Python进阶知识

python numpy模块数组

python numpy模块数组（array）

在机器学习或数据分析中，我们经常会用到一维向量、二维矩阵等数学计算。Python作为一种编程语言，并没有内置数学上常用的一维向量、二维矩阵等对象，但是可以使用numpy模块的数组（array）。数组（array）使用连续的存储空间存储一组相同类型的值，其中的每一个元素即为值本身，操作和list基本相似，同样支持基本符号运算和索引、切片，但比list具有更快的读写速度和更少的占用空间，如果存储的数据类型相同，使用数组是更好的选择。

数组的创建

数组创建首先需要导入numpy，然后通过np.arange()函数进行创建。创建数组后，可以通过“type()函数”、“.ndim”、“.shape”分别查看数组的类型、维度和形状。

示例

参阅教材内容

数组的计算

我们基于数组可以进行计算，需要注意的是，数组不同于列表，数组的计算是基于每个元素的值，比如之前我们讲解列表时，如果针对列表乘以2，那么会将列表进行复制，而针对数组乘以2，则会使得其中的每个元素的值都变为原来的2倍。计算不仅包括加减乘除等四则运算，也包括使用np.sqrt()进行开平方计算、使用np.exp()进行指数计算、使用np.sum进行求和计算、使用np.mean进行求均值计算等。这种可按元素进行计算的函数，称为通用函数，类型为“numpy.ufunc”，简称“ufunc”。

示例

参阅教材内容

使用数组开展矩阵运算

二维数组体现为矩阵，我们完全可以通过生成二维数组、并运用数组计算的相关方法来完成矩阵的运算。具体包括不限于矩阵的转置、相乘、求逆矩阵、求特征值和特征向量等。

示例

参阅教材内容

创建ndarray对象

1. ndarray数据类型

- NumPy比原生Python支持更丰富的数据类型。这些数据类型大多以数字结尾，表示其在内存中占有的位数。同时，为了能够更容易确定一个nharray所需的存储空间，同一个ndarray中所有元素的类型必须是一致的。NumPy基本数据类型与其取值范围，如下表所示。

类型	描述
布尔型	bool，用一位存储的布尔类型（值为TRUE或FALSE）
整型	int32或int64，由所在平台决定其精度的整数，取值范围为-2^{31}至$2^{31}-1$和-2^{63}至$2^{63}-1$，同样属于整型的还有int8和int16
无符号整型	uint32或uint64，非负整数，取值范围为0至$2^{32}-1$和0至$2^{64}-1$，同样属于无符号整型的还有uint8和uint16
浮点数	包括float16（16位半精度浮点数）、float32（32位单精度浮点数）和float64或float（64位双精度浮点数）。其中float16用1位表示正负号，5位表示指数，10位表示尾数；float32用1位表示正负号，8位表示指数，23位表示尾数；float64用1位表示正负号，11位表示指数，52位表示尾数
复数	complex64、complex128或complex，其中complex64用两个32位浮点数表示实部和虚部，complex128用两个64位浮点数表示实部和虚部

创建ndarray对象

2. Ndarray创建

- NumPy提供了多种创建ndarray的方式，其中array函数可以创建一维或多维ndarray，其基本语法格式如下。

```
numpy.array(object, dtype=None, copy=True, order='K', subok=False, ndmin=0)
```

- array函数的常用参数及其说明如下表所示。

参数名称	说明
object	接收array，list，tuple等。表示用于创建ndarray的数据。无默认值
dtype	接收data-type。表示创建的ndarray的数据类型。如果未给定，那么选择保存对象所需的最小字节数的数据类型。无默认值
ndmin	接收int。指定生成ndarray应该具有的最小维数。默认为0

创建ndarray对象

2. Ndarray创建

- 常用的ndarray属性主要有维数、尺寸、元素总数，数据类型，每个元素的存储字节数等，分别用ndim、shape、size、dtype和itemsize来表示，这些属性的详细作用如下表所示。

属性	说明
ndim	返回int。表示ndarray的维数
shape	返回tuple。表示ndarray的尺寸，对于n行m列的矩阵，形状将为（n，m）
size	返回int。表示ndarray的元素总数，这等于形状元素的乘积
dtype	返回data-type。描述ndarray中元素类型的对象
itemsize	返回int。表示ndarray的每个元素的大小（以字节为单位）。例如，数据类型float64具有itemsize 8（= 64/8），数据类型的一个complex32具有itemsize 4（= 32/8）。相当于ndarray.dtype.itemsize

创建ndarray对象

2. Ndarray创建

- array函数创建ndarray虽然通用，但并不方便。针对一些特殊的ndarray，NumPy提供了其他的ndarray创建函数，如下表所示。

函数	说明
arange	创建等差数列（指定开始值，终值和步长）
linspace	创建等差数列（指定开始值，终值和元素个数）
logspace	创建等比数列
zeros	创建值全部为0的矩阵
eye	创建单位矩阵（对角线元素为1，其余为0）
diag	创建对角矩阵（对角线元素为指定值，其余为0）
ones	创建值全部为1的矩阵

创建ndarray对象

2. 随机数

- NumPy提供了强大的生成随机数的功能，使用随机数也可以创建ndarray。随机数相关的函数都在random模块中，包括可以生成服从多种概率分布的随机数的函数，numpy.random中的部分函数下如表所示。

函数名称	说明
seed	确定随机数生成器的种子
permutation	返回一个序列的随机排列或返回一个随机排列的范围
shuffle	对一个序列进行随机排序
random	产生0-1之间的随机浮点数
rand	产生指定形状的随机数ndarray
randint	产生给定上下限范围的随机整数ndarray
randn	产生正态分布的随机数
binomial	产生二项分布的随机数
normal	产生正态（高斯）分布的随机数
beta	产生beta分布的随机数
chisquare	产生卡方分布的随机数
gamma	产生gamma分布的随机数
uniform	产生在[0,1)中均匀分布的随机数

创建ndarray对象

2. 随机数

- randint函数能够根据给定的上下限范围生成随机整数ndarray，它稍比其他生成随机数的函数复杂，共有3个参数，其基本语法格式如下。

```
numpy.random.randint(low, high=None, size=None, dtype='l')
```

- randint函数的常用参数及其说明如下表所示。

参数名称	说明
low	接收int。表示随机范围下限。无默认值
high	接收int。表示随机范围上限。无默认值
size	接收整数tuple，指定生成ndarray的shape。无默认值

数组的排序、索引和切片

同列表一样，我们也可以针对数组中的元素进行排序、索引和切片。

示例

参阅教材内容

ndarray的索引与切片

1. 一维ndarray的索引

- 一维ndarray的索引方法很简单，与list的索引方法一致，如下代码所示。

In[25]:	**arr = np.arange(10)** **print('使用元素位置索引结果为：', arr[5])**
Out[25]:	使用元素位置索引结果为： 5
In[26]:	print('使用元素位置切片结果为：', arr[3:5])
Out[26]:	使用元素位置切片结果为： [3 4]
In[27]:	print('省略单个位置切片结果为：', arr[:5])
Out[27]:	省略单个位置切片结果为： [0 1 2 3 4]
In[28]:	print('使用元素反向位置切片结果为：', arr[:-1])
Out[28]:	使用元素反向位置切片结果为： [0 1 2 3 4 5 6 7 8]

In[29]:	arr[2:4] = 100, 101 # 修改对应下标的值
	print('修改后的ndarrayarr为：', arr)

Out[29]:	修改后的ndarrayarr为： [0 1 100 101 4 5 6 7 8 9]

ndarray的索引与切片

1. 一维ndarray的索引

In[30]:	**print('元素位置等差索引结果为：', arr[1:-1:2])**
Out[30]:	元素位置等差索引结果为： [1 101 5 7]

In[31]:	# 步长为负数时，开始位置必须大于结束位置
	print('元素位置负数步长等差索引结果为：', arr[5:1:-2])

Out[31]:	元素位置负数步长等差索引结果为： [5 101]

- 多维ndarray的每一个维度都有一个索引，各个维度的索引之间用逗号隔开，如下代码所示。

In[32]:	**arr = np.array([[1, 2, 3, 4, 5], [4, 5, 6, 7, 8], [7, 8, 9, 10, 11]])** **print('创建的二维ndarray arr为：\n', arr)**
Out[32]:	创建的二维ndarray arr为： [[1 2 3 4 5] [4 5 6 7 8] [7 8 9 10 11]]
In[33]:	print('切片结果为：', arr[0, 3:5]) # 访问第0行中第3和第4列的元素
Out[33]:	切片结果为： [4 5]
In[34]	print('切片结果为：\n', arr[1:, 2:]) # 访问第1和第2行中第2列、第3列和第4列的元素
Out[34]:	切片结果为： [[6 7 8] [9 10 11]]
In[35]:	print('切片结果为：\n', arr[:, 2]) # 访问第3列所有的元素
Out[35]:	切片结果为： [3 6 9]

ndarray的索引与切片

2. 多维ndarray的索引

- ndarray在索引与切片的时候除了使用整型的数据外，还可以使用布尔型，如下代码所示。

In[36]:	**# 索引第1、3行中第2列的元素** **mask = np.array([1, 0, 1], dtype=np.bool)** **print('使用布尔值ndarray索引结果为：', arr[mask, 2])**
Out[36]:	使用布尔值ndarray索引结果为： [3 9]

ndarray的索引与切片

3. 花式索引

- 花式索引（Fancy indexing）是一个NumPy术语，是在基础索引方式上衍生出的功能更强大的索引方式。它能够利用整数ndarray进行索引，如下代码所示。

In[37]:	**arr = np.empty((8, 4))** **for i in range(8):** **　　arr[i] = i** **print('创建的二维ndarray arr为：\n', arr)**
Out[37]:	创建的二维ndarray arr为： [[0. 0. 0. 0.] [1. 1. 1. 1.] [2. 2. 2. 2.] [3. 3. 3. 3.] [4. 4. 4. 4.] [5. 5. 5. 5.] [6. 6. 6. 6.] [7. 7. 7. 7.]]

ndarray的索引与切片

3. 花式索引

In[38]:	**print('以特定顺序索引arr结果为：\n', arr[[4, 3, 0, 6]])**
Out[38]:	以特定顺序索引arr结果为： [[4. 4. 4. 4.] [3. 3. 3. 3.] [0. 0. 0. 0.] [6. 6. 6. 6.]]
In[39]:	print('以特定逆序索引arr结果为：\n', arr[[-3, -5, -7]])
Out[39]:	以特定逆序索引arr结果为： [[5. 5. 5. 5.] [3. 3. 3. 3.] [1. 1. 1. 1.]]

ndarray的索引与切片

3. 花式索引

- 对二维ndarray传入两个索引ndarray进行花式索引，返回的是一个一维ndarray。此时，其中两个ndarray的元素一一对应，如下代码所示。

In[40]:	**arr = np.array([np.arange(i*4, i*4+4) for i in np.arange(6)])** **print('创建的二维ndarray arr为：\n', arr)**
Out[40]:	创建的二维ndarray arr为： [[0 1 2 3] [4 5 6 7] [8 9 10 11] [12 13 14 15] [16 17 18 19] [20 21 22 23]]

In[41]:	# 返回一个ndarray最终的元素(1,0)、(5,3)、(4,1)、(2,2)
	print('使用二维ndarray索引arr结果为：', arr[[1, 5, 4, 2], [0, 3, 1, 2]])

Out[41]:	使用二维ndarray索引arr结果为： [4 23 17 10]

ndarray的索引与切片

3. 花式索引

- 若需要使用ndarray索引的方式来达成获取某个区域数据的效果，则需要配合ix函数来达成目的，如下代码所示。

In[42]:	# 利用np.ix函数将两个一维的整数ndarray转化为方形区域的索引器
	print('使用ix成片索引arr结果为：\n', arr[np.ix_([1, 5, 4, 2], [0, 3, 1, 2])])

Out[42]:	使用ix成片索引arr结果为： [[4 7 5 6] [20 23 21 22] [16 19 17 18] [8 11 9 10]]

- 在使用NumPy的过程中使用花式索引常常会引起误会，所以在实际使用过程中应尽可能使用一般索引方式，在必须使用花式索引时，需做好代码注释。

变换ndarray的形态

1. 设置ndarray形状

- NumPy提供了reshape方法用于改变ndarray的形状，reshape方法仅改变原始数据的形状，不改变原始数据的值，且不改变原ndarray，如下代码所示。

In[1]:	**arr = np.arange(12) # 创建一维ndarray** **print('创建的一维ndarray arr为：', arr)**
Out[1]:	创建的一维ndarray arr为： [0 1 2 ..., 9 10 11]

In[2]:	arr1 = arr.reshape(3, 4) # 设置ndarray的维度
	print('改变形状后的ndarray arr1为：\n', arr1)

Out[2]:	改变形状后的ndarray arr1为： [[0 1 2 3] [4 5 6 7] [8 9 10 11]]
In[3]:	print('形状改变后ndarray arr1的维度为：', arr1.ndim)
Out[3]:	形状改变后ndarray arr1的维度为： 2

变换ndarray的形态

1. 设置ndarray形状

- resize方法也提供了类似reshape方法的功能，但resize方法会直接作用于所操作的ndarray，如下代码所示。

In[4]:	arr.resize(2, 6) print('resize改变原ndarray形状，ndarray arr变为：\n', arr)
Out[4]:	resize改变原ndarray形状，ndarray arr变为： [[0 1 2 3 4 5] [6 7 8 9 10 11]]

变换ndarray的形态

1. 设置ndarray形状

- 通过修改ndarray的shape属性也可以实现ndarray形状的更改，但这种方法将直接作用于所操作的ndarray，如下代码所示。

In[5]:	arr.shape = (4, 3) print('通过重新设置shape属性后，ndarray arr为：\n', arr)
Out[5]:	通过重新设置shape属性后，ndarray arr为： [[0 1 2] [3 4 5] [6 7 8] [9 10 11]]

变换ndarray的形态

2. 展平ndarray

- 展平是指将多维ndarray转换成一维ndarray的操作过程，似乎一种特殊的ndarray形状变换。在NumPy中可以使用ravel方法完成ndarray的横向展平工作，如下代码所示。

In[6]:	**arr = np.arange(12).reshape(3, 4)** **print('创建的二维ndarray arr为：\n', arr)**
Out[6]:	创建的二维ndarray arr为： [[0 1 2 3] [4 5 6 7] [8 9 10 11]]
In[7]:	print('ndarray arr横向展平后为：', arr.ravel())
Out[7]:	ndarray arr横向展平后为： [0 1 2 3 4 5 6 7 8 9 10 11]

变换ndarray的形态

2. 展平ndarray

- flatten方法也可以完成ndarray展平工作，区别在于flatten方法可以选择横向或纵向展平，如下代码所示。

In[8]:	**print('ndarray arr使用flatten方法横向展平后为：', arr.flatten())**
Out[8]:	ndarray arr使用flatten方法横向展平后为： [0 1 2 3 4 5 6 7 8 9 10 11]
In[9]:	print('ndarray arr使用flatten方法纵向展平后为：', arr.flatten('F'))
Out[9]:	ndarray arr使用flatten方法纵向展平后为： [0 4 8 1 5 9 2 6 10 3 7 11]

变换ndarray的形态

3. 组合ndarray

- 将多个ndarray组合为一个全新的ndarray，这一操作称为组合ndarray。在NumPy中，提供了横向组合、纵向组合、深度组合等多种组合方式，分别使用hstack、vstack、concatenate、dstack函数来完成。除了深度组合方式外，其余组合方式结果的维度与原ndarray相同。
- hstack函数可实现ndarray的横向组合，如下代码所示。

In[10]:	**arr1 = np.arange(12).reshape(3, 4)** **print('创建的ndarray arr1为：\n', arr1)**
Out[10]:	创建的ndarray arr1为： [[0 1 2 3] [4 5 6 7] [8 9 10 11]]
In[11]:	arr2 = arr1*3 print('创建的ndarray arr2为：\n', arr2)

