

pandas安装

Anaconda 中自带 pandas 库，无需另行安装

Python标准版本，需要手动安装 pandas:

- 查看当前版本: `print(pd.__version__)`
- 安装最新的稳定版本: `pip install pandas`
- 安装指定版本: `pip install pandas==版本号`

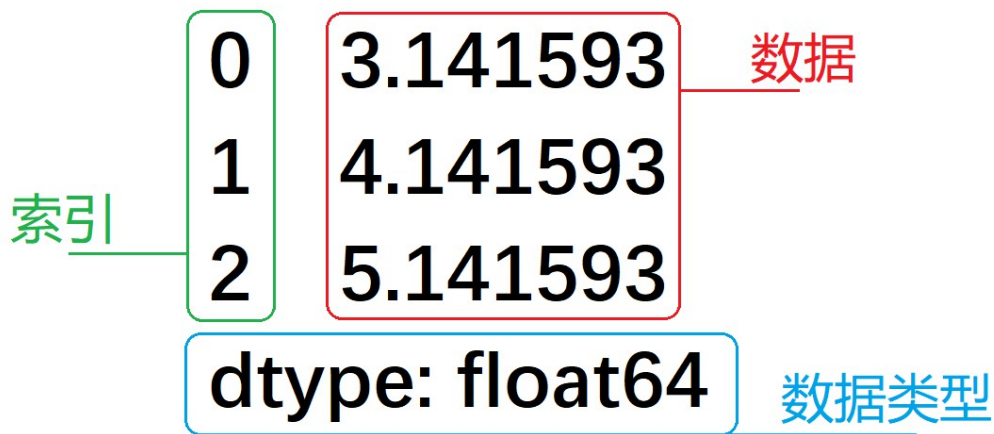
pandas简介

Pandas 是 Python 进行数据分析的一个扩展库，是基于 NumPy 的一种工具。能够快速得从不同格式的文件中加载数据（比如 CSV、Excel文件等），然后将其转换为可处理的对象。

Pandas 在 ndarray 的基础上构建出了两种更适用于数据分析的存储结构，分别是 Series（一维数据结构）和 DataFrame（二维数据结构）。在操作 Series 和 DataFrame 时，基本上可以看成是 NumPy 中的一维和二维数组来操作，数组的绝大多数操作它们都可以适用。

Pandas Series

- Series是一种一维数据结构，每一个元素都带有一个索引，与 NumPy 中的一维数组类似
- Series 可以保存任何数据类型，比如整数、字符串、浮点数、Python 对象等，它的索引默认为整数，从 0 开始依次递增。Series 的结构图，如下所示：



创建 **Series** 对象

`pd.Series(data=None, index=None, dtype=None, name=None)`

- **data**: array-like, dict, or scalar value
- **index**: 索引必须是不可变数据类型，允许相同。不指定时，默认为从 0 开始依次递增的整数
- **dtype**: 数据类型，如果没有指定，则会自动推断得出
- **name**: 设置 Series 的名称

```
import numpy as np
import pandas as pd

""" 创建一个空Series对象 """
ser = pd.Series()
print(ser)

""" 标量创建Series对象:
标量值按照 index 的数量进行重复，并与其一一对应
如果没有指定index，就只有一个数据 """
d = 99
ser = pd.Series(data=d)
print(ser)
ser = pd.Series(data=d, index=[1, 2, 3])
print(ser)
```

```

""" str创建Series对象：当作标量一样处理 """
d = 'abc'
ser = pd.Series(data=d, index=[1, 2, 3])
print(ser)

""" list创建Series对象 """
d = ['a', 'b', 'c']
ser = pd.Series(data=d)
print(ser)

""" ndarray创建Series对象 """
d = np.array([1, 2, 3])
ser = pd.Series(data=d, dtype=np.float64, index=('one', 'two', 'three'), name='test-series')
print(ser)

""" dict创建Series对象：
默认用字典的键作为index，对应字典的值作为数据 """
d = {'a': 1, 'b': 2, 'c': 3}
ser = pd.Series(data=d) # index=['a', 'b', 'c'] 可省略
print(ser)

""" dict创建Series对象：
如果指定索引不是字典的键，那么会得到缺失值NaN """
d = {'a': 1, 'b': 2, 'c': 3}
ser = pd.Series(data=d, index=['a', 'y', 'z'])
print(ser)

```

访问 Series 数据

两种方式：位置索引访问、索引标签访问

- 位置索引访问

```
import numpy as np
import pandas as pd

d = np.array([1, 2, 3, 4, 5])
ser = pd.Series(data=d, index=('a', 'e', 'c', 'd', 'e'))
print(ser)
print(ser[1])
print(ser[1:3])
print(ser[:-2:2])
print(ser[[2, 1, 3]])
```

- 索引标签访问

```
import numpy as np
import pandas as pd

d = np.array([1, 2, 3, 4, 5])
ser = pd.Series(data=d, index=('a', 'e', 'c', 'd', 'e'))
print(ser)
print(ser['c'])
print(ser['e'])

""" 索引标签切片时，右边不是开区间哦 """
print(ser['a':'d']) # ser['e':'d']报错，因为'e'不唯一
print(ser[:'c':2])
print(ser[['c', 'e', 'd']])
```

修改 Series 索引

- 可以通过给 index 属性重新赋值达到修改索引的目的

```
import pandas as pd

ser = pd.Series([4, 7, -5, 3], index=['a', 'b', 'c', 'd'])
print(ser)
ser.index = ['aa', 'bb', 'cc', 'dd'] # 修改原数据
print(ser)
```

修改 Series 数据

- 可以通过索引和切片的方式修改数据

```
import pandas as pd

ser = pd.Series([2, 3, 4, 5], index=['a', 'b', 'c', 'd'])
ser['a'] = 8
print(ser)
ser['b':'d'] = 9
print(ser)
ser['b':'d'] = [1, 1, 1]
print(ser)
```

