电子科技大学
University of Electronic Science and Technology of China

Lecture 9 数据处理技术

- 合并数据集
- 数据转换

教学目标

- 了解数据处理技术的概念和特点
- 了解其基本原理、主要功能特点
- 让学生对数据处理技术有一个初步理解

数据处理技术

- 存放在文件或者数据库中的原始数据并不总能满足数据分析应用的要求。
- 通常，原始数据中存在不符合规范的数据格式，或者存在数据缺失的情况。
- 在这些情况下，必须对原始数据进行包括加载、清理、转换和重塑等处理。

9.1 合并数据集

- 数据存储时，往往会按照数据的物理含义，将数据分别存储在不同的表中，以便于管理和操作。

- 在数据分析和数据建模时，往往需要将不同的数据表进行关联或合并，从而找出不同数据项之间的内在关联。

电子科技大学
University of Electronic Science and Technology of China

Pandas库

Pandas是python的一个数据分析包，是基于NumPy 的一种数据分析工具，它包含大量库函数和一些标准的数据模型，提供了高效地操作大型数据集、快速便捷地处理数据的函数和方法。

Pandas网页： https://pandas.pydata.org/
Pandas源代码：https://github.com/pandas-dev/pandas

Pandas使用的数据结构

Series：一维数组，能保存不同数据类型如字符串、boolean值、数值等

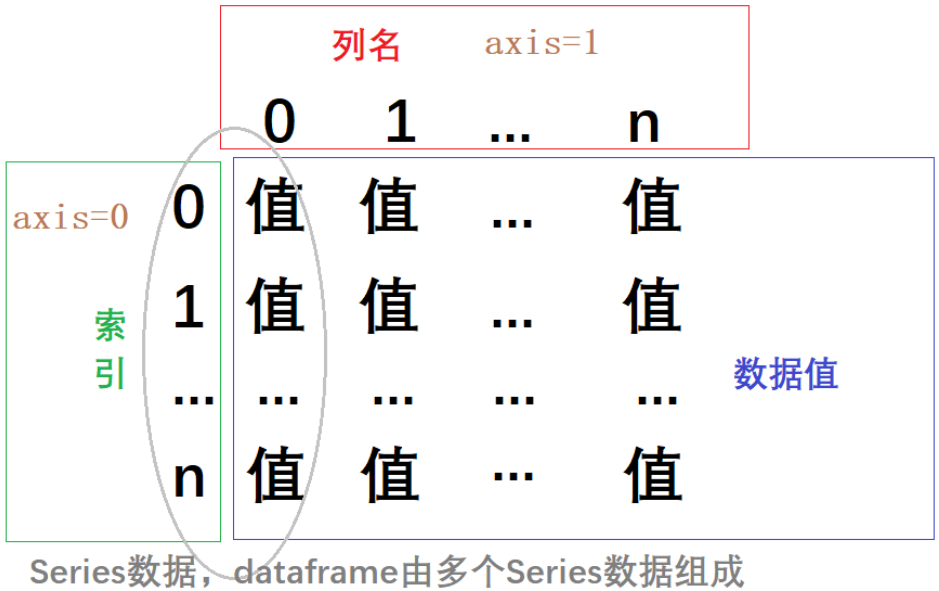
DataFrame：二维的表格型数据结构。可以将DataFrame理解为Series的容器

