# 12.Pandas库

## 12.1.对象创建

### 12.1.1.Pandas Series对象

基于Numpy构建的Pandas库，提供了使得数据分析变得更快更简单的高级数据结构和操作工具

Series 是带标签数据的一维数组

**Series对象的创建**

通用结构: pd.Series(data, index=index, dtype=dtype)

data：数据，可以是列表，字典或Numpy数组

index：索引，为可选参数

dtype: 数据类型，为可选参数

**1、用列表创建**

index缺省，默认为整数序列

**import** **pandas** **as** **pd**

data = pd.Series([1.5, 3, 4.5, 6])

data

0 1.5

1 3.0

2 4.5

3 6.0

dtype: float64

增加index

data = pd.Series([1.5, 3, 4.5, 6], index=["a", "b", "c", "d"])

data

a 1.5

b 3.0

c 4.5

d 6.0

dtype: float64

增加数据类型

缺省则从传入的数据自动判断

data = pd.Series([1, 2, 3, 4], index=["a", "b", "c", "d"])

data

a 1

b 2

c 3

d 4

dtype: int64

data = pd.Series([1, 2, 3, 4], index=["a", "b", "c", "d"], dtype="float")

data

a 1.0

b 2.0

c 3.0

d 4.0

dtype: float64

注意：数据支持多种类型

data = pd.Series([1, 2, "3", 4], index=["a", "b", "c", "d"])

data

a 1

b 2

c 3

d 4

dtype: object

data["a"]

1

data["c"]

'3'

数据类型可被强制改变

data = pd.Series([1, 2, "3", 4], index=["a", "b", "c", "d"], dtype=float)

data

a 1.0

b 2.0

c 3.0

d 4.0

dtype: float64

data["c"]

3.0

data = pd.Series([1, 2, "a", 4], index=["a", "b", "c", "d"], dtype=float)

data

**ValueError**: could not convert string to float: 'a'

**2、用一维numpy数组创建**

**import** **numpy** **as** **np**

x = np.arange(5)

pd.Series(x)

0 0

1 1

2 2

3 3

4 4

dtype: int32

**3、用字典创建**

默认以键为index 值为data

population\_dict = {"BeiJing": 2154,

"ShangHai": 2424,

"ShenZhen": 1303,

"HangZhou": 981 }

population = pd.Series(population\_dict)

population

BeiJing 2154

ShangHai 2424

ShenZhen 1303

HangZhou 981

dtype: int64

字典创建，如果指定index，则会到字典的键中筛选，找不到的，值设为NaN

population = pd.Series(population\_dict, index=["BeiJing", "HangZhou", "c", "d"])

population

BeiJing 2154.0

HangZhou 981.0

c NaN

d NaN

dtype: float64

**4、data为标量的情况**

pd.Series(5, index=[100, 200, 300])

100 5

200 5

300 5

dtype: int64

### 12.1.2.Pandas DataFrame对象

DataFrame 是带标签数据的多维数组

DataFrame对象的创建

通用结构: pd.DataFrame(data, index=index, columns=columns)

data：数据，可以是列表，字典或Numpy数组

index：索引，为可选参数

columns: 列标签，为可选参数

**1、通过Series对象创建**

population\_dict = {"BeiJing": 2154,

"ShangHai": 2424,

"ShenZhen": 1303,

"HangZhou": 981 }

population = pd.Series(population\_dict)

pd.DataFrame(population)

|  | **0** |
| --- | --- |
| **BeiJing** | 2154 |
| **ShangHai** | 2424 |
| **ShenZhen** | 1303 |
| **HangZhou** | 981 |

pd.DataFrame(population, columns=["population"])

|  | **population** |
| --- | --- |
| **BeiJing** | 2154 |
| **ShangHai** | 2424 |
| **ShenZhen** | 1303 |
| **HangZhou** | 981 |

**2、通过Series对象字典创建**

GDP\_dict = {"BeiJing": 30320,

"ShangHai": 32680,

"ShenZhen": 24222,

"HangZhou": 13468 }

GDP = pd.Series(GDP\_dict)

GDP

BeiJing 30320

ShangHai 32680

ShenZhen 24222

HangZhou 13468

dtype: int64

pd.DataFrame({"population": population,

"GDP": GDP})

|  | **population** | **GDP** |
| --- | --- | --- |
| **BeiJing** | 2154 | 30320 |
| **ShangHai** | 2424 | 32680 |
| **ShenZhen** | 1303 | 24222 |
| **HangZhou** | 981 | 13468 |

注意：数量不够的会自动补齐

pd.DataFrame({"population": population,

"GDP": GDP,

"country": "China"})

|  | **population** | **GDP** | **country** |
| --- | --- | --- | --- |
| **BeiJing** | 2154 | 30320 | China |
| **ShangHai** | 2424 | 32680 | China |
| **ShenZhen** | 1303 | 24222 | China |
| **HangZhou** | 981 | 13468 | China |

**3、通过字典列表对象创建**

字典索引作为index，字典键作为columns

**import** **numpy** **as** **np**

**import** **pandas** **as** **pd**

data = [{"a": i, "b": 2\*i} **for** i **in** range(3)]

data

[{'a': 0, 'b': 0}, {'a': 1, 'b': 2}, {'a': 2, 'b': 4}]

data = pd.DataFrame(data)

data

|  | **a** | **b** |
| --- | --- | --- |
| **0** | 0 | 0 |
| **1** | 1 | 2 |
| **2** | 2 | 4 |

data1 = data["a"].copy()

data1

0 0

1 1

2 2

Name: a, dtype: int64

data1[0] = 10

data1

0 10

1 1

2 2

Name: a, dtype: int64

data

|  | **a** | **b** |
| --- | --- | --- |
| **0** | 0 | 0 |
| **1** | 1 | 2 |
| **2** | 2 | 4 |

不存在的键，会默认值为NaN

data = [{"a": 1, "b":1},{"b": 3, "c":4}]

data

[{'a': 1, 'b': 1}, {'b': 3, 'c': 4}]

pd.DataFrame(data)

|  | **a** | **b** | **c** |
| --- | --- | --- | --- |
| **0** | 1.0 | 1 | NaN |
| **1** | NaN | 3 | 4.0 |

4、通过Numpy二维数组创建

data = np.random.randint(10, size=(3, 2))

data

array([[1, 6],

[2, 9],

[4, 0]])

pd.DataFrame(data, columns=["foo", "bar"], index=["a", "b", "c"])

|  | **foo** | **bar** |
| --- | --- | --- |
| **a** | 1 | 6 |
| **b** | 2 | 9 |
| **c** | 4 | 0 |

## 12.2.DataFrame性质

### 12.2.1.属性

data = pd.DataFrame({"pop": population, "GDP": GDP})

data

|  | **pop** | **GDP** |
| --- | --- | --- |
| **BeiJing** | 2154 | 30320 |
| **ShangHai** | 2424 | 32680 |
| **ShenZhen** | 1303 | 24222 |
| **HangZhou** | 981 | 13468 |

**（1）df.values 返回numpy数组的数据**

data.values

array([[ 2154, 30320],

[ 2424, 32680],

[ 1303, 24222],

[ 981, 13468]], dtype=int64)

**（2）df.index 返回行索引**

data.index

Index(['BeiJing', 'ShangHai', 'ShenZhen', 'HangZhou'], dtype='object')

**（3）df.columns 返回列索引**

data.columns

Index(['pop', 'GDP'], dtype='object')

**（4）df.shape 形状**

data.shape

(4, 2)

**（5） pd.size 大小**

data.size

8

**（6）pd.dtypes 返回每列数据类型**

data.dtypes

pop int64

GDP int64

dtype: object

### 12.2.2.索引

data

|  | **pop** | **GDP** |
| --- | --- | --- |
| **BeiJing** | 2154 | 30320 |
| **ShangHai** | 2424 | 32680 |
| **ShenZhen** | 1303 | 24222 |
| **HangZhou** | 981 | 13468 |

**（1）获取列**

字典式

data["pop"]

BeiJing 2154

ShangHai 2424

ShenZhen 1303

HangZhou 981

Name: pop, dtype: int64

data[["GDP", "pop"]]

|  | **GDP** | **pop** |
| --- | --- | --- |
| **BeiJing** | 30320 | 2154 |
| **ShangHai** | 32680 | 2424 |
| **ShenZhen** | 24222 | 1303 |
| **HangZhou** | 13468 | 981 |

对象属性式

data.GDP

BeiJing 30320

ShangHai 32680

ShenZhen 24222

HangZhou 13468

Name: GDP, dtype: int64

**（2）获取行**

绝对索引 df.loc

data.loc["BeiJing"]

pop 2154

GDP 30320

Name: BeiJing, dtype: int64

data.loc[["BeiJing", "HangZhou"]]

|  | **pop** | **GDP** |
| --- | --- | --- |
| **BeiJing** | 2154 | 30320 |
| **HangZhou** | 981 | 13468 |

相对索引 df.iloc

data.iloc[0]

pop 2154

GDP 30320

Name: BeiJing, dtype: int64

data.iloc[[1, 3]]

|  | **pop** | **GDP** |
| --- | --- | --- |
| **ShangHai** | 2424 | 32680 |
| **HangZhou** | 981 | 13468 |

**（3）获取标量**

data.loc["BeiJing", "GDP"]

30320

data.iloc[0, 1]

30320

data.values[0][1]

30320

**（4）Series对象的索引**

type(data.GDP)

pandas.core.series.Series

GDP

BeiJing 30320

ShangHai 32680

ShenZhen 24222

HangZhou 13468

dtype: int64

GDP["BeiJing"]

30320

### 12.2.3.切片

dates = pd.date\_range(start='2019-01-01', periods=6)

dates

DatetimeIndex(['2019-01-01', '2019-01-02', '2019-01-03', '2019-01-04',

'2019-01-05', '2019-01-06'],

dtype='datetime64[ns]', freq='D')

df = pd.DataFrame(np.random.randn(6,4), index=dates, columns=["A", "B", "C", "D"])

df

|  | **A** | **B** | **C** | **D** |
| --- | --- | --- | --- | --- |
| **2019-01-01** | -0.935378 | -0.190742 | 0.925984 | -0.818969 |
| **2019-01-02** | -0.234414 | -1.194674 | 1.080779 | -2.294395 |
| **2019-01-03** | -0.141572 | 0.058118 | 1.102248 | 1.207726 |
| **2019-01-04** | 0.305088 | 0.535920 | -0.978434 | 0.177251 |
| **2019-01-05** | 0.313383 | 0.234041 | 0.163155 | -0.296649 |
| **2019-01-06** | 0.250613 | -0.904400 | -0.858240 | -1.573342 |