重新索引

`Series.reindex(index=None, fill_value=np.NaN)`

- **index**: 要获取数据的索引，传入列表
- **fill_value**: 填充的缺失值（标量），默认为 `np.NaN`
- 返回重新索引组成的新的 **Series** 对象
- 注意: **reindex**操作的series对象不能有重复的索引

```
import pandas as pd
import numpy as np

ser = pd.Series([2, 3, 4, 5], index=['a', 'b', 'c', 'd'])
print(ser)

new_ser1 = ser.reindex(index=['a', 'c', 'd'])
print(new_ser1)

new_ser2 = ser.reindex(index=['g', 'c', 'b', 'a', 'f'],
fill_value=np.e)
print(new_ser2)
```

Series 常用属性

属性	描述
dtype	返回 Series 对象数据类型
empty	返回一个布尔值，判断 Series 对象是否为空
name	返回 Series 对象名称
size	返回 Series 中的元素数量
values	以 ndarray 数组的形式返回 Series 中的数据
index	返回 index

```
import pandas as pd

d = [1, 2, 3, 4]
ser = pd.Series(data=d, index=['a', 'b', 'c', 'd'], name="Test-
Series")
print(ser.dtype)
print(ser.empty)
print(ser.name)
print(ser.size)
print(ser.values)
print(ser.index)
```

Series 常用方法

Series.head(n=5)

- 返回前 n 行数据
- 如果 n 为负数，则等价于 Series[:n]

```
import pandas as pd
import numpy as np

ser = pd.Series(np.random.randn(7))
print(ser)
print(ser.head()) # 返回前5行数据
print(ser.head(3)) # 返回前3行数据
print(ser.head(-2)) # 等价于ser[:-2]
```

Series.tail(n=5)

- 返回后 n 行数据
- 如果 n 为负数，则等价于 Series[-n:]

```
import pandas as pd
import numpy as np

ser = pd.Series(np.random.randn(7))
print(ser)
print(ser.tail()) # 返回后5行数据
print(ser.tail(3)) # 返回后3行数据
print(ser.tail(-2)) # 等价于ser[2:]
```

Series.isnull() / Series.notnull()

- 检测 Series 中的缺失值

```
import pandas as pd
import numpy as np

ser = pd.Series([1, 2, 5, np.nan])
print(ser)
print(ser.isnull())
print(ser.notnull())
```

Series 的运算

Series 保留了 NumPy 中的数组运算，且 Series 进行数组运算的时候，索引与值之间的映射关系不会发生改变。在进行 Series 和 Series 的运算时，把两个 Series 中索引一样的值进行运算，其他不一样的做并集，对应的值为 NaN

```
import pandas as pd

ser1 = pd.Series([15, 20], index=["a", "b"])
print(ser1 + 1)
print(ser1 - 1)
print(ser1 * 2)
```



```
print(ser1 / 2)

ser2 = pd.Series([1, 2], index=["c", "a"])
print(ser1 + ser2)
print(ser1 - ser2)
print(ser1 * ser2)
print(ser1 / ser2)
```

Pandas DataFrame

DataFrame 是一种表格型的二维数据结构，既有行索引（**index**），又有列索引（**columns**），且默认都是从0开始递增的整数。可以把每一列看作是共用一个索引的 **Series**，且不同列的数据类型可以不同。**DataFrame** 的结构图，如下所示：

列索引		0	1	2	3
行索引	0	Tom	18	188	75
	1	Bob	19	179	68
	2	Linda	17	177	62

数据

创建 **DataFrame** 对象

```
pd.DataFrame(data=None, index=None, columns=None, dtype=None)
```

- data: array-like, dict, or DataFrame

- **index**: 行索引。不指定时，默认为从 0 开始依次递增的整数
- **columns**: 列索引。不指定时，默认为从 0 开始依次递增的整数
- **dtype**: 数据类型，如果没有指定，则会自动推断得出

```
import numpy as np
import pandas as pd

""" 创建一个空DataFrame对象 """
df = pd.DataFrame()
print(df)

""" ndarray创建DataFrame对象 """
d = np.array([[1, 2, 3], [4, 5, 6]])
df = pd.DataFrame(data=d, dtype=np.float64)
print(df)

""" 单一列表创建DataFrame对象 """
d = ['Tom', 'Bob', 'Linda']
df = pd.DataFrame(data=d)
print(df)

""" 嵌套列表创建DataFrame对象 """
d = [['Tom', 17], ['Bob', 18], ['Linda', 26]]
df = pd.DataFrame(data=d, index=['p1', 'p2', 'p3'], columns=
['name', 'age'])
print(df)

""" 字典嵌套列表创建DataFrame对象:
字典data中，所有键对应的值的元素个数必须相同
默认情况下，字典的键被用作列索引 """
d = {'name': ['Tom', 'Bob', 'Linda'], 'age': [17, 18, 26]}
df = pd.DataFrame(data=d, index=['p1', 'p2', 'p3'])
print(df)

""" 列表嵌套字典创建DataFrame对象:
默认情况下，字典的键被用作列名
如果某个字典的key无法找到对应的value，则为NaN """
d = [{'name': 'Tom', 'age': 17}, {'name': 'Bob', 'age': 18},
{'name': 'Linda', 'ages': 26}]
```

```

df = pd.DataFrame(data=d, index=['p1', 'p2', 'p3'])
print(df)

""" Series创建DataFrame对象 """
d = {'name': pd.Series(['Tom', 'Bob', 'Linda'], index=['p1', 'p2', 'p3']), 'age': pd.Series([17, 18, 26], index=['p1', 'p2', 'p3'])}
df = pd.DataFrame(data=d)
print(df)

d = [pd.Series(['Tom', 'Bob', 'Linda'], index=['p1', 'p2', 'p3'], name="name"), pd.Series([17, 18, 26], index=['p1', 'p2', 'p3'], name='age')]
df = pd.DataFrame(data=d)
print(df)

