二维数组中每一行为一个索引，多行就是多个索引。

## [示例] 行层次化索引

```
import pandas as pd
import numpy as np
data = {'name':['小明','小花','小兰','小胜'],
        'gender':['男','女','女','男'],
        'age':[20,22,19,23],
        'clazz':['1班','1班','2班','1班']}

df = pd.DataFrame(data, \
     index=[['a组','a组','b组','a组'],\
            ['1号','2号','1号','3号']])
print(df)
print('------------------------------')
print(df.loc['a组'])
print('------------------------------')
print(df.loc['a组','2号'])
```

```
          name gender  age clazz
a组 1号   小明     男     20    1班
   2号   小花     女     22    1班
b组 1号   小兰     女     19    2班
a组 3号   小胜     男     23    1班
------------------------------
    name gender  age clazz
1号  小明     男     20    1班
2号  小花     女     22    1班
3号  小胜     男     23    1班
------------------------------
name      小花
gender     女
age       22
clazz     1班
Name: (a组, 2号), dtype: object
```

## [示例] 列层次化索引

```
import pandas as pd
import numpy as np

i = ['张三','王五','李四','马六']
c = [['python','python','java','java'],\
     ['平时','考试','平时','考试']]
data = np.random.randint(60,100, size=(4,4))
df = pd.DataFrame(data,index = i, columns=c)
print(df)
print('**********')
print('python分数:')
print(df['python'])
print('**********')
print('python总评分:')
print(df['python','平时']*0.3+df['python','考试']*0.7)
```

```
      python       java
      平时  考试   平时  考试
张三    73    94    83    92
王五    72    65    80    77
李四    95    75    71    70
马六    74    64    91    84
**********
python分数:
      平时  考试
张三    73    94
王五    72    65
李四    95    75
马六    74    64
**********
python总评分:
张三    87.7
王五    67.1
李四    81.0
马六    67.0
dtype: float64
```

## 2. reset_index 方法

1）数据清洗时，可能会将带空值的行删除，此时DataFrame或Series类型的数据不再是连续的索引，可以使用reset_index()重置索引。

2）重置索引后，原行索引可以保留为列数据或删除。

DataFrame.reset_index(level=None, drop=False, inplace=False)

- level：可以将层次化索引还原为普通索引，level控制了具体要还原的的索引等级 。
- drop为False则索引列会被还原为普通列，否则会丢失。

## [示例] 重置索引（原行索引变成index数据列，保留下来）

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.arange(12).reshape(4,3),\
                  index=[1,3,4,6])
print('df=\n',df,sep='')

df1 = df.reset_index()
print('\n新索引从0开始增1，连续排列：')
print(df1) # 原索引列变成列名为index的数据列
```

```
df=
   0  1  2
1  0  1  2
3  3  4  5
4  6  7  8
6  9  10 11

新索引从0开始增1，连续排列：
   index  0  1  2
0      1  0  1  2
1      3  3  4  5
2      4  6  7  8
3      6  9  10 11
```

## [示例] 重置索引(不保留原索引)

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.arange(12).reshape(4,3),\
                  index=[1,3,4,6])
print('df=\n',df,sep='')

df1 = df.reset_index(drop=True)
print('\n新索引从0开始增1，连续排列：')
print(df1)  # 原索引列被删除
```

```
df=
   0   1   2
1  0   1   2
3  3   4   5
4  6   7   8
6  9  10  11

新索引从0开始增1，连续排列：
   0   1   2
0  0   1   2
1  3   4   5
2  6   7   8
3  9  10  11
```

## [示例] 重置索引(还原层次化索引)

```
import pandas as pd
import numpy as np
data = {'name':['小明','小花','小兰','小胜'],
        'gender':['男','女','女','男'],
        'age':[20,22,19,23],
        'clazz':['1班','1班','2班','1班']}
df = pd.DataFrame(data)
print(df)
df2 = df.set_index(['clazz','name'])
print(df2)
df3 = df2.reset_index(level=0)
print(df3)
df4 = df2.reset_index(level=1)
print(df4)
df5 = df2.reset_index()
print(df5)
```

```
  name gender  age clazz
0   小明      男   20    1班
1   小花      女   22    1班
2   小兰      女   19    2班
3   小胜      男   23    1班
             gender  age
clazz name
1班    小明       男   20
       小花       女   22
2班    小兰       女   19
1班    小胜       男   23
       clazz gender  age
name
小明      1班      男    20
小花      1班      女    22
小兰      2班      女    19
小胜      1班      男    23
       name gender  age
clazz
1班      小明      男   20
1班      小花      女   22
2班      小兰      女   19
1班      小胜      男   23
   clazz  name gender age
0    1班   小明      男    20
1    1班   小花      女    22
2    2班   小兰      女    19
3    1班   小胜      男    23
```