变换ndarray的形态

3. 组合ndarray

Out[11]:	**创建的ndarray arr2为：** **[[0 3 6 9]** **[12 15 18 21]** **[24 27 30 33]]**
In[12]:	print('hstack横向组合ndarray arr1与arr2为： \n', np.hstack((arr1, arr2)))
Out[12]:	hstack横向组合ndarray arr1与arr2为： [[0 1 2 3 0 3 6 9] [4 5 6 7 12 15 18 21] [8 9 10 11 24 27 30 33]]

变换ndarray的形态

3. 组合ndarray

- vstack函数可实现ndarray的纵向组合，如下代码所示。

In[13]:	print('vstack纵向组合ndarray arr1与arr2为：\n', np.vstack((arr1, arr2)))
Out[13]:	vstack纵向组合ndarray arr1与arr2为： [[0 1 2 3] [4 5 6 7] [8 9 10 11] [0 3 6 9] [12 15 18 21] [24 27 30 33]]

变换ndarray的形态

3. 组合ndarray

- concatenate函数既可以实现横向组合，也能巩固实现纵向组合。当参数axis=1时为横向组合，参数axis=0时则为纵向组合，如下代码所示。

In[14]:	**print('concatenate横向组合arr1与arr2为：\n', np.concatenate((arr1, arr2), axis=1))**
Out[14]:	concatenate横向组合arr1与arr2为： [[0 1 2 3 0 3 6 9] [4 5 6 7 12 15 18 21] [8 9 10 11 24 27 30 33]]
In[15]:	print('concatenate纵向组合arr1与arr2为：\n', np.concatenate((arr1, arr2), axis=0))
Out[15]:	concatenate纵向组合arr1与arr2为： [[0 1 2 3] [4 5 6 7] [8 9 10 11] [0 3 6 9] [12 15 18 21] [24 27 30 33]]

变换ndarray的形态

3. 组合ndarray

- 所谓深度组合，就是将一系列ndarray沿着纵轴方向进行层叠组合，类似Python内置函数zip。NumPy提供了dstack函数可实现深度组合，如下代码所示。

In[16]:	print('dstack深度组合arr1与arr2为：\n', np.dstack((arr1, arr2)))
Out[16]:	dstack深度组合arr1与arr2为： [[[0 0] [1 3] [2 6] [3 9]] [[4 12] [5 15] [6 18] [7 21]]

变换ndarray的形态

3. 组合ndarray

	[[8 24] [9 27] [10 30] [11 33]]]
In[17]:	arr3 = [] for x,y in list(zip(arr1, arr2)): arr3.append(list(zip(x, y))) arr3 = np.array(arr3) print('zip函数实现深度合并的arr3与dstack实现的等价：\n', arr3 == np.dstack((arr1, arr2)))

变换ndarray的形态

3. 组合ndarray

```
Out[17]: zip函数实现深度合并的arr3与dstack实现的等价：
         [[[ True  True]
           [ True  True]
           [ True  True]
           [ True  True]]

          [[ True  True]
           [ True  True]
           [ True  True]
           [ True  True]]

          [[ True  True]
           [ True  True]
           [ True  True]
           [ True  True]]]
```

变换ndarray的形态

4. 分割ndarray

- 将一个数组拆分为多个，这一操作称为分割ndarray。在NumPy中ndarray可以进行横向、纵向或深度分割，分别使用hsplit、vsplit、split和dsplit函数实现。通过这些函数可以将ndarray分割成相同大小的子ndarray，也可以根据位置分割为目标形状。分割后的每一个ndarray维度与原ndarray相同。
- hsplit函数可实现ndarray的横向分割，如下代码所示。

In[18]:	**arr = np.arange(16).reshape(4, 4)** **print('创建的二维ndarray arr为：\n', arr)**
Out[18]:	创建的二维ndarray arr为： [[0 1 2 3] [4 5 6 7] [8 9 10 11] [12 13 14 15]]
In[19]:	print('hsplit横向分割arr为：\n', np.hsplit(arr, 2))

变换ndarray的形态

4. 分割ndarray

```
Out[19]: hsplit横向分割arr为：
 [array([[ 0,  1],
        [ 4,  5],
        [ 8,  9],
        [12, 13]]),
 array([[ 2,  3],
        [ 6,  7],
        [10, 11],
        [14, 15]])]
```

变换ndarray的形态

4. 分割ndarray

- vsplit函数可实现ndarray的纵向分割，如下代码所示。

In[20]:	print('hsplit纵向分割arr为：\n', np.vsplit(arr, 2))
Out[20]:	hsplit纵向分割arr为： [array([[0, 1, 2, 3], [4, 5, 6, 7]]), array([[8, 9, 10, 11], [12, 13, 14, 15]])]

变换ndarray的形态

4. 分割ndarray

- split函数同样可以实现ndarray的横向纵向分割，当axis=1时为横向分割，当axis=0时为纵向分割，如下代码所示。

In[21]:	**print('split横向分割arr为：\n', np.split(arr, 2, axis=1))**
Out[21]:	split横向分割arr为： [array([[0, 1], [4, 5], [8, 9], [12, 13]]), array([[2, 3], [6, 7], [10, 11], [14, 15]])]
In[22]:	print('split纵向分割arr为：\n', np.split(arr, 2, axis=0))
Out[22]:	split纵向分割arr为： [array([[0, 1, 2, 3], [4, 5, 6, 7]]), array([[8, 9, 10, 11], [12, 13, 14, 15]])]

变换ndarray的形态

4. 分割ndarray

- dsplit函数可实现ndarray的深度分割，但分割的ndarray必须是三维ndarray，且分割的数目必须为shape属性中下标为2的值的公约数，如下代码所示。

In[23]:	**arr = np.arange(12).reshape(2, 2, 3)** **print('创建的三维ndarray arr为：\n', arr)**
Out[23]:	创建的三维ndarray arr为： [[[0 1 2] [3 4 5]] [[6 7 8] [9 10 11]]]
In[24]:	print('dsplit深度分割arr为：\n', np.dsplit(arr, 3))

变换ndarray的形态

4. 分割ndarray

Out[24]:	dsplit深度分割arr为： [array([[[0], [3]], [[6], [9]]]), array([[[1], [4]],[[7],[10]]]), array([[[2],[5]],[[8], [11]]])]

排序与搜索

1. 排序

- NumPy的主要排序方式可以概括为直接排序和间接排序两种。直接排序指对数值直接进行排序；间接排序是指根据一个或多个键对数据集进行排序。NumPy中的常用排序函数有sort、argsort和lexsort函数。
- 其中，sort函数是最常用的排序方法，其基本语法格式如下。

```
numpy.sort(a, axis=-1, kind='quicksort', order=None)
```

- sort函数的常用参数及其说明如下表所示。

参数名称	说明
a	接收array。表示想要排序的ndarray。无默认值
axis	接收int。表示指定的轴，axis=1时指定横轴，axis=0时指定纵轴。默认为-1，即ndarray被横向展开排序
kind	接收str。表示排序的算法，可取“quicksort”，“mergesort”，“heapsort”，“stable”。默认为“quicksort”

排序与搜索

1. 排序

- argsort函数和lexsort函数可以实现在给定一个或多个键下，得到一个由整数构成的索引ndarray，索引值表示数据在新的顺序下的位置。配合花式索引即可实现基于某一列或者某一行的排序。
- argsort函数的基本语法格式如下。

```
numpy.argsort(a, axis=-1, kind='quicksort', order=None)
```

- sort 函数的常用参数及其说明如下表所示。

参数名称	说明
a	接收array。表示想要排序的ndarray。无默认值
axis	接收int。表示指定的轴，axis=1时指定横轴，axis=0时指定纵轴。默认为-1，即ndarray被展开排序
kind	接收string。表示排序的方法，可取“quicksort”，“mergesort”，“heapsort”，“stable”。默认为“quicksort”

排序与搜索

2. 搜索

- NumPy模块提供了一些用于在ndarray内搜索的函数，包括用于找到最大值、最小值以及满足给定条件的元素的函数。
- 其中，argmax和argmin函数可以求最大和最小元素的索引，基本语法格式如下。

```
numpy.argmax(a, axis=None, out=None)
numpy.argmin(a, axis=None, out=None)
```

- argmax和argmin函数的常用参数及其说明如下表所示。

参数名称	说明
a	接收array。表示想要搜索的ndarray。无默认值
axis	接收int。表示指定的轴，axis=1时指定横轴，axis=0时指定纵轴。默认情况下，索引的是平铺的ndarray

排序与搜索

2. 搜索

- where函数返回输入ndarray中满足给定条件的元素的索引，其基本语法格式如下。

```
np.where(condition, x, y)
```

- where函数的常用参数及其说明如下表所示。

参数名称	说明
condition	接收array、bool，表示搜索的条件。无默认值
x	接收array，表示搜索对象。无默认值
y	接收array，表示不满足条件的值的替换值。无默认值

排序与搜索

2. 搜索

- extract函数返回输入ndarray中满足给定条件的元素。extract函数的基本语法格式如下。

```
numpy.extract(condition,arr)
```

- extract函数的常用参数及其说明如下表所示。

参数名称	说明
condition	接收array，表示搜索的条件。无默认值
arr	接收array，表示满足条件的元素。无默认值

常用ufunc

1. 算术运算

- ufunc支持算术运算，并且有运算符和函数两种方式，和数值运算的使用方式一样，但输入ndarray必须具有相同的形状或符合ndarray广播规则，如下表所示。

运算符	函数格式	说明
+	add(x,y)	ndarray x各元素与ndarray y各元素的和，返回数值型ndarray
–	subtract(x,y)	ndarray x各元素与ndarray y各元素的的差，返回数值型ndarray
*	multiply(x,y)	ndarray x各元素与ndarray y各元素的的积，返回数值型ndarray
/	divide(x,y)	ndarray x各元素与ndarray y各元素的的商，返回数值型ndarray
**	power(x,y)	ndarray x各元素的ndarray y各元素次幂，返回数值型ndarray

常用ufunc

1. 算术运算

- 除基础的四则运算外，NumPy还提供了其他数学运算的函数，如下表所示。

函数格式	说明	函数格式	说明
negative(x)	求ndarray x各元素的相反数，返回数值型ndarray	exp(x)	求自然数E的ndarray x各元素次幂，返回数值型ndarray
absolute(x)	求ndarray x各元素的绝对值，返回0或正整数ndarray	sqrt(x)	求ndarray x各元素的平方根，返回数值型ndarray
fabs(x)	求ndarray x各元素的绝对值，返回0或正整数ndarray	curt(x)	求ndarray x各元素的立方根，返回数值型ndarray
rint(x)	求ndarray x各元素最近的整数，返回整数ndarray	reciprocal(x)	求ndarray x各元素的倒数，返回数值型ndarray
sign(x)	求ndarray x个元素的符号，返回仅含-1，1的ndarray	conj(x)	求ndarray x各元素的共轭复数，返回数值型ndarray
log1p(x)	求ndarray x各元素的自然数E为底的对数，返回数值型ndarray	log2(x)	返回ndarray x各元素以2为底的对数，返回数值型ndarray
log(x)	求ndarray x个元素的对数，返回数值型ndarray	log10(x)	返回ndarray x各元素以10为底的对数，返回数值型ndarray

常用ufunc

2. 三角函数

- ufunc提供标准的三角函数与双曲三角函数功能，双曲函数是一类与常见的三角函数类似的函数，双曲函数经常出现于某些重要的线性微分方程的解中。ufunc提供的三角函数，如下表所示。

函数格式	说明	函数格式	说明
sin(x)	三角正弦运算，返回数值型ndarray	sinh(x)	双曲正弦运算，返回数值型ndarray
cos(x)	三角余弦运算，返回数值型ndarray	cosh(x)	双曲余弦运算，返回数值型ndarray
tan(x)	三角正切，，返回数值型ndarray	tanh(x)	双曲正切运算，返回数值型ndarray
arcsin(x)	三角反正弦运算，返回数值型ndarray	arcsinh(x)	反双曲正弦运算，返回数值型ndarray
arccos(x)	三角反余弦运算，返回数值型ndarray	arccosh(x)	反双曲余弦运算，返回数值型ndarray

常用ufunc

2. 三角函数

函数格式	说明	函数格式	说明
arctan(x)	三角反正切运算，返回数值型ndarray	arctanh(x)	反双曲正切运算，返回数值型ndarray
degrees(x)	弧度转换为角度，返回数值型ndarray	rad2deg(x)	弧度转换为角度，返回数值型ndarray
radians(x)	角度转换为弧度，返回数值型ndarray	deg2rad(x)	角度转换为弧度，返回数值型ndarray
hypot(x, y)	通过直角三角形的直角边x，y求斜边，返回数值型ndarray		

3. 集合运算

- ufunc支持基本集合运算，集合函数及其说明，如下表所示。

函数格式	说明
unique(x)	去重并排序，返回与x类型相同ndarray
intersect1d(x,y)	ndarray x与ndarray y的交集，返回与x类型相同ndarray
union1d(x,y)	ndarray x与ndarray y的并集，返回与x类型相同ndarray
in1d(x,y)	判断ndarray x中的元素是否存在于ndarray y中，返回布尔型ndarray
setdiff1d(x,y)	ndarray x中的元素减去ndarray x与ndarray y交集的元素，返回与x类型相同ndarray
setxor1d(x,y)	ndarray x与ndarray y的对称差集。返回与x类型相同ndarray

常用ufunc

4. 比较运算

- ufunc也可以进行完整的比较运算，并且有运算符和比较函数两种方式，如下表所示。

运算符	函数格式	说明
==	equal(x, y)	ndarray x与ndarray y各元素是否相等，返回bool型ndarray
!=	not_equal(x, y)	ndarray x与ndarray y各元素是否不相等，返回bool型ndarray
<	less(x, y)	ndarray x各元素是否小于ndarray y中的元素，返回bool型ndarray
<=	less_equal(x, y)	ndarray x各元素是否小于等于ndarray y中的元素，返回bool型ndarray
>	greater(x, y)	ndarray x各元素是否大于ndarray y中的元素，返回bool型ndarray
>=	greater_equal(x, y)	ndarray x各元素是否大于等于ndarray y中的元素，返回bool型ndarray

常用ufunc

5. 逻辑运算

- ufunc支持基本逻辑运算，函数及其说明，如下表所示。

函数格式	说明
any(x)	ndarray x中是否存在一个为True的元素，返回bool值
all(x)	ndarray x内元素是否全为True，返回bool值
logical_and(x, y)	ndarray x内元素与ndarray y内的对应元素的与运算，返回布尔型ndarray
logical_or(x, y)	ndarray x内元素与ndarray y内的对应元素的或运算，返回布尔型ndarray
logical_not(x)	ndarray x对应元素的非运算，返回布尔型ndarray
logical_xor(x, y)	ndarray x内元素与ndarray y内的对应元素的异或运算，返回布尔型ndarray

常用ufunc

5. 逻辑运算

- 此外，ufunc提供ndarray内容测试函数，如下表所示。

函数格式	说明
isfinite(x)	判断ndarray x内的有穷值，返回bool型ndarray
isinf(x)	判断ndarray x内的无穷值，返回bool型ndarray
isnan(x)	判断ndarray x内的空值，返回bool型ndarray
isneginf(x)	判断ndarray x内的负无穷值，返回bool型ndarray
isposinf(x)	判断ndarray x内的正无穷值，返回bool型ndarray

常用ufunc

6. 统计运算

- 在NumPy中有许多可以用于统计分析的函数，如下表所示。

函数格式	说明	函数格式	说明
sum(x)	ndarray x内元素和，返回数值型数据或数值型ndarray	var(x)	ndarray x内元素方差，返回数值型数据或数值型ndarray
ptp(x)	ndarray x内元素极差，返回数值型数据或数值型ndarray	min(x)	ndarray x内元素最小值，返回数值型数据或数值型ndarray
mean(x)	ndarray x内元素均值，返回数值型数据或数值型ndarray	max(x)	ndarray x内元素最大值，返回数值型数据或数值型ndarray
median(x)	ndarray x内元素中位数，返回数值型数据或数值型ndarray	cumsum(x)	ndarray x内元素累计和，返回数值型数据或数值型ndarray
percentile(x,y)	ndarray x内元素的对应y元素值的百分位数，返回数值型数据或数值型ndarray	cumprod(x)	ndarray x内元素累计积，返回数值型数据或数值型ndarray
std(x)	ndarray x内元素标准差，返回数值型数据或数值型ndarray		