```

访问 DataFrame 数据

- 索引获取列数据，切片获取行数据

```

import pandas as pd

d = {'name': ['Tom', 'Bob', 'Linda'], 'age': [17, 18, 26], 'height': [172, 176, 188]}
df = pd.DataFrame(data=d, index=['p1', 'p2', 'p3'])
print(df)

# 索引获取列数据
print(df['age'])
print(df[['age', 'name']])

# 切片获取行数据
print(df[0: 1]) # 下标切片左闭右开
print(df['p1': 'p2']) # 标签切片两边都是闭区间

# 组合使用
print(df[['name', 'age']][0: : 2])
print(df[0: : 2][['name', 'age']])

```

- loc指定标签获取数据，iloc指定下标获取数据

```
import pandas as pd

d = {'name': ['Tom', 'Bob', 'Linda'], 'age': [17, 18, 26],
     'height': [172, 176, 188]}
df = pd.DataFrame(data=d, index=['p1', 'p2', 'p3'])
print(df)

""" loc允许接两个参数分别是行和列，且只能接收标签索引 """
# 选取行索引为'p1'的数据
print(df.loc['p1'])

# 选取行索引为'p2'且列索引为'age'的数据
print(df.loc['p2', 'age'])

# 选取行索引为'p2'且列索引分别为'age'和'name'的数据
print(df.loc['p2', ['age', 'name']])

# 选取行索引分别为'p3'和'p2'且列索引分别为'age'和'name'的数据
print(df.loc[['p3', 'p2'], ['age', 'name']])

""" iloc允许接两个参数分别是行和列，且只能接收整数索引 """
# 选取行索引为0的数据
print(df.iloc[0])

# 选取行索引为1且列索引为1的数据
print(df.iloc[1, 1])

# 选取行索引为1且列索引分别为1和0的数据
print(df.iloc[1, [1, 0]])

# 选取行索引分别为2和1且列索引分别为1和0的数据
print(df.iloc[[2, 1], [1, 0]])
```

修改 DataFrame 索引

- 修改对应的属性即可

```
import pandas as pd

d = {'name': ['Tom', 'Bob', 'Linda'], 'age': [17, 18, 26],
     'height': [172, 176, 188]}
df = pd.DataFrame(data=d, index=['p1', 'p2', 'p3'])
print(df)

""" 修改行索引 """
df.index = ['n1', 'n2', 'n3']

""" 修改列索引 """
df.columns = ['names', 'ages', 'heights']

print(df)
```

修改 DataFrame 数据

- 对访问的数据重新赋值，即可修改数据；如果访问数据不存在，则会添加数据
- 如果修改时用下标访问，替换的数据不需要指定索引；如果修改时用标签访问，需要指定索引

```
import pandas as pd

d = {'name': ['Tom', 'Bob', 'Linda'], 'age': [17, 18, 26],
     'height': [172, 176, 188]}
df = pd.DataFrame(data=d, index=['p1', 'p2', 'p3'])
print(df)

""" 修改单列数据 """
df['height'] = pd.Series([1.72, 1.88, 1.76], index=df.index)
print(df)

""" 修改多列数据 """
```

```
df[['name', 'age']] = pd.DataFrame({'name': ['Bob', 'Tom', 'Jack'], 'age': [19, 22, 27]}, ['p1', 'p2', 'p3'])
print(df)
```

""" 添加单列数据 """

```
df['weight'] = pd.Series([65, 75, 60], index=df.index)
print(df)
```

""" 添加多列数据 """

```
df[['BMI', 'adres']] = pd.DataFrame({'BMI': df['weight'] / df['height']**2, 'adres': ['威宁路', '长宁路', '大马路']})
print(df)
```

""" 修改单行数据 """

```
df[1:2] = pd.DataFrame([['Linda', 23, 1.78, 65, 22, '徐汇路']])
df.loc['p2'] = pd.Series(['Linda', 23, 1.78, 65, 22, '徐汇路'], index=df.columns)
print(df)
```

如果修改时用下标访问，替换的数据不需要指定索引

```
df.iloc[1] = pd.Series(['Linda', 23, 1.78, 65, 22, '徐汇路'])
print(df)
```

""" 修改多行数据 """

```
df[::2] = pd.DataFrame([['Jack', 27, 1.76, 60, 19, '大马路'], ['Bob', 19, 1.72, 65, 21, '威宁路']])
print(df)
df.iloc[::2] = pd.DataFrame([['Jack', 27, 1.76, 60, 19, '大马路'], ['Bob', 19, 1.72, 65, 21, '威宁路']])
print(df)
```

如果修改时用标签访问，需要指定索引

```
data = [['Jack', 27, 1.76, 60, 19, '大马路'], ['Bob', 19, 1.72, 65, 21, '威宁路']]
df.loc['p1':'p3':2] = pd.DataFrame(data, index=['p1', 'p3'], columns=df.columns)
```

```
print(df)

""" 添加单行数据(这里就只能用loc), 添加多行可以使用对象方法 """
df.loc['p4'] = pd.Series(['Tony', 23, 1.78, 65, 22, '徐汇路'],
index=df.columns)
print(df)
```

重新索引

DataFrame.reindex(labels=None, axis=0, index=None, columns=None, fill_value=np.NaN)

- labels: 要获取数据的列标签或者行标签, 传入列表, 与axis对应
- axis: 轴的方向, 0为行, 1为列
- index: 要获取数据的行索引, 传入列表
- columns: 要获取数据的列索引, 传入列表
- fill_value: 填充的缺失值(标量), 默认为 np.NaN
- 返回重新索引组成的新的 DataFrame 对象

```
import pandas as pd
import numpy as np

data = np.arange(12).reshape(3, 4)
df = pd.DataFrame(data, index=['n1', 'n2', 'n3'], columns=['a', 'b', 'c', 'd'])
print(df)

# 重新索引行标签为'n2'的数据行
df2 = df.reindex(labels=['n2'], axis=0)
print(df2)

df2 = df.reindex(index=['n2'])
print(df2)

# 重新索引列标签为'c'的数据行
df2 = df.reindex(labels=['c'], axis=1)
print(df2)
```

```
df2 = df.reindex(columns=['c'])
print(df2)

# 分别重新索引行标签为'n2'、'n1'、'n4'的数据行
df2 = df.reindex(index=['n2', 'n1', 'n4'], fill_value=np.pi)
print(df2)
```