Panel ：三维的数组，可以理解为DataFrame的容器

PanelND：拥有factory集合，可以创建N维命名容器的模块

DataFrame数据结构

DataFrame是由一组数据和一对索引（行索引和列索引）组成的表格型数据结构，常用于表达二维数据，同时也可以表达多维数据，行列索引有自动索引和自定义索引。



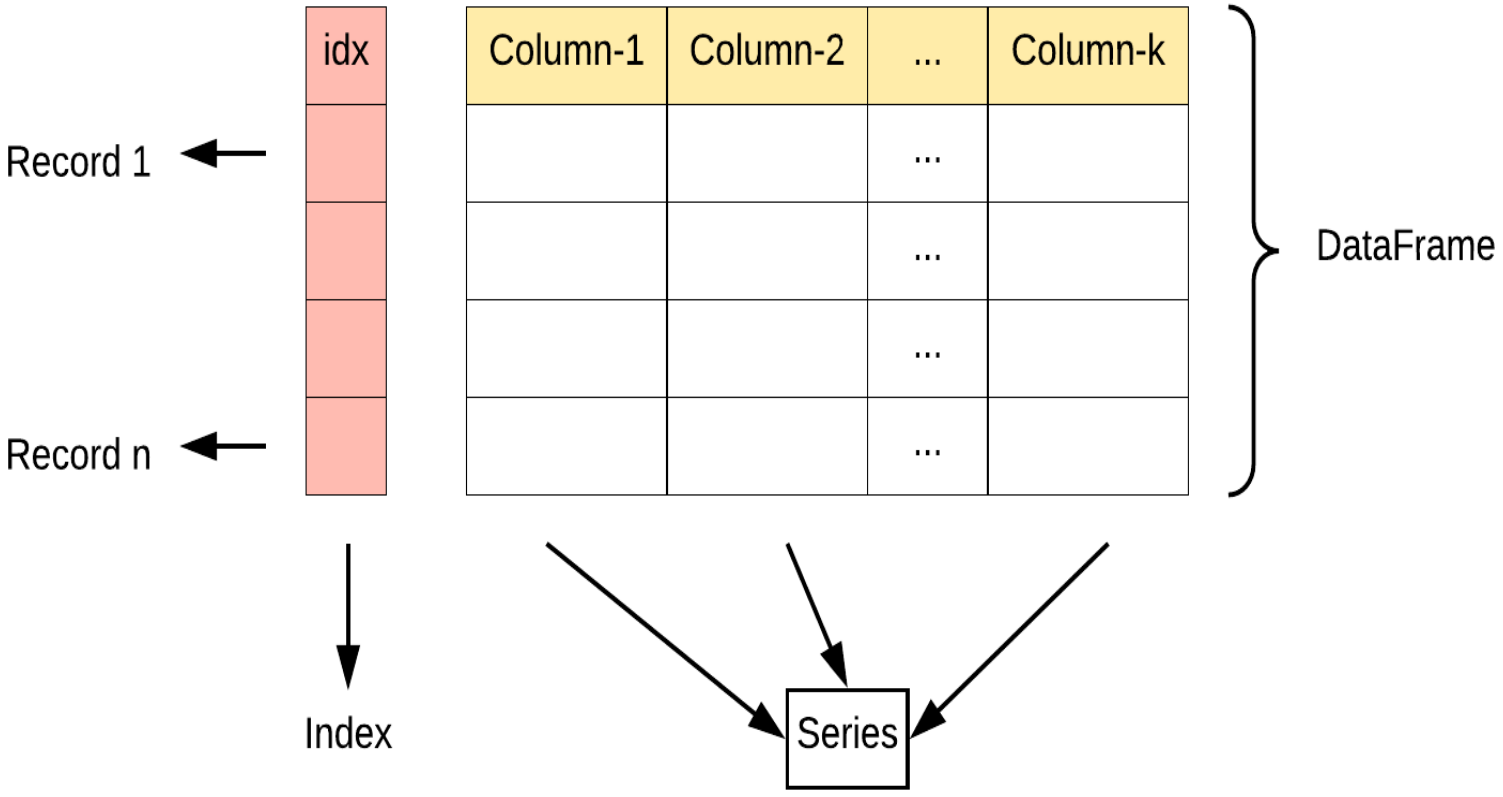
Series数据，dataframe由多个Series数据组成

如果series索引跟datafarame索引相同，那么series的name值会被合理设置为dataframe的列名

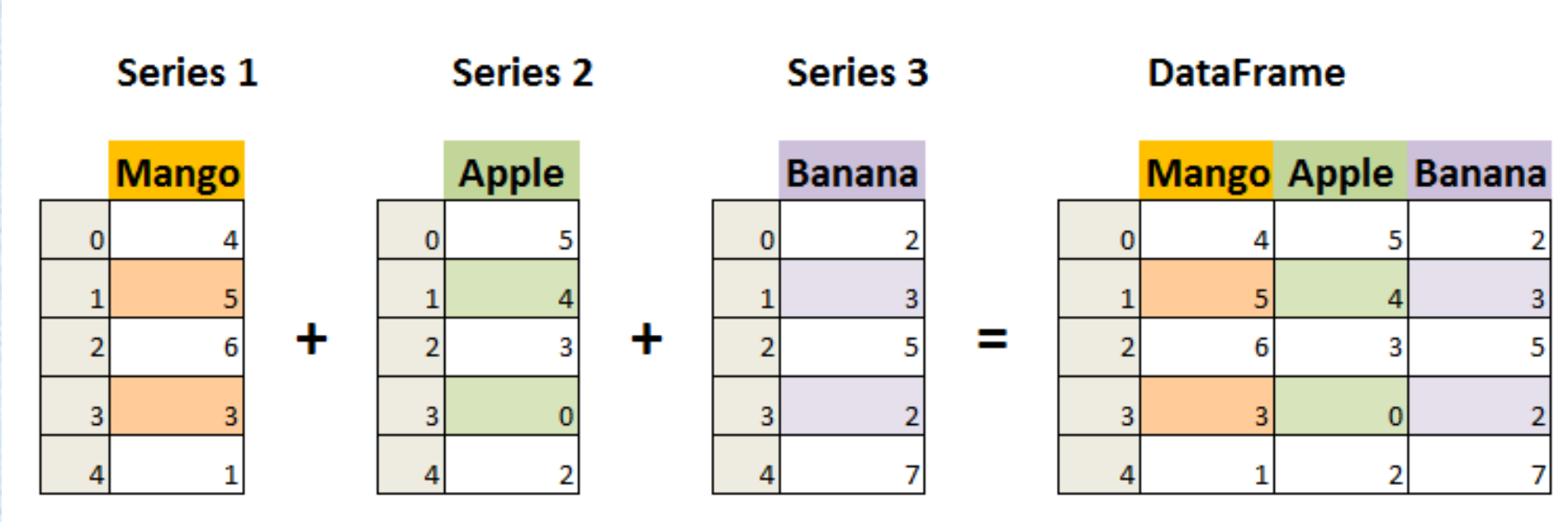
df=pd.DataFrame(数据序列 [, columns=序列 , index=序列])

定义数据值　　定义列名　　定义索引

DataFrame数据结构



DataFrame数据结构



DataFrame数据结构

DataFrame 构造方法如下：

pandas.DataFrame(data, index, columns, dtype, copy)

参数：

data：一组数据（ndarray,series,map,lists,dict 等类型）

index：索引值，或者可以称为行标签

columns：列标签，默认为 RangeIndex（0, 1, 2, …, n）

dtype：数据类型

copy：拷贝数据，默认为 False

DataFrame数据结构

实例 - 使用列表创建

```
import pandas as pd

data = [[ 'Google' ,10], [ 'Runoob' ,12], [ 'Wiki' ,13]]

df = pd.DataFrame(data, columns=[ 'Site' , 'Age' ], dtype=float)

print(df)
```

输出结果如下：

```
     Site   Age
0  Google  10.0
1  Runoob  12.0
2    Wiki  13.0
```

DataFrame数据结构

实例 - 使用ndarrays创建

```
import pandas as pd

data = {'Site':['Google', 'Runoob', 'Wiki'], 'Age':[10, 12, 13]}

df = pd.DataFrame(data)

print (df)
```