PART 05

pandas模块序列与数据框

python pandas模块序列（series）与数据框（DataFrame）

我们在实务中开展机器学习时，面对的通常是各种数据表，多以“.csv”的形式（EXCEL可打开）存在。在数据表中，通常用每一行来表示一个样本示例，每一列表示一个变量，比如一张30行3列的数据表，其中包括30个样本示例，每个样本示例都有3个变量维度。上节中我们介绍的数组，虽然也可以生成二维数组（矩阵）的方式对数据予以展现，但最大的缺点在于无法展现出变量（列）的名称，这时候我们就可以用到pandas模块中的序列（series）与数据框（DataFrame），其中序列为一维数据（注意此处讲的序列是pandas模块中的序列（series），不同于第一章中讲的序列），而且每个元素都有相应的标签（index，也称行编号）；数据框则为多维数据，针对数据中包含多个变量的情形，针对每个样本示例不仅可以展示行编号，也可以设置各个列变量的名称。

- pandas提供了众多类，满足不同的使用需求，其中常用的类，如下表所示。

类	说明
Series	基本数据结构，一维标签数组，能够保存任何数据类型
DataFrame	基本数据结构，一般为二维数组，是一组有序的列
Index	索引对象，负责管理轴标签和其他元数据（比如轴名称）
groupby	分组对象，通过传入需要分组的参数实现对数据分组
Timestamp	时间戳对象，表示时间轴上的一个时刻
Timedelta	时间差对象，用来计算两个时间点的差值

序列（series）的相关操作

- 创建序列（series）
- 序列中元素的索引（index）和值（values）
- 序列中元素值的基本统计

示例

参阅教材内容

Series

1. 创建Series

- Series类用于创建Series对象，其主要参数为data和index，基本语法格式如下。

```
class pandas.Series(data=None, index=None, dtype=None, name=None, copy=False, fastpath=False)
```

- Series类常用的参数及其说明，如下表所示。

参数名称	说明
data	接收array或dict。表示接收的数据。默认为None
index	接收array或list。表示索引，它必须与数据长度相同。默认为None
name	接收str或list。表示Series对象的名称。默认为None

Series

1. 创建Series

- Series本质上是一个ndarray，通过ndarray创建Series对象，如下代码所示。

In[1]:	import pandas as pd import numpy as np print('通过ndarray创建的Series为：\n', pd.Series(np.arange(5), index = ['a', 'b', 'c', 'd', 'e'], name = 'ndarray'))
Out[1]:	通过ndarray创建的Series为： a 0 b 1 c 2 d 3 e 4 Name: ndarray, dtype: int32

- 若数据存放于一个dict中，则可以通过dict创建Series，此时dict的键名（key）作为Sereis的索引，其值会作为Sereis的值，因此无需传入index参数。通过dict创建Series对象，如下代码所示。

In[2]:	dit = {'a': 0, 'b': 1, 'c': 2, 'd': 3, 'e': 4} print('通过dict创建的Series为： \n', pd.Series(dit))
Out[2]:	通过dict创建的Series为： a 0 b 1 c 2 d 3 e 4 dtype: int64

- 通过list创建Series，类似于通过ndarray创建Series，如下代码所示。

In[3]:	list1 = [0, 1, 2, 3, 4] print('通过list创建的Series为：\n', pd.Series(list1, index = ['a', 'b', 'c', 'd', 'e'], name = 'list'))
Out[3]:	通过list创建的Series为： a 0 b 1 c 2 d 3 e 4 Name: list, dtype: int64

- Series拥有8个常用属性，如下表所示。

属性	说明
values	以ndarray的格式返回Series对象的所有元素
index	返回Series对象的索引
dtype	返回Series对象的数据类型
shape	返回Series对象的形状
nbytes	返回Series对象的字节数
ndim	返回Series对象的维度
size	返回Series对象的个数
T	返回Series对象的转置

- 索引与切片是Series最常用的操作之一。通过索引位置访问Series的数据，与ndarray相同，如下代码所示。

In[8]:	print('Series位于第1位置的数据为：', series[0])
Out[8]:	Series位于第1位置的数据为： 0

- 相比ndarray，通过索引名称（标签）也可以访问Series数据，如下代码所示。

In[9]:	print('Series中Index为a的数据为：', series['a'])
Out[9]:	Series中Index为a的数据为： 0

- 索引与切片是Series最常用的操作之一。通过索引位置访问Series的数据，与ndarray相同，如下代码所示。

In[10]:	**bool = (series < 4)** **print('bool类型的Series为：\n', bool)**
Out[10]:	bool类型的Series为： a True b True c True d True e False Name: list, dtype: bool
In[11]:	print('通过bool数据访问Series结果为：\n', series[bool])
Out[11]:	通过bool数据访问Series结果为： a 0 b 1 c 2 d 3 Name: list, dtype: int64

- 更新Series的方法十分简单，采用赋值的方式对指定索引标签（或位置）对应的数据进行修改即可，如下代码所示。

In[12]:	**# 更新元素** **series['a'] = 3** **print('更新后的Series为：\n', series)**
Out[12]:	更新后的Series为： a 3 b 1 c 2 d 3 e 4 Name: list, dtype: int64

- 类似list，通过append方法能够在原Series上插入（追加）新的Series。若只在原Series上插入单个值，则采用赋值方式即可，如下代码所示。

In[13]:	series1 = pd.Series([4, 5], index = ['f', 'g'])
	# 追加Series
	print('在series插入series1后为：\n', series.append(series1))

Out[13]:	在series插入series1后为： a 3 b 1 c 2 d 3 e 4 f 4 g 5 dtype: int64

更新、插入和删除

In[14]:	# 新增单个数据 series1['h'] = 7 print('在series1插入单个数据后为：\n', series1)
Out[14]:	在series1插入单个数据后为： f 4 g 5 h 7 dtype: int64

- 一般使用drop方法删除Series元素，它接收被删除元素对应的索引，inplace=True表示对原Series起作用，如下代码所示。

In[15]:	**# 删除数据** **series.drop('e', inplace = True)** **print('删除索引e对应数据后的series为：\n', series)**
Out[15]:	删除索引e对应数据后的series为： a 3 b 1 c 2 d 3 Name: list, dtype: int64

数据框（DataFrame）的相关操作

数据框（DataFrame）针对多维数据，其特色在于针对每个样本示例不仅可以展示行编号，也可以设置各个列变量的名称，非常契合商业运营实践中的实际数据存储方式，所以应用非常广泛。

- 创建数据框（DataFrame）
- 数据框索引（index）、列（columns）和值（values）
- 提取数据框中的数据
- 从数据框中提取子数据框
- 数据框中变量列的编辑操作

示例

参阅教材内容

- DataFrame类用于创建DataFrame对象，其基本语法格式如下。

```
class pandas.DataFrame(data=None, index=None, columns=None, dtype=None, copy=False)
```

- DataFrame类常用的参数及其说明，如下表所示。

参数名称	说明
data	接收ndarray，dict，list或DataFrame。表示输入数据。默认为None
index	接收Index，ndarray。表示索引。默认为None
columns	接收Index，ndarray。表示列标签（列名）。默认为None

- 由于DataFrame是二维数据结构，包含列索引（列名），因此较Series拥有更多的属性。DataFrame常用的属性及其说明，如下表所示。

属性	说明
values	以ndarray的格式返回DataFrame对象的所有元素
index	返回DataFrame对象的Index
columns	返回DataFrame对象的列标签
dtypes	返回DataFrame对象的数据类型
axes	返回DataFrame对象的轴标签
ndim	返回DataFrame对象的轴尺寸数
size	返回DataFrame对象的个数
shape	返回DataFrame对象的形状

- head和tail方法用于访问DataFrame前n行和后n行数据，默认返回5行数据，如下代码所示。

In[23]:	**print('默认返回前5行数据为：\n', df.head())**
Out[23]:	默认返回前5行数据为： col1 col2 a 0 5 b 1 6 c 2 7 d 3 8 e 4 9
In[24]:	print('返回后3行数据为：\n', df.tail(3))
Out[24]:	返回后3行数据为： col1 col2 c 2 7 d 3 8 e 4 9

- 类似Series，更新DataFrame列也采用赋值的方法，对指定列赋值即可，如下代码所示。

In[25]:	# 更新列 df['col1'] = [10, 11, 12, 13, 14] print('更新列后的DataFrame为：\n', df)
Out[25]:	更新列后的DataFrame为： col1 col2 a 10 5 b 11 6 c 12 7 d 13 8 e 14 9

- 插入列也可以采用赋值方法，如下代码所示。

In[26]:	# 插入列 df['col3'] = [15, 16, 17, 18, 19] print('插入列后的DataFrame为：\n', df)
Out[26]:	插入列后的DataFrame为： col1 col2 col3 a 10 5 15 b 11 6 16 c 12 7 17 d 13 8 18 e 14 9 19

- 删除列方法有多种，如del、pop、drop等，常用的是drop方法，它可以删除行或者列，基本语法格式如下。

```
DataFrame.drop(labels, axis=0, level=None, inplace=False, errors='raise')
```

- drop方法常用的参数及其说明，如下表所示。

参数名称	说明
labels	接收str或array。表示删除的行或列的标签。无默认值
axis	接收0或1。表示执行操作的轴向，其中0表示删除行，1表示删除列。默认为0
levels	接收int或者索引名。表示索引级别。默认为None
inplace	接收bool。表示操作是否对原数据生效。默认为False

- Index对象为其他pandas对象的存储轴标签，管理轴标签和其他元数据（如轴名称）。创建Series或DataFrame等对象时，索引都会被转换为Index对象。主要Index对象及其说明如下表所示。

对象名称	说明
Index	一般的Index对象
MultiIndex	层次化Index对象
DatetimeIndex	Timestamp索引对象
PeriodIndex	Period索引对象

- Index对象可以通过pandas模块中的Index类创建，也可以通过创建数据对象如Series、DataFrame时接收index（或column）参数创建，前者属于显式创建，后者属于隐式创建。隐式创建中，通过访问index（或针对DataFrame的column）属性即可得到Index。创建的Index对象不可修改，保证了Index对象在各个数据结构之间的安全共享。Series的索引是一个Index对象，访问Series索引，如下代码所示。

In[29]:	print('series的Index为： \n', series.index)
Out[29]:	series的Index为： Index(['a', 'b', 'c', 'd'], dtype='object')

- Index对象常用的属性及其说明，如下表所示。

属性	说明
is_monotonic	当各元素均大于前一个元素时，返回True
is_unique	当Index没有重复值时，返回True

- Index对象的常用方法及其说明，如下表所示。

方法名称	说明
append	连接另一个Index对象，产生一个新的Index
difference	计算两个Index对象的差集，得到一个新的Index
intersection	计算两个Index对象的交集
union	计算两个Index对象的并集
isin	计算一个Index是否在另一个Index，返回bool数组
delete	删除指定Index的元素，并得到新的Index
drop	删除传入的值，并得到新的Index
insert	将元素插入到指定Index处，并得到新的Index
unique	计算Index中唯一值的数组

- 根据DataFrame的定义可知，DataFrame是一个带有标签的二维数组，每个标签相当每一列的列名。只要以dict访问某一个键（key）的值的方式访问对应的列名，就可以访问DataFrame的某列数据，它返回的是Series，如下代码所示。

In[1]:	**import pandas as pd** **df = pd.DataFrame({'col1': [0, 1, 2, 3, 4], 'col2': [5, 6, 7, 8, 9]},** **index = ['a', 'b', 'c', 'd', 'e'])** **print('创建的DataFrame为：\n', df)**
Out[1]:	创建的DataFrame为： col1 col2 a 0 5 b 1 6 c 2 7 d 3 8 e 4 9

In[2]:	# 访问单列数据
	print('DataFrame中col1列数据为：\n', df['col1'])

Out[2]:	DataFrame中col1列数据为： a 0 b 1 c 2 d 3 e 4 Name: col1, dtype: int64

- 此外，还能够以属性的方式访问单列数据，如下代码所示。

In[3]:	**# 以属性的方式访问单列数据** **print('DataFrame中col1列数据为：\n', df.col1)**
Out[3]:	DataFrame中col1列数据为： a 0 b 1 c 2 d 3 e 4 Name: col1, dtype: int64

- 以上两种基础的索引方法均可以获得DataFrame中的某一列数据，但是不建议使用属性的方法访问数据，因为通常列名为英文，以属性方式访问某一列的形式和访问属性相同，若列名与属性相同，则会引起程序混乱，也使得代码晦涩难懂。

- 访问DataFrame中某一列的某几行时，可以采用适用于Series的索引方式，如下代码所示。

In[4]:	**# 访问单列多行数据** **print('DataFrame中col1列前3行数据为：\n', df['col1'][0: 3])**
Out[4]:	DataFrame中col1列前3行数据为： a 0 b 1 c 2 Name: col1, dtype: int64

- 访问DataFrame多列数据时，可以将多个列的列名（列标签）视为一个list，因此接收多个列名组成的list即可访问多列数据，与此同时，也可选择多行数据，如下代码所示。

In[5]:	**# 访问多列多行数据** **print('DataFrame中col1列、col2列前3行数据为：\n', df[['col1', 'col2']][0:3])**
Out[5]:	DataFrame中col1列、col2列前3行数据为： col1 col2 a 0 5 b 1 6 c 2 7

- 若只访问DataFrame某几行数据，则方式与访问多列多行相似，此时不用接收所有列名组成的list，使用“:”代替即可，如下代码所示。

In[6]:	**# 访问多行数据** **print('DataFrame的前3行为：\n', df[:][0: 3])**
Out[6]:	DataFrame的前3行为： col1 col2 a 0 5 b 1 6 c 2 7

loc方法

- loc方法是基于名称的索引方法，它接收索引名称（标签），若索引名称不存在则会报错。loc方法也能够接收整数，但这个整数必须是已存在的索引名称。loc方法的基本语法格式如下。

```
DataFrame.loc[行索引名称或条件，列索引名称]
```

loc方法

- loc方法可以像基础索引方式一样访问数据子集。行索引在前，列索引在后，整行或整列用“:”代替，当只查看行数据时“:”可以省略，如下代码所示。

In[7]:	**# 访问单列数据** **print('DataFrame中col1列数据为：\n', df.loc[: , 'col1'])**
Out[7]:	DataFrame中col1列数据为： a 0 b 1 c 2 d 3 e 4 Name: col1, dtype: int64

In[8]:	# 访问多列数据
	print('DataFrame中col1列、col2数据为：\n', df.loc[: , ['col1', 'col2']])

loc方法

Out[8]:	**DataFrame中col1列、 col2数据为：** **col1 col2** **a 0 5** **b 1 6** **c 2 7** **d 3 8** **e 4 9**

In[9]:	# 访问单行数据
	print('DataFrame中a行对应数据为： \n', df.loc['a', :])

Out[9]:	DataFrame中a行对应数据为： col1 0 col2 5 Name: a, dtype: int64

In[10]:	# 访问多行数据
	print('DataFrame中a行、 b行对应数据为： \n', df.loc[['a', 'b'], :])

loc方法

Out[10]:	**DataFrame中a行、b行对应数据为：** **col1 col2** **a 0 5** **b 1 6**
In[11]:	# 行列结合访问数据 print('DataFrame中a行、b行，col1列、col2列对应的数据为：\n', df.loc[['a', 'b'], ['col1', 'col2']])
Out[11]:	DataFrame中a行、b行，col1列、col2列对应的数据为： col1 col2 a 0 5 b 1 6

loc方法

- loc方法接收多种输入形式，输入形式包括单个索引名称、索引名称组成的list、名称切片、bool类型的数据（Series、list或array）、包含一个参数的函数这5种。
- 使用loc方法允许的5种输入形式进行索引操作，如下代码所示。

In[12]:	**# 接收bool数据** **print('DataFrame中col1列大于0的数据为：\n', df.loc[df['col1'] > 0, :])**
Out[12]:	DataFrame中col1列大于0的数据为： col1 col2 b 1 6 c 2 7 d 3 8 e 4 9