**（1）行切片**

df["2019-01-01": "2019-01-03"]

|  | **A** | **B** | **C** | **D** |
| --- | --- | --- | --- | --- |
| **2019-01-01** | -0.935378 | -0.190742 | 0.925984 | -0.818969 |
| **2019-01-02** | -0.234414 | -1.194674 | 1.080779 | -2.294395 |
| **2019-01-03** | -0.141572 | 0.058118 | 1.102248 | 1.207726 |

df.loc["2019-01-01": "2019-01-03"]

|  | **A** | **B** | **C** | **D** |
| --- | --- | --- | --- | --- |
| **2019-01-01** | -0.935378 | -0.190742 | 0.925984 | -0.818969 |
| **2019-01-02** | -0.234414 | -1.194674 | 1.080779 | -2.294395 |
| **2019-01-03** | -0.141572 | 0.058118 | 1.102248 | 1.207726 |

df.iloc[0: 3]

|  | **A** | **B** | **C** | **D** |
| --- | --- | --- | --- | --- |
| **2019-01-01** | -0.935378 | -0.190742 | 0.925984 | -0.818969 |
| **2019-01-02** | -0.234414 | -1.194674 | 1.080779 | -2.294395 |
| **2019-01-03** | -0.141572 | 0.058118 | 1.102248 | 1.207726 |

**（2）列切片**

df.loc[:, "A": "C"]

|  | **A** | **B** | **C** |
| --- | --- | --- | --- |
| **2019-01-01** | -0.935378 | -0.190742 | 0.925984 |
| **2019-01-02** | -0.234414 | -1.194674 | 1.080779 |
| **2019-01-03** | -0.141572 | 0.058118 | 1.102248 |
| **2019-01-04** | 0.305088 | 0.535920 | -0.978434 |
| **2019-01-05** | 0.313383 | 0.234041 | 0.163155 |
| **2019-01-06** | 0.250613 | -0.904400 | -0.858240 |

df.iloc[:, 0: 3]

|  | **A** | **B** | **C** |
| --- | --- | --- | --- |
| **2019-01-01** | -0.935378 | -0.190742 | 0.925984 |
| **2019-01-02** | -0.234414 | -1.194674 | 1.080779 |
| **2019-01-03** | -0.141572 | 0.058118 | 1.102248 |
| **2019-01-04** | 0.305088 | 0.535920 | -0.978434 |
| **2019-01-05** | 0.313383 | 0.234041 | 0.163155 |
| **2019-01-06** | 0.250613 | -0.904400 | -0.858240 |

**（3）多种多样的取值**

行、列同时切片

df.loc["2019-01-02": "2019-01-03", "C":"D"]

|  | **C** | **D** |
| --- | --- | --- |
| **2019-01-02** | 1.080779 | -2.294395 |
| **2019-01-03** | 1.102248 | 1.207726 |

df.iloc[1: 3, 2:]

|  | **C** | **D** |
| --- | --- | --- |
| **2019-01-02** | 1.080779 | -2.294395 |
| **2019-01-03** | 1.102248 | 1.207726 |

行切片，列分散取值

df.loc["2019-01-04": "2019-01-06", ["A", "C"]]

|  | **A** | **C** |
| --- | --- | --- |
| **2019-01-04** | 0.305088 | -0.978434 |
| **2019-01-05** | 0.313383 | 0.163155 |
| **2019-01-06** | 0.250613 | -0.858240 |

df.iloc[3:, [0, 2]]

|  | **A** | **C** |
| --- | --- | --- |
| **2019-01-04** | 0.305088 | -0.978434 |
| **2019-01-05** | 0.313383 | 0.163155 |
| **2019-01-06** | 0.250613 | -0.858240 |

行分散取值，列切片

df.loc[["2019-01-02", "2019-01-06"], "C": "D"]

df.iloc[[1, 5], 0: 3]

|  | **A** | **B** | **C** |
| --- | --- | --- | --- |
| **2019-01-02** | -0.234414 | -1.194674 | 1.080779 |
| **2019-01-06** | 0.250613 | -0.904400 | -0.858240 |

行、列均分散取值

df.loc[["2019-01-04", "2019-01-06"], ["A", "D"]]

df.iloc[[1, 5], [0, 3]]

|  | **A** | **D** |
| --- | --- | --- |
| **2019-01-02** | -0.234414 | -2.294395 |
| **2019-01-06** | 0.250613 | -1.573342 |

### 12.2.4.布尔索引

df

|  | **A** | **B** | **C** | **D** |
| --- | --- | --- | --- | --- |
| **2019-01-01** | -0.935378 | -0.190742 | 0.925984 | -0.818969 |
| **2019-01-02** | -0.234414 | -1.194674 | 1.080779 | -2.294395 |
| **2019-01-03** | -0.141572 | 0.058118 | 1.102248 | 1.207726 |
| **2019-01-04** | 0.305088 | 0.535920 | -0.978434 | 0.177251 |
| **2019-01-05** | 0.313383 | 0.234041 | 0.163155 | -0.296649 |
| **2019-01-06** | 0.250613 | -0.904400 | -0.858240 | -1.573342 |

df > 0

|  | **A** | **B** | **C** | **D** |
| --- | --- | --- | --- | --- |
| **2019-01-01** | False | False | True | False |
| **2019-01-02** | False | False | True | False |
| **2019-01-03** | False | True | True | True |
| **2019-01-04** | True | True | False | True |
| **2019-01-05** | True | True | True | False |
| **2019-01-06** | True | False | False | False |

df[df > 0]

|  | **A** | **B** | **C** | **D** |
| --- | --- | --- | --- | --- |
| **2019-01-01** | NaN | NaN | 0.925984 | NaN |
| **2019-01-02** | NaN | NaN | 1.080779 | NaN |
| **2019-01-03** | NaN | 0.058118 | 1.102248 | 1.207726 |
| **2019-01-04** | 0.305088 | 0.535920 | NaN | 0.177251 |
| **2019-01-05** | 0.313383 | 0.234041 | 0.163155 | NaN |
| **2019-01-06** | 0.250613 | NaN | NaN | NaN |

df.A > 0

2019-01-01 False

2019-01-02 False

2019-01-03 False

2019-01-04 True

2019-01-05 True

2019-01-06 True

Freq: D, Name: A, dtype: bool

df[df.A > 0]

|  | **A** | **B** | **C** | **D** |
| --- | --- | --- | --- | --- |
| **2019-01-04** | 0.305088 | 0.535920 | -0.978434 | 0.177251 |
| **2019-01-05** | 0.313383 | 0.234041 | 0.163155 | -0.296649 |
| **2019-01-06** | 0.250613 | -0.904400 | -0.858240 | -1.573342 |

isin（）方法

df2 = df.copy()

df2['E'] = ['one', 'one', 'two', 'three', 'four', 'three']

df2

|  | **A** | **B** | **C** | **D** | **E** |
| --- | --- | --- | --- | --- | --- |
| **2019-01-01** | -0.935378 | -0.190742 | 0.925984 | -0.818969 | one |
| **2019-01-02** | -0.234414 | -1.194674 | 1.080779 | -2.294395 | one |
| **2019-01-03** | -0.141572 | 0.058118 | 1.102248 | 1.207726 | two |
| **2019-01-04** | 0.305088 | 0.535920 | -0.978434 | 0.177251 | three |
| **2019-01-05** | 0.313383 | 0.234041 | 0.163155 | -0.296649 | four |
| **2019-01-06** | 0.250613 | -0.904400 | -0.858240 | -1.573342 | three |

ind = df2["E"].isin(["two", "four"])

ind

2019-01-01 False

2019-01-02 False

2019-01-03 True

2019-01-04 False

2019-01-05 True

2019-01-06 False

Freq: D, Name: E, dtype: bool

df2[ind]

|  | **A** | **B** | **C** | **D** | **E** |
| --- | --- | --- | --- | --- | --- |
| **2019-01-03** | -0.141572 | 0.058118 | 1.102248 | 1.207726 | two |
| **2019-01-05** | 0.313383 | 0.234041 | 0.163155 | -0.296649 | four |

### 12.2.5.赋值

**DataFrame 增加新列**

s1 = pd.Series([1, 2, 3, 4, 5, 6], index=pd.date\_range('20190101', periods=6))

s1

2019-01-01 1

2019-01-02 2

2019-01-03 3

2019-01-04 4

2019-01-05 5

2019-01-06 6

Freq: D, dtype: int64

df["E"] = s1

df

|  | **A** | **B** | **C** | **D** | **E** |
| --- | --- | --- | --- | --- | --- |
| **2019-01-01** | -0.935378 | -0.190742 | 0.925984 | -0.818969 | 1 |
| **2019-01-02** | -0.234414 | -1.194674 | 1.080779 | -2.294395 | 2 |
| **2019-01-03** | -0.141572 | 0.058118 | 1.102248 | 1.207726 | 3 |
| **2019-01-04** | 0.305088 | 0.535920 | -0.978434 | 0.177251 | 4 |
| **2019-01-05** | 0.313383 | 0.234041 | 0.163155 | -0.296649 | 5 |
| **2019-01-06** | 0.250613 | -0.904400 | -0.858240 | -1.573342 | 6 |

**修改赋值**

df.loc["2019-01-01", "A"] = 0

df

|  | **A** | **B** | **C** | **D** | **E** |
| --- | --- | --- | --- | --- | --- |
| **2019-01-01** | 0.000000 | -0.190742 | 0.925984 | -0.818969 | 1 |
| **2019-01-02** | -0.234414 | -1.194674 | 1.080779 | -2.294395 | 2 |
| **2019-01-03** | -0.141572 | 0.058118 | 1.102248 | 1.207726 | 3 |
| **2019-01-04** | 0.305088 | 0.535920 | -0.978434 | 0.177251 | 4 |
| **2019-01-05** | 0.313383 | 0.234041 | 0.163155 | -0.296649 | 5 |
| **2019-01-06** | 0.250613 | -0.904400 | -0.858240 | -1.573342 | 6 |

df.iloc[0, 1] = 0

df

|  | **A** | **B** | **C** | **D** | **E** |
| --- | --- | --- | --- | --- | --- |
| **2019-01-01** | 0.000000 | 0.000000 | 0.925984 | -0.818969 | 1 |
| **2019-01-02** | -0.234414 | -1.194674 | 1.080779 | -2.294395 | 2 |
| **2019-01-03** | -0.141572 | 0.058118 | 1.102248 | 1.207726 | 3 |
| **2019-01-04** | 0.305088 | 0.535920 | -0.978434 | 0.177251 | 4 |
| **2019-01-05** | 0.313383 | 0.234041 | 0.163155 | -0.296649 | 5 |
| **2019-01-06** | 0.250613 | -0.904400 | -0.858240 | -1.573342 | 6 |

df["D"] = np.array([5]\*len(df)) *# 可简化成df["D"] = 5*

df

|  | **A** | **B** | **C** | **D** | **E** |
| --- | --- | --- | --- | --- | --- |
| **2019-01-01** | 0.000000 | 0.000000 | 0.925984 | 5 | 1 |
| **2019-01-02** | -0.234414 | -1.194674 | 1.080779 | 5 | 2 |
| **2019-01-03** | -0.141572 | 0.058118 | 1.102248 | 5 | 3 |
| **2019-01-04** | 0.305088 | 0.535920 | -0.978434 | 5 | 4 |
| **2019-01-05** | 0.313383 | 0.234041 | 0.163155 | 5 | 5 |
| **2019-01-06** | 0.250613 | -0.904400 | -0.858240 | 5 | 6 |