DataFrame 常用属性

属性	描述
T	转置
dtypes	返回返回每一列的数据类型
empty	返回一个布尔值，判断 DataFrame 对象是否为空
shape	返回一个代表 DataFrame 维度的元组
size	返回 DataFrame 中的元素数量
index	返回行索引
columns	返回列索引
axes	以列表形式返回行索引和列索引
values	以 ndarray 数组的形式返回 DataFrame 中的数据

```
import pandas as pd
import numpy as np

d = [['Tom', 17], ['Bob', 18], ['Linda', 26]]
df = pd.DataFrame(data=d, index=['p1', 'p2', 'p3'], columns=
['name', 'age'])
print(df)
print(df.T)
print(df.dtypes)
print(df.empty)
print(df.shape)
print(df.size)
```



```
print(df.index)
print(df.columns)
print(df.axes)
print(df.values)
```

DataFrame 常用方法

DataFrame.head(n=5)

- 返回前 n 行数据
- 如果 n 为负数，则等价于 DataFrame[:n]

```
import pandas as pd
import numpy as np

d = np.random.randn(7, 2)
df = pd.DataFrame(data=d)
print(df)
print(df.head()) # 返回前5行数据
print(df.head(3)) # 返回前3行数据
print(df.head(-2)) # 等价于df[:-2]
```

DataFrame.tail(n=5)

- 返回后 n 行数据
- 如果 n 为负数，则等价于 DataFrame[-n:]

```
import pandas as pd
import numpy as np

d = np.random.randn(7, 2)
df = pd.DataFrame(data=d)
print(df)
print(df.tail()) # 返回后5行数据
print(df.tail(3)) # 返回后3行数据
print(df.tail(-2)) # 等价于df[2:]
```

DataFrame.isnull() / DataFrame.notnull()

- 检测 DataFrame 中的缺失值

```
import pandas as pd
import numpy as np

d = np.random.randn(7, 2)
df = pd.DataFrame(data=d).reindex(columns=[1, 2])
print(df)
print(df.isnull())
print(df.notnull())
```

DataFrame.insert(loc, column, value)

- loc: int, 整数列索引, 指定插入数据列的位置
- column: 新插入的数据列的名字
- value: int, Series, or array-like, 插入的数据

```
import pandas as pd

d = {'name': ['Tom', 'Bob', 'Linda'], 'age': [17, 18, 26]}
df = pd.DataFrame(data=d, index=['p1', 'p2', 'p3'])
print(df)

# insert()方法插入新的列
df.insert(2, 'weight', [65, 75, 60])
print(df)
```

DataFrame.pop(item)

- item: 要删除的列标签
- 删除并返回 item 指定的数据

```
import pandas as pd

d = {'name': ['Tom', 'Bob', 'Linda'], 'age': [17, 18, 26],
      'height': [172, 176, 188]}
df = pd.DataFrame(data=d, index=['p1', 'p2', 'p3'])
print(df)

# del语句删除
del df['name']
print(df)

# pop()方法删除
res = df.pop('height')
print(df)
print(res)
```

DataFrame.append(other, ignore_index=False)

- other: DataFrame or Series/dict-like, 要添加的数据
- ignore_index: 如果指定为 True, 则索引将变为从0开始递增的整数
- 返回一个新的 DataFrame

pd.concat(objs, axis=0, join='outer', ignore_index=False)

- **objs**: Series 或 DataFrame对象的序列或映射
- **axis**: 要拼接的轴
- **join**: 外连接 ('outer') 保留两个表中的所有信息；内连接 ('inner') 只保留共有信息
- **ignore_index**: 如果指定为 True, 则索引将变为从0开始递增的整数
- 返回一个新的 DataFrame

```
import pandas as pd

df = pd.DataFrame([[1, 2], [3, 4]], index=['p1', 'p2'],
columns=list('AB'))
print(df)

df2 = pd.DataFrame([[5, 6], [7, 8]], columns=list('AB'))
print(df2)

print(df.append(df2))
print(df.append(df2, ignore_index=True))

print(df.append([{'A': 5, 'B': 6}, {'A': 7, 'B': 8}],
ignore_index=True))

print(df.append([pd.Series([5, 6], index=list('AB')),
pd.Series([7, 8], index=list('AB'))]))

print(df.append([pd.Series([5, 6], index=list('AB')),
pd.Series([7, 8], index=list('AB'))], ignore_index=True))

df2['c'] = pd.Series([9, 0])
print(df2)

print(pd.concat([df, df2]))
print(pd.concat([df, df2], join='inner'))
print(pd.concat([df, df2], axis=1))
df2.index = [0, 'p1'] # 修改索引
print(df2)
```

```
print(pd.concat([df, df2], axis=1, join='inner'))
```

`pd.merge(left, right, how='inner', on=None)`

- **left**: 左侧 DataFrame 对象
- **right**: 右侧 DataFrame 对象
- **how**: 要执行的合并类型。'inner'为内连接，取左右两个 DataFrame 的键的交集进行合并；'left'为左连接，以左侧 DataFrame 的键为基准进行合并，如果左侧 DataFrame 中的键在右侧不存在，则用缺失值NaN 填充；'right'为右连接，以右侧 DataFrame 的键为基准进行合并，如果右侧 DataFrame 中的键在左侧不存在，则用缺失值NaN 填充；'outer'为外连接，取左右两个 DataFrame 的键的并集进行合并
- **on**: 指定用于连接的键（即列标签的名字），该键必须同时存在于左右两个 DataFrame 中，如果没有指定，那么将会以两个 DataFrame 的列名交集做为连接键

```
import pandas as pd

d1 = {'name': ['Tom', 'Bob', 'Jack'], 'age': [18, 17, 19],
      'weight': [65, 66, 67]}
df1 = pd.DataFrame(data=d1)

d2 = {'name': ['Tom', 'Jack'], 'height': [168, 187], 'weight':
      [65, 68]}
df2 = pd.DataFrame(data=d2)

print(df1)
print(df2)
print(pd.merge(df1, df2, how='inner', on='name'))
print(pd.merge(df1, df2, how='left', on='name'))
print(pd.merge(df1, df2, how='right', on='name'))
print(pd.merge(df1, df2, how='outer', on='name'))
```