输出结果如下：

```
     Site   Age
0  Google  10.0
1  Runoob  12.0
2    Wiki  13.0
```

9.1.1 索引上的合并

DataFrame的merge( )函数

pd.merge (left, right, how='inner', on=None, left_on=None, right_on=None, left_index=False, right_index=False, sort=True, suffixes=('_x', '_y'), copy=True, indicator=False, validate=None)

left: 拼接的左侧 DataFrame 对象

right: 拼接的右侧 DataFrame 对象

on: 要加入的列或索引级别名称。 必须在左侧和右侧 DataFrame 对象中找到。 如果未传递且 left_index 和 right_index 为 False，则 DataFrame 中的列的交集将被推断为连接键。

left_on: 左侧 DataFrame 中的列或索引级别用作键。 可以是列名，索引级名称，也可以是长度等于 DataFrame 长度的数组。一般用于当数据值相同但是列名不同的情况，需和right_on一起使用。

right_on: 左侧 DataFrame 中的列或索引级别用作键。 可以是列名，索引级名称，也可以是长度等于 DataFrame 长度的数组。

9.1.1 索引上的合并

pd.merge (left, right, how='inner', on=None, left_on=None, right_on=None, left_index=False, right_index=False, sort=True, suffixes=('_x', '_y'), copy=True, indicator=False, validate=None)

left_index: 如果为 True，则使用左侧 DataFrame 的索引作为其连接键。

right_index: 与 left_index 功能相似。

how: ‘left’, ‘right’, ‘outer’, ‘inner’中四选一， 默认 inner。inner 是取交集，outer 取并集，left是左连接，right是右连接。

sort: 按字典顺序通过连接键对结果 DataFrame 进行排序。 默认为 True，设置为 False 将在很多情况下显着提高性能。

suffixes: 用于重叠列的字符串后缀元组。 默认为（‘x’，’ y’）。

copy: 始终从传递的 DataFrame 对象复制数据（默认为 True），即使不需要重建索引也是如此。

indicator: 将一列添加到名为_merge 的输出 DataFrame，其中包含有关每行源的信息。

9.1.1 索引上的合并

- 在DataFrame中，两个或多个表的连接键有时会位于其索引中。在这种情况下，需要传入left_index = True 或 right_index = True（两个都传）以说明索引应该被用作连接键。

索引上的合并

left1 = pd.DataFrame({'key': ['a', 'b', 'a', 'a', 'b', 'c'], 'value':range(6)})

right1 = pd.DataFrame({'group_val':[3.5, 7]}, index=['a', 'b'])

	key	value
0	a	0
1	b	1
2	a	2
3	a	3
4	b	4
5	c	5

	group_val
a	3.5
b	7.0

索引上的合并

- *pd.merge (left1, right1, left_on='key', right_index=True)*

	key	value
0	a	0
1	b	1
2	a	2
3	a	3
4	b	4
5	c	5

∩

	group_val
a	3.5
b	7.0

→

	key	value	group_val
0	a	0	3.5
2	a	2	3.5
3	a	3	3.5
1	b	1	7.0
4	b	4	7.0

索引上的合并

- 默认的merge方法是求取两张关联表的交集部分(共有部分)。
- 如果需要求取关联表的并集部分(或有部分)，可通过外连接的方式得到它们的并集 。

索引上的合并

pd.merge (left1, right1, left_on='key',
right_index=True, how='outer')

	key	value
0	a	0
1	b	1
2	a	2
3	a	3
4	b	4
5	c	5

∪

	group_val
a	3.5
b	7.0

	key	value	group_val
0	a	0	3.5
2	a	2	3.5
3	a	3	3.5
1	b	1	7.0
4	b	4	7.0
5	c	5	NaN

索引上的合并

一种比较复杂的情况，即某个表中的index是复合键进行索引的：

```
lefth = pd.DataFrame({'key1': ['Ohio', 'Ohio', 'Ohio',
    'Nevada', 'Nevada'], 'key2': [2000, 2001, 2002,
    2001, 2002], 'data': np.arange(5.)})

righth = pd.DataFrame(np.arange(12).reshape((6, 2)),
    index=[['Nevada', 'Nevada', 'Ohio', 'Ohio', 'Ohio',
    'Ohio'], [2001, 2000, 2000, 2000, 2001, 2002]],
    columns=['event1', 'event2'])
```