**修改index和columns**

df.index = [i **for** i **in** range(len(df))]

df

|  | **A** | **B** | **C** | **D** | **E** |
| --- | --- | --- | --- | --- | --- |
| **0** | 0.000000 | 0.000000 | 0.925984 | 5 | 1 |
| **1** | -0.234414 | -1.194674 | 1.080779 | 5 | 2 |
| **2** | -0.141572 | 0.058118 | 1.102248 | 5 | 3 |
| **3** | 0.305088 | 0.535920 | -0.978434 | 5 | 4 |
| **4** | 0.313383 | 0.234041 | 0.163155 | 5 | 5 |
| **5** | 0.250613 | -0.904400 | -0.858240 | 5 | 6 |

df.columns = [i **for** i **in** range(df.shape[1])]

df

|  | **0** | **1** | **2** | **3** | **4** |
| --- | --- | --- | --- | --- | --- |
| **0** | 0.000000 | 0.000000 | 0.925984 | 5 | 1 |
| **1** | -0.234414 | -1.194674 | 1.080779 | 5 | 2 |
| **2** | -0.141572 | 0.058118 | 1.102248 | 5 | 3 |
| **3** | 0.305088 | 0.535920 | -0.978434 | 5 | 4 |
| **4** | 0.313383 | 0.234041 | 0.163155 | 5 | 5 |
| **5** | 0.250613 | -0.904400 | -0.858240 | 5 | 6 |

## 12.3.数值运算及统计分析

### 12.3.1.数据的查看

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

dates = pd.date\_range(start='2019-01-01', periods=6)

df = pd.DataFrame(np.random.randn(6,4), index=dates, columns=["A", "B", "C", "D"])

df

|  | **A** | **B** | **C** | **D** |
| --- | --- | --- | --- | --- |
| **2019-01-01** | -0.854043 | 0.412345 | -2.296051 | -0.048964 |
| **2019-01-02** | 1.371364 | -0.121454 | -0.299653 | 1.095375 |
| **2019-01-03** | -0.714591 | -1.103224 | 0.979250 | 0.319455 |
| **2019-01-04** | -1.397557 | 0.426008 | 0.233861 | -1.651887 |
| **2019-01-05** | 0.434026 | 0.459830 | -0.095444 | 1.220302 |
| **2019-01-06** | -0.133876 | 0.074500 | -1.028147 | 0.605402 |

**（1）查看前面的行**

df.head() *# 默认5行*

|  | **A** | **B** | **C** | **D** |
| --- | --- | --- | --- | --- |
| **2019-01-01** | -0.854043 | 0.412345 | -2.296051 | -0.048964 |
| **2019-01-02** | 1.371364 | -0.121454 | -0.299653 | 1.095375 |
| **2019-01-03** | -0.714591 | -1.103224 | 0.979250 | 0.319455 |
| **2019-01-04** | -1.397557 | 0.426008 | 0.233861 | -1.651887 |
| **2019-01-05** | 0.434026 | 0.459830 | -0.095444 | 1.220302 |

df.head(2)

|  | **A** | **B** | **C** | **D** |
| --- | --- | --- | --- | --- |
| **2019-01-01** | -0.854043 | 0.412345 | -2.296051 | -0.048964 |
| **2019-01-02** | 1.371364 | -0.121454 | -0.299653 | 1.095375 |

**（2）查看后面的行**

df.tail() *# 默认5行*

|  | **A** | **B** | **C** | **D** |
| --- | --- | --- | --- | --- |
| **2019-01-02** | 1.371364 | -0.121454 | -0.299653 | 1.095375 |
| **2019-01-03** | -0.714591 | -1.103224 | 0.979250 | 0.319455 |
| **2019-01-04** | -1.397557 | 0.426008 | 0.233861 | -1.651887 |
| **2019-01-05** | 0.434026 | 0.459830 | -0.095444 | 1.220302 |
| **2019-01-06** | -0.133876 | 0.074500 | -1.028147 | 0.605402 |

df.tail(3)

|  | **A** | **B** | **C** | **D** |
| --- | --- | --- | --- | --- |
| **2019-01-04** | -1.397557 | 0.426008 | 0.233861 | -1.651887 |
| **2019-01-05** | 0.434026 | 0.459830 | -0.095444 | 1.220302 |
| **2019-01-06** | -0.133876 | 0.074500 | -1.028147 | 0.605402 |

（3）查看总体信息

df.iloc[0, 3] = np.nan

df

|  | **A** | **B** | **C** | **D** |
| --- | --- | --- | --- | --- |
| **2019-01-01** | -0.854043 | 0.412345 | -2.296051 | NaN |
| **2019-01-02** | 1.371364 | -0.121454 | -0.299653 | 1.095375 |
| **2019-01-03** | -0.714591 | -1.103224 | 0.979250 | 0.319455 |
| **2019-01-04** | -1.397557 | 0.426008 | 0.233861 | -1.651887 |
| **2019-01-05** | 0.434026 | 0.459830 | -0.095444 | 1.220302 |
| **2019-01-06** | -0.133876 | 0.074500 | -1.028147 | 0.605402 |

df.info()

<class 'pandas.core.frame.DataFrame'>

DatetimeIndex: 6 entries, 2019-01-01 to 2019-01-06

Freq: D

Data columns (total 4 columns):

A 6 non-null float64

B 6 non-null float64

C 6 non-null float64

D 5 non-null float64

dtypes: float64(4)

memory usage: 240.0 bytes

### 12.3.2.Numpy通用函数同样适用于Pandas

**（1）向量化运算**

x = pd.DataFrame(np.arange(4).reshape(1, 4))

x

|  | **0** | **1** | **2** | **3** |
| --- | --- | --- | --- | --- |
| **0** | 0 | 1 | 2 | 3 |

x+5

|  | **0** | **1** | **2** | **3** |
| --- | --- | --- | --- | --- |
| **0** | 5 | 6 | 7 | 8 |

np.exp(x)

|  | **0** | **1** | **2** | **3** |
| --- | --- | --- | --- | --- |
| **0** | 1.0 | 2.718282 | 7.389056 | 20.085537 |

y = pd.DataFrame(np.arange(4,8).reshape(1, 4))

y

|  | **0** | **1** | **2** | **3** |
| --- | --- | --- | --- | --- |
| **0** | 4 | 5 | 6 | 7 |

x\*y

|  | **0** | **1** | **2** | **3** |
| --- | --- | --- | --- | --- |
| **0** | 0 | 5 | 12 | 21 |

（2）矩阵化运算

np.random.seed(42)

x = pd.DataFrame(np.random.randint(10, size=(3, 3)))

x

|  | **0** | **1** | **2** |
| --- | --- | --- | --- |
| **0** | 6 | 3 | 7 |
| **1** | 4 | 6 | 9 |
| **2** | 2 | 6 | 7 |

转置

z = x.Tz

|  | **0** | **1** | **2** |
| --- | --- | --- | --- |
| **0** | 6 | 4 | 2 |
| **1** | 3 | 6 | 6 |
| **2** | 7 | 9 | 7 |

np.random.seed(1)

y = pd.DataFrame(np.random.randint(10, size=(3, 3)))

y

|  | **0** | **1** | **2** |
| --- | --- | --- | --- |
| **0** | 5 | 8 | 9 |
| **1** | 5 | 0 | 0 |
| **2** | 1 | 7 | 6 |

x.dot(y)

|  | **0** | **1** | **2** |
| --- | --- | --- | --- |
| **0** | 52 | 97 | 96 |
| **1** | 59 | 95 | 90 |
| **2** | 47 | 65 | 60 |

%**timeit** x.dot(y)

133 µs ± 2.05 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)

%**timeit** np.dot(x, y)

40.2 µs ± 1.73 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)

执行相同运算，Numpy与Pandas的对比

x1 = np.array(x)

x1

array([[6, 3, 7],

[4, 6, 9],

[2, 6, 7]])

y1 = np.array(y)

y1

array([[5, 8, 9],

[5, 0, 0],

[1, 7, 6]])

%**timeit** x1.dot(y1)

22.1 µs ± 992 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)

%**timeit** np.dot(x1, y1)

22.6 µs ± 766 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)

%**timeit** np.dot(x.values, y.values)

42.9 µs ± 1.24 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)

x2 = list(x1)y2 = list(y1)x3 = []y3 = []**for** i **in** x2:

res = []

**for** j **in** i:

res.append(int(j))

x3.append(res)**for** i **in** y2:

res = []

**for** j **in** i:

res.append(int(j))

y3.append(res)

**def** f(x, y):

res = []

**for** i **in** range(len(x)):

row = []

**for** j **in** range(len(y[0])):

sum\_row = 0

**for** k **in** range(len(x[0])):

sum\_row += x[i][k]\*y[k][j]

row.append(sum\_row)

res.append(row)

**return** res

%**timeit** f(x3, y3)

4.29 ms ± 207 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)

一般来说，纯粹的计算在Numpy里执行的更快

Numpy更侧重于计算，Pandas更侧重于数据处理

**（3）广播运算**

np.random.seed(42)

x = pd.DataFrame(np.random.randint(10, size=(3, 3)), columns=list("ABC"))

x

|  | **A** | **B** | **C** |
| --- | --- | --- | --- |
| **0** | 6 | 3 | 7 |
| **1** | 4 | 6 | 9 |
| **2** | 2 | 6 | 7 |

按行广播

x.iloc[0]

A 6

B 3

C 7

Name: 0, dtype: int32

x/x.iloc[0]

|  | **A** | **B** | **C** |
| --- | --- | --- | --- |
| **0** | 1.000000 | 1.0 | 1.000000 |
| **1** | 0.666667 | 2.0 | 1.285714 |
| **2** | 0.333333 | 2.0 | 1.000000 |

按列广播

x.A

0 6

1 4

2 2

Name: A, dtype: int32

x.div(x.A, axis=0) *# add sub div mul*

|  | **A** | **B** | **C** |
| --- | --- | --- | --- |
| **0** | 1.0 | 0.5 | 1.166667 |
| **1** | 1.0 | 1.5 | 2.250000 |
| **2** | 1.0 | 3.0 | 3.500000 |

x.div(x.iloc[0], axis=1)

|  | **A** | **B** | **C** |
| --- | --- | --- | --- |
| **0** | 1.000000 | 1.0 | 1.000000 |
| **1** | 0.666667 | 2.0 | 1.285714 |
| **2** | 0.333333 | 2.0 | 1.000000 |

### 12.3.3.其他用法

**（1）索引对齐**

A = pd.DataFrame(np.random.randint(0, 20, size=(2, 2)), columns=list("AB"))

A

|  | **A** | **B** |
| --- | --- | --- |
| **0** | 3 | 7 |
| **1** | 2 | 1 |

B = pd.DataFrame(np.random.randint(0, 10, size=(3, 3)), columns=list("ABC"))