# 第3章 Pandas数据分析

Python

# 3.6 Pandas数据运算

## 3.6.1 算术运算

- Pandas的**Series**数据对象在进行算术运算时，如果有相同的索引，则对相同索引的数据进行运算，如果没有相同索引，则引入缺失值。
- Pandas的**DataFrame**数据对象进行算术运算时，相同的行索引和列索引对应的数据进行运算，若没有相同的索引和列名，则引入缺失值。
- Pandas的**Series和DataFrame**对象也可以进行算术运算，因为维度不同，所以运算规则遵循广播规则，即Series对象根据DataFrame对象结构扩展为二维，变为多行的DataFrame，**每行数据都是Series本身**。

## [示例] Series的算术运算（索引相同）

```
import pandas as pd


s1 = pd.Series([1, 2, 3])
s2 = pd.Series([11,12,13])
print('s1=\n', s1, sep='')
print('s2=\n',s2, sep='')
print('s1+s2=\n', s1+s2, sep='')
```

```
s1=
0    1
1    2
2    3
dtype: int64
```

```
s2=
0    11
1    12
2    13
dtype: int64
```

```
s1+s2=
0    12
1    14
2    16
dtype: int64
```

## [示例] Series的算术运算（索引不相同，引入NaN值）

```
import pandas as pd


s3 = pd.Series([1, 2, 3], index=['a', 'b', 'c'])
s4 = pd.Series([11, 12, 13], index=['a', 'b', 'd'])
print('s3=\n', s3, sep='')
print('s4=\n', s4, sep='')
print('s3+s4=\n', s3+s4, sep='')
```

```
s3=
a    1
b    2
c    3
dtype: int64
```

```
s4=
a    11
b    12
d    13
dtype: int64
```

```
s3+s4=
a    12.0
b    14.0
c     NaN
d     NaN
dtype: float64
```

## [示例] Series的算术运算（索引不相同，add函数可引入填充值）

```
import pandas as pd

s3 = pd.Series([1, 2, 3], index=['a', 'b', 'c'])
s4 = pd.Series([11, 12, 13], index=['a', 'b', 'd'])

print(s3.add(s4, fill_value=4))
```

s3=

```
a    1
b    2
c    3
dtype: int64
```

s4=

```
a    11
b    12
d    13
dtype: int64
```

```
a    12.0
b    14.0
c     7.0
d    17.0
dtype: float64
```

## [示例] DataFrame算术运算（索引不相同，引入NaN值）

```
import pandas as pd

d1 = {'a':[1,2],'b':[3,4]}
df1 = pd.DataFrame(d1)
print('df1=\n', df1, sep='')

d2 = {'a':[11,12,13],'b':[13,14,15],'c':[15,16,17]}
df2 = pd.DataFrame(d2)
print('df2=\n', df2, sep='')

print('df1+df2=\n', df1+df2, sep='')
```

```
df1=
   a  b
0  1  3
1  2  4

df2=
    a    b   c
0  11  13  15
1  12  14  16
2  13  15  17

df1+df2=
     a     b    c
0  12.0  16.0  NaN
1  14.0  18.0  NaN
2   NaN   NaN  NaN
```

## [示例] Series和DataFrame进行算术运算

```
import pandas as pd

s = pd.Series([1,2,3],index=['a','b','c'])
print('s=\n', s, sep='')

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

print('df+s=\n', df+s, sep='')
```

```
s=
a    1
b    2
c    3
dtype: int64

df=
   a  b  c
0  1  3  5
1  2  4  6

df+s=
   a  b  c
0  2  5  8
1  3  6  9
```

s将自己的数据根据df行数广播，成为一个数据帧：

| | a | b | c |
|---|---|---|---|
| 0 | 1 | 2 | 3 |
| 1 | 1 | 2 | 3 |

# 3.6.2 排序

Pandas有两种排序方式，它们分别是：**按索引排序**和**按实际值排序**。

1. sort_index函数

sort_index(axis=0, level=None, ascending=True, inplace=False, kind='quicksort', na_position='last', sort_remaining=True, by=None)

- **axis**：0按照行名排序；1按照列名排序。
- level：默认None，否则按照给定的level顺序排列。
- **ascending**：默认True升序排列；False降序排列。
- inplace：默认False，否则排序之后的数据直接替换原来的数据集。
- kind：默认quicksort，排序的方法。
- na_position：缺失值默认排在最后{"first","last"}。
- **by**：按照哪一列数据进行排序。

## 2. sort_values函数

DataFrame.sort_values(by, axis=0, ascending=True, inplace=False, kind='quicksort', na_position='last')