In[13]:	# 接收函数
	print('DataFrame中col1列大于0的数据为：\n', df.loc[lambda df: df['col1'] > 0, :])

loc方法

Out[13]:	DataFrame中col1列大于0的数据为： col1 col2 b 1 6 c 2 7 d 3 8 e 4 9

iloc方法

- 另外一种常用的索引方法是iloc，与loc方法基于索引名称不同，iloc方法完全基于位置，它接收int，不能接收索引名称，否则会报错。iloc的用法完全与NumPy中ndarray的数字索引方式相同。iloc方法的基本语法格式如下。

```
DataFrame.iloc[行索引位置, 列索引位置]
```

使用iloc方法访问DataFrame的数据子集，基本用法与loc方法类似，行在前，列在后，它们的主要区别有两个。

- loc方法传入的是索引名称，而iloc方法限定为是索引位置。
- loc方法传入的行索引名称如果为一个区间，那么前后均为闭区间，而iloc方法为前闭后开区间。

iloc方法

- 类似loc方法，iloc方法也允许多种输入形式，输入形式包括单个int、int组成的list、int切片、bool类型的数据（list或array）、包含一个参数的函数这5种。
- 使用iloc方法允许的5种输入形式进行索引操作，如下代码所示。

In[19]:	# 接收bool数据
	print('DataFrame 中 col1 列 大 于 0 的 数 据 为 : \n', df.iloc[df['col1'].values>0, :])

Out[19]:	DataFrame中col1列大于0的数据为: col1 col2 b 1 6 c 2 7 d 3 8 e 4 9

iloc方法

In[20]:	# 接收函数
	print('DataFrame中col1列大于0的数据为：\n'，df.iloc[lambda df: df['col1'].values>0, :])

Out[20]:	DataFrame中col1列大于0的数据为： col1 col2 b 1 6 c 2 7 d 3 8 e 4 9

- sort_index方法用于对DataFrame按索引排序，其基本语法格式如下。

```
DataFrame.sort_index(axis = 0, level = None, ascending = True, inplace = False)
```

- sort_index方法常用的参数及其说明，如下表所示。

参数名称	说明
axis	接收0或1。表示排序作用的轴，其中0表示对行排序，1表示对列排序。默认为0
level	接收int、list或str。表示索引级别。默认为None
ascending	接收bool。表示排序方式，False表示升序，True表示降序。默认为False
inplace	接收bool。表示操作是否对原数据生效。默认为False

- sort_values方法用于按值排序，其基本语法格式如下。

```
DataFrame.sort_values(by, axis=0, ascending=True, inplace=False)
```

- sort_values方法常用的参数及其说明，如下表所示。

参数名称	说明
by	接收str或由str组成的list。表示排序依据的值，可以为列名或索引名。无默认值
axis	接收0或1。表示排序作用的轴，其中0表示对行排序，1表示对列排序。默认为0
ascending	接收bool。表示排序方式，True表示降序，False表示升序。默认为False
inplace	接收bool。表示操作是否对原数据生效。默认为False

- nlargest方法和nsmallest方法也可用于按列排序，它们返回DataFrame的前n个最大值和最小值，如下代码所示。

In[34]:	print('按col2列排序,返回前2个最小值：\n', df.nsmallest(2, 'col2'))
Out[34]:	按col2列排序,返回前2个最小值： col1 col2 a 0 5 b 1 6
In[35]:	print('按col2列排序,返回前2个最大值：\n', df.nlargest(2, 'col2'))
Out[35]:	按col2列排序,返回前2个最大值： col1 col2 e 4 9 d 3 8

- 堆叠就是简单地把两个表拼在一起，也被称作轴向连接，绑定或连接。依照连接轴的方向，数据堆叠可分为横向堆叠和纵向堆叠。pandas提供concat函数，用于表堆叠，其基本语法格式如下。

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

- concat函数的具体用法及相关参数说明如下表所示。

参数名称	说明
objs	接收多个Series，DataFrame，Panel的组合。表示参与链接的pandas对象的list的组合。无默认值
axis	接收0或1。表示连接的轴向，默认为0
join	接收特定str（“inner”，“outer”）。表示其他轴向上的索引是按交集（inner）还是并（outer）集进行合并。默认为outer。
join_axes	接收索引对象。表示用于其他n-1条轴的索引，不执行并集/交集运算
ignore_index	接收bool。表示是否不保留连接轴上的索引，产生一组新Indexrange(total_length)。默认为False

参数名称	说明
keys	接收sequence。表示与连接对象有关的值，用于形成连接轴向上的层次化索引。默认为None
levels	接收包含多个sequence的list。表示在指定keys参数后，指定用作层次化索引各级别上的索引。默认为None
names	接收list。表示在设置了keys和levels参数后，用于创建分级别别的名称。默认为None
verify_integrity	接收boolearn。表示是否检查结果对象新轴上的重复情况，如果发现则引发异常。默认为False

横向堆叠

- 横向堆叠，即将两个表在x轴向拼接在一起，concat函数能够实现横向堆叠。当axis=1的时候，concat函数做行对齐，然后将不同列名称的两张或多张表合并，从而实现横向堆叠。当两个表索引不完全一样时，可以使用join参数选择是内连接还是外连接，默认为外连接。在内连接的情况下，仅仅返回索引重叠部分。在外连接的情况下，则显示的并集部分数据，不足的地方则使用空值填补，其原理示意如下图所示，图中表1和表2使用外连接横向堆叠合并后，形成表3。

□,

	<	=	>	?
,	<,	=,	>,	?,
–	<–	=–	>–	?–
.	<.	=.	>.	?.
/	</	=/	>/	?/

□–

	=	?	A
–	=–	?–	A–
/	=/	?/	A/
1	=1	?1	A1
3	=3	?3	A3

凌薜令□.

	<	=	>	?	=	?	A
,	<,	=,	>,	?,	I \I	I \I	I \I
–	<–	=–	>–	?–	=–	?–	A–
.	<.	=.	>.	?.	I \I	I \I	I \I
/	</	=/	>/	?/	=/	?/	A/
1	I \I	I \I	I \I	I \I	=1	?1	A1
3	I \I	I \I	I \I	I \I	=3	?3	A3

横向堆叠

- 除了concat函数，join方法也可用于简单的横向堆叠，其基本语法格式如下。

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

横向堆叠

- join方法常用的参数及其说明，如下表所示。

参数名称	说明
other	接收DataFrame、Series或者包含了多个DataFrame的list。表示参与连接的其他DataFrame。无默认值
on	接收列名或者包含列名的list或tuple。表示用于连接的列名。默认为None
how	接收特定str。inner代表内连接；outer代表外连接；left和right分别代表左连接和右连接。默认为inner
lsuffix	接收sring。表示用于追加到左侧重叠列名的末尾。无默认值
rsuffix	接收str。表示用于追加到右侧重叠列名的末尾。无默认值
sort	根据连接键对合并后的数据进行排序，默认为True。

横向堆叠

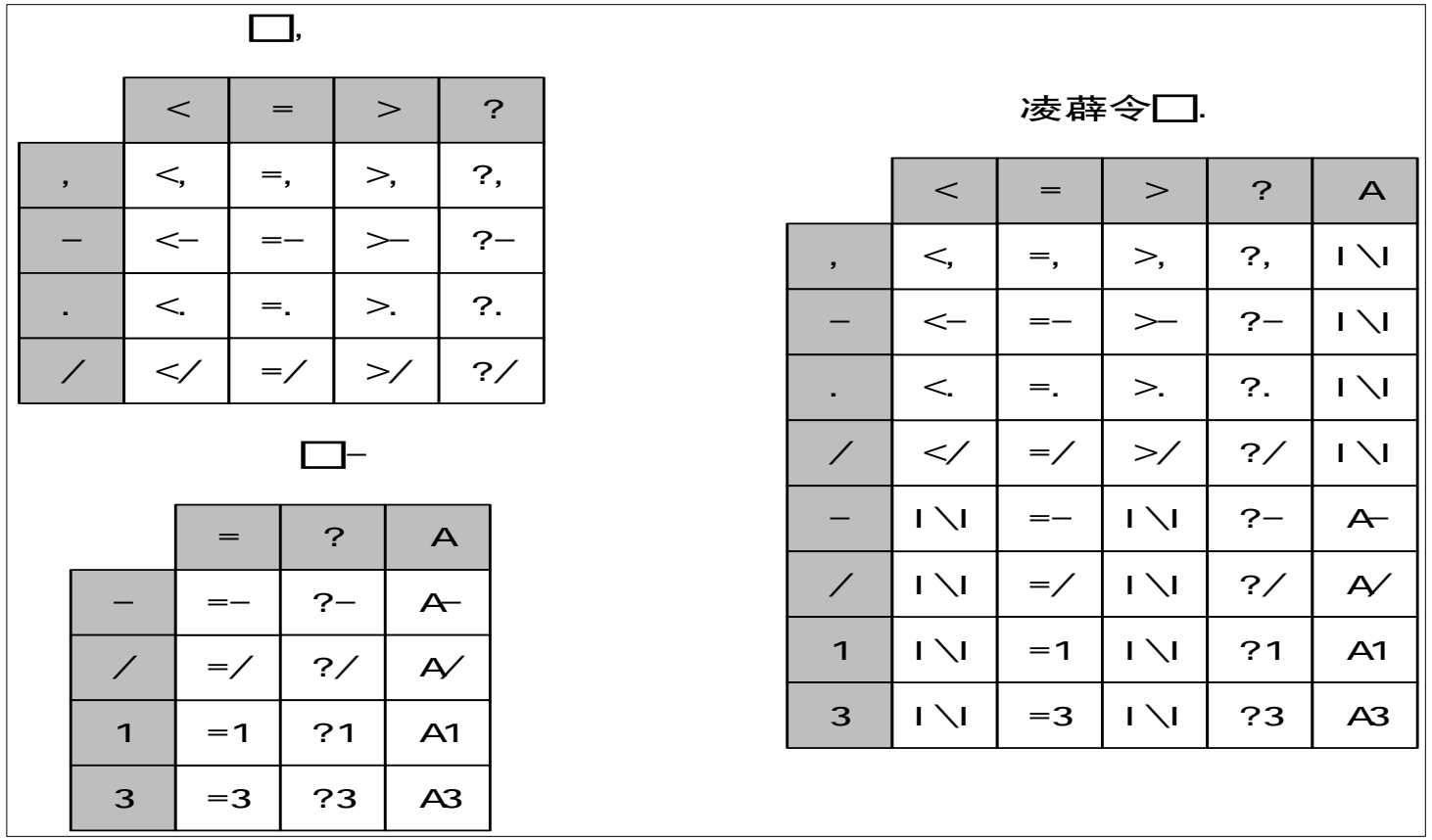
- 当横向堆叠的两个表的列名有相同时，需设置lsuffix或rsuffix参数以示区别，否则会报错。使用join方法横向堆叠，如下代码所示。

In[38]:	print('横向堆叠df2、df3后的DataFrame为：\n', df2.join(df3, rsuffix = '_2'))
Out[38]:	横向堆叠df2、df3后的DataFrame为： key A key_2 B 0 K0 A0 K0 B0 1 K1 A1 K1 B1 2 K2 A2 K2 B2 3 K3 A3 NaN NaN 4 K4 A4 NaN NaN 5 K5 A5 NaN NaN

纵向堆叠

- 与横向堆叠不同，纵向堆叠是将两个数据表在y轴向上拼接，concat函数可以实现纵向堆叠。使用concat函数时，在默认情况下，即axis=0时，concat做列对齐，将不同行索引的两张或多张表纵向合并，从而实现纵向堆叠。在两张表的列名并不完全相同的情况下，可join参数取值为inner时，返回的仅仅是列名交集所代表的列。取值为outer时，它是默认值，返回的是两者列名的并集所代表的列，其原理示意如图 6-2所示，图中表1和表2使用外连接纵向堆叠合并后，形成表3。

纵向堆叠



□,

	<	=	>	?
,	<,	=,	>,	?,
–	<–	=–	>–	?–
.	<.	=.	>.	?.
/	</	=/	>/	?/

□–

	=	?	A
–	=–	?–	A–
/	=/	?/	A/
1	=1	?1	A1
3	=3	?3	A3

凌薜令□.

	<	=	>	?	A
,	<,	=,	>,	?,	I \I
–	<–	=–	>–	?–	I \I
.	<.	=.	>.	?.	I \I
/	</	=/	>/	?/	I \I
–	I \I	=–	I \I	?–	A–
/	I \I	=/	I \I	?/	A/
1	I \I	=1	I \I	?1	A1
3	I \I	=3	I \I	?3	A3

纵向堆叠

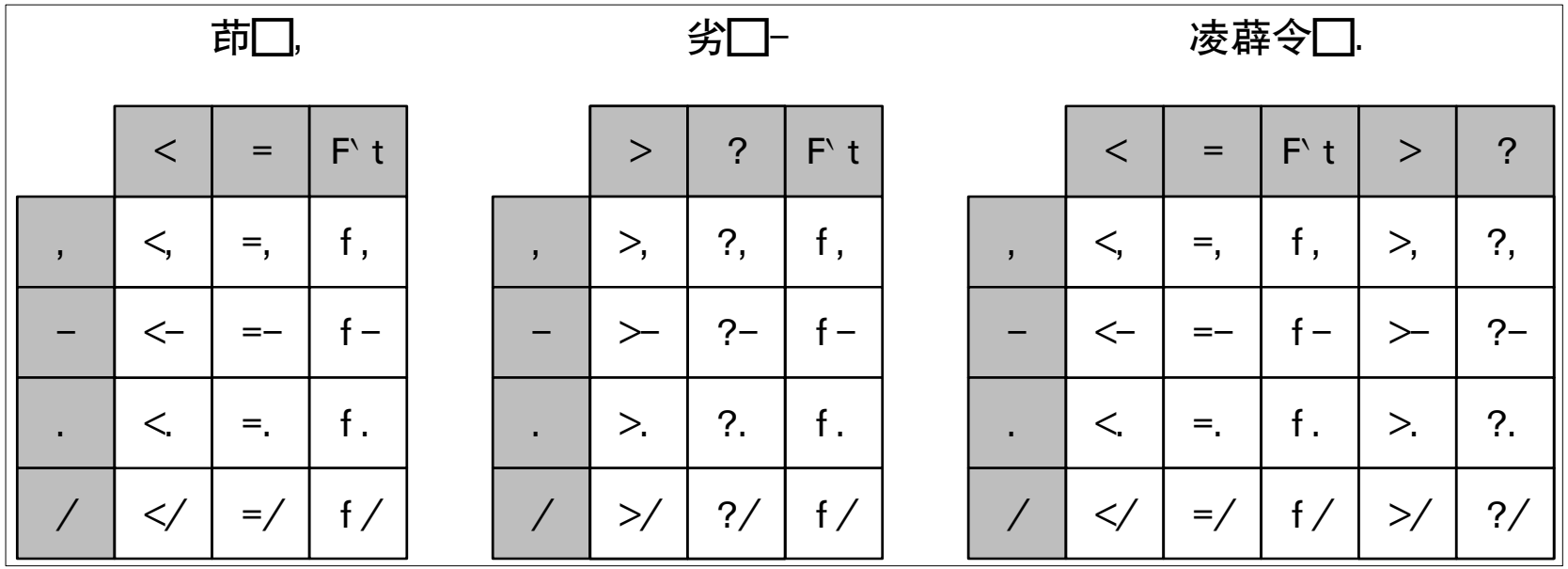
- 除了concat函数，append方法也可用于简单的纵向堆叠，这对列名完全相同的两张表特别有用，列名不同则会被空值替代，其基本语法如下。

```
pandas.DataFrame.append(self, other, ignore_index=False, verify_integrity=False)
```

- append方法常用的参数及其说明，如下表所示。

参数名称	说明
other	接收DataFrame或Series。表示要添加的新数据。无默认值
ignore_index	接收boolean。如果输入True，会对新生成的DataFrame使用新的索引（自动产生）而忽略原来数据的索引。默认为False
verify_integrity	接收boolean。如果输入True，那么当ignore_index为False时，会检查添加的数据索引是否冲突，如果冲突，则会添加失败。默认为False