B

|  | **A** | **B** | **C** |
| --- | --- | --- | --- |
| **0** | 7 | 5 | 1 |
| **1** | 4 | 0 | 9 |
| **2** | 5 | 8 | 0 |

pandas会自动对齐两个对象的索引，没有的值用np.nan表示

A+B

|  | **A** | **B** | **C** |
| --- | --- | --- | --- |
| **0** | 10.0 | 12.0 | NaN |
| **1** | 6.0 | 1.0 | NaN |
| **2** | NaN | NaN | NaN |

缺省值也可用fill\_value来填充

A.add(B, fill\_value=0)

|  | **A** | **B** | **C** |
| --- | --- | --- | --- |
| **0** | 10.0 | 12.0 | 1.0 |
| **1** | 6.0 | 1.0 | 9.0 |
| **2** | 5.0 | 8.0 | 0.0 |

A\*B

|  | **A** | **B** | **C** |
| --- | --- | --- | --- |
| **0** | 21.0 | 35.0 | NaN |
| **1** | 8.0 | 0.0 | NaN |
| **2** | NaN | NaN | NaN |

**（2）统计相关**

数据种类统计

y = np.random.randint(3, size=20)

y

array([2, 2, 2, 1, 2, 1, 1, 2, 1, 2, 2, 0, 2, 0, 2, 2, 0, 0, 2, 1])

np.unique(y)

array([0, 1, 2])

**from** **collections** **import** Counter

Counter(y)

Counter({2: 11, 1: 5, 0: 4})

y1 = pd.DataFrame(y, columns=["A"])

y1

|  | **A** |
| --- | --- |
| **0** | 2 |
| **...** | ... |
| **19** | 1 |

np.unique(y1)

array([0, 1, 2])

y1["A"].value\_counts()

2 11

1 5

0 4

Name: A, dtype: int64

产生新的结果，并进行排序

population\_dict = {"BeiJing": 2154,

"ShangHai": 2424,

"ShenZhen": 1303,

"HangZhou": 981 }

population = pd.Series(population\_dict)

GDP\_dict = {"BeiJing": 30320,

"ShangHai": 32680,

"ShenZhen": 24222,

"HangZhou": 13468 }

GDP = pd.Series(GDP\_dict)

city\_info = pd.DataFrame({"population": population,"GDP": GDP})

city\_info

|  | **population** | **GDP** |
| --- | --- | --- |
| **BeiJing** | 2154 | 30320 |
| **ShangHai** | 2424 | 32680 |
| **ShenZhen** | 1303 | 24222 |
| **HangZhou** | 981 | 13468 |

city\_info["per\_GDP"] = city\_info["GDP"]/city\_info["population"]

city\_info

|  | **population** | **GDP** | **per\_GDP** |
| --- | --- | --- | --- |
| **BeiJing** | 2154 | 30320 | 14.076137 |
| **ShangHai** | 2424 | 32680 | 13.481848 |
| **ShenZhen** | 1303 | 24222 | 18.589409 |
| **HangZhou** | 981 | 13468 | 13.728848 |

递增排序

city\_info.sort\_values(by="per\_GDP")

|  | **population** | **GDP** | **per\_GDP** |
| --- | --- | --- | --- |
| **ShangHai** | 2424 | 32680 | 13.481848 |
| **HangZhou** | 981 | 13468 | 13.728848 |
| **BeiJing** | 2154 | 30320 | 14.076137 |
| **ShenZhen** | 1303 | 24222 | 18.589409 |

递减排序

city\_info.sort\_values(by="per\_GDP", ascending=**False**)

|  | **population** | **GDP** | **per\_GDP** |
| --- | --- | --- | --- |
| **ShenZhen** | 1303 | 24222 | 18.589409 |
| **BeiJing** | 2154 | 30320 | 14.076137 |
| **HangZhou** | 981 | 13468 | 13.728848 |
| **ShangHai** | 2424 | 32680 | 13.481848 |

**按轴进行排序**

data = pd.DataFrame(np.random.randint(20, size=(3, 4)), index=[2, 1, 0], columns=list("CBAD"))

data

|  | **C** | **B** | **A** | **D** |
| --- | --- | --- | --- | --- |
| **2** | 3 | 13 | 17 | 8 |
| **1** | 1 | 19 | 14 | 6 |
| **0** | 11 | 7 | 14 | 2 |

行排序

data.sort\_index()

|  | **C** | **B** | **A** | **D** |
| --- | --- | --- | --- | --- |
| **0** | 11 | 7 | 14 | 2 |
| **1** | 1 | 19 | 14 | 6 |
| **2** | 3 | 13 | 17 | 8 |

列排序

data.sort\_index(axis=1)

|  | **A** | **B** | **C** | **D** |
| --- | --- | --- | --- | --- |
| **2** | 17 | 13 | 3 | 8 |
| **1** | 14 | 19 | 1 | 6 |
| **0** | 14 | 7 | 11 | 2 |

data.sort\_index(axis=1, ascending=**False**)

|  | **D** | **C** | **B** | **A** |
| --- | --- | --- | --- | --- |
| **2** | 8 | 3 | 13 | 17 |
| **1** | 6 | 1 | 19 | 14 |
| **0** | 2 | 11 | 7 | 14 |

**统计方法**

df = pd.DataFrame(np.random.normal(2, 4, size=(6, 4)),columns=list("ABCD"))

df

|  | **A** | **B** | **C** | **D** |
| --- | --- | --- | --- | --- |
| **0** | 1.082198 | 3.557396 | -3.060476 | 6.367969 |
| **1** | 13.113252 | 6.774559 | 2.874553 | 5.527044 |
| **2** | -2.036341 | -4.333177 | 5.094802 | -0.152567 |
| **3** | -3.386712 | -1.522365 | -2.522209 | 2.537716 |
| **4** | 4.328491 | 5.550994 | 5.577329 | 5.019991 |
| **5** | 1.171336 | -0.493910 | -4.032613 | 6.398588 |

非空个数

df.count()

A 6

B 6

C 6

D 6

dtype: int64

求和

df.sum()

A 14.272224

B 9.533497

C 3.931385

D 25.698741

dtype: float64

df.sum(axis=1)

0 7.947086

1 28.289408

2 -1.427283

3 -4.893571

4 20.476806

5 3.043402

dtype: float64

最大值 最小值

df.min()

A -3.386712

B -4.333177

C -4.032613

D -0.152567

dtype: float64

df.max(axis=1)

0 6.367969

1 13.113252

2 5.094802

3 2.537716

4 5.577329

5 6.398588

dtype: float64

df.idxmax()

A 1

B 1

C 4

D 5

dtype: int64

均值

df.mean()

A 2.378704

B 1.588916

C 0.655231

D 4.283124

dtype: float64

方差

df.var()

A 34.980702

B 19.110656

C 18.948144

D 6.726776

dtype: float64

标准差

df.std()

A 5.914449

B 4.371574

C 4.352947

D 2.593603

dtype: float64

中位数

df.median()

A 1.126767

B 1.531743

C 0.176172

D 5.273518

dtype: float64

众数

data = pd.DataFrame(np.random.randint(5, size=(10, 2)), columns=list("AB"))

data

|  | **A** | **B** |
| --- | --- | --- |
| **0** | 4 | 2 |
| **1** | 3 | 2 |
| **2** | 2 | 0 |
| **3** | 2 | 4 |
| **4** | 2 | 0 |
| **5** | 4 | 1 |
| **6** | 2 | 0 |
| **7** | 1 | 1 |
| **8** | 3 | 4 |
| **9** | 2 | 0 |

data.mode()

|  | **A** | **B** |
| --- | --- | --- |
| **0** | 2 | 0 |

75%分位数

df.quantile(0.75)

A 3.539202

B 5.052594

C 4.539740

D 6.157738

Name: 0.75, dtype: float64

统计所有

df.describe()

|  | **A** | **B** | **C** | **D** |
| --- | --- | --- | --- | --- |
| **count** | 6.000000 | 6.000000 | 6.000000 | 6.000000 |
| **mean** | 2.378704 | 1.588916 | 0.655231 | 4.283124 |
| **std** | 5.914449 | 4.371574 | 4.352947 | 2.593603 |
| **min** | -3.386712 | -4.333177 | -4.032613 | -0.152567 |
| **25%** | -1.256706 | -1.265251 | -2.925910 | 3.158284 |
| **50%** | 1.126767 | 1.531743 | 0.176172 | 5.273518 |
| **75%** | 3.539202 | 5.052594 | 4.539740 | 6.157738 |
| **max** | 13.113252 | 6.774559 | 5.577329 | 6.398588 |

data\_2 = pd.DataFrame([["a", "a", "c", "d"],

["c", "a", "c", "b"],

["a", "a", "d", "c"]], columns=list("ABCD"))

data\_2

|  | **A** | **B** | **C** | **D** |
| --- | --- | --- | --- | --- |
| **0** | a | a | c | d |
| **1** | c | a | c | b |
| **2** | a | a | d | c |

data\_2.describe()

|  | **A** | **B** | **C** | **D** |
| --- | --- | --- | --- | --- |
| **count** | 3 | 3 | 3 | 3 |
| **unique** | 2 | 1 | 2 | 3 |
| **top** | a | a | c | d |
| **freq** | 2 | 3 | 2 | 1 |

相关性系数和协方差

df.corr()

|  | **A** | **B** | **C** | **D** |
| --- | --- | --- | --- | --- |
| **A** | 1.000000 | 0.831063 | 0.331060 | 0.510821 |
| **B** | 0.831063 | 1.000000 | 0.179244 | 0.719112 |
| **C** | 0.331060 | 0.179244 | 1.000000 | -0.450365 |
| **D** | 0.510821 | 0.719112 | -0.450365 | 1.000000 |

df.corrwith(df["A"])

A 1.000000

B 0.831063

C 0.331060

D 0.510821

dtype: float64

**自定义输出**

apply（method）的用法：使用method方法默认对每一列进行相应的操作

df

|  | **A** | **B** | **C** | **D** |
| --- | --- | --- | --- | --- |
| **0** | 1.082198 | 3.557396 | -3.060476 | 6.367969 |
| **1** | 13.113252 | 6.774559 | 2.874553 | 5.527044 |
| **2** | -2.036341 | -4.333177 | 5.094802 | -0.152567 |
| **3** | -3.386712 | -1.522365 | -2.522209 | 2.537716 |
| **4** | 4.328491 | 5.550994 | 5.577329 | 5.019991 |
| **5** | 1.171336 | -0.493910 | -4.032613 | 6.398588 |

df.apply(np.cumsum)

|  | **A** | **B** | **C** | **D** |
| --- | --- | --- | --- | --- |
| **0** | 1.082198 | 3.557396 | -3.060476 | 6.367969 |
| **1** | 14.195450 | 10.331955 | -0.185923 | 11.895013 |
| **2** | 12.159109 | 5.998778 | 4.908878 | 11.742447 |
| **3** | 8.772397 | 4.476413 | 2.386669 | 14.280162 |
| **4** | 13.100888 | 10.027406 | 7.963999 | 19.300153 |
| **5** | 14.272224 | 9.533497 | 3.931385 | 25.698741 |

df.apply(np.cumsum, axis=1)

|  | **A** | **B** | **C** | **D** |
| --- | --- | --- | --- | --- |
| **0** | 1.082198 | 4.639594 | 1.579117 | 7.947086 |
| **1** | 13.113252 | 19.887811 | 22.762364 | 28.289408 |
| **2** | -2.036341 | -6.369518 | -1.274717 | -1.427283 |
| **3** | -3.386712 | -4.909077 | -7.431287 | -4.893571 |
| **4** | 4.328491 | 9.879485 | 15.456814 | 20.476806 |
| **5** | 1.171336 | 0.677427 | -3.355186 | 3.043402 |

df.apply(sum)