- axis：坐标轴，取值为0（或‘index’）和1(或‘columns’)，**默认为0**，按照索引排序，即纵向排序，如果为1，则是横向排序。
- by：是一个字符串或字符串列表，如果axis=0，那么by="列名"；如果axis=1，那么by="行名"。
- ascending:布尔型，True则升序，可以是[True,False]，即第一字段升序，第二个降序。
- inplace:布尔型，是否用排序后的数据框替换现有的数据框。
- kind：排序方法，{‘quicksort’，‘mergesort’，‘heapsort’}，默认为“quicksort”。
- na_position：{‘first’，‘last’}, 默认值为“last”，默认缺失值排在最后。

## [示例] 按行索引排序

```
import pandas as pd
import numpy as np

data = np.array([[2,5,3,7],[16,14,2,16],[29,27,2,25]])
df = pd.DataFrame(data,index=[0,4,2],\
                  columns=['a','c','b','d'])
print('原数据帧：')
print(df)

print('行索引升序：')
print(df.sort_index())

print('行索引降序：')
print(df.sort_index(ascending=False))
```

原数据帧：

| | a | c | b | d |
|---|---|---|---|---|
| 0 | 2 | 5 | 3 | 7 |
| 4 | 16 | 14 | 2 | 16 |
| 2 | 29 | 27 | 2 | 25 |

行索引升序：

| | a | c | b | d |
|---|---|---|---|---|
| 0 | 2 | 5 | 3 | 7 |
| 2 | 29 | 27 | 2 | 25 |
| 4 | 16 | 14 | 2 | 16 |

行索引降序：

| | a | c | b | d |
|---|---|---|---|---|
| 4 | 16 | 14 | 2 | 16 |
| 2 | 29 | 27 | 2 | 25 |
| 0 | 2 | 5 | 3 | 7 |

## [示例] 按列索引排序

```
import pandas as pd
import numpy as np

data = np.array([[2,5,3,7],[16,14,2,16],[29,27,2,25]])
df = pd.DataFrame(data, \
                          index=[0,4,2],\
                          columns=['a','c','b','d'])
print('原数据帧：')
print(df)

print('按列索引升序：')
print(df.sort_index(axis=1))

print('按列索引降序：')
print(df.sort_index(ascending=False, axis=1))
```

**原数据帧：**

| | a | c | b | d |
|---|---|---|---|---|
| 0 | 2 | 5 | 3 | 7 |
| 4 | 16 | 14 | 2 | 16 |
| 2 | 29 | 27 | 2 | 25 |

**按列索引升序：**

| | a | b | c | d |
|---|---|---|---|---|
| 0 | 2 | 3 | 5 | 7 |
| 4 | 16 | 2 | 14 | 16 |
| 2 | 29 | 2 | 27 | 25 |

**按列索引降序：**

| | d | c | b | a |
|---|---|---|---|---|
| 0 | 7 | 5 | 3 | 2 |
| 4 | 16 | 14 | 2 | 16 |
| 2 | 25 | 27 | 2 | 29 |

## [示例] 按值排序

```
import pandas as pd
import numpy as np

data = np.array([[2,5,3,7],[16,27,2,16],[29,14,2,25]])
df = pd.DataFrame(data,index=[0,4,2],\
                  columns=['a','c','b','d'])
print('原数据帧：')
print(df)

print('按b列排序后：')
print(df.sort_values(by='b'))

print('先按b列排序，若b列有相同值，则按c列排序:')
print(df.sort_values(by=['b','c']))
```

原数据帧:

| | a | c | b | d |
|---|---|---|---|---|
| 0 | 2 | 5 | 3 | 7 |
| 4 | 16 | 27 | 2 | 16 |
| 2 | 29 | 14 | 2 | 25 |

按b列排序后:

| | a | c | b | d |
|---|---|---|---|---|
| 4 | 16 | 27 | 2 | 16 |
| 2 | 29 | 14 | 2 | 25 |
| 0 | 2 | 5 | 3 | 7 |

先按b列排序，若b列有相同值，则按c列排序:

| | a | c | b | d |
|---|---|---|---|---|
| 2 | 29 | 14 | 2 | 25 |
| 4 | 16 | 27 | 2 | 16 |
| 0 | 2 | 5 | 3 | 7 |