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

- **labels**: 要删除的列标签或者行标签，如果要删除多个，传入列表，与axis对应
- **axis**: 轴的方向，0为行，1为列

- **index**: 要删除的行索引，如果要删除多个，传入列表
- **columns**: 要删除的列索引，如果要删除多个，传入列表
- **inplace**: `inplace=True`时，对原数据操作，返回None

```
import pandas as pd

df = pd.DataFrame([[1, 2], [3, 4], [5, 6]], index=['n1', 'n2', 'n3'], columns=['a', 'b'])
print(df)

# 删除行索引为'n2'的数据行
print(df.drop(labels='n2', axis=0))
print(df.drop(index='n2'))

# 删除列索引为'b'的数据列
print(df.drop(labels='b', axis=1))
print(df.drop(columns='b'))

# 分别删除行索引为'n2'和'n1'的数据行
print(df.drop(labels=['n2', 'n1'], axis=0))
print(df.drop(index=['n2', 'n1']))

# 分别删除列索引为'a'和'b'的数据列
print(df.drop(labels=['a', 'b'], axis=1))
print(df.drop(columns=['a', 'b']))

# inplace=True时，对原数据操作，返回None
df.drop(index='n1', inplace=True)
print(df)
```

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

- **axis**: 0表示删除包含缺失值的行，1表示删除包含缺失值的列
- **how**: 'any'表示如果存在任何缺失值，则删除该行或列；'all'表示如果所有值都是缺失值，则删除该行或列
- **thresh**: 只保留至少n个非NaN值的行或列，n由该参数指定
- **subset**: 定义要根据哪些列（行）中的缺失值来删除行（列），和axis成行列对应关系

- inplace: 如果为 True 表示对原数据操作，返回 None
- 删除缺失值所在的行或列

```
import pandas as pd
import numpy as np

d = {'name': ['Tom', np.nan, 'Bob'], 'age': [np.nan, np.nan, 19], 'height': [177, 182, 179]}
df = pd.DataFrame(data=d)
print(df)

# 删除缺失值所在的行
print(df.dropna())

# 删除缺失值所在的列
print(df.dropna(axis=1))

# 修改数据
df.loc[2, 'age'] = np.nan
print(df)

# 删除所有值都是缺失值的列
print(df.dropna(axis=1, how='all'))

# 只保留至少2个非NaN值的列
print(df.dropna(axis=1, thresh=2))

# 根据'name'、'height'列中的缺失值来删除行
print(df.dropna(subset=['name', 'height']))

# 根据1、2行中的缺失值来删除列
print(df.dropna(axis=1, subset=[1, 2]))

# 对原数据操作
df.dropna(axis=1, inplace=True)
print(df)
```

DataFrame.fillna(value=None, method=None, axis=None, inplace=False, limit=None)

- **value:** 需要填充的数据
- **method:** 填充方式。'pad'/'ffill' 表示用前一个非缺失值去填充该缺失值；'backfill'/'bfill' 表示用后一个非缺失值填充该缺失值
- **axis:** 指定填充方向，0表示行方向，1表示列方向，默认行方向
- **inplace:** 如果为 True 表示对原数据操作，返回 None
- **limit:** 限制填充个数

```
import pandas as pd
import numpy as np

df = pd.DataFrame([[np.nan, 2, np.nan, 0],
                   [3, 4, np.nan, 1],
                   [np.nan, np.nan, np.nan, np.nan],
                   [np.nan, 3, np.nan, 4]],
                  columns=list("ABCD"))

print(df)

# 用标量填充：将所有NaN填充为0
print(df.fillna(0))

# 用字典填充：指定不同列的NaN填充值
dic = {'A': 6, 'B': 7}
print(df.fillna(dic))

# 当使用DataFrame填充时，替换会沿着相同的行列索引进行
df2 = pd.DataFrame(np.zeros((4, 4)), columns=list("ABCE"))
print(df2)
print(df.fillna(df2))

# 用前一个非缺失值去填充该缺失值
print(df.fillna(method='ffill'))
# 用后一个非缺失值去填充该缺失值
print(df.fillna(method='bfill'))

# 用前一个非缺失值去填充该缺失值
print(df.fillna(method='ffill', axis=1))
# 用后一个非缺失值去填充该缺失值
print(df.fillna(method='bfill', axis=1))
```



```
# 指定'A'列填充为6,'C'列填充7, 且限制每列最多填充2个
print(df.fillna({'A': 6, 'C': 7}, limit=2))
```

DataFrame.info(verbose=None, show_counts=None)

- **verbose**: 是否打印完整的摘要, 为None时表示打印完整摘要, 为False则打印简短摘要
- **show_counts**: 是否显示Non-Null Count, 为None时表示显示, 为False则不显示
- 打印 DataFrame 的简明摘要

```
import pandas as pd
import numpy as np

df = pd.DataFrame(data={'name': ['Tom', 'Bob', np.nan], 'age': [18, 19, 17], 'height': [167, 177, 178]}, index=['n1', 'n2', 'n3'])
print(df)
df.info()
df.info(verbose=False)
df.info(show_counts=False)
```

DataFrame.describe(percentiles=None, include=None, exclude=None)