A 14.272224

B 9.533497

C 3.931385

D 25.698741

dtype: float64

df.sum()

A 14.272224

B 9.533497

C 3.931385

D 25.698741

dtype: float64

df.apply(**lambda** x: x.max()-x.min())

A 16.499965

B 11.107736

C 9.609942

D 6.551155

dtype: float64

**def** my\_describe(x):

**return** pd.Series([x.count(), x.mean(), x.max(), x.idxmin(), x.std()], \

index=["Count", "mean", "max", "idxmin", "std"])

df.apply(my\_describe)

|  | **A** | **B** | **C** | **D** |
| --- | --- | --- | --- | --- |
| **Count** | 6.000000 | 6.000000 | 6.000000 | 6.000000 |
| **mean** | 2.378704 | 1.588916 | 0.655231 | 4.283124 |
| **max** | 13.113252 | 6.774559 | 5.577329 | 6.398588 |
| **idxmin** | 3.000000 | 2.000000 | 5.000000 | 2.000000 |
| **std** | 5.914449 | 4.371574 | 4.352947 | 2.593603 |

## 12.4.缺失值处理

### 12.4.1.发现缺失值

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

data = pd.DataFrame(np.array([[1, np.nan, 2],

[np.nan, 3, 4],

[5, 6, **None**]]), columns=["A", "B", "C"])

data

|  | **A** | **B** | **C** |
| --- | --- | --- | --- |
| **0** | 1 | NaN | 2 |
| **1** | NaN | 3 | 4 |
| **2** | 5 | 6 | None |

注意：有None、字符串等，数据类型全部变为object，它比int和float更消耗资源

data.dtypes

A object

B object

C object

dtype: object

data.isnull()

|  | **A** | **B** | **C** |
| --- | --- | --- | --- |
| **0** | False | True | False |
| **1** | True | False | False |
| **2** | False | False | True |

data.notnull()

|  | **A** | **B** | **C** |
| --- | --- | --- | --- |
| **0** | True | False | True |
| **1** | False | True | True |
| **2** | True | True | False |

### 12.4.2.删除缺失值

data = pd.DataFrame(np.array([[1, np.nan, 2, 3],

[np.nan, 4, 5, 6],

[7, 8, np.nan, 9],

[10, 11 , 12, 13]]), columns=["A", "B", "C", "D"])

data

|  | **A** | **B** | **C** | **D** |
| --- | --- | --- | --- | --- |
| **0** | 1.0 | NaN | 2.0 | 3.0 |
| **1** | NaN | 4.0 | 5.0 | 6.0 |
| **2** | 7.0 | 8.0 | NaN | 9.0 |
| **3** | 10.0 | 11.0 | 12.0 | 13.0 |

注意：np.nan是一种特殊的浮点数

data.dtypes

A float64

B float64

C float64

D float64

dtype: object

**（1）删除整行**

data.dropna()

|  | **A** | **B** | **C** | **D** |
| --- | --- | --- | --- | --- |
| **3** | 10.0 | 11.0 | 12.0 | 13.0 |

（2）删除整列

data.dropna(axis="columns")

|  | **D** |
| --- | --- |
| **0** | 3.0 |
| **1** | 6.0 |
| **2** | 9.0 |
| **3** | 13.0 |

data["D"] = np.nan

data

|  | **A** | **B** | **C** | **D** |
| --- | --- | --- | --- | --- |
| **0** | 1.0 | NaN | 2.0 | NaN |
| **1** | NaN | 4.0 | 5.0 | NaN |
| **2** | 7.0 | 8.0 | NaN | NaN |
| **3** | 10.0 | 11.0 | 12.0 | NaN |

data.dropna(axis="columns", how="all")

|  | **A** | **B** | **C** |
| --- | --- | --- | --- |
| **0** | 1.0 | NaN | 2.0 |
| **1** | NaN | 4.0 | 5.0 |
| **2** | 7.0 | 8.0 | NaN |
| **3** | 10.0 | 11.0 | 12.0 |

data.dropna(axis="columns", how="any")

|  |
| --- |
| **0** |
| **1** |
| **2** |
| **3** |

data.loc[3] = np.nan

data

|  | **A** | **B** | **C** | **D** |
| --- | --- | --- | --- | --- |
| **0** | 1.0 | NaN | 2.0 | NaN |
| **1** | NaN | 4.0 | 5.0 | NaN |
| **2** | 7.0 | 8.0 | NaN | NaN |
| **3** | NaN | NaN | NaN | NaN |

data.dropna(how="all")

|  | **A** | **B** | **C** | **D** |
| --- | --- | --- | --- | --- |
| **0** | 1.0 | NaN | 2.0 | NaN |
| **1** | NaN | 4.0 | 5.0 | NaN |
| **2** | 7.0 | 8.0 | NaN | NaN |

### 12.4.3.填充缺失值

data = pd.DataFrame(np.array([[1, np.nan, 2, 3],

[np.nan, 4, 5, 6],

[7, 8, np.nan, 9],

[10, 11 , 12, 13]]), columns=["A", "B", "C", "D"])

data

|  | **A** | **B** | **C** | **D** |
| --- | --- | --- | --- | --- |
| **0** | 1.0 | NaN | 2.0 | 3.0 |
| **1** | NaN | 4.0 | 5.0 | 6.0 |
| **2** | 7.0 | 8.0 | NaN | 9.0 |
| **3** | 10.0 | 11.0 | 12.0 | 13.0 |

data.fillna(value=5)

|  | **A** | **B** | **C** | **D** |
| --- | --- | --- | --- | --- |
| **0** | 1.0 | 5.0 | 2.0 | 3.0 |
| **1** | 5.0 | 4.0 | 5.0 | 6.0 |
| **2** | 7.0 | 8.0 | 5.0 | 9.0 |
| **3** | 10.0 | 11.0 | 12.0 | 13.0 |

用均值进行替换

fill = data.mean()

fill

A 6.000000

B 7.666667

C 6.333333

D 7.750000

dtype: float64

data.fillna(value=fill)

|  | **A** | **B** | **C** | **D** |
| --- | --- | --- | --- | --- |
| **0** | 1.0 | 7.666667 | 2.000000 | 3.0 |
| **1** | 6.0 | 4.000000 | 5.000000 | 6.0 |
| **2** | 7.0 | 8.000000 | 6.333333 | 9.0 |
| **3** | 10.0 | 11.000000 | 12.000000 | 13.0 |

fill = data.stack().mean()

fill

7.0

data.fillna(value=fill)

|  | **A** | **B** | **C** | **D** |
| --- | --- | --- | --- | --- |
| **0** | 1.0 | 7.0 | 2.0 | 3.0 |
| **1** | 7.0 | 4.0 | 5.0 | 6.0 |
| **2** | 7.0 | 8.0 | 7.0 | 9.0 |
| **3** | 10.0 | 11.0 | 12.0 | 13.0 |

## 12.5.合并数据

构造一个生产DataFrame的函数

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

**def** make\_df(cols, ind):

"一个简单的DataFrame"

data = {c: [str(c)+str(i) **for** i **in** ind] **for** c **in** cols}

**return** pd.DataFrame(data, ind)

make\_df("ABC", range(3))

|  | **A** | **B** | **C** |
| --- | --- | --- | --- |
| **0** | A0 | B0 | C0 |
| **1** | A1 | B1 | C1 |
| **2** | A2 | B2 | C2 |

**垂直合并**

df\_1 = make\_df("AB", [1, 2])

df\_2 = make\_df("AB", [3, 4])

print(df\_1)

print(df\_2)

A B

1 A1 B1

2 A2 B2

A B

3 A3 B3

4 A4 B4

pd.concat([df\_1, df\_2])

|  | **A** | **B** |
| --- | --- | --- |
| **1** | A1 | B1 |
| **2** | A2 | B2 |
| **3** | A3 | B3 |
| **4** | A4 | B4 |

**水平合并**

pd.concat([df\_1, df\_2], axis=1)

|  | **A** | **B** | **A** | **B** |
| --- | --- | --- | --- | --- |
| **1** | A1 | B1 | A1 | B1 |
| **2** | A2 | B2 | A2 | B2 |

**索引重叠**

行重叠

df\_5 = make\_df("AB", [1, 2])

df\_6 = make\_df("AB", [1, 2])

print(df\_5)

print(df\_6)

A B

1 A1 B1

2 A2 B2

A B

1 A1 B1

2 A2 B2

pd.concat([df\_5, df\_6])

|  | **A** | **B** |
| --- | --- | --- |
| **1** | A1 | B1 |
| **2** | A2 | B2 |
| **1** | A1 | B1 |
| **2** | A2 | B2 |

pd.concat([df\_5, df\_6],ignore\_index=**True**)

|  | **A** | **B** |
| --- | --- | --- |
| **0** | A1 | B1 |
| **1** | A2 | B2 |
| **2** | A1 | B1 |
| **3** | A2 | B2 |

列重叠

df\_7 = make\_df("ABC", [1, 2])

df\_8 = make\_df("BCD", [1, 2])

print(df\_7)

print(df\_8)

A B C

1 A1 B1 C1

2 A2 B2 C2

B C D

1 B1 C1 D1

2 B2 C2 D2

pd.concat([df\_7, df\_8], axis=1)

|  | **A** | **B** | **C** | **B** | **C** | **D** |
| --- | --- | --- | --- | --- | --- | --- |
| **1** | A1 | B1 | C1 | B1 | C1 | D1 |
| **2** | A2 | B2 | C2 | B2 | C2 | D2 |

pd.concat([df\_7, df\_8],axis=1, ignore\_index=**True**)

|  | **0** | **1** | **2** | **3** | **4** | **5** |
| --- | --- | --- | --- | --- | --- | --- |
| **1** | A1 | B1 | C1 | B1 | C1 | D1 |
| **2** | A2 | B2 | C2 | B2 | C2 | D2 |

**对齐合并merge()**

df\_9 = make\_df("AB", [1, 2])

df\_10 = make\_df("BC", [1, 2])

print(df\_9)

print(df\_10)

A B

1 A1 B1

2 A2 B2

B C

1 B1 C1

2 B2 C2

pd.merge(df\_9, df\_10)

|  | **A** | **B** | **C** |
| --- | --- | --- | --- |
| **0** | A1 | B1 | C1 |
| **1** | A2 | B2 | C2 |

df\_9 = make\_df("AB", [1, 2])

df\_10 = make\_df("CB", [2, 1])

print(df\_9)

print(df\_10)