索引上的合并

- 两个表分别如下

lefth

	key1	key2	data
0	Ohio	2000	0.0
1	Ohio	2001	1.0
2	Ohio	2002	2.0
3	Nevada	2001	3.0
4	Nevada	2002	4.0

righth

		event1	event2
Nevada	**2001**	0	1
	2000	2	3
Ohio	**2000**	4	5
	2000	6	7
	2001	8	9
	2002	10	11

索引上的合并

righth表中的index是由key1和key2两个键的复合键组成的，必须以列表的形式指明用作合并键的多个列(默认是交集)：

pd.merge(lefth, righth, left_on=['key1', 'key2'], right_index=True)

	key1	key2	data	event1	event2
0	Ohio	2000	0.0	4	5
0	Ohio	2000	0.0	6	7
1	Ohio	2001	1.0	8	9
2	Ohio	2002	2.0	10	11
3	Nevada	2001	3.0	0	1

索引上的合并

pd.merge(lefth, righth, left_on=['key1', 'key2'], right_index=True, how='outer') #明确是合集

	key1	key2	data	event1	event2
0	Ohio	2000	0.0	4.0	5.0
0	Ohio	2000	0.0	6.0	7.0
1	Ohio	2001	1.0	8.0	9.0
2	Ohio	2002	2.0	10.0	11.0
3	Nevada	2001	3.0	0.0	1.0
4	Nevada	2002	4.0	NaN	NaN
4	Nevada	2000	NaN	2.0	3.0

索引上的合并

可以采用合并双方的索引，实现多个表之间的关联

```
left2 = pd.DataFrame([[1., 2.], [3., 4.],
    [5., 6.]], index=[‘a’, ‘c’, ‘e’],
    columns=[‘Ohio’, ‘Nevada’])

right2 = pd.DataFrame([[7., 8.], [9., 10.],
    [11., 12.], [13, 14]], index=['b', 'c',
    'd', 'e'], columns=['Missouri',
    'Alabama'])
```

索引上的合并

left2

	Ohio	Nevada
a	1.0	2.0
c	3.0	4.0
e	5.0	6.0

right2

	Missouri	Alabama
b	7.0	8.0
c	9.0	10.0
d	11.0	12.0
e	13.0	14.0

索引上的合并

pd.merge (left2, right2, how='outer', left_index=True, right_index=True)

	Ohio	Nevada
a	1.0	2.0
c	3.0	4.0
e	5.0	6.0

merge

	Missouri	Alabama
b	7.0	8.0
c	9.0	10.0
d	11.0	12.0
e	13.0	14.0

	Ohio	Nevada	Missouri	Alabama
a	1.0	2.0	NaN	NaN
b	NaN	NaN	7.0	8.0
c	3.0	4.0	9.0	10.0
d	NaN	NaN	11.0	12.0
e	5.0	6.0	13.0	14.0

索引上的合并

DataFrame的join( )函数

merge是基于columns来连接DataFrame，join函数是基于index连接DataFrame。

DataFrame.join (other, on=None, how='left', lsuffix='', rsuffix='', sort=False)

other: DataFrame，或者带有名字的Series，或者DataFrame的list。如果传递的是Series，那么其name属性应当是一个集合，并且该集合将会作为结果DataFrame的列名