- 主键合并，即通过一个或多个键将两个数据集的行连接起来，类似于SQL中的JOIN。针对同一个主键存在两张包含不同字段的表，将其根据某几个字段一一对应拼接起来，结果集列数为两个元数据的列数和减去连接键的数量，如下图所示，图中左表1和右表2通过列key合并起来，形成表3。



	茚□,		
	<	=	F‘ t
,	<,	=,	f ,
–	<–	=–	f –
.	<.	=.	f .
/	</	=/	f /

	劣□–		
	>	?	F‘ t
,	>,	?,	f ,
–	>–	?–	f –
.	>.	?.	f .
/	>/	?/	f /

	凌薜令□.				
	<	=	F‘ t	>	?
,	<,	=,	f ,	>,	?,
–	<–	=–	f –	>–	?–
.	<.	=.	f .	>.	?.
/	</	=/	f /	>/	?/

- pandas提供merge函数用于主键合并，其基本语法格式如下。

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

- 与SQL中的JOIN一样，merge函数也有左连接（left）、右连接（right）、内连接（inner）和外连接（outer），但比起SQL中的JOIN，merge函数还有其自身独到之处，例如可以在合并过程中对数据集中的数据进行排序等。merge函数常用的参数及其说明，如下表所示。

参数名称	说明
left	接收DataFrame或Series。表示要添加的新数据1。无默认值
right	接收DataFrame或Series。表示要添加的新数据2。无默认值
how	接收inner、outer、left、right。表示数据的连接方式。默认为inner
on	接收str或sequence。表示两个数据合并的主键（必须一致）。默认为None
left_on	接收str或sequence。表示left参数接收数据用于合并的主键。默认为None
right_on	接收str或sequence。表示 right 参数接收数据用于合并的主键。默认为None

参数名称	说明
left_index	接收boolean。表示是否将left参数接收数据的index作为连接主键。默认为False
right_index	接收boolean。表示是否将right参数接收数据的index作为连接主键。默认值False
sort	接收boolean。表示是否根据连接键对合并后的数据进行排序。默认为False
suffixes	接收tuple。表示用于追加到left和right参数接收数据重叠列名的尾缀。默认为('_x', '_y')

- 如日期或时间等时间类型数据在金融、经济等领域用途十分广泛。pandas提供多种时间类，最基本的是Timestamp、Timedelta和Period，它们是单个时间标量，由它们可以组成时间Series和时间索引。基本时间类的说明，如下表所示。

类	名称	说明	对应索引类
Timestamp	时间戳	具体时间点	DatetimeIndex
Timedelta	时间差	两个时间点的差	TimedeltaIndex
Period	时间段	时间段，如2018年全年	PeriodIndex

创建

- 时间戳指特定的时刻，可以理解为时间点。pandas中时间戳的类是Timestamp，它是Python基本库datetime的datetime类的替代品，在很多情况下可以互换。Timestamp类可以作为DatetimeIndex以及时间序列导向的数据结构的输入类型。使用Timestamp类可以创建Timestamp对象，其基本语法格式如下。

```
pandas.Timestamp(ts_input, freq=None, tz=None, unit=None, year=None, month=None, day=None, hour=None, minute=None, second=None, microsecond=None, tzinfo=None, offset=None)
```

创建

- Timestamp类常用的参数及其说明，如下表所示。

参数名称	说明
ts_input	接收datetime-like的str、int或float。表示被转换成Timestamp的值。无默认值
freq	接收str或DateOffset。表示时间频率。默认为None
tz	接收str、pytz.timezone或dateutil.tz.tzfile。表示时区。默认为None
unit	接收str。表示用于转换的NumPy时间单位。默认为None

创建

- 使用Timestamp类创建Timestamp对象，ts_input参数一般接收4个值，分别代表年、月、日、小时，与创建datetime.datetime类似。
- 需要注意的是，Timestamp类型时间是有限制的，最早只能够取值至1677年9月21日，最晚只能取值至2262年4月11日（根据当前日期发生改变）

创建

- 创建Timestamp对象的另一个方法是通过类型转换。很多情况下，需要特定的数据类型转换为Timestamp对象，pandas提供to_datetime函数能够实现这一目的。to_datetime函数的基本语法格式如下。

```
pandas.to_datetime(arg, errors='raise', dayfirst=False, yearfirst=False, utc=None, box=True, format=None, exact=True, unit=None, infer_datetime_format=False, origin='unix')
```

创建

- to_datetime函数常用的参数及其说明，如下表所示。

参数名称	说明
arg	接收integer、float、str、datetime、list、tuple、1-d array和Series。表示被转换成Timestamp的值。无默认值
dayfirst	接收bool。表示解析形式，如10/11/12会被解析成2012-11-10。默认为None
yearfirst	接收bool。表示解析形式，如10/11/12会被解析成2010-11-12。默认为None

属性

- Timestamp对象拥有一些常用的属性，如下表所示，这些属性有助于从Timestamp对象中提取出需要的信息。

属性名称	说明	属性名称	说明
year	年	quarter	季节
month	月	weekofyear	一年中第几周
day	日	dayofyear	一年中的第几天
hour	小时	dayofweek	一周第几天
minute	分钟	weekday_name	星期名称
second	秒	is_leap_year	是否闰年

属性

- 当Series由Timestamp对象组成时，即Series类型为datetime64时，可以使用Series的dt属性（pandas.Series.dt.）访问Timestamp的常用属性，如代下码所示。

In[12]:	**print('从Series提取的年为：\n', series1.dt.year)**
Out[12]:	从Series提取的年为： 0 2016.0 1 2017.0 2 2015.0 3 NaN 4 2016.0 dtype: float64
In[13]:	print('从Series提取的月为：\n', series1.dt.month)

属性

Out[13]:	**从Series提取的月为：** **0 8.0** **1 9.0** **2 7.0** **3 NaN** **4 8.0** **dtype: float64**
In[14]:	print('从Series提取的日为：\n', series1.dt.day)
Out[14]:	从Series提取的日为： 0 1.0 1 2.0 2 3.0 3 NaN 4 1.0 dtype: float64

创建

- 时间差即两个时间点的差。pandas中时间差的类是Timedelta，它是Python标准库datetime的timedelta类的子类，在很多情况下可以互换。Timedelta类可以创建Timedelta对象，其基本语法格式如下。

```
class pandas.Timedelta(value, unit=None, **kwargs)
```

- Timedelta类常用的参数及其说明，如下表所示。

参数	说明
value	接收Timedelta、timedelta、np.timedelta64、str或int。表示被转换成Timedelta的值。无默认值
unit	接收str。当接收值为int类型时，表示传入值的时间单位。默认为None

时间操作

2. 时间差

创建

- 通过转换的方式也可以创建Timedelta。pandas提供的to_timedelta函数能够将可识别为timedelta形式的scalar、array、list、或Series转换为Timedelta，其基本语法格式如下。

```
pandas.to_timedelta(arg, unit='ns', box=True, errors='raise')
```

- to_timedelta函数常用的参数及其说明，如下表所示。

参数名称	说明
arg	接收integer、float、str、datetime、list、tuple、1-d array和Series。表示被转换成Timedelta的值。无默认值
unit	接收str。当接收值为int类型时，表示传入值的时间单位。默认为ns

加减及转换频率

- 通过Timestamp对象的相减也可以得到Timedelta，如下代码所示。

In[18]:	**print('Timestamp相减后结果为：TimeDelta({0})'.format (pd.Timestamp(2018, 8, 15, 12) - pd.Timestamp(2018, 8, 14, 12)))**
Out[18]:	Timestamp相减后结果为：TimeDelta(1 days 00:00:00)

- Timedelta相加减后得到的也是Timedelta，如下表所示。

In[19]:	**print('Timedelta 相 减 后 为 ： TimeDelta({0})'. format(timedelta[1] - timedelta[0]))**
Out[19]:	Timedelta相减后为：TimeDelta(1 days 00:02:00)

加减及转换频率

- 此外，Timedelta或Timedelta组成的Series还能被转换成指定频率（时间单位）的数值，实现方法是用Timedelta除以另一个Timedelta对象（或NumPy的Timedelta对象），如下代码所示。

In[21]:	**# 转换为小时** **print('Timedelta 频率转换为小时后的数值为：\n', timedelta / pd.Timedelta('1 hour'))**
Out[21]:	Timedelta频率转换为小时后的数值为： 0 24.016667 1 48.050000 dtype: float64

In[22]:	# 转换为分钟
	print('Timedelta频率转换为分钟后的数值为：\n', timedelta / pd.Timedelta('1 minute'))

加减及转换频率

Out[22]:	Timedelta频率转换为分钟后的数值为： 0 1441.0 1 2883.0 dtype: float64

加减及转换频率

- 当Timedelta组成Series时，还可以使用astype方法转换时间频率，如下代码所示。

In[23]:	**# 转换为小时** **print('Timedelta频率转换为小时后的数值为：\n', timedelta.astype(('timedelta64[h]')))**
Out[23]:	Timedelta频率转换为小时后的数值为： 0 24.0 1 48.0 dtype: float64

In[24]:	# 转换为分钟
	print('Timedelta频率转换为分钟后的数值为：\n', timedelta.astype(('timedelta64[m]')))

Out[24]:	Timedelta频率转换为分钟后的数值为： 0 1441.0 1 2883.0 dtype: float64

属性

- Timedelta对象拥有一些常用的属性，如下表所示，这些属性有助于从Timedelta对象中提取出需要的信息。

属性名称	说明	属性名称	说明
days	天数	seconds	秒数
microseconds	毫秒数	nanoseconds	纳秒数

属性

- 当Series由Timedelta对象组成时，即Series类型为timedelta64，可以使用Series的dt属性（pandas.Series.dt.）访问Timedelta的常用属性，如下代码所示。

In[29]:	**print('从Series提取的天为：\n', timedelta.dt.days)**
Out[29]:	从Series提取的天为： 0 1 1 2 dtype: int64
In[30]:	print('从Series提取的秒为：\n', timedelta.dt.seconds)
Out[30]:	从Series提取的秒为： 0 60 1 180 dtype: int64

- 时间索引对象包括DatetimeIndex、PeriodIndex以及TimedeltaIndex，其中最基本的是DatetimeIndex。通过DatetimeIndex类创建DatetimeIndex对象，如下代码所示。

In[31]:	**timeindex = pd.DatetimeIndex(['2018-01-01', '2018-01-02', '2018-01-03', '2018-01-04'])** **print('创建的DatetimeIndex为：\n', timeindex)**
Out[31]:	创建的DatetimeIndex为： DatetimeIndex(['2018-01-01', '2018-01-02', '2018-01-03', '2018-01-04'], dtype='datetime64[ns]', freq=None)

data_range

- pandas更为推荐通过date_range函数创建，它能返回一个包含固定频率的DatetimeIndex，这在时间序列处理上十分有用，其基本语法格式如下。

```
pandas.date_range(start=None, end=None, periods=None, freq='D', tz=None, normalize=False, name=None, closed=None, **kwargs)
```

- date_range函数常用的参数及其说明，如下表所示。

参数名称	说明
start	接收日期形式的str。表示开始时间。默认为None
end	接收日期形式的str。表示截止时间。默认为None
periods	接收int。表示周期。默认为None
freq	接收str或DateOffset对象。表示时间频率，如D表示以天为频率。默认为D

时间序列索引操作

- DatetimeIndex一般与时间序列结合。以DatetimeIndex为索引的Series，是一个最基本的时间序列，如下代码所示。

In[35]:	**date = pd.date_range(start = '2018-01-10 01:02:03', periods = 5 , freq = 'W')** **list1 = [1, 2, 3, 4, 5]** **arr = pd.Series(list1, index = date)** **print('创建的时间序列为：\n', arr)**
Out[35]:	创建的时间序列为： 2018-01-14 01:02:03 1 2018-01-21 01:02:03 2 2018-01-28 01:02:03 3 2018-02-04 01:02:03 4 2018-02-11 01:02:03 5 Freq: W-SUN, dtype: int64

时间序列索引操作

- 时间序列是Series的子类，所以它们在基本索引操作是一样的，如下代码所示。

In[36]:	print('访问2018-01-14 01:02:03的数据为：', arr['2018-01-21 01:02:03'])
Out[36]:	访问2018-01-14 01:02:03的数据为： 2

- 时间序列的索引操作有其独特性，对于较长的时间序列，只需传入代表年、月等单位的字符串即可访问数据，如下代码所示。

In[37]:	print('访问2018年1月份数据为：\n', arr['2018-1'])
Out[37]:	访问2018年1月份数据为： 2018-01-14 01:02:03 1 2018-01-21 01:02:03 2 2018-01-28 01:02:03 3 Freq: W-SUN, dtype: int64

时间序列索引操作

- 由于大部分时间序列的数据都是按照先后顺序排列的，因此可以用不存在于DatetimeIndex中的时间（字符串）对其切片，这个切片可以理解为时间范围，切片不必工整，如下代码所示。

In[38]:	print('访问2017年12月到2018年2月3号的数据为：\n', arr['2017-12': '2018-02-03'])
Out[38]:	访问2017年12月到2018年2月3号的数据为： 2018-01-14 01:02:03 1 2018-01-21 01:02:03 2 2018-01-28 01:02:03 3 Freq: W-SUN, dtype: int64

- 通过str属性访问Series的文本方法，其基本语法格式如下。

pandas.Series.str.文本处理方法

- 访问upper方法，将Series各元素转换为大写，如下代码所示。

In[39]:	series3 = pd.Series(['a', 'abb', 'Ab12']) print('大写后的Series为：\n', series3.str.upper())
Out[39]:	大写后的Series为： 0 A 1 ABB 2 AB12 dtype: object

- Series的文本处理方法的名称大部分与Python内建的str数据类型的方法相同，作用也基本相同，但某些用法可能稍不同，例如replace方法、split方法等，它们通过Series.str访问时能够接收正则表达式，而通过普通字符串访问时则不能，如下代码所示。

In[40]:	**# 匹配以小写a开头的元素，将ab替换为F，作用于Series** **print('替换后的Series为：\n', series3.str.replace(r'^ab', 'F'))**
Out[40]:	替换后的Series为： 0 a 1 Fb 2 Ab12 dtype: object
In[41]:	print('替换后的元素为：\n', series3.str.replace(r'^ab', 'F')[1])
Out[41]:	替换后的元素为： Fb

In[42]:	# 匹配以小写a开头的元素，将ab替换为F，作用于str
	print('替换后的str为：', series3[1].replace(r'^ab', 'F'))

Out[42]:	替换后的str为： abb

- 部分文本处理方法与Python内建的str数据类型的方法不同，它们是特有的，如下表所示。

参数名称	说明	参数名称	说明
cat	实现元素级str连接操作，可指定分隔符	slice_replace	替换截取的str
get	抽取指定字符串位置的元素	findall	找到所有匹配模式所匹配的值
get_dummies	通过分割符分割str返回哑变量构成的DataFrame	match	根据指定的正则表达式对各元素执行re.match
contains	返回表示各str是否含有指定模式的bool型数组	extract	将正则表达式匹配的第一个组取出
repeat	对每个str重复指定次数	extractall	将正则表达式匹配的所有组取出
pad	在str的左边、右边或两边添加空白符	len	计算字符长度
wrap	按照指定长度分割字符	normalize	返回Unicode标准形式
slice	对Series中的各个str进行子串截取		

- 通过“[ ]”符号也可实现索引操作，它是基于位置的。若输入位置超过索引，则返回NaN，如下代码所示。

In[49]:	**# 位置索引** **print('第一个字符为：\n', series.str[0])**
Out[49]:	第一个字符为： 0 2 1 2 2 2 3 NaN 4 2 dtype: object

In[50]:	# 切片索引
	print('前两个字符为：\n', series3.str[0: 2])

Out[50]:	前两个字符为： 0 a 1 ab 2 Ab dtype: object

- 通过Series、DataFrame、Categorical类皆可创建category，其中使用Categorical类创建属于显式创建。本小节主要介绍通过Series创建，它最为典型，包含4个创建方法：指定Series数据类型，转换Series数据类型，接收Categorical对象，使用cut方法。
- 指定Series数据类型创建category的方法十分简单，设置dtype参数为category即可，如下代码所示。