A B

1 A1 B1

2 A2 B2

C B

2 C2 B2

1 C1 B1

pd.merge(df\_9, df\_10)

|  | **A** | **B** | **C** |
| --- | --- | --- | --- |
| **0** | A1 | B1 | C1 |
| **1** | A2 | B2 | C2 |

【例】 合并城市信息

population\_dict = {"city": ("BeiJing", "HangZhou", "ShenZhen"),

"pop": (2154, 981, 1303)}

population = pd.DataFrame(population\_dict)

population

|  | **city** | **pop** |
| --- | --- | --- |
| **0** | BeiJing | 2154 |
| **1** | HangZhou | 981 |
| **2** | ShenZhen | 1303 |

GDP\_dict = {"city": ("BeiJing", "ShangHai", "HangZhou"),

"GDP": (30320, 32680, 13468)}

GDP = pd.DataFrame(GDP\_dict)

GDP

|  | **city** | **GDP** |
| --- | --- | --- |
| **0** | BeiJing | 30320 |
| **1** | ShangHai | 32680 |
| **2** | HangZhou | 13468 |

city\_info = pd.merge(population, GDP)

city\_info

|  | **city** | **pop** | **GDP** |
| --- | --- | --- | --- |
| **0** | BeiJing | 2154 | 30320 |
| **1** | HangZhou | 981 | 13468 |

city\_info = pd.merge(population, GDP, how="outer")

city\_info

|  | **city** | **pop** | **GDP** |
| --- | --- | --- | --- |
| **0** | BeiJing | 2154.0 | 30320.0 |
| **1** | HangZhou | 981.0 | 13468.0 |
| **2** | ShenZhen | 1303.0 | NaN |
| **3** | ShangHai | NaN | 32680.0 |

## 12.6.分组和数据透视表

### 12.6.1.分组

df = pd.DataFrame({"key":["A", "B", "C", "C", "B", "A"],

"data1": range(6),

"data2": np.random.randint(0, 10, size=6)})

df

|  | **key** | **data1** | **data2** |
| --- | --- | --- | --- |
| **0** | A | 0 | 1 |
| **1** | B | 1 | 4 |
| **2** | C | 2 | 9 |
| **3** | C | 3 | 9 |
| **4** | B | 4 | 1 |
| **5** | A | 5 | 9 |

**延迟计算**

df.groupby("key")

<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000002276795A240>

df.groupby("key").sum()

|  | **data1** | **data2** |
| --- | --- | --- |
| **key** |  |  |
| **A** | 5 | 10 |
| **B** | 5 | 6 |
| **C** | 5 | 11 |

df.groupby("key").mean()

|  | **data1** | **data2** |
| --- | --- | --- |
| **key** |  |  |
| **A** | 2.5 | 5.0 |
| **B** | 2.5 | 3.0 |
| **C** | 2.5 | 5.5 |

**for** i **in** df.groupby("key"):

print(str(i))

('A', key data1 data2

0 A 0 2

5 A 5 8)

('B', key data1 data2

1 B 1 2

4 B 4 4)

('C', key data1 data2

2 C 2 8

3 C 3 3)

**按列取值**

df.groupby("key")["data2"].sum()

key

A 10

B 6

C 11

Name: data2, dtype: int32

**按组迭代**

**for** data, group **in** df.groupby("key"):

print("**{0:5}** shape=**{1}**".format(data, group.shape))

A shape=(2, 3)

B shape=(2, 3)

C shape=(2, 3)

**调用方法**

df.groupby("key")["data1"].describe()

|  | **count** | **mean** | **std** | **min** | **25%** | **50%** | **75%** | **max** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **key** |  |  |  |  |  |  |  |  |
| **A** | 2.0 | 2.5 | 3.535534 | 0.0 | 1.25 | 2.5 | 3.75 | 5.0 |
| **B** | 2.0 | 2.5 | 2.121320 | 1.0 | 1.75 | 2.5 | 3.25 | 4.0 |
| **C** | 2.0 | 2.5 | 0.707107 | 2.0 | 2.25 | 2.5 | 2.75 | 3.0 |

**支持更复杂的操作**

df.groupby("key").aggregate(["min", "median", "max"])

|  | **data1** | | | **data2** | | |
| --- | --- | --- | --- | --- | --- | --- |
|  | **min** | **median** | **max** | **min** | **median** | **max** |
| **key** |  |  |  |  |  |  |
| **A** | 0 | 2.5 | 5 | 2 | 5.0 | 8 |
| **B** | 1 | 2.5 | 4 | 2 | 3.0 | 4 |
| **C** | 2 | 2.5 | 3 | 3 | 5.5 | 8 |

**过滤**

**def** filter\_func(x):

**return** x["data2"].std() > 3

df.groupby("key")["data2"].std()

key

A 4.242641

B 1.414214

C 3.535534

Name: data2, dtype: float64

df.groupby("key").filter(filter\_func)

|  | **key** | **data1** | **data2** |
| --- | --- | --- | --- |
| **0** | A | 0 | 2 |
| **2** | C | 2 | 8 |
| **3** | C | 3 | 3 |
| **5** | A | 5 | 8 |

**apply（）方法**

df

|  | **key** | **data1** | **data2** |
| --- | --- | --- | --- |
| **0** | A | 0 | 2 |
| **1** | B | 1 | 2 |
| **2** | C | 2 | 8 |
| **3** | C | 3 | 3 |
| **4** | B | 4 | 4 |
| **5** | A | 5 | 8 |

df.groupby("key").apply(**lambda** x: x-x.mean())

|  | **data1** | **data2** |
| --- | --- | --- |
| **0** | -2.5 | -3.0 |
| **1** | -1.5 | -1.0 |
| **2** | -0.5 | 2.5 |
| **3** | 0.5 | -2.5 |
| **4** | 1.5 | 1.0 |
| **5** | 2.5 | 3.0 |

**def** norm\_by\_data2(x):

x["data1"] /= x["data2"].sum()

**return** x

df.groupby("key").apply(norm\_by\_data2)

|  | **key** | **data1** | **data2** |
| --- | --- | --- | --- |
| **0** | A | 0.000000 | 2 |
| **1** | B | 0.166667 | 2 |
| **2** | C | 0.181818 | 8 |
| **3** | C | 0.272727 | 3 |
| **4** | B | 0.666667 | 4 |
| **5** | A | 0.500000 | 8 |

**将列表、数组设为分组键**

L = [0, 1, 0, 1, 2, 0]

df.groupby(L).sum()

|  | **data1** | **data2** |
| --- | --- | --- |
| **0** | 7 | 18 |
| **1** | 4 | 5 |
| **2** | 4 | 4 |

**用字典将索引映射到分组**

df2 = df.set\_index("key")

df2

|  | **data1** | **data2** |
| --- | --- | --- |
| **key** |  |  |
| **A** | 0 | 2 |
| **B** | 1 | 2 |
| **C** | 2 | 8 |
| **C** | 3 | 3 |
| **B** | 4 | 4 |
| **A** | 5 | 8 |

mapping = {"A": "first", "B": "constant", "C": "constant"}

df2.groupby(mapping).sum()

|  | **data1** | **data2** |
| --- | --- | --- |
| **constant** | 10 | 17 |
| **first** | 5 | 10 |

**任意Python函数**

df2.groupby(str.lower).mean()

|  | **data1** | **data2** |
| --- | --- | --- |
| **a** | 2.5 | 5.0 |
| **b** | 2.5 | 3.0 |
| **c** | 2.5 | 5.5 |

**多个有效值组成的列表**

df2.groupby([str.lower, mapping]).mean()

|  |  | **data1** | **data2** |
| --- | --- | --- | --- |
| **a** | **first** | 2.5 | 5.0 |
| **b** | **constant** | 2.5 | 3.0 |
| **c** | **constant** | 2.5 | 5.5 |

### 12.6.2.案例

【例1】 行星观测数据处理

**import** **seaborn** **as** **sns**

**import** **pandas** **as** **pd**

*# planets = sns.load\_dataset("planets")*

planets = pd.read\_csv("data/planets.csv") *# 读取本地的csv文件*

planets.shape

(1035, 6)

planets.head()

|  | **method** | **number** | **orbital\_period** | **mass** | **distance** | **year** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | Radial Velocity | 1 | 269.300 | 7.10 | 77.40 | 2006 |
| **1** | Radial Velocity | 1 | 874.774 | 2.21 | 56.95 | 2008 |
| **2** | Radial Velocity | 1 | 763.000 | 2.60 | 19.84 | 2011 |
| **3** | Radial Velocity | 1 | 326.030 | 19.40 | 110.62 | 2007 |
| **4** | Radial Velocity | 1 | 516.220 | 10.50 | 119.47 | 2009 |

planets.describe()

|  | **number** | **orbital\_period** | **mass** | **distance** | **year** |
| --- | --- | --- | --- | --- | --- |
| **count** | 1035.000000 | 992.000000 | 513.000000 | 808.000000 | 1035.000000 |
| **mean** | 1.785507 | 2002.917596 | 2.638161 | 264.069282 | 2009.070531 |
| **std** | 1.240976 | 26014.728304 | 3.818617 | 733.116493 | 3.972567 |
| **min** | 1.000000 | 0.090706 | 0.003600 | 1.350000 | 1989.000000 |
| **25%** | 1.000000 | 5.442540 | 0.229000 | 32.560000 | 2007.000000 |
| **50%** | 1.000000 | 39.979500 | 1.260000 | 55.250000 | 2010.000000 |
| **75%** | 2.000000 | 526.005000 | 3.040000 | 178.500000 | 2012.000000 |
| **max** | 7.000000 | 730000.000000 | 25.000000 | 8500.000000 | 2014.000000 |

decade = 10 \* (planets["year"] // 10)

decade.head()

0 2000

1 2000

2 2010

3 2000

4 2000

Name: year, dtype: int64

decade = decade.astype(str) + "s"

decade.name = "decade"

decade.head()