on: 列名称，或者列名称的list/tuple，或者类似形状的数组连接的列，默认使用索引连接

how: {'left', 'right', 'outer', 'inner'}, default:'left' 连接的方式，默认为左连接

lsuffix: string 左DataFrame中重复列的后缀

rsuffix: string 右DataFrame中重复列的后缀

sort: boolean, default=False 按照字典顺序对结果在连接键上排序。如果为False，连接键的顺序取决于连接类型（关键字）。

索引上的合并

left2

	Ohio	Nevada
a	1.0	2.0
c	3.0	4.0
e	5.0	6.0

right2

	Missouri	Alabama
b	7.0	8.0
c	9.0	10.0
d	11.0	12.0
e	13.0	14.0

索引上的合并

join函数还可用于合并多个带有相同或相似索引的DataFrame对象，而不管它们之间有没有重叠的列。

left2.join(right2, how='outer')

	Ohio	Nevada
a	1.0	2.0
c	3.0	4.0
e	5.0	6.0

join

	Missouri	Alabama
b	7.0	8.0
c	9.0	10.0
d	11.0	12.0
e	13.0	14.0

	Ohio	Nevada	Missouri	Alabama
a	1.0	2.0	NaN	NaN
b	NaN	NaN	7.0	8.0
c	3.0	4.0	9.0	10.0
d	NaN	NaN	11.0	12.0
e	5.0	6.0	13.0	14.0

索引上的合并

由于一些历史原因（早期版本pandas规定的），DataFrame的join函数默认是通过连接键做左连接，对多个表进行关联的

left1.join(right1, on='key')

	key	value
0	a	0
1	b	1
2	a	2
3	a	3
4	b	4
5	c	5

join

	group_val
a	3.5
b	7.0

	key	value	group_val
0	a	0	3.5
1	b	1	7.0
2	a	2	3.5
3	a	3	3.5
4	b	4	7.0
5	c	5	NaN

轴向连接

Concat（ ）连接函数

有行连接和列连接，默认是行连接，连接方法默认是外连接（并集），连接的对象是pandas数据类型。

pd.concat (objs, axis=0, join='outer', join_axes=None, ignore_index=False, keys=None, levels=None, names=None, verify_integrity=False, copy=True)

objs: series，dataframe或者是panel构成的序列lsit。

axis： 需要合并连接的轴，0是行（默认），1是列。

join： 连接的方式 inner （默认）， 或者outer。

ignore_index：boolean，default False。如果为True，请不要使用并置轴上的索引值。结果轴将被标记为0，...，n-1。

join_axes：Index对象列表。用于其他n-1轴的特定索引，而不是执行内部/外部设置逻辑。

9.1.2 轴向连接

数据合并运算也被称作连接（concatenation）、绑定（binding）或堆叠（stacking）。NumPy有一个用于合并原始NumPy数组的concatenation函数

arr = np.arange(12).reshape((3, 4))

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

轴向连接

np.concatenate([arr, arr]) *#默认 axis=0*

```
array ( [ [  0,   1,   2,   3  ],
          [  4,   5,   6,   7  ],
          [  8,   9,   10,  11 ],
          [  0,   1,   2,   3  ],
          [  4,   5,   6,   7  ],
          [  8,   9,   10,  11 ] ] )
```

轴向连接

np.concatenate([arr, arr], axis=1)

```
array([[ 0,  1,  2,  3,  0,  1,  2,  3],
       [ 4,  5,  6,  7,  4,  5,  6,  7],
       [ 8,  9, 10, 11,  8,  9, 10, 11]])
```

9.1.3 合并重叠数据

当两个数据集的索引全部或部分重叠时，它们的数据组合问题就不能用简单的合并（merge）或连接（concatenation）运算来处理

```
a = pd.Series([np.nan, 2.5, np.nan, 3.5, 4.5,
    np.nan], index=['f', 'e', 'd', 'c', 'b', 'a'])
b = pd.Series(np.arange(len(a),
    dtype=np.float64), index=['f', 'e', 'd', 'c',
    'b', 'a'])
b[-1] = np.nan
```