# 3.6.3 迭代

## 1. 迭代DataFrame的列

（1）DataFrame的数据本身就是**多列构成**的，迭代DataFrame的列可直接用**foreach**循坏。

（2）**items()**返回(key，value)对，将每个列名作为键，将列数据的Series对象作为值。

## [示例] DataFrame迭代

```
import pandas as pd
import numpy as np

data = np.array([[2,5,3],[16,14,2],[29,27,2]])
df = pd.DataFrame(data, \
                          index=[0,4,2],\
                          columns=['a','c','b'])
print(df)

for col in df:  # 取的是列名
    print(f'df[\'{col}\']=\n{df[col]}')
```

```
    a   c  b
0   2   5  3
4  16  14  2
2  29  27  2
```

```
col= a
df[col]=
0     2
4    16
2    29
Name: a, dtype: int32
```

```
col= c
df[col]=
0     5
4    14
2    27
Name: c, dtype: int32
```

```
col= b
df[col]=
0    3
4    2
2    2
Name: b, dtype: int32
```

## [示例] iteritems迭代列

```
import pandas as pd
import numpy as np

data = np.array([[2,5,3],[16,14,2],[29,27,2]])
df = pd.DataFrame(data,index=[0,4,2],\
                          columns=['a','c','b'])
print(df)

for col,value in df.items():
    print(f'df[\'{col}\']=\n{value}')
```

```
 a   c  b
0   2   5  3
4  16  14  2
2  29  27  2

df['a']=
0     2
4    16
2    29
Name: a, dtype: int32

df['c']=
0     5
4    14
2    27
Name: c, dtype: int32

df['b']=
0    3
4    2
2    2
Name: b, dtype: int32
```

# 3.6.3 迭代

## 2. 迭代DataFrame的行

遍历DataFrame的行可以使用以下函数。

**（1）iterrows()**

将行迭代为(索引，系列)对，产生每个行索引值以及包含每行数据的序列。

**（2）itertuples()**

以namedtuples的形式迭代行，其中的值是行的数据。

## [示例] iterrows迭代行

```
import pandas as pd
import numpy as np

data = np.array([[2,5,3],[16,14,2],[29,27,2]])
df = pd.DataFrame(data,index=[0,4,2],\
                       columns=['a','c','b'])
print(df)

for row_index, row in df.iterrows():
    print('row_index=', row_index)
    print(row)
```

```
    a   c  b
0   2   5  3
4  16  14  2
2  29  27  2
row_index= 0
a    2
c    5
b    3
Name: 0, dtype: int32
row_index= 4
a   16
c   14
b    2
Name: 4, dtype: int32
row_index= 2
a   29
c   27
b    2
Name: 2, dtype: int32
```

## [示例] itertuples迭代行

```
import pandas as pd
import numpy as np

data = np.array([[2,5,3],[16,14,2],[29,27,2]])
df = pd.DataFrame(data,index=[0,4,2],  columns=['a','c','b'])
print(df)

for row in df.itertuples():
    print(row)
```

```
    a   c  b
0   2   5  3
4  16  14  2
2  29  27  2
```

```
Pandas(Index=0, a=2, c=5, b=3)
Pandas(Index=4, a=16, c=14, b=2)
Pandas(Index=2, a=29, c=27, b=2)
```

# 3.6.4 唯一值与值计数

唯一值（unique）函数的作用是为**Series**数据去重，只留下不重复的元素组成的数组。 'DataFrame' object has no attribute 'unique'

值计数（value_counts）函数的作用是计算**Series中**每一个元素的个数，默认会自动按照计数值降序排列（从大到小排序）。

## [示例] 唯一值和值计数

```
import pandas as pd

s = s = pd.Series(['apple', 'banana', 'apple', 'orange', 'banana', 'banana'])
print(s)
print('s.unique=',s.unique())
print(s.value_counts())
```

```
0     apple
1    banana
2     apple
3    orange
4    banana
5    banana
dtype: object
```

```
s.unique= ['apple' 'banana' 'orange']
```

```
banana    3
apple     2
orange    1
Name: count, dtype: int64
```

## [实例] 检查数据中的唯一ID是否重复

```
import pandas as pd

# 用户ID Series
user_ids = pd.Series([101, 102, 103, 101, 104, 102])

# 检查是否有重复ID
if len(user_ids) != len(user_ids.unique()):
    print("存在重复的用户ID")
else:
    print("没有重复的用户ID")
```

存在重复的用户ID

## [示例] 行或列的唯一值和值计数

```
import pandas as pd


# 断行之后是字符串，可不加反斜杠续行
df = pd.DataFrame({'a':[1,2,3,4,3,2,2],
                                'b':[3,2,3,3,2,3,4]})
print('df=\n', df, sep='')
print('a列去重后：',df['a'].unique())
print('a列元素重复数：\n',df['a'].value_counts())
print('第1行去重后：', df.iloc[1].unique())
```

```
df=
   a  b
0  1  3
1  2  2
2  3  3
3  4  3
4  3  2
5  2  3
6  2  4
a列去重后：  [1 2 3 4]
a列元素重复数:
2    3
3    2
1    1
4    1
Name: a, dtype: int64
第1行去重后：  [2]
```

# END