0 2000s

1 2000s

2 2010s

3 2000s

4 2000s

Name: decade, dtype: object

planets.groupby(["method", decade]).sum()

|  |  | **number** | **orbital\_period** | **mass** | **distance** | **year** |
| --- | --- | --- | --- | --- | --- | --- |
| **method** | **decade** |  |  |  |  |  |
| **Astrometry** | **2010s** | 2 | 1.262360e+03 | 0.00000 | 35.75 | 4023 |
| **Eclipse Timing Variations** | **2000s** | 5 | 1.930800e+04 | 6.05000 | 261.44 | 6025 |
| **2010s** | 10 | 2.345680e+04 | 4.20000 | 1000.00 | 12065 |
| **Imaging** | **2000s** | 29 | 1.350935e+06 | 0.00000 | 956.83 | 40139 |
| **2010s** | 21 | 6.803750e+04 | 0.00000 | 1210.08 | 36208 |
| **Microlensing** | **2000s** | 12 | 1.732500e+04 | 0.00000 | 0.00 | 20070 |
| **2010s** | 15 | 4.750000e+03 | 0.00000 | 41440.00 | 26155 |
| **Orbital Brightness Modulation** | **2010s** | 5 | 2.127920e+00 | 0.00000 | 2360.00 | 6035 |
| **Pulsar Timing** | **1990s** | 9 | 1.900153e+02 | 0.00000 | 0.00 | 5978 |
| **2000s** | 1 | 3.652500e+04 | 0.00000 | 0.00 | 2003 |
| **2010s** | 1 | 9.070629e-02 | 0.00000 | 1200.00 | 2011 |
| **Pulsation Timing Variations** | **2000s** | 1 | 1.170000e+03 | 0.00000 | 0.00 | 2007 |
| **Radial Velocity** | **1980s** | 1 | 8.388800e+01 | 11.68000 | 40.57 | 1989 |
| **1990s** | 52 | 1.091561e+04 | 68.17820 | 723.71 | 55943 |
| **2000s** | 475 | 2.633526e+05 | 945.31928 | 15201.16 | 619775 |
| **2010s** | 424 | 1.809630e+05 | 316.47890 | 11382.67 | 432451 |
| **Transit** | **2000s** | 64 | 2.897102e+02 | 0.00000 | 31823.31 | 124462 |
| **2010s** | 712 | 8.087813e+03 | 1.47000 | 102419.46 | 673999 |
| **Transit Timing Variations** | **2010s** | 9 | 2.393505e+02 | 0.00000 | 3313.00 | 8050 |

planets.groupby(["method", decade])[["number"]].sum().unstack().fillna(0)

|  | **number** | | | |
| --- | --- | --- | --- | --- |
| **decade** | **1980s** | **1990s** | **2000s** | **2010s** |
| **method** |  |  |  |  |
| **Astrometry** | 0.0 | 0.0 | 0.0 | 2.0 |
| **Eclipse Timing Variations** | 0.0 | 0.0 | 5.0 | 10.0 |
| **Imaging** | 0.0 | 0.0 | 29.0 | 21.0 |
| **Microlensing** | 0.0 | 0.0 | 12.0 | 15.0 |
| **Orbital Brightness Modulation** | 0.0 | 0.0 | 0.0 | 5.0 |
| **Pulsar Timing** | 0.0 | 9.0 | 1.0 | 1.0 |
| **Pulsation Timing Variations** | 0.0 | 0.0 | 1.0 | 0.0 |
| **Radial Velocity** | 1.0 | 52.0 | 475.0 | 424.0 |
| **Transit** | 0.0 | 0.0 | 64.0 | 712.0 |
| **Transit Timing Variations** | 0.0 | 0.0 | 0.0 | 9.0 |

【例2】泰坦尼克号乘客数据分析

**import** **seaborn** **as** **sns**

*# titanic = sns.load\_dataset("titanic")*

titanic = pd.read\_csv("data/titanic.csv") *# 读取本地的csv文件*

titanic.head()

|  | **survived** | **pclass** | **sex** | **age** | **sibsp** | **parch** | **fare** | **embarked** | **class** | **who** | **adult\_male** | **deck** | **embark\_town** | **alive** | **alone** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0 | 3 | male | 22.0 | 1 | 0 | 7.2500 | S | Third | man | True | NaN | Southampton | no | False |
| **1** | 1 | 1 | female | 38.0 | 1 | 0 | 71.2833 | C | First | woman | False | C | Cherbourg | yes | False |
| **2** | 1 | 3 | female | 26.0 | 0 | 0 | 7.9250 | S | Third | woman | False | NaN | Southampton | yes | True |
| **3** | 1 | 1 | female | 35.0 | 1 | 0 | 53.1000 | S | First | woman | False | C | Southampton | yes | False |
| **4** | 0 | 3 | male | 35.0 | 0 | 0 | 8.0500 | S | Third | man | True | NaN | Southampton | no | True |

titanic.describe()

|  | **survived** | **pclass** | **age** | **sibsp** | **parch** | **fare** |
| --- | --- | --- | --- | --- | --- | --- |
| **count** | 891.000000 | 891.000000 | 714.000000 | 891.000000 | 891.000000 | 891.000000 |
| **mean** | 0.383838 | 2.308642 | 29.699118 | 0.523008 | 0.381594 | 32.204208 |
| **std** | 0.486592 | 0.836071 | 14.526497 | 1.102743 | 0.806057 | 49.693429 |
| **min** | 0.000000 | 1.000000 | 0.420000 | 0.000000 | 0.000000 | 0.000000 |
| **25%** | 0.000000 | 2.000000 | 20.125000 | 0.000000 | 0.000000 | 7.910400 |
| **50%** | 0.000000 | 3.000000 | 28.000000 | 0.000000 | 0.000000 | 14.454200 |
| **75%** | 1.000000 | 3.000000 | 38.000000 | 1.000000 | 0.000000 | 31.000000 |
| **max** | 1.000000 | 3.000000 | 80.000000 | 8.000000 | 6.000000 | 512.329200 |

titanic.groupby("sex")[["survived"]].mean()

|  | **survived** |
| --- | --- |
| **sex** |  |
| **female** | 0.742038 |
| **male** | 0.188908 |

titanic.groupby("sex")["survived"].mean()

sex

female 0.742038

male 0.188908

Name: survived, dtype: float64

titanic.groupby(["sex", "class"])["survived"].aggregate("mean").unstack()

| **class** | **First** | **Second** | **Third** |
| --- | --- | --- | --- |
| **sex** |  |  |  |
| **female** | 0.968085 | 0.921053 | 0.500000 |
| **male** | 0.368852 | 0.157407 | 0.135447 |

数据透视表

titanic.pivot\_table("survived", index="sex", columns="class")

| **class** | **First** | **Second** | **Third** |
| --- | --- | --- | --- |
| **sex** |  |  |  |
| **female** | 0.968085 | 0.921053 | 0.500000 |
| **male** | 0.368852 | 0.157407 | 0.135447 |

titanic.pivot\_table("survived", index="sex", columns="class", aggfunc="mean", margins=**True**)

| **class** | **First** | **Second** | **Third** | **All** |
| --- | --- | --- | --- | --- |
| **sex** |  |  |  |  |
| **female** | 0.968085 | 0.921053 | 0.500000 | 0.742038 |
| **male** | 0.368852 | 0.157407 | 0.135447 | 0.188908 |
| **All** | 0.629630 | 0.472826 | 0.242363 | 0.383838 |

titanic.pivot\_table(index="sex", columns="class", aggfunc={"survived": "sum", "fare": "mean"})

|  | **fare** | | | **survived** | | |
| --- | --- | --- | --- | --- | --- | --- |
| **class** | **First** | **Second** | **Third** | **First** | **Second** | **Third** |
| **sex** |  |  |  |  |  |  |
| **female** | 106.125798 | 21.970121 | 16.118810 | 91 | 70 | 72 |
| **male** | 67.226127 | 19.741782 | 12.661633 | 45 | 17 | 47 |

## 12.7.其他

**（1） 多级索引：用于多维数据**

base\_data = np.array([[1771, 11115 ],

[2154, 30320],

[2141, 14070],

[2424, 32680],

[1077, 7806],

[1303, 24222],

[798, 4789],

[981, 13468]])

data = pd.DataFrame(base\_data, index=[["BeiJing","BeiJing","ShangHai","ShangHai","ShenZhen","ShenZhen","HangZhou","HangZhou"], [2008, 2018]\*4], columns=["population", "GDP"])

data

|  |  | **population** | **GDP** |
| --- | --- | --- | --- |
| **BeiJing** | **2008** | 1771 | 11115 |
| **2018** | 2154 | 30320 |
| **ShangHai** | **2008** | 2141 | 14070 |
| **2018** | 2424 | 32680 |
| **ShenZhen** | **2008** | 1077 | 7806 |
| **2018** | 1303 | 24222 |
| **HangZhou** | **2008** | 798 | 4789 |
| **2018** | 981 | 13468 |

data.index.names = ["city", "year"]

data

|  |  | **population** | **GDP** |
| --- | --- | --- | --- |
| **city** | **year** |  |  |
| **BeiJing** | **2008** | 1771 | 11115 |
| **2018** | 2154 | 30320 |
| **ShangHai** | **2008** | 2141 | 14070 |
| **2018** | 2424 | 32680 |
| **ShenZhen** | **2008** | 1077 | 7806 |
| **2018** | 1303 | 24222 |
| **HangZhou** | **2008** | 798 | 4789 |
| **2018** | 981 | 13468 |

data["GDP"]

city year

BeiJing 2008 11115

2018 30320

ShangHai 2008 14070

2018 32680

ShenZhen 2008 7806

2018 24222

HangZhou 2008 4789

2018 13468

Name: GDP, dtype: int32

data.loc["ShangHai", "GDP"]

year

2008 14070

2018 32680

Name: GDP, dtype: int32

data.loc["ShangHai", 2018]["GDP"]

32680

**（2） 高性能的Pandas：eval（）**

df1, df2, df3, df4 = (pd.DataFrame(np.random.random((10000,100))) **for** i **in** range(4))

%**timeit** (df1+df2)/(df3+df4)

17.6 ms ± 120 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)

减少了复合代数式计算中间过程的内存分配

%**timeit** pd.eval("(df1+df2)/(df3+df4)")

10.5 ms ± 153 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)

np.allclose((df1+df2)/(df3+df4), pd.eval("(df1+df2)/(df3+df4)"))

True

**（3） 高性能的Pandas：query（）**

df = pd.DataFrame(np.random.random((1000, 3)), columns=list("ABC"))

df.head()

|  | **A** | **B** | **C** |
| --- | --- | --- | --- |
| **0** | 0.418071 | 0.381836 | 0.500556 |
| **1** | 0.059432 | 0.749066 | 0.302429 |
| **2** | 0.489147 | 0.739153 | 0.777161 |
| **3** | 0.175441 | 0.016556 | 0.348979 |
| **4** | 0.766534 | 0.559252 | 0.310635 |

df.eval("D=(A+B)/(C-1)", inplace=**True**)

df.head()

|  | **A** | **B** | **C** | **D** |
| --- | --- | --- | --- | --- |
| **0** | 0.418071 | 0.381836 | 0.500556 | -1.601593 |
| **1** | 0.059432 | 0.749066 | 0.302429 | -1.159019 |
| **2** | 0.489147 | 0.739153 | 0.777161 | -5.512052 |
| **3** | 0.175441 | 0.016556 | 0.348979 | -0.294917 |
| **4** | 0.766534 | 0.559252 | 0.310635 | -1.923199 |

%**timeit** df[(df.A < 0.5) & (df.B > 0.5)]

1.11 ms ± 9.38 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)

%**timeit** df.query("(A < 0.5)&(B > 0.5)")