- **percentiles**: 默认值为 [.25, .5, .75], 它返回第 25、第 50 和第 75 个百分位数
- **include**: 包含在结果中的数据类型; 默认None表示结果将包括所有数字列; 'all'表示包括所有列; 'number'表示包括所有数字列; 'object'表示包括所有字符列
- **exclude**: 不包含在结果中的数据类型; 默认None表示结果不会排除任何列; 'number'表示不包括所有数字列; 'object'表示不包括所有字符列
- 返回描述性统计

```
import pandas as pd

df = pd.DataFrame(data={'name': ['Tom', 'Bob', 'Bob'], 'age': [18, 19, 17], 'height': [167, 177, 178]}, index=['n1', 'n2', 'n3'])
print(df)
print(df.describe())
print(df.describe(include='all'))
print(df.describe(include='object'))

# 也可以是数据类型的组合
print(df.describe(include=['number', 'object']))
```

DataFrame.count(axis=0)	返回指定轴的非空值的数量
DataFrame.max(axis=0)	返回指定轴的最大值
DataFrame.min(axis=0)	返回指定轴的最小值
DataFrame.mean(axis=0)	返回指定轴的平均值
DataFrame.var(axis=0)	返回指定轴的方差
DataFrame.std(axis=0)	返回指定轴的标准差

```
import pandas as pd
import numpy as np

df = pd.DataFrame(data={'name': ['Tom', np.nan, 'Linda'], 'age': [18, 19, 17]}, index=['n1', 'n2', 'n3'])
print(df)
print(df.count())
print(df.count(axis=1))

d = np.random.randn(7, 2)
df = pd.DataFrame(data=d)
print(df)
print(df.max(axis=0)) # 返回每列最大值
print(df.max(axis=1)) # 返回每行最大值
print(df.min(axis=0)) # 返回每列最小值
print(df.min(axis=1)) # 返回每行最小值
print(df.mean(axis=0)) # 返回每列平均值
```

```
print(df.mean(axis=1)) # 返回每行平均值
print(df.var(axis=0)) # 返回每列方差
print(df.var(axis=1)) # 返回每行方差
print(df.std(axis=0)) # 返回每列标准差
print(df.std(axis=1)) # 返回每行标准差
```

DataFrame.sample(n=None, frac=None, replace=False, random_state=None, axis=None)

- **n**: 默认为1, 表示要采样的行数, 不能和 **frac** 参数一起使用
- **frac**: 表示要采用的比例, 不能和 **n** 参数一起使用
- **replace**: 表示是否有放回采样
- **random_state**: 随机数种子
- **axis**: 表示采样的方向, 默认为0, 行采样
- 从指定的轴返回随机样本

```
import pandas as pd

df = pd.DataFrame(data={'name': ['Tom', 'Bob', 'Jack', 'Linda'],
                        'age': [18, 19, 17, 21], 'height': [167, 177, 178, 188]}, index=
['n1', 'n2', 'n3', 'n4'])
print(df)

# 默认n=1, 随机采样一行数据
print(df.sample())

# 随机采样75%的数据, 即这里即3行
print(df.sample(frac=0.75))

# 随机采样2行数据
print(df.sample(n=2))

# 随机有放回采样2行数据(有可能会重复采样)
print(df.sample(n=2, replace=True))

# 随机采样2列数据
print(df.sample(n=2, axis=1))
```

```
# 随机采样2行数据，设置随机数种子seed为3
print(df.sample(n=2, random_state=3))
```

DataFrame.drop_duplicates(subset=None, keep='first', inplace=False)

- **subset**: 表示要进行去重的列名，默认为 **None**，表示所有列
- **keep**: 保留哪些副本。'first'表示只保留第一次出现的重复项，删除其余重复项；'last'表示只保留最后一次出现的重复项；**False** 则表示删除所有重复项
- **inplace**: **False** 表示删除重复项后返回一个副本；**True** 表示直接在原数据上删除重复项
- 返回去重（删除重复行）之后的 DataFrame

```
import pandas as pd

d = {'A': [1, 0, 0, 1], 'B': [0, 2, 5, 0],
      'C': [4, 0, 4, 4], 'D': [1, 0, 0, 1]}

df = pd.DataFrame(data=d)
print(df)

# 第一行和第四行重复，删除第4行
print(df.drop_duplicates())

# 第一行和第四行重复，删除第1行
print(df.drop_duplicates(keep='last'))

# 第一行和第四行重复，两行都删除
print(df.drop_duplicates(keep=False))

# 指定columns为'A'和'D'的两列去重
# 第二行和第三行重复，保留第三行
# 第一行和第四行重复，保留第四行
print(df.drop_duplicates(subset=['A', 'D'], keep='last'))

# 直接在原数据上删除重复项
df.drop_duplicates(subset=['A', 'D'], keep='last', inplace=True)
print(df)
```

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

- **by**: 要排序的名称或名称列表
如果 axis=0 或 'index', by可指定列标签
如果 axis=1 或 'columns', by可指定行标签
- **axis**: 要排序的轴, 可选择 0 或 'index', 1 或 'columns'
- **ascending**: False 则为降序; 如果这是一个 bool 列表, 则必须匹配 by 的长度
- **inplace**: 是否原地操作
- **na_position**: 设置缺失值的排序位置, 'first'表示开头, 'last'表示结尾

```
import pandas as pd
import numpy as np

df = pd.DataFrame({'col1': [4, 1, 2, np.nan, 5, 2],
                   'col2': [2, 1, 9, 8, 7, 6],
                   'col3': [0, 1, 9, 4, 2, 3],
                   'col4': ['a', 'B', 'c', 'D', 'e', 1]})

print(df)
print(df.sort_values(by=['col1']))
print(df.sort_values(by='col1'))
print(df.sort_values(by=['col1', 'col2']))
print(df.sort_values(by=5, axis=1))
print(df.sort_values(by=5, axis=1, ascending=False))
print(df.sort_values(['col1', 'col2'], ascending=[True, False]))
print(df.sort_values(by='col1', na_position='first'))
df.sort_values(by='col1', inplace=True)
print(df)
```

DataFrame.apply(func, axis=0, result_type=None)

- **func**: 应用于每一个列或行的函数
- **axis**: 0 or 'index'表示函数处理的是每一列; 1 or 'columns'表示函数处理的是每一行