In[51]:	series4 = pd.Series(['a', 'b', 'b', 'c'], dtype = 'category') print('指定Series数据类型创建的category为：\n', series4)
Out[51]:	指定Series数据类型创建的category为： 0 a 1 b 2 b 3 c dtype: category Categories (3, object): [a, b, c]

- 对于已经创建的Series，将其数据类型转换为category类型即可创建category。这里供转换的类型有两种：category，CategoricalDtype。CategoricalDtype是category的专用类型，它还可以锁定排序，如下代码所示。

In[52]:	**series = pd.Series(['a', 'b', 'b', 'c'])** **series1 = series.astype('category')** **print('转换Series数据类型创建的category为：\n', series1)**
Out[52]:	转换Series数据类型创建的category为： 0 a 1 b 2 b 3 c dtype: category Categories (3, object): [a, b, c]

In[53]:	**from pandas.api.types import CategoricalDtype** **cat_type = CategoricalDtype(categories=['b', 'c', 'd'], ordered = True)** **series1 = series.astype(cat_type)** **print('转换Series数据类型为CategoricalDtype创建的category结果为：\n', series1)**
Out[53]:	转换Series数据类型为CategoricalDtype创建的category结果为： 0 NaN 1 b 2 b 3 c dtype: category Categories (3, object): [b < c < d]

- Series通过接收Categorical对象也可以创建category，如下代码所示。

In[54]:	raw_cat = pd.Categorical(['a', 'b', 'b', 'c'], categories = ['a', 'b', 'c'], ordered = False) series = pd.Series(raw_cat) print('接收Categorical对象创建category为： \n', series)
Out[54]:	接收Categorical对象创建category为： 0 a 1 b 2 b 3 c dtype: category Categories (3, object): [a, b, c]

- cut函数将Series的连续数据分成不同的区间变成离散数据，一般用在直方图展示。cut函数作用后的Series数据类型变成了category，如下代码所示。

In[55]:	**series = pd.Series(range(9))** **series1 = pd.cut(series, [0, 3, 6, 9, 12], right = False)** **print('cut函数作用Series创建category结果为：\n', series1)**
Out[55]:	cut函数作用Series创建category结果为： 0 [0, 3) 1 [0, 3) 2 [0, 3) 3 [3, 6) 4 [3, 6) 5 [3, 6) 6 [6, 9) 7 [6, 9) 8 [6, 9) dtype: category Categories (4, interval[int64]): [[0, 3) < [3, 6) < [6, 9) < [9, 12)]

重命名

- category拥有一个类别属性和一个顺序属性，类别属性是category可能的值，顺序属性是指类别顺序是否指定。通过访问器访问相应属性，如下代码所示。

In[56]:	**series = pd.Series(['a', 'b', 'b', 'c'], dtype = 'category')** **print('category的类别为：\n', series.cat.categories)**
Out[56]:	category的类别为： Index(['a', 'b', 'c'], dtype='object')
In[57]:	print('category的类别是否指定：', series.cat.ordered)
Out[57]:	category的类别是否指定： False

- 使用rename_categories方法对类别重命名，如下代码所示。

In[58]:	**series1 = series.cat.rename_categories([1, 2, 3])** **print('重命名category的类别为：', series1.cat.categories)**
Out[58]:	重命名category的类别为： Int64Index([1, 2, 3], dtype='int64')

增删及设置类别

- 若要对category增加类别，则使用add_categories即可实现，如下代码所示。

In[59]:	series1 = series.cat.add_categories(['e']) print('新增后的category的类别为：\n', series1.cat.categories)
Out[59]:	新增后的category的类别为： Index(['a', 'b', 'c', 'e'], dtype='object')

- 移除类别和增加类别是对应的，移除类别使用的是remove_categories方法，本来对应类别的category变为NaN，如下代码所示。

In[60]:	series1 = series.cat.remove_categories(['c']) print('删除后的category的类别为：\n', series1.cat.categories)
Out[60]:	删除后的category的类别为： Index(['a', 'b'], dtype='object')

增删及设置类别

- 通过set_categories方法，可以一步到位实现类别增加或移除，或者改变类别值，甚至可以设置类别是否有序，如下代码所示。

In[61]:	series1 = series.cat.set_categories(['c', 'd'], ordered = True) print('设置后的category的类别为：\n', series1.cat.categories)
Out[61]:	设置后的category的类别为： Index(['c', 'd'], dtype='object')

排序

- 使用sort_values方法对category进行排序，如下代码所示。

In[62]:	series1 = series.cat.set_categories(['c', 'e', 'a', 'b'], ordered = True) print('指定顺序后的category为：\n', series1.sort_values())
Out[62]:	指定顺序后的category为： 3 c 0 a 1 b 2 b dtype: category Categories (4, object): [c < e < a < b]

读取文本文件（csv或者txt）

读取csv或者txt需要用到pandas模块中的pd.read_csv( )函数或者pd.read_table( )函数，其中pd.read_csv( )函数主要用来读取csv，而pd.read_table( )函数主要用来读取txt。

pd.read_csv( )函数基本语法格式为：pd.read_csv('文件.csv',sep=',')

其中的参数sep用于指定分隔符，一定要与拟读取的csv文件中实际的分隔符完全一致，如不特别设置则默认为英文状态下的逗号。

pd.read_table( )函数基本语法格式为： pd.read_table('文件.txt',sep='\t')

其中的参数sep用于指定分隔符，一定要与拟读取的txt文件中实际的分隔符完全一致，如不特别设置则默认为水平制表符“\t”

示例

参阅教材内容

- CSV是一种字符分隔文件，文件以纯文本形式存储表格数据（数字和文本）。它是一种通用、相对简单的文件格式，最广泛的应用是在程序之间转移表格数据，而这些程序本身是在不兼容的格式上进行操作的（往往是私有的和 / 或无规范的格式）。因为大量程序都支持CSV或者其变体，可以作为大多数程序的输入和输出格式。
- read_csv函数用于读取CSV文件，其基本语法格式如下。

```
pandas.read_csv(filepath_or_buffer, sep=', ', delimiter=None, header='infer', names=None, index_col=None, usecols=None, squeeze=False, prefix=None, mangle_dupe_cols=True, dtype=None, engine=None, converters=None, true_values=None, false_values=None, skipinitialspace=False, skiprows=None, nrows=None, na_values=None, keep_default_na=True, na_filter=True, verbose=False, skip_blank_lines=True, parse_dates=False, infer_datetime_format=False, keep_date_col=False, date_parser=None, dayfirst=False, iterator=False, chunksize=None, compression='infer', thousands=None, decimal=b'.', lineterminator=None, quotechar='"', quoting=0, escapechar=None, comment=None, encoding=None, dialect=None, tupleize_cols=None, error_bad_lines=True, warn_bad_lines=True, skipfooter=0, doublequote=True, delim_whitespace=False, low_memory=True, memory_map=False, float_precision=None)
```

- read_csv函数常用的参数及其说明，如下表所示。

参数名称	说明
filepath	接收str。表示文件路径。无默认值
sep	接收str。表示文件的分隔符。默认为“,”
header	接收int或sequence。表示将某行数据作为列名，为int时表示将第n列作为列名；为sequence时，将sequence作为列名。默认为infer，表示自动识别
names	接收array。表示列名。默认为None
index_col	接收int、sequence或False。表示索引列的位置，为sequence时表示多重索引。默认为None
dtype	接收dict。表示写入的数据类型（列名为key，数据格式为values）。默认为None
engine	接收c或者python。表示数据解析引擎。默认为c
nrows	接收int。表示读取前n行。默认为None
encoding	接收str。表示文件的编码格式。无默认值

- read_csv函数中的sep参数是指定文本的分隔符的，如果分隔符指定错误，在读取数据的时候，每一行数据将连成一片。Encoding参数表示文件的编码格式，常用的编码有utf-8、utf-16、gbk、gb2312、gb18030等，如果编码指定错误，那么数据将无法读取。
- 除CSV外，read_csv函数还能读取其他文本文件，只需改变encoding、sep等对应参数即可。

- 文本文件的存储与读取类似，结构化数据可以通过pandas中的to_csv方法实现以CSV文件格式存储文件，Series和DatFrame数据都可以写入CSV文件，DatFrame的to_csv方法的基本语法格式如下。

```
DataFrame.to_csv(path_or_buf=None, sep=',', na_rep='', columns=None, header=True, index=True, index_label=None, mode='w', encoding=None)
```

- to_csv方法常用的参数及其说明，如下表所示。

参数名称	说明
path_or_buf	接收str。表示文件路径。无默认值
sep	接收str。表示分隔符。默认为“,”
na_rep	接收str。表示缺失值。默认为“”
columns	接收list。表示写出的列名。默认为None
header	接收bool，表示是否将列名写出。默认为True
index	接收bool，表示是否将行名（索引）写出。默认为True
index_labels	接收sequence。表示索引名。默认为None
mode	接收特定str。表示数据写入模式。默认为w
encoding	接收特定str。表示存储文件的编码格式。默认为None

读取excel数据

很多情况下，我们需要调用excel数据，可以直接调用默认的sheet表，也可以特别指定需要调用的sheet表，还可以在确定sheet表后、单独调用其中的部分列而不是载入整个sheet表。

示例

参阅教材内容

Excel是微软公司的办公软件Microsoft Office的组件之一，它可以进行各种数据的处理、统计分析和辅助决策操作，广泛地应用于管理、统计财经和金融等众多领域。其文件保存依照程序版本的不同分为两种。

- Microsoft Office Excel 2007之前的版本（不包括2007）默认保存的文件名后缀为.xls。
- Microsoft Office Excel 2007之后的版本默认保存的文件名后缀为.xlsx。

pandas提供了read_excel函数来读取“xls”“xlsx”两种Excel文件，其基本语法格式如下。

```
pandas.read_excel(io, sheet_name=0, header=0, names=None, index_col=None,
usecols=None, squeeze=False, dtype=None, engine=None, converters=None,
true_values=None, false_values=None, skiprows=None, nrows=None, na_values=None,
parse_dates=False, date_parser=None, thousands=None, comment=None,
skipfooter=0, convert_float=True, **kwds)
```

- read_excel函数常用的参数及其说明，如下表所示，部分与read_csv函数相同。

参数名称	说明
io	接收str。表示文件路径。无默认值
sheetname	接收str或int。表示excel表内数据的分表位置。默认为0
header	接收int或sequence。表示将某行数据作为列名，为int时表示将第n列作为列名；为sequence时，将sequence作为列名。默认为infer，表示自动识别
names	接收array。表示列名。默认为None
index_col	接收int、sequence或者False。表示。默认为None
dtype	接收dict。表示写入的数据类型（列名为key，数据格式为values）。默认为None

- pandas提供to_excel方法用于将DataFrame写出到Excel文件，其基本语法格式如下。

```
DataFrame.to_excel(excel_writer, sheet_name='Sheet1', na_rep='',
float_format=None, columns=None, header=True, index=True, index_label=None,
startrow=0, startcol=0, engine=None, merge_cells=True, encoding=None,
inf_rep='inf', verbose=True, freeze_panes=None)
```

- to_excel方法和to_csv方法的常用参数基本一致，区别之处在于指定存储文件的文件路径参数名称为excel_writer，并且没有sep参数，增加了一个sheetnames参数用来指定存储的Excel sheet的名称，默认为sheet1。

- 在生产环境中，绝大多数的数据都存储在数据库中。pandas提供了读取与存储关系型数据库数据的函数与方法。除了pandas库外，还需要使用SQLAlchemy库建立对应的数据库连接。SQLAlchemy配合相应数据库的Python连接工具（例如MySQL数据库需要安装mysqlclient或者pymysql库，Oracle数据库需要安装cx_oracle库），使用create_engine函数，建立一个数据库连接。支持MySQL、postgresql、Oracle、SQLServer和SQLite等主流数据库。本小节以MySQL数据库为例，介绍使用pandas库数读取与存储据库的数据。
- pandas实现数据库数据读取有3个函数：read_sql、read_sql_table和read_sql_query。其中，read_sql_table函数只能够读取数据库的某一个表格，不能实现查询的操作；read_sql_query函数只能实现查询操作，不能直接读取数据库中的某个表；read_sql函数是两者的综合，既能够读取数据库中的某一个表，也能够实现查询操作，其基本语法格式如下。

```
pandas.read_sql(sql, con, index_col=None, coerce_float=True, params=None, parse_dates=None, columns=None, chunksize=None)
```

- pandas的3个数据库数据读取函数的参数几乎完全一致，唯一的区别在于传入的是语句还是表名。read_sql函数常用的参数及其说明，如下表所示。

参数名称	说明
sql	接收str。表示读取的数据的表名或者sql语句。无默认值
con	接收数据库连接。表示数据库连接信息。无默认值
index_col	接收int，sequence或者False。表示设定的列作为行名，如果是一个数列则是多重索引。默认为None
coerce_float	接收boolean。将数据库中的decimal类型的数据转换为pandas中的float64类型的数据。默认为True
columns	接收list。表示读取数据的列名。默认为None

- read_sql函数的con参数是一个数据库连接，需要提前建立完成才能够正常读取数据库数据。使用SQLAlchemy建立mysql数据库连接，如下代码所示，注意需要在MySQL中先创建数据库test_db，然后创建数据表orders。

In[5]:	from sqlalchemy import create_engine
	# 创建一个mysql连接器，用户名为root，密码为12345
	# 地址为127.0.0.1，数据库名称为test_db，编码为utf-8
	# 创建数据表orders
	engine = create_engine(
	'mysql+pymysql:'
	'//root:12345@127.0.0.1:3306/test_db?charset=utf8mb4')
	print(engine)

Out[5]:	Engine(mysql+pymysql://root:***@127.0.0.1:3306/test_db?charset=utf8mb4)

- creat_engine函数中填入的是一个连接字符串。在使用Python的SQLAlchemy时，MySQL和Oracle数据库连接字符串的格式如下。

数据库产品名+连接工具名：//用户名:密码@数据库IP地址:数据库端口号/数据库名称?

charset = 数据库数据编码

- 将pandas的数据写入数据库同样需要依赖SQLAlchemy的数据库连接。to_sql方法用于将数据写入数据库，其基本语法格式如下。

```
DataFrame.to_sql(name, con, schema=None, if_exists='fail', index=True,
index_label=None, chunksize=None, dtype=None)
```

- to_sql方法常用的参数及其说明，如下表所示。

参数名称	说明
name	接收str。表示数据库表名。无默认值
con	接收数据库连接。无默认值
if_exists	接收fail，replace，append。fail表示如果表名存在则不执行写入操作；replace表示如果存在，将原数据库表删除，再重新创建；append则表示在原数据库表的基础上追加数据。默认为fail
index	接收bool。表示是否将行索引作为数据传入数据库。默认True。
index_label	接收str或者sequence。表示是否引用索引名称，若index参数为True，且此参数为None时，则使用默认名称。若为多重索引，则必须使用数列形式。默认为None
dtype	接收dict。表示写入的数据类型（列名为key，数据格式为values）。默认为None

读取spss数据

很多情况下，我们需要调用SPSS软件产生的数据，下面我们以示例的方式进行讲解。

示例

参阅教材内容

读取stata数据

很多情况下，我们需要调用stata软件产生的数据，下面我们以示例的方式进行讲解。

示例

参阅教材内容

python数据检索

在机器学习和数据统计分析中，我们在正式使用相关的算法或方法之前，往往需要对数据进行观察。

示例

参阅教材内容

- 描述性统计是用来概括、表述事物整体状况以及事物间关联、类属关系的统计方法，通过几个统计值简洁的表示一组数据的集中趋势和离散程度。在第5章介绍的NumPy库提供了大量统计学函数，用于处理数值型特征数据，部分函数如下表所示。

函数名称	说明	函数名称	说明
np.min	最小值	np.max	最大值
np.mean	均值	np.ptp	极差
np.median	中位数	np.std	标准差
np.var	方差	np.cov	协方差

- 此外，pandas也提供了专门的描述性统计方法，如下表所示。

方法名称	说明	方法名称	说明
min	最小值	max	最大值
mean	均值	median	中位数
std	标准差	var	方差
cov	协方差	sem	标准误差
mode	众数	skew	样本偏度
kurt	样本峰度	quantile	四分位数
count	非空值数目	mad	平均绝对离差