2.55 ms ± 199 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)

df.query("(A < 0.5)&(B > 0.5)").head()

|  | **A** | **B** | **C** | **D** |
| --- | --- | --- | --- | --- |
| **1** | 0.059432 | 0.749066 | 0.302429 | -1.159019 |
| **2** | 0.489147 | 0.739153 | 0.777161 | -5.512052 |
| **7** | 0.073950 | 0.730144 | 0.646190 | -2.272672 |
| **10** | 0.393200 | 0.610467 | 0.697096 | -3.313485 |
| **11** | 0.065734 | 0.764699 | 0.179380 | -1.011958 |

np.allclose(df[(df.A < 0.5) & (df.B > 0.5)], df.query("(A < 0.5)&(B > 0.5)"))

True

**（4）eval（）和query（）的使用时机**

小数组时，普通方法反而更快

df.values.nbytes

32000

df1.values.nbytes

8000000

## 12.8.作业练习

**创建DataFrame数组并进行相应操作：**

1、创建一个30\*6的DataFrame数组，元素由70~100之间均匀分布的随机整数构成，行标签按030201（初三.二班1号）~030230格式顺序排列，列标签分别为语文、数学、英语、物理、化学、计算机。

2、输出其纯数据、行标签、列标签、形状、大小和数据类型

3、获取全班数学成绩、获取学号为030205的同学的所有成绩；

4、增加总成绩的新列，并建立按总成绩降序排列的副本（注意是获得副本，不是获得视图），切片获得前十名学生的全部成绩；

5、创建一个DataFrame对象（记为B），行标签与上文DataFrame对象（记为A）一致，列标签为性别，数据为30个学生的随机性别，将A和B进行水平合并，获得新的DataFrame对象（记为C）；

6、输出数据C的info和describe信息，尝试自定义my\_describe，输出自己感兴趣的统计信息；

7、按性别进行分组，对比男生女生所有科目及总成绩的平均值。

**DataFrame数组操作**

8.下载titanic数据集，执行下列操作：

**import** **seaborn** **as** **sns**

*# titanic = sns.load\_dataset("titanic")*

titanic = pd.read\_csv("data/titanic.csv") *# 读取本地的csv文件*

titanic.head()

|  | **survived** | **pclass** | **sex** | **age** | **sibsp** | **parch** | **fare** | **embarked** | **class** | **who** | **adult\_male** | **deck** | **embark\_town** | **alive** | **alone** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0 | 3 | male | 22.0 | 1 | 0 | 7.2500 | S | Third | man | True | NaN | Southampton | no | False |
| **1** | 1 | 1 | female | 38.0 | 1 | 0 | 71.2833 | C | First | woman | False | C | Cherbourg | yes | False |
| **2** | 1 | 3 | female | 26.0 | 0 | 0 | 7.9250 | S | Third | woman | False | NaN | Southampton | yes | True |
| **3** | 1 | 1 | female | 35.0 | 1 | 0 | 53.1000 | S | First | woman | False | C | Southampton | yes | False |
| **4** | 0 | 3 | male | 35.0 | 0 | 0 | 8.0500 | S | Third | man | True | NaN | Southampton | no | True |

（1）获得一个删除了无年龄数据的所有行的副本；

（2）创建一个名为Age的Series对象，其数据来源于对数据集中的年龄按下列规则进行映射（参照行星数据集案例中decade的处理办法）：

If <10 : “0s”

elif <20 : “10s”

…

elif <60 : “50s”

…

（3）通过sex和Age对titanic数据集进行分组，获得不同性别、不同年龄段乘客的幸存比例，请分别使用groupby和pivot\_table（如果直接用Age不行的话，换个思路）两种方法。

答案：

1.

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

classes = ["03020" + str(i) **for** i **in** range(1, 10)] + ["0302" + str(i) **for** i **in** range(10, 31)]

classes

['030201',

'030202',

...

'030230']

objectes = ["语文", "数学", "英语", "物理", "化学", "计算机"]

df = pd.DataFrame(np.random.randint(70, 100, (30, 6)), classes, objectes)

df

|  | **语文** | **数学** | **英语** | **物理** | **化学** | **计算机** |
| --- | --- | --- | --- | --- | --- | --- |
| **030201** | 91 | 74 | 85 | 92 | 84 | 78 |
| **030202** | 71 | 72 | 75 | 93 | 97 | 99 |
| **...** | ... | ... | ... | ... | ... | ... |
| **030230** | 74 | 86 | 90 | 85 | 80 | 82 |

2.

print("row\_index : **{}**, **\n**col\_index : **{}**, **\n**shape : **{}**, **\n**size : **{}**, **\n**dtype : **{}**".format(df.index, df.columns,

df.shape, df.size,

df.dtypes))

row\_index : Index(['030201', '030202', '030203', '030204', '030205', '030206', '030207',

'030208', '030209', '030210', '030211', '030212', '030213', '030214',

'030215', '030216', '030217', '030218', '030219', '030220', '030221',

'030222', '030223', '030224', '030225', '030226', '030227', '030228',

'030229', '030230'],

dtype='object'),

col\_index : Index(['语文', '数学', '英语', '物理', '化学', '计算机'], dtype='object'),

shape : (30, 6),

size : 180,

dtype : 语文 int32

数学 int32

英语 int32

物理 int32

化学 int32

计算机 int32

dtype: object

3.

print("math\_score = **\n{}**, **\n**030205\_score = **\n{}**".format(df["数学"], df.loc["030205"]))

math\_score =

030201 74

030202 72

...

030230 86

Name: 数学, dtype: int32,

030205\_score =

语文 76

数学 96

英语 77

物理 87

化学 82

计算机 93

Name: 030205, dtype: int32

4.

df\_cp = df.copy()

df\_cp['总成绩'] = df\_cp.apply(**lambda** x : x.sum(), axis=1)

df\_cp

|  | **语文** | **数学** | **英语** | **物理** | **化学** | **计算机** | **总成绩** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **030201** | 91 | 74 | 85 | 92 | 84 | 78 | 504 |
| **030202** | 71 | 72 | 75 | 93 | 97 | 99 | 507 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **030230** | 74 | 86 | 90 | 85 | 80 | 82 | 497 |

*# 按总成绩降序排列的副本，切片获得前十名学生的全部成绩*

df2 = df\_cp.sort\_values(by=['总成绩'], ascending=**False**)

*# print(df2)*

print(df2.iloc[0: 10, :])

语文 数学 英语 物理 化学 计算机 总成绩

030208 80 93 92 85 98 98 546

030203 95 95 81 80 97 90 538

030216 72 97 92 83 95 97 536

030220 89 96 80 83 93 90 531

030222 81 96 97 81 73 98 526

030226 89 99 90 85 86 76 525

030211 88 94 95 73 83 91 524

030213 98 95 76 85 94 75 523

030207 91 86 81 98 87 76 519

030210 76 99 82 88 95 78 518

5.

gender = ['男' **if** np.random.random() < 0.5 **else** '女' **for** i **in** range(30)]

gender

['男',

'男',

...

'女']

df2 = pd.DataFrame(gender, index=classes, columns=["性别"])

print(df2)

性别

030201 男

030202 男

...

030230 女

c = pd.concat([df\_cp, df2], axis=1)

print(c)

语文 数学 英语 物理 化学 计算机 总成绩 性别

030201 91 74 85 92 84 78 504 男

030202 71 72 75 93 97 99 507 男

...

030230 74 86 90 85 80 82 497 女

6.

print("C\_info=**{}**, C\_describe=**{}**".format(c.info(), c.describe()))

<class 'pandas.core.frame.DataFrame'>

Index: 30 entries, 030201 to 030230

Data columns (total 8 columns):

语文 30 non-null int32

数学 30 non-null int32

...

总成绩

count 30.000000

...

max 546.000000

**def** my\_describe(x):

**return** pd.Series([x.count()], index=["Count"])

c.apply(my\_describe)

|  | **语文** | **数学** | **英语** | **物理** | **化学** | **计算机** | **总成绩** | **性别** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Count** | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 |

7.

c.groupby('性别').sum()

|  | **语文** | **数学** | **英语** | **物理** | **化学** | **计算机** | **总成绩** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **性别** |  |  |  |  |  |  |  |
| **女** | 1189 | 1223 | 1155 | 1188 | 1220 | 1185 | 7160 |
| **男** | 1305 | 1353 | 1376 | 1356 | 1403 | 1345 | 8138 |

c.groupby('性别').mean()

|  | **语文** | **数学** | **英语** | **物理** | **化学** | **计算机** | **总成绩** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **性别** |  |  |  |  |  |  |  |
| **女** | 84.928571 | 87.357143 | 82.5 | 84.857143 | 87.142857 | 84.642857 | 511.428571 |
| **男** | 81.562500 | 84.562500 | 86.0 | 84.750000 | 87.687500 | 84.062500 | 508.625000 |

8.（1）

data\_no\_age = titanic.drop(columns='age').copy()

data\_no\_age.head(10)

|  | **survived** | **pclass** | **sex** | **sibsp** | **parch** | **fare** | **embarked** | **class** | **who** | **adult\_male** | **deck** | **embark\_town** | **alive** | **alone** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0 | 3 | male | 1 | 0 | 7.2500 | S | Third | man | True | NaN | Southampton | no | False |
| **1** | 1 | 1 | female | 1 | 0 | 71.2833 | C | First | woman | False | C | Cherbourg | yes | False |
| **2** | 1 | 3 | female | 0 | 0 | 7.9250 | S | Third | woman | False | NaN | Southampton | yes | True |
| **3** | 1 | 1 | female | 1 | 0 | 53.1000 | S | First | woman | False | C | Southampton | yes | False |
| **4** | 0 | 3 | male | 0 | 0 | 8.0500 | S | Third | man | True | NaN | Southampton | no | True |
|  | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **9** | 1 | 2 | female | 1 | 0 | 30.0708 | C | Second | child | False | NaN | Cherbourg | yes | False |

（2）

Age = titanic[titanic['age'].notnull()]

Age = 10 \* (Age['age'] // 10)

Age = Age.astype(int)

Age = Age.astype(str) + 's'

print(Age)

0 20s

1 30s

...

890 30s

Name: age, Length: 714, dtype: object

（3）

t = titanic[titanic.age.notnull()]

t.groupby(["sex", Age])["survived"].mean().unstack()

| **age** | **0s** | **10s** | **20s** | **30s** | **40s** | **50s** | **60s** | **70s** | **80s** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **sex** |  |  |  |  |  |  |  |  |  |
| **female** | 0.633333 | 0.755556 | 0.722222 | 0.833333 | 0.687500 | 0.888889 | 1.000000 | NaN | NaN |
| **male** | 0.593750 | 0.122807 | 0.168919 | 0.214953 | 0.210526 | 0.133333 | 0.133333 | 0.0 | 1.0 |

t.age = Age.copy()

t.pivot\_table("survived", index="sex", columns="age")

| **age** | **0s** | **10s** | **20s** | **30s** | **40s** | **50s** | **60s** | **70s** | **80s** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **sex** |  |  |  |  |  |  |  |  |  |
| **female** | 0.633333 | 0.755556 | 0.722222 | 0.833333 | 0.687500 | 0.888889 | 1.000000 | NaN | NaN |
| **male** | 0.593750 | 0.122807 | 0.168919 | 0.214953 | 0.210526 | 0.133333 | 0.133333 | 0.0 | 1.0 |