合并重叠数据

```
f    NaN
e    2.5
d    NaN
c    3.5
b    4.5
a    NaN
dtype: float64
```

```
f    0.0
e    1.0
d    2.0
c    3.0
b    4.0
a    NaN
dtype: float64
```

下面实现完全重叠的两个数据集的合并，当第一个数据集非空时，取第一个数据集的值，否则取第二个数据集的值

```
In  [28]: np.where(pd.isnull(a), b, a)
Out[28]: array([0. , 2.5, 2. , 3.5, 4.5, nan])
```

9.2 数据转换

除了数据合并以外，数据处理工作还包括对数据进行转换。具体的工作包括对数据进行过滤、清理以及其它的转换工作

在数据转换工作中，最常见的是移除重复数据的工作。通常来说，数据集中总会出现重复的数据行。

9.2.1 移除重复数据

```
data = pd.DataFrame({'k1':['one'] * 3 + ['two'] * 4, 'k2':[1, 1, 2, 3, 3, 4, 4]})
```

	k1	k2
0	one	1
1	one	1
2	one	2
3	two	3
4	two	3
5	two	4
6	two	4

移除重复数据

上面的DataFrame中存在6个数据行，其中一部分是重复的。通常来说，我们可以通过duplicated方法返回一个布尔型Series，每行中的布尔值表示该行是否是重复的

data.duplicated()

```
0    False
1     True
2    False
3    False
4     True
5    False
6     True
dtype: bool
```

数据去重方法

第1、4、6行不是第一次出现的数据行，在后面的去重工作中可以考虑去除。如果想要直接去除数据中的重复行，可以考虑使用drop_duplicates方法，它用于返回一个移除了重复行的DataFrame。

	k1	k2
0	one	1
1	one	1
2	one	2
3	two	3
4	two	3
5	two	4
6	two	4

	k1	k2
0	one	1
2	one	2
3	two	3
5	two	4

数据去重方法

上面的结果显示，重复的数据行的全部列都已经被移除。在实际的数据处理案例中，可能只希望根据某一列来过滤重复项：

```
data['v1'] = range(7)
```

	k1	k2	v1
0	one	1	0
1	one	1	1
2	one	2	2
3	two	3	3
4	two	3	4
5	two	4	5
6	two	4	6

数据去重方法

data.drop_duplicates(['k1'])

	k1	k2	v1
0	one	1	0
3	two	3	3

上面的方法中，通过drop_duplicates([‘k1’])可以将k1中的重复值去掉。此外，duplicated和drop_duplicates还可以通过多列的联合取值来筛选数据，并且通过keep= ‘last’保留重复数据中的最后一个。

数据去重方法

data.drop_duplicates(['k1', 'k2'], keep='last')

	k1	k2	v1
1	one	1	1
2	one	2	2
4	two	3	4
6	two	4	6

9.2.3 数据替换方法

利用fillna方法填充缺失数据可以看做值替换的一种特殊情况。在通常的值替换时，往往采用replace方法，它提供了一种实现替换功能的简单、灵活的方式。来看下面的Series：

data = pd.Series([1., -999, 2., -999, -1000., 3.])

```
0       1.0
1    -999.0
2       2.0
3    -999.0
4   -1000.0
5       3.0
dtype: float64
```

假设-999这个值是一个表示缺失数据的标记值。要将其替换为Pandas能够理解的NA值，可以利用replace来产生一个新的Series：

data.replace(-999, np.nan)

```
0       1.0
1    -999.0
2       2.0
3    -999.0
4   -1000.0
5       3.0
dtype: float64
```

```
0       1.0
1       NaN
2       2.0
3       NaN
4   -1000.0
5       3.0
dtype: float64
```

当然，如果希望一次性替换多个值（例如-999和-1000替换为NaN），可以传入一个由待替换值组成的列表以及一个替换值：

data.replace([-999, -1000], np.nan)

```
0       1.0
1    -999.0
2       2.0
3    -999.0
4   -1000.0
5       3.0
dtype: float64
```

```
0    1.0
1    NaN
2    2.0
3    NaN
4    NaN
5    3.0
dtype: float64
```