- 对于类别型数据，即category，也可以通过describe方法进行描述性统计，它返回4种特征，如下表所示。

特征	说明	特征	说明
count	类别计数	unique	不重复类别个数
top	个数最多的类别	freq	个数最多的类别的数量

- 除describe方法外，还有info方法，也可用于描述性统计，能够对数据的类型，索引，内存信息有一个直观的了解，如下代码所示。

In[14]:	print('DataFrame的info信息为：\n') df.info()
Out[14]:	DataFrame的info信息为： <class 'pandas.core.frame.DataFrame'> RangeIndex: 1030 entries, 0 to 1029 Data columns (total 4 columns): Cement 1030 non-null float64 Blast Furnace Slag 1030 non-null float64 Fly Ash 1030 non-null float64 Water 1030 non-null float64 dtypes: float64(4) memory usage: 32.3 KB

- 为了提升数据的准确性，将某个点的取值扩大到包含这个点的一段区间，用区间来进行判断，这个区间就是窗口。移动窗口就是窗口向一端滑行，每次滑行并不是区间整块的滑行，而是一个单位一个单位的滑行。移动窗口在处理时间序列相关数据时特别有用，可与描述性统计方法相结合。pandas提供的rolling方法可用于对DataFrame进行移动窗口操作，其基本语法格式如下。

```
DataFrame.rolling(window, min_periods=None, freq=None, center=False, win_type=None, on=None, axis=0, closed=None)
```

统计分析

2. 移动窗口rolling方法

- rolling方法常用的参数及其说明，如下表所示。

参数名称	说明
window	接收int或offset。表示移动窗口的大小，当接收offset时，它是每个窗口的时间段，此时的索引必须为时间类型。无默认值
min_periods	接收int。表示窗口中需要有值的观测点数量的最小值。默认为None
center	接收bool。表示窗口中间设置标签。默认为Fasle
win_type	接收str。表示窗口类型。默认为None
on	接收str。表示对于DataFrame做移动窗口计算的列。无默认值
axis	接收int。表示作用的轴方向。默认为0
closed	接收str。表示区间的开闭。无默认值

- 使用rolling方法对DataFrame进行移动窗口操作，一般设置window参数即可，它返回窗口对象，如下代码所示。

In[14]:	**arr = np.array([[2,2,2], [4,4,4], [6,6,6], [8,8,8], [10,10,10]])** **df2 = pd.DataFrame(arr, columns = ['one', 'two', 'three'],** **index = pd.date_range('1/1/2018', periods = 5))** **print('创建的移动窗口对象为：', df2.rolling(2))**
Out[14]:	创建的移动窗口对象为： Rolling [window=2,center=False,axis=0]

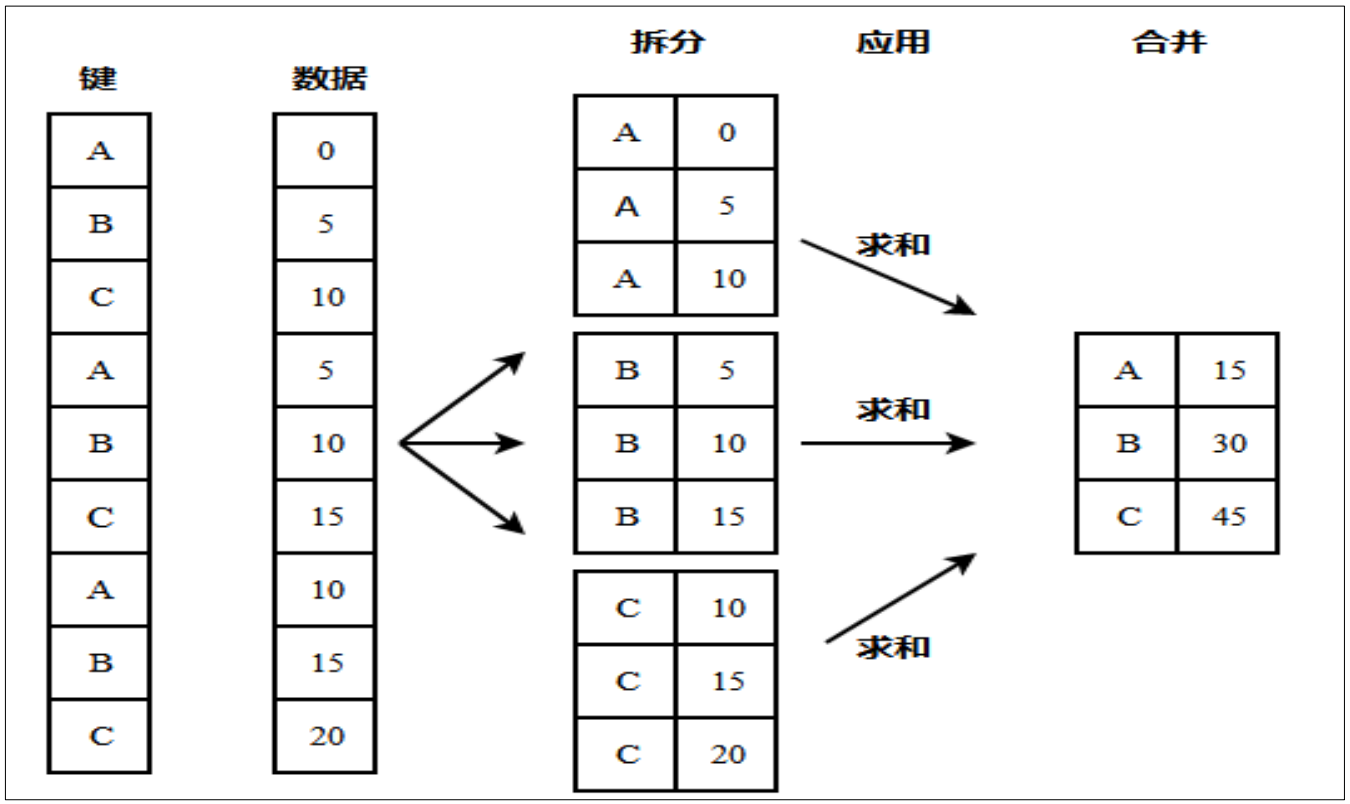
- 返回的窗口对象一般与统计描述方法结合使用，如sum、mean、max等，结果返回原来的数据对象，如下代码所示。

In[16]:	print('移动窗口为2，使用mean方法计算后结果为：\n', df2.rolling(2).mean())
Out[16]:	移动窗口为2，使用mean方法计算后结果为： one two three 2018-01-01 NaN NaN NaN 2018-01-02 3.0 3.0 3.0 2018-01-03 5.0 5.0 5.0 2018-01-04 7.0 7.0 7.0 2018-01-05 9.0 9.0 9.0
In[17]:	print('移动窗口为2，使用sum方法计算后结果为：\n', df2.rolling(2).sum())
Out[17]:	移动窗口为2，使用sum方法计算后结果为： one two three 2018-01-01 NaN NaN NaN 2018-01-02 6.0 6.0 6.0 2018-01-03 10.0 10.0 10.0 2018-01-04 14.0 14.0 14.0 2018-01-05 18.0 18.0 18.0

- window参数除了接收int，还可以接收offset，它是时间偏移对象，通常用于时间序列。与接收int相比，offset更为灵活，且计算结果无空值，空值会被第一个或最后一个数据替代，如下代码所示。

In[18]:	**print('移动窗口为2天，使用mean方法计算后结果为：\n', df2.rolling('2D').mean())**
Out[18]:	移动窗口为2天，使用mean方法计算后结果为： one two three 2018-01-01 2.0 2.0 2.0 2018-01-02 3.0 3.0 3.0 2018-01-03 5.0 5.0 5.0 2018-01-04 7.0 7.0 7.0 2018-01-05 9.0 9.0 9.0
In[19]:	print('移动窗口为2天，使用sum方法计算后结果为：\n', df2.rolling('2D').sum())
Out[19]:	移动窗口为2天，使用sum方法计算后结果为： one two three 2018-01-01 2.0 2.0 2.0 2018-01-02 6.0 6.0 6.0 2018-01-03 10.0 10.0 10.0 2018-01-04 14.0 14.0 14.0 2018-01-05 18.0 18.0 18.0

- 分组运算是依据某个或者某几个字段对数据集进行分组，并对各组应用函数，从而得到特定结果。分组聚合是分组运算的一种，其原理如下图所示。



groupby方法

- GroupBy对象提供分组运算步骤中的拆分功能，groupby方法用于创建分组对象GroupBy，其基本语法格式如下。

```
DataFrame.groupby(by=None, axis=0, level=None, as_index=True, sort=True, group_keys=True, squeeze=False, **kwargs)
```

groupby方法

- groupby方法常用的参数及其说明，如下表所示。

参数名称	说明
by	接收list，str，mapping或generator。用于确定进行分组的键值。如果传入的是一个函数则对索引进行计算并分组，如果传入的是一个字典或者Series则字典或者Series的值用来做分组依据。如果传入一个NumPy数组则数据的元素作为分组依据，如果传入的是字符串或者字符串列表则使用这些字符串所代表的字段作为分组依据。无默认值
axis	接收int。表示操作的轴向，默认对列进行操作。默认为0
level	接收int或者索引名。表示标签所在级别。默认为None
as_index	接收bool。表示聚合后的聚合标签是否以DataFrame索引形式输出。默认为True
sort	接收bool。表示是否对分组依据分组标签进行排序。默认为True
group_keys	接收bool。表示是否显示分组标签的名称。默认为True
squeeze	接收bool。表示是否在允许的情况下对返回数据进行降维。默认为False

分组聚合

- 从分组产生标量值（即单个数值）的数据转换过程被称为分组聚合，GroupBy对象结合描述统计方法从各个分组中产生标量值，这个标量值可以是平均值、数量、中位数等。GroupBy对象常用的描述性统计方法，如表所示。

方法	说明	方法	说明
count	计算分组的数目，包括缺失值	median	返回每组的中位数
head	返回每组的前n个值	cumcount	对每个分组中组员的进行标记，0至n-1
max	返回每组最大值	size	返回每组的大小
mean	返回每组的均值	min	返回每组最小值
sum	返回每组的和	std	返回每组的标准差

agg

- agg方法是一个既能作用于Series 、 DataFrame，也能作用于GroupBy的据合方法。agg方法接收函数并应用于每个分组，返回标量值，其基本语法格式如下。

```
GroupBy.agg(func, *args, **kwargs)
```

agg

- agg方法的常用参数是func，它接收函数，表示作用于分组的函数，无默认值。
- func参数可以接收NumPy库提供的函数，如下代码所示。

```
In[23]: print('分组的均值前5行结果为：\n', group.agg(np.mean).head())
Out[23]: 分组的均值前5行结果为：
              on_man    off_man
station
ST001      38.327982  34.410550
ST002     130.263761  10.126147
ST003      57.839450  26.869266
ST004     307.619377  18.121107
ST005      75.855505  22.520642
```

agg

- func参数同时也可以接收自定义函数，但函数返回的结果需为单个标量值，如下代码所示。

In[24]:	**def f(x):** ** return x.max() - x.min()** **group1 = group.agg(f)** **print('分组的极差前5行结果为：\n', group1.head())**
Out[24]:	分组的极差前5行结果为： on_man off_man station ST001 101.0 185 ST002 513.0 81 ST003 179.0 94 ST004 781.0 134 ST005 239.0 93

agg

- agg方法可同时接收多个函数，如下代码所示。

In[25]:

```
group2 = group.agg([np.mean, np.sum])
print('分组的均值和总和前5行结果为：\n', group2.head())
```

Out[25]:

分组的均值和总和前5行结果为：

	on_man		off_man	
	mean	sum	mean	sum
station				
ST001	38.327982	16711.0	34.410550	15003
ST002	130.263761	56795.0	10.126147	4415
ST003	57.839450	25218.0	26.869266	11715
ST004	307.619377	88902.0	18.121107	5237
ST005	75.855505	33073.0	22.520642	9819

agg

- 同时，agg方法可对不同的列指定不同的函数，如下代码所示。

In[26]:	group3 = group.agg({'on_man': np.mean, 'off_man': np.sum}) print('列on_man应用均值函数，列off_man应用汇总函数前5行结果为：\n', group3.head())
Out[26]:	列on_man应用均值函数，列off_man应用汇总函数前5行结果为： on_man off_man station ST001 38.327982 15003 ST002 130.263761 4415 ST003 57.839450 11715 ST004 307.619377 5237 ST005 75.855505 9819

apply

- apply方法是一个既能接收返回标量值的函数，又能接收返回数组的函数的聚合方法。相比之下，agg方法仅能接收返回标量值的函数。apply方法的基本语法格式如下。

```
GroupBy.apply(func, *args, **kwargs)
```

apply

- apply方法的常用参数是func，它接收函数，表示作用于分组的函数，无默认值。
- 当apply方法func参数接收NumPy函数，如下代码所示。

```
In[27]: print('分组的均值前5行结果为：\n', group.apply(np.mean).head())
Out[27]: 分组的均值前5行结果为：
              on_man    off_man
station
ST001     38.327982  34.410550
ST002    130.263761  10.126147
ST003     57.839450  26.869266
ST004    307.619377  18.121107
ST005     75.855505  22.520642
```

apply

- apply方法的func参数也可接收自定义函数，如下代码所示。

In[28]:
```
def f(x):
    result = x[0: 2]
    return result
print('分组的前两个数据前5行结果为：\n', group.apply(f).head())
```

Out[28]:
```
分组的前两个数据前5行结果为：
            station  on_man  off_man  train
station
ST001   735    ST001    40.0       21  PK07
        1853   ST001    32.0       18  PK07
ST002   326    ST002   112.0       11  PK04
        1181   ST002   318.0       16  PK04
ST003   98     ST003    95.0       37  PK02
```

transform

- transform方法返回的对象与被分组对象的形状相同（除去为分组键的列），这与agg、apply方法不同。transform方法的基本语法格式如下。

```
GroupBy.transform(func, *args, **kwargs)
```

transform

- transform方法的常用参数是func，表示接收的作用于分组的函数，无默认值。
- 需要注意的是，func参数接收的函数返回的结果若为标量值，则会广播到整个分组上。
- transform方法的func参数接收返回标量值的函数作用于GroupBy分组对象，如下代码所示。

In[29]:	print('对分组应用均值函数，返回的DataFrame前5行数据为：\n', group.transform(np.mean).head())
Out[29]:	对分组应用均值函数，返回的DataFrame前5行数据为： on_man off_man 0 1225.300459 11.410550 1 100.594037 109.727064 2 136.396789 66.511468 3 180.149083 43.433486 4 445.440367 226.125382

transform

- 相比apply方法，transform方法接收的函数若返回数组结果，则结果必须和分组对象形状相同，如下代码 所示。

In[30]:

```
def f(x):
    result = x*2
    return result
print('对分组的每个元组乘以2，返回的DataFrame前5行数据为：\n',
      group.transform(f).head())
```

Out[30]:

对分组的每个元组乘以2，返回的DataFrame前5行数据为：

	on_man	off_man	train
0	1782.0	0	PK11 PK11
1	138.0	322	PK11 PK11
2	300.0	80	PK11 PK11
3	144.0	50	PK11 PK11
4	864.0	712	PK11 PK11

- 透视表与分组聚合类似，不同的是分组聚合只能指定一个轴做分组键，而透视表可以同时指定两个轴做分组键。pivot_table函数用于创建透视表，其基本语法格式如下。

```
pands.pivot_table(data, values=None, index=None, columns=None, aggfunc='mean',
fill_value=None, margins=False, dropna=True, margins_name='All')
```

- pivot_table函数常用的参数及其说明，如下表示。

参数名称	说明
data	接收DataFrame。表示透视表的数据。无默认值
values	接收字符串。用于指定想要聚合的数据字段名，默认使用全部数据。默认为None
index	接收str或list。表示行分组键。默认为None
columns	接收str或list。表示列分组键。默认为None
aggfunc	接收functions。表示聚合函数。默认为mean
margins	接收bool。表示汇总（Total）功能的开关，设为True后结果集中会出现名为“ALL”的行和列。默认为True
dropna	接收bool。表示是否删掉全为NaN的列。默认为False

