貴州大學
GUIZHOU UNIVERSITY

Lecture 7 数据处理技术

- 合并数据集
- 数据转换

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。

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。

合并数据集

- 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（或两个都传）以说明索引应该被用作连接键：

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

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

合并数据集

```
In [10]: left1
Out[10]:
   key  value
0    a      0
1    b      1
2    a      2
3    a      3
4    b      4
5    c      5

[6 rows x 2 columns]
```

```
In [11]: right1
Out[11]:
   group_val
a        3.5
b        7.0

[2 rows x 1 columns]
```

合并数据集

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

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

合并数据集

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

	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 rows x 3 columns]

合并数据集

	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

[6 rows x 3 columns]

合并数据集

下面我们再来看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'])
```

合并数据集

	data	key1	key2
0	0	Ohio	2000
1	1	Ohio	2001
2	2	Ohio	2002
3	3	Nevada	2001
4	4	Nevada	2002

[5 rows x 3 columns]

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

[6 rows x 2 columns]

合并数据集

*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：

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

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

[5 rows x 5 columns]

合并数据集

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

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

[7 rows x 5 columns]

合并数据集

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'])

合并数据集

```
In [23]: left2
Out[23]:
   Ohio  Nevada
a     1       2
c     3       4
e     5       6

[3 rows x 2 columns]
```

```
In [24]: right2
Out[24]:
   Missouri  Alabama
b         7        8
c         9       10
d        11       12
e        13       14

[4 rows x 2 columns]
```

合并数据集

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

	Ohio	Nevada	Missouri	Alabama
a	1	2	NaN	NaN
b	NaN	NaN	7	8
c	3	4	9	10
d	NaN	NaN	11	12
e	5	6	13	14

[5 rows x 4 columns]

合并数据集

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	Missouri	Alabama
a	1	2	NaN	NaN
b	NaN	NaN	7	8
c	3	4	9	10
d	NaN	NaN	11	12
e	5	6	13	14

[5 rows x 4 columns]

合并数据集

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

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

	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

[6 rows x 3 columns]

轴向连接

另一种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))

```
In [33]: arr
Out[33]:
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

轴向连接

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

```
In [34]: np.concatenate([arr, arr], axis=1)
Out[34]:
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]])
```

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'])
```

合并重叠数据

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

```
Out[67]:
f    NaN
e    2.5
d    NaN
c    3.5
b    4.5
a    NaN
dtype: float64
```

```
Out[68]:
f    0
e    1
d    2
c    3
b    4
a  NaN
dtype: float64
```

合并重叠数据

np.where(pd.isnull(a), b, a)

array([0., 2.5, 2. , 3.5, 4.5, nan])

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。

数据转换

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

```
Out[5]:
    k1   k2
0   one   1
1   one   1
2   one   2
3   two   3
4   two   3
5   two   4
6   two   4

[7 rows x 2 columns]
```

数据转换

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
```

数据转换

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*data.drop_duplicates()*

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

[4 rows x 2 columns]

数据转换

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

[7 rows x 3 columns]

数据转换

data.drop_duplicates(['k1'])

```
   k1   k2  v1
0  one   1   0
3  two   3   3

[2 rows x 3 columns]
```

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take_last=True保留重复数据中的最后一个。

数据转换

In [11]: data.drop_duplicates(['k1', 'k2'], take_last=True)

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

[4 rows x 3 columns]

替换值

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。我们来看看下面这个Series：

In [18]: data = pd.Series([1., -999, 2., -999, -1000., 3.])

替换值

In [18]: data = pd.Series([1., -999, 2., -999, -1000., 3.])

```
Out[19]:
0       1
1    -999
2       2
3    -999
4   -1000
5       3
dtype: float64
```

替换值

-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，我们可以利用replace来产生一个新的Series：

data.replace(-999, np.nan)

```
0      1
1    NaN
2      2
3    NaN
4   -100
5      3
dtype: float64
```

替换值

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
1    NaN
2      2
3    NaN
4    NaN
5      3
dtype: float64
```

替换值

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])

```
Out[22]:
0      1
1    NaN
2      2
3    NaN
4      0
5      3
dtype: float64
```

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))
```

检测异常值

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

[8 rows x 4 columns]

检测异常值

下面我们要选出全部含有“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

[11 rows x 4 columns]

检测异常值

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

[8 rows x 4 columns]

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))

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的行调换

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

[5 rows x 4 columns]

排列和随机采样

如果不想用替换的方式选取随机子集，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

[3 rows x 4 columns]