- **result_type**: 默认为 `None`。下面选项只在 `'axis=1'` 时起作用:
 `'expand'`: 类似列表的结果将转换为列
 `'broadcast'`: 结果将被广播到 `DataFrame` 的原始形状, 原始索引和列将被保留
- 返回 `Series` 或者 `DataFrame`: 沿着给定的 `DataFrame` 轴应用 `func` 的结果

```
import pandas as pd
import numpy as np

df = pd.DataFrame([[4, 9]] * 3, columns=['A', 'B'])
print(df)

print(df.apply(np.sum))
print(df.apply(np.sum, axis=1))
print(df.apply(lambda x: [1, 2], axis=1))
print(df.apply(lambda x: [1, 2], axis=1, result_type='expand'))

# 在函数内部返回一个Series类似于传递'result_type='expand', 结果列名将是Series索引
print(df.apply(lambda x: pd.Series([1, 2], index=['foo', 'bar']), axis=1))

# 传递'result_type='broadcast'将确保函数返回相同的形状结果,
# 无论是类列表还是标量, 并沿轴广播它, 列名将是原始列名
print(df.apply(lambda x: [1, 2], axis=1, result_type='broadcast'))
```

`DataFrame.groupby(by=None, axis=0, as_index=True, sort=True, dropna=True)`

- **by**: 指定根据哪个或者哪些标签分组
- **axis**: 沿着分组的轴, 0表示行, 1表示列
- **as_index**: 对于聚合操作的输出结果, 默认将分组列的值作为索引, 如果将 `as_index` 设置为 `False`, 可以重置索引 (0, 1, 2...)
- **sort**: 结果按分组标签的值升序排列, 设置为 `False` 则不排序
- **dropna**: 默认为 `True` 时, 分组标签那列的 `NaN` 在分组结果中不保留, 设置为 `False`, 可以保留 `NaN` 分组
- 返回一个包含分组信息的 `DataFrameGroupBy` 对象

```
import pandas as pd
import numpy as np

d = {
    'company': ['A', 'B', 'A', 'C', 'C', 'B', 'C', 'A'],
    'salary': [8, 15, 10, 15, 15, 28, 30, 15],
    'age': [26, 29, 26, 30, 50, 30, 30, 35]
}

df = pd.DataFrame(data=d)
print(df)

# 根据 'company' 列, 按行分组
res = df.groupby('company')
# 返回包含组分组信息的 DataFrameGroupBy 对象
print(res)

# 根据 'company', 'age' 列, 按行分组
res2 = df.groupby(['company', 'age'])
print(res2)

# 迭代 DataFrameGroupBy 对象
print(list(res))
for g, i in res:
    print(g)
    print(i)

print(list(res2))
for g, i in res2:
    print(g)
    print(i)

# 通过标签索引的方式获取各个分组的部分信息
print(res['company'])
for g, i in res['company']:
    print(g)
    print(i)
```

```
print(res['age'])
for g, i in res['age']:
    print(g)
    print(i)

# 根据'age'列，按行分组，结果不按照age大小排序
res3 = df.groupby('age', sort=False)
for g, i in res3:
    print(g)
    print(i)

# 把'age'列的第一个数据改为NaN
df.loc[0, 'age'] = np.nan
print(df)

# dropna设置为False，按照'age'列进行分组时，会保留NaN组
res4 = df.groupby('age', dropna=False)
for g, i in res4:
    print(g)
    print(i)

# groups实例属性以字典的形式包含了分组信息
print(res.groups)

# get_group对象方法获取指定分组的数据
print(res.get_group('A'))
print(res.get_group('B'))
print(res.get_group('C'))

# agg聚合方法可以同时对各组分别操作
print(res.agg('max'))
print(res.agg(np.max)) # 同上
print(res.agg('min'))
print(res.agg('sum'))
print(res.agg('mean'))
print(res.agg('median')) # 中位数
print(res.agg('std'))
print(res.agg('var'))
```



```

print(res.agg('count'))

# transform转换方法对各组分别操作后，得到的结果是同一组内的样本共有的，
再按照原索引的顺序返回结果
avg_salary = res['salary'].transform('mean')
print(avg_salary)

# 添加新的数据列
df['avg_salary'] = avg_salary
print(df)

def get_oldest(x):
    df = x.sort_values(by='age', ascending=False,
na_position='last')
    # print(df)
    return df.iloc[0, :]

# 获取各个公司年龄最大的员工的数据
# apply对象方法把每个分组作为参数传递给指定的函数来调用
oldest = df.groupby('company', as_index=False).apply(get_oldest)
print(oldest)

```

DataFrame 的运算

DataFrame 保留了 NumPy 中的数组运算，且 DataFrame 进行数组运算的时候，索引与值之间的映射关系不会发生改变。在进行 DataFrame 和 DataFrame 的运算时，把两个 DataFrame 中行索引名和列索引名一样的值进行运算，其他不一样的做并集且对应的值为NaN

```

import pandas as pd
import numpy as np

d = np.arange(9).reshape((3, 3))

```

```
df1 = pd.DataFrame(data=d, columns=list('abc'), index=['n1',
'n2', 'n3'])
print(df1)

print(df1 + 1)
print(df1 - 1)
print(df1 * 2)
print(df1 / 2)

d = np.arange(16).reshape((4, 4))
df2 = pd.DataFrame(data=d, columns=list('dacf'), index=['n1',
'n2', 'n3', 'n4'])
print(df2)

print(df1 + df2)
print(df1 - df2)
print(df1 * df2)
print(df1 / df2)
```