- 参数aggfunc因为默认为mean，所以它会自动过滤掉非数值类型数据。也可以自定义设置聚合函数，如下代码所示。

In[32]:	tdf = pd.pivot_table(df, index=['four'], columns=['one'], aggfunc = np.sum) print('分组和的透视表为：\n', tdf)
Out[32]:	分组和的透视表为： three two one a b a b four x 5 6 0 1 y 9 15 4 5

- 交叉表是一种特殊的透视表，主要用于计算分组频率，功能与透视表类似。crosstab函数用于制作交叉表，其基本语法格式如下。

```
pandas.crosstab(index, columns, values=None, rownames=None, colnames=None, aggfunc=None, margins=False, dropna=True, normalize=False)
```

- pivot_table函数常用的参数及其说明，如下表示。

参数名称	说明
index	接收str或list。表示行索引键。无默认值
columns	接收str或list。表示列索引键。无默认值
values	接收array。表示聚合数据。默认为None
aggfunc	接收function。表示聚合函数。默认为None
rownames	表示行分组键名。无默认值
colnames	表示列分组键名。无默认值
dropna	接收bool。表示是否删掉全为NaN的。默认为False
margins	接收bool。默认为True。汇总（Total）功能的开关，设为True后结果集中会出现名为“ALL”的行和列
normalize	接收bool。表示是否对值进行标准化。默认为False

- 由于交叉表是透视表的一种，其参数基本保持一致，不同之处在于crosstab函数中的index、columns、values参数填入的都是对应的从Dataframe中取出的某一列。当不指定aggfunc参数时，分组返回的是频数。使用crosstab函数制作交叉表，如下代码所示。

In[33]:	cdf = pd.crosstab(index = df['four'], columns = df['one']) print('创建的交叉表为：\n', cdf)
Out[33]:	创建的交叉表为： one a b four x 1 1 y 1 2

- 由于交叉表是一种特殊的透视表，使用pivot_table函数也可以创建与上一页代码 结果相同的交叉表，如下代码所示。

In[34]:	**cdf = pd.pivot_table(df, values = 'two', index = ['four'], columns = ['one'],** **aggfunc = (lambda x: len(x)))** **print('使用pivot_table函数创建的交叉表为：\n', cdf)**
Out[34]:	使用pivot_table函数创建的交叉表为： one a b four x 1 1 y 1 2

python数据缺失值处理

在很多时候，受数据收集整理环节中种种因素的影响，数据集中会含有缺失值，缺失值在一定程度上会影响数据的处理效能，我们需要针对缺失值进行处理。缺失值的处理一般有以下方法：一是直接忽略掉缺失值，很多算法并不会因为缺失值的存在而导致效能的显著下降；二是将缺失值所在行（样本示例）或者列（变量）做删除处理，以确保进入机器学习或统计分析的所有样本示例或变量没有缺失值；三是对缺失值进行补充或者估算。下面我们针对相应的操作代码进行一一讲解。

- 数据中的某个或某些特征的值是不完整的，这些值称为缺失值。对缺失值处理前需先识别缺失值，pandas提供isnull方法，能够识别出缺失值，返回bool。isnull方法结合其他操作，找出缺失值的数量及占比，如下代码所示。

In[1]:	**import pandas as pd** **dit = {'col1': [0, 1, 2, None, 4], 'col2': [5, None, 6, 7, None]}** **df = pd.DataFrame(dit)** **print('缺失值数量为：\n', df.isnull().sum())**
Out[1]:	缺失值数量为： col1　1 col2　2 dtype: int64
In[2]:	print('缺失值占比为：\n', df.isnull().sum() / len(df))
Out[2]:	缺失值占比为： col1　0.2 col2　0.4 dtype: float64

- 删除法是指将含有缺失值的特征或者记录删除。删除法分为删除观测记录和删除特征两种，观测记录指删除行，特征指删除列，它属于利用减少样本量来换取信息完整度的一种方法，是一种最简单的缺失值处理方法。pandas提供简便的删除缺失值的方法dropna，通过参数控制，该方法既可以删除观测记录，亦可以删除特征，其基本语法格式如下。

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

- dropna方法常用的参数及其说明，如下表示。

参数名称	说明
axis	接收0或1。表示轴向，0为删除观测记录（行），1为删除特征（列）。默认为0
how	接收特定str。表示删除的形式。any表示只要有缺失值存在就执行删除操作。all表示当且仅当全部为缺失值时执行删除操作。默认为any
subset	接收类array数据。表示进行删除缺失值的列。默认为None，表示所有列
inplace	接收bool。表示是否在原表上进行操作。默认为False

- 替换法是指用一个特定的值替换缺失值。特征可分为数值型和类别型，两者出现缺失值时的处理方法也是不同的。缺失值所在特征为数值型时，通常利用其均值、中位数和众数等描述其集中趋势的统计量来代替缺失值；缺失值所在特征为类别型时，则选择使用众数来替换缺失值。fillna方法用于替换缺失值，其基本语法格式如下。

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

- fillna方法常用的参数及其说明，如下表所示。

参数名称	说明
value	接收数字，dict，Series或者DataFrame。表示用来替换缺失值的值。无默认值
method	接收特定str。表示确实值填充的方法，当value参数未填写时起效。取值为“backfill”或“bfill”时，表示使用下一个非缺失值填补缺失值；取值为“pad”或“ffill”时，表示使用上一个非缺失值填补缺失值。默认为None
axis	接收0或1。表示轴向。默认为1
inplace	接收bool。表示是否在原表上进行操作。默认为False
limit	接收int。表示填补缺失值个数上限，超过则不进行填补。默认为None

- 删除法简单易行，但是会引起数据结构变动，样本减少；替换法使用难度较低，但是会影响数据的标准差，导致信息量变动。在面对数据缺失问题时，除了这两种方法之外，还有一种常用的方法—插值法。
- interpolate方法用于对缺失值进行插值。针对DataFrame的interpolate方法，其基本语法格式如下。

```
DataFrame.interpolate(method='linear', axis=0, limit=None, inplace=False,
limit_direction='forward', limit_area=None, downcast=None, **kwargs)
```

- interpolate方法常用的参数及其说明，如下表所示。

参数名称	说明
method	接收str。表示插值方法。默认为“linear”
axis	接收int。表示插值的轴。默认为0
limit	接收int。表示遇到连续NaN插值的最大数。默认为None
inplace	接收bool。表示是否更新原DataFrame。默认为Fasle

- interpolate方法提供了多种插值方法，常用插值方法及使用场景，如下表所示。

方法	说明
linear	线性插值。忽视索引，将所有值看做等距隔开。若DataFrame或Series为多重索引，则只支持此种插值方法
time	时间插值。索引为时间类型，按给定时间间隔插值
Index、values	索引插值。按照数值化的索引值来插值

查看数据集中的缺失值

示例

参阅教材内容

填充数据集中的缺失值

一、用字符串'缺失数据'代替

二、用前后值填充缺失数据

三、用变量均值或者中位数填充缺失数据

四、用线性插值法填充缺失数据

示例

参阅教材内容

删除数据集中的缺失值

我们还可以通过删除的方式解决数据集中的缺失值问题。

示例

参阅教材内容

PART 10

python数据重复值处理

python数据重复值处理

我们在实务中经常会遇到数据重复录入，出现重复值的情形。很多情况下，这些重复值对于机器学习或统计分析来说是没有意义的，需要予以剔除，那么就需要用到数据重复值处理的系列方法

- 查看数据集中的重复值
- 删除数据集中的重复值

示例

参阅教材内容

python数据行列处理

我们在应用python开展机器学习或统计分析时，经常需要对数据行列进行处理，下面介绍几种常用的python数据行列处理操作。

- 删除变量列、样本示例行
- 更改变量列名称、调整变量列顺序
- 改变列的数据格式
- 多列转换
- 数据百分比格式转换

示例

参阅教材内容

- 重复数据处理是机器学习经常面对的问题之一。对重复数据进行处理前，需要分析重复数据产生的原因以及去除这部分数据后可能造成的不良影响。
- drop_duplicates方法用于去除一个或多个特征的重复记录。针对DataFrame的drop_duplicates方法的基本语法格式如下。

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

- drop_duplicates方法常用的参数及其说明，如下表所示。

参数名称	说明
subset	接收列标签或者标签的sequence。表示参与去重操作的列名。默认为None，表示全部列
keep	接收特定str。表示重复时保留第几个数据。取值为“first”时表示第一个；取值为“last”时表示最后一个；取值为False时表示只要有重复都不保留。默认为first
inplace	接收bool。表示是否在原表上进行操作。默认为False

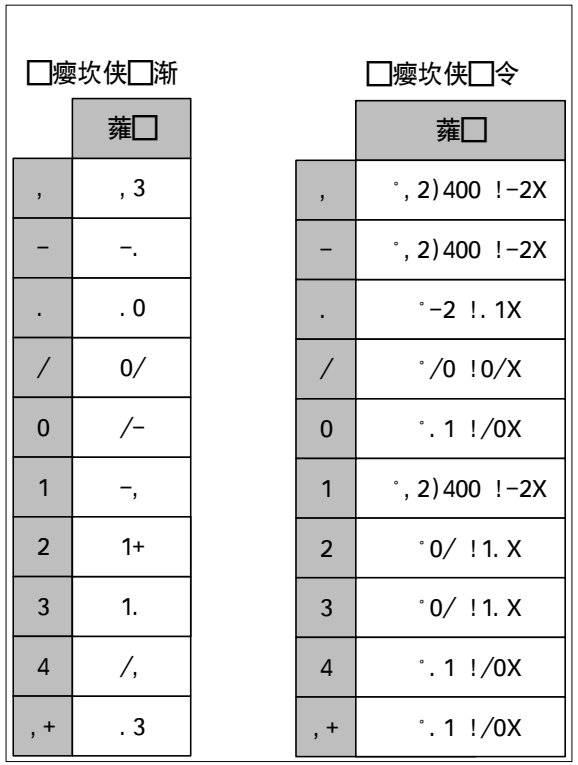
- 使用drop_duplicates方法去除数据中的重复记录，默认对所有特征起作用，即只有所有特征的重复记录对应的索引（行）相同的情况下才会执行去除操作，如下代码所示。

In[10]:	df = pd.read_csv('../data/Station.csv', encoding = 'gbk') df1 = df.drop_duplicates() print('去重前数据框的长度为：', len(df), '\n', '去重后数据框的长度为：', len(df1))
Out[10]:	去重前数据框的长度为： 396060 去重后数据框的长度为： 357672

- drop_duplicates函数除了能够针对所有特征去重外，能够针对某个或某几个特征进行去重，只需要将指定的特征名称传给subset参数即可，如下代码所示。

In[11]:	df2 = df.drop_duplicates(subset = ['train']) print('去重前数据框长度为：', len(df), '\n', '指定特征col1去重后数据框长度为：', len(df2))
Out[11]:	去重前数据框长度为： 396060 指定特征col1去重后数据框长度为： 138

- 某些算法如ID3决策树算法和Apriori算法等，要求数据是离散的，此时就需要将连续型特征（数值型）转换成离散型特征（类别型），即连续特征离散化。
- 连续特征离散化是在数据的取值范围内设定若干个离散的划分点，将取值范围划分为一些离散化的区间，最后用不同的符号或整数值代表落在每个子区间中的数据值。因此离散化涉及两个子任务，即确定分类数以及如何将连续型数据映射到这些类别型数据上。其原理如下图所示。



□癮坎侠□渐

	薤□
,	, 3
–	–.
.	. 0
/	0/
0	/–
1	–,
2	1+
3	1.
4	/,
, +	. 3

□癮坎侠□令

	薤□
,	°, 2)400 !–2X
–	°, 2)400 !–2X
.	°–2 !. 1X
/	°/0 !0/X
0	°. 1 !/0X
1	°, 2)400 !–2X
2	°0/ !1. X
3	°0/ !1. X
4	°. 1 !/0X
, +	°. 1 !/0X

- 等宽法指将数据的值域分成具有相同宽度的区间，与制作频率分布表类似。cut函数用于实现等宽法，其基础语法格式如下。

```
pandas.cut(x, bins, right=True, labels=None, retbins=False, precision=3, include_lowest=False)
```

- cut函数常用的参数及其说明，如下表所示。

参数名称	说明
x	接收数组或Series。表示需要进行离散化处理的数据。无默认值
bins	接收int，list，array，tuple。若为int，表示离散化后的类别数目；若为序列类型的数据，则表示进行切分的区间，每两个数间隔为一个区间。无默认值
right	接收bool。表示右侧是否为闭区间。默认为True
labels	接收list，array类型数据。表示离散化后各个类别的名称。默认为空
retbins	接收bool。表示是否返回区间标签。默认为False
precision	接收int。显示的标签的精度。默认为3

- 等频法在等宽法的基础上实现，将切分区间指定为被切分数据的分位数，这样能保证切分区间的数据大体相等。结合函数对数据应用等频法，如下代码所示。

```
In[14]:  import numpy as np
         def SameRateCut(data, k):
             w = data.quantile(np.arange(
                     0, 1 + 1.0 / k, 1.0 / k))
             data = pd.cut(data, w)
             return data
         series1 = SameRateCut(series, 3)
         print('等频离散化后数据为：\n', series1, '\n',
               '离散化后数据各区间数目为：\n', series1.value_counts())
```

Out[14]:
```
等频离散化后数据为：
 0              NaN
1      (1.0, 6.667]
2    (6.667, 8.333]
3    (6.667, 8.333]
4     (8.333, 15.0]
5     (8.333, 15.0]
dtype: category
Categories (3, interval[float64]): [(1.0, 6.667] < (6.667, 8.333] < (8.333, 15.0]]
 离散化后数据各区间数目为：
 (8.333, 15.0]     2
(6.667, 8.333]     2
(1.0, 6.667]       1
dtype: int64
```

- 哑变量又称虚拟变量，通常取0或1。机器学习中有相当一部分的算法模型都要求输入的特征为数值型，但实际数据中特征的类型不一定只有数值型，还会存在相当一部分的类别型，这部分的特征需要经过哑变量处理才可以放入模型之中。哑变量处理的原理如下图所示。

悯粮□侠□渐

	踏荤
,	蘧苜
–	笔□
.	鲔苜
／	砍嘲
0	□逝
1	砍嘲
2	笔□
3	鲔苜
4	蘧苜
, +	□逝

悯粮□侠□令

	踏荤Z蘧苜	踏荤Z笔□	踏荤Z鲔苜	踏荤Z砍嘲	踏荤Z□逝
,	,	+	+	+	+
–	+	,	+	+	+
.	+	+	,	+	+
／	+	+	+	,	+
0	+	+	+	+	,
1	+	+	+	,	+
2	+	,	+	+	+
3	+	+	,	+	+
4	,	+	+	+	+
, +	+	+	+	+	,

- get_dummies函数用于哑变量处理，其基本语法格式如下。

```
pandas.get_dummies(data, prefix=None, prefix_sep='_', dummy_na=False, columns=None, sparse=False, drop_first=False)
```

哑变量处理

- cut函数常用的参数及其说明，如下表所示。

参数名称	说明
data	接收array、DataFrame或者Series。表示需要哑变量处理的数据。无默认值
prefix	接收str、str的列表或者str的dict。表示哑变量化后列名的前缀。默认为None
prefix_sep	接收str。表示前缀的连接符。默认为’_’
dummy_na	接收bool。表示是否为Nan值添加一列。默认为False
columns	接收类似list的数据。表示DataFrame中需要编码的列名。默认为None，表示对所有object和category类型进行编码
sparse	接收bool。表示虚拟列是否是稀疏的。默认为False
drop_first	接收bool，表示是否通过从k个分类级别中删除第一级来获得k-1个分类级别。默认为False

PART 12

习　题

作业3

1、读取csv数据“数据2.3.csv”，并开展以下操作：

（1）查看数据集中的缺失值

（2）填充数据集中的缺失值（用字符串'缺失数据'代替、用变量均值或者中位数填充缺失数据）

注意：在采取不同方式填充缺失数据时，均需重新读取数据。

（3）将var5的数据格式设置成百分比格式。

感谢聆听

THANKS