如果希望对不同的值进行不同的替换（例如-999替换为NaN，-1000替换为0），则传入一个由替换关系组成的列表即可：

data.replace([-999, -1000], [np.nan, 0])

```
0       1.0
1    -999.0
2       2.0
3    -999.0
4   -1000.0
5       3.0
dtype: float64
```

```
0    1.0
1    NaN
2    2.0
3    NaN
4    0.0
5    3.0
dtype: float64
```

9.2.6 检测异常值

- 异常值（outlier）的过滤或变换运算在很大程度上其实就是数组运算。
- 我们首先来看一个含有正态分布数据的DataFrame：

```
np.random.seed(12345)
data = pd.DataFrame(np.random.randn(1000, 4))
data.describe()
```

	0	1	2	3
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	-0.067684	0.067924	0.025598	-0.002298
std	0.998035	0.992106	1.006835	0.996794
min	-3.428254	-3.548824	-3.184377	-3.745356
25%	-0.774890	-0.591841	-0.641675	-0.644144
50%	-0.116401	0.101143	0.002073	-0.013611
75%	0.616366	0.780282	0.680391	0.654328
max	3.366626	2.653656	3.260383	3.927528

下面要选出全部含有“超过3或-3的值”的行，可以利用布尔型DataFrame及any方法：

```
data[(np.abs(data) > 3).any(1)]
```

	0	1	2	3
5	-0.539741	0.476985	3.248944	-1.021228
97	-0.774363	0.552936	0.106061	3.927528
102	-0.655054	-0.565230	3.176873	0.959533
305	-2.315555	0.457246	-0.025907	-3.399312
324	0.050188	1.951312	3.260383	0.963301
400	0.146326	0.508391	-0.196713	-3.745356
499	-0.293333	-0.242459	-3.056990	1.918403
523	-3.428254	-0.296336	-0.439938	-0.867165
586	0.275144	1.179227	-3.184377	1.369891
808	-0.362528	-3.548824	1.553205	-2.186301
900	3.366626	-2.372214	0.851010	1.332846

根据这些条件，我们可以轻松地对值进行设置。下面的代码将值限制在区间-3到3以内：

```
data[np.abs(data) > 3] = np.sign(data) * 3
data.describe()
```

	0	1	2	3
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	-0.067623	0.068473	0.025153	-0.002081
std	0.995485	0.990253	1.003977	0.989736
min	-3.000000	-3.000000	-3.000000	-3.000000
25%	-0.774890	-0.591841	-0.641675	-0.644144
50%	-0.116401	0.101143	0.002073	-0.013611
75%	0.616366	0.780282	0.680391	0.654328
max	3.000000	2.653656	3.000000	3.000000

9.2.7 排列和随机采样

利用numpy.random.permutation函数可以实现对Series或DataFrame的排列工作。通过需要排列的轴的长度调用permutation，可产生一个表示新顺序的整数数组：

```
df = pd.DataFrame(np.arange(5 * 4).reshape(5, 4))
```

In: df

Out:

	0	1	2	3
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15
4	16	17	18	19

```
sampler = np.random.permutation(5)
```

sampler 显示 array([1, 0, 2, 3, 4])

排列和随机采样

我们可以采用take函数操作来完成原数组行调换

df.take(sampler)

	0	1	2	3
1	4	5	6	7
0	0	1	2	3
2	8	9	10	11
3	12	13	14	15
4	16	17	18	19

排列和随机采样

如果不想用替换方式选取随机子集，则可以使用permutation：从permutation返回的数组中切下前k个元素，k为期望的子集大小

df.take(np.random.permutation(len(df))[:3])

	0	1	2	3
1	4	5	6	7
3	12	13	14	15
4	16	17	18	19