Pandas 文件读写

CSV文件读写

```
from typing import Iterator
import pandas as pd
import numpy as np

df = pd.read_csv('./test01.csv')
print(df)

# sep参数默认为逗号
# 但是test02.csv文件是用分号分隔的，所以需要设置这个参数
```

```
df = pd.read_csv('./test02.csv', sep=';')
print(df)

# header参数默认自行推断，会把第一行数据作为列索引(表头)
# 如果不想使用数据为列索引，可设置为None，那么列索引就会是(0, 1...)
df = pd.read_csv('./test03.csv', sep=';', header=None)
print(df)

# header=2，指定第3行数据作为列索引(表头)，之后的行才为数据
df = pd.read_csv('./test03.csv', sep=';', header=2)
print(df)

# names参数可以指定列索引，如果指定names，则header不会把第一行推断为列索引
df = pd.read_csv('./test02.csv', sep=';', names=['name', 'age', 'height'])
print(df)

# 读取大文件片段
# nrows参数用来读取指定行数的数据：这里读取前两行数据，不包括表头
df = pd.read_csv('./test01.csv', nrows=2)
print(df)

# skiprows参数用来指定需要跳过的行，包括表头
# 当skiprows为整数时，表示跳过对应的前几行：这里跳过前两行
df = pd.read_csv('./test01.csv', skiprows=2)
print(df)

# 当skiprows为索引时，表示跳过对应的索引行：这里跳过第1行和第3行
df = pd.read_csv('./test01.csv', skiprows=[0, 2])
print(df)

# usecols参数用来读取指定列的数据：这里读取第1列和第3列
df = pd.read_csv('./test01.csv', usecols=[0, 2])
print(df)

# chunksize参数指定时，read_csv会返回TextFileReader对象
# TextFileReader对象是个迭代器，可以按照chunksize迭代
obj = pd.read_csv('./test01.csv', chunksize=2)
print(isinstance(obj, Iterator))
```

```
for i in obj:
    print(i)
```

```
import pandas as pd

d = {
    '名字': ['张三', '李四', '王五', '赵六', '孙七'],
    '年龄': [18, 19, 20, 22, 17],
    '身高': [188, 178, 189, 175, 177]
}
df = pd.DataFrame(data=d)
print(df)

# 把DataFrame写入csc文件
# sep参数默认为逗号，所以文件中分隔符为逗号
# index参数默认为True，所以行索引也被写入了文件
# header参数默认为True，所以列索引也被写入了文件
df.to_csv('./test04.csv')

# 可以给sep参数指定其他的分隔符(分隔符必须是一个字符)
df.to_csv('./test05.csv', sep=';')

# 如果不想把行索引写入文件可以把index参数指定为False
df.to_csv('./test06.csv', index=False)

# 如果不想把列索引写入文件可以把header参数指定为False
df.to_csv('./test07.csv', header=False)
```

EXCEL文件读写

```
import pandas as pd

d = {
```

```

    '名字': ['张三', '李四', '王五', '赵六', '孙七'],
    '年龄': [18, 19, 20, 22, 17],
    '身高': [188, 178, 189, 175, 177]
}
df = pd.DataFrame(data=d)
print(df)

# 把DataFrame写入excel文件
# index参数默认为True, 所以行索引也被写入了文件
# header参数默认为True, 所以列索引也被写入了文件
df.to_excel('./test08.xlsx')

# 如果不想把行索引写入文件可以把index参数指定为False
df.to_excel('./test09.xlsx', index=False)

# 如果不想把列索引写入文件可以把header参数指定为False
df.to_excel('./test10.xlsx', header=False)

# 把DataFrame写入excel文件
with pd.ExcelWriter('./test11.xlsx') as writer:
    # 写入工作表1
    df.to_excel(writer, sheet_name='工作表1', index=False)
    # 写入工作表2
    df.iloc[:, :2].to_excel(writer, sheet_name='工作表2',
index=False)

# 等价于上面with语句的写法
writer = pd.ExcelWriter('./test11.xlsx')
df.to_excel(writer, sheet_name='工作表1', index=False)
df.iloc[:, :2].to_excel(writer, sheet_name='工作表2',
index=False)
writer.close()

```

```

import pandas as pd

# header参数默认为0, 会把第一行作为列索引(表头)
df = pd.read_excel('./test11.xlsx')

```

```
print(df)
```

如果不想使用数据为列索引，可设置为None，那么列索引就会是(0, 1...)

```
df = pd.read_excel('./test11.xlsx', header=None)
```

```
print(df)
```

header=2，指定第3行数据作为列索引(表头)，之后的行才为数据

```
df = pd.read_excel('./test11.xlsx', header=2)
```

```
print(df)
```

names参数可以指定列索引(表头)

如果此时header为默认值0，会被names参数指定的值覆盖掉

```
df = pd.read_excel('./test11.xlsx', names=['name', 'age',  
      'height'])
```

```
print(df)
```

如果此时header为None，第一行仍会当作数据

```
df = pd.read_excel('./test11.xlsx', header=None, names=['name',  
      'age', 'height'])
```

```
print(df)
```

sheet_name指定要读取的工作表

如果为整数，则为工作表的索引；默认为0，表示读取第一个工作表

```
df = pd.read_excel('./test11.xlsx', sheet_name=1)
```

```
print(df)
```

如果为字符串，则为工作表的名称

```
df = pd.read_excel('./test11.xlsx', sheet_name='工作表2')
```

```
print(df)
```

如果想要读取多个工作表，可以指定为列表

```
df = pd.read_excel('./test11.xlsx', sheet_name=[0, '工作表2'])
```

```
print(df)
```

读取大文件片段

nrows参数用来读取指定行数的数据：这里读取前两行数据，不包括表头

```
df = pd.read_excel('./test11.xlsx', nrows=2)
```

```
print(df)
```

```
# skiprows参数用来指定需要跳过的行，包括表头
# 当skiprows为整数时，表示跳过对应的前几行：这里跳过前两行
df = pd.read_excel('./test11.xlsx', skiprows=2)
print(df)

# 当skiprows为索引时，表示跳过对应的索引行：这里跳过第1行和第3行
df = pd.read_excel('./test11.xlsx', skiprows=[0, 2])
print(df)

# usecols参数用来读取指定列的数据：这里读取第1列和第3列
df = pd.read_excel('./test11.xlsx', usecols=[0, 2])
print(df)
```