第 5 章

pandas 数据整理

采集到的原始数据通常来自多个异构数据源，数据在准确性、完整性和一致性等方面存在着多种多样的问题。在数据分析挖掘之前，首先要做的就是对数据进行整理，处理缺失值、重复值、噪声数据以及规范数据，得到一个高质量的数据集，从而提高数据分析的准确性和有效性。本章主要介绍数据运算与排序，数据集成，数据筛选，删除指定的行或列，重新命名列名、行名和重新索引，缺失值处理，异常值处理，重复值处理，数据替换、更新与转换，噪声数据处理，数据规范化，数据离散化，数据归约，以及数据降维。

5.1 数据运算与排序

5.1.1 数据运算

DataFrame 对象的数据运算方法如表 5-1 所示，表格中的 df 表示一个 DataFrame 对象。

表 5-1 DataFrame 对象的数据运算方法

数据运算方法	描　　述
df * N	df 的所有元素乘以 N
df1＋df2	将 df1 和 df2 的行名和列名都相同的元素相加，其他位置的元素用 NaN 填充
df1.add(other, axis＝'columns', level＝None, fill_value＝None)	将 df1 中的元素与 other 中的元素相加，other 的类型可以是 scalar(标量)、sequence(序列)、Series、DataFrame 等形式
df1.sub(other, axis＝'columns', level＝None, fill_value＝None)	将 df1 中的元素与 other 中的元素相减
df1.div(other, axis＝'columns', level＝None, fill_value＝None)	将 df1 中的元素与 other 中的元素相除
df1.mul (other, axis＝'columns', level＝None, fill_value＝None)	将 df1 中的元素与 other 中的元素相乘

1. df1＋df2

作用：将 df1 和 df2 的行名和列名都相同的元素相加，其他位置的元素用 NaN 填充。

```
>>>import pandas as pd
>>>import numpy as np
```

```
>>>df1=pd.DataFrame(np.arange(9).reshape((3,3)),columns=list('bcd'), index=
['A','B','C'])
>>>df2=pd.DataFrame(np.arange(12).reshape((4,3)),columns=list('bde'), index
=['A','D','B','E'])
>>>df1
   b  c  d
A  0  1  2
B  3  4  5
C  6  7  8
>>>df2
   b   d   e
A  0   1   2
D  3   4   5
B  6   7   8
E  9  10  11
>>>df1+df2   #行名和列名都相同的元素相加,其他位置的元素用 NaN 填充
     b   c     d   e
A  0.0 NaN   3.0 NaN
B  9.0 NaN  12.0 NaN
C  NaN NaN   NaN NaN
D  NaN NaN   NaN NaN
E  NaN NaN   NaN NaN
```

2. df1.add(other，axis='columns'，level=None，fill_value=None)

作用：若 df1 和 other 对象的行名和列名都相同的位置上的元素都存在时，直接将相应的元素相加；同一行名和列名处，若其中一个对象在该处不存在元素，则先用 fill_value 指定的值填充，然后相加；若同一行名和列名处，两个对象都不存在元素，则相加的结果用 NaN 填充。

参数说明如下。

other：取值集合{scalar(标量)，sequence(序列)，Series，DataFrame}

axis：取值集合{ 'index'或 'columns'}，axis=0 或 'index'表示按行比较，axis=1 或'columns'表示按列比较。

level：int 或 name，选择不同的索引，一个 DataFrame 对象可能有两个索引。

fill_value：用于填充缺失值，默认将缺失值填充为 NaN。

```
>>>df1.add(10)          #other 为标量 10
    b   c   d
A  10  11  12
B  13  14  15
C  16  17  18
>>>df1.add([5,5,5])     #other 为列表
    b   c   d
A   5   6   7
B   8   9  10
C  11  12  13
>>>df1.add(df2)         #与 df1+df2 的计算结果一样
```

```
     b   c     d   e
A  0.0 NaN   3.0 NaN
B  9.0 NaN  12.0 NaN
C  NaN NaN   NaN NaN
D  NaN NaN   NaN NaN
E  NaN NaN   NaN NaN
>>>df1.add(df2, fill_value=100)
       b      c      d      e
A    0.0  101.0    3.0  102.0
B    9.0  104.0   12.0  108.0
C  106.0  107.0  108.0    NaN
D  103.0    NaN  104.0  105.0
E  109.0    NaN  110.0  111.0
```

5.1.2　数据排序

DataFrame 对象的数据排序方法如表 5-2 所示，表格中的 df 表示一个 DataFrame 对象。

表 5-2　DataFrame 对象的数据排序方法

数据排序方法	描　　述
df.sort_index(axis=0, ascending=True)	按行索引进行升序排序
df.sort_values(by, axis=0, ascending=True)	按指定的列或行进行值排序
df.rank(axis=0, method='average', ascending=True)	沿着行计算元素值的排名，对于相同的两个元素值，沿着行顺序排在前面的数据排名高，返回各个位置上元素值从小到大排序对应的序号

1. df.sort_index(axis=0, ascending=True)

作用：按行索引进行升序排序。

参数说明如下。

axis：axis=0，对行进行排序；axis=1，对列进行排序。

ascending：ascending=True，升序排序；ascending=False，降序排序。

```
>>>df =pd.DataFrame({ 'col1' : ['A', 'A', 'B',  'D', 'C'],'col2' : [2, 9, 8, 7, 4],
'col3': [1, 9, 4, 2, 3],})
>>>df
  col1  col2  col3
0    A     2     1
1    A     9     9
2    B     8     4
3    D     7     2
4    C     4     3
>>>df.sort_index(axis=1,ascending=False) #按列索引进行降序排列
   col3  col2 col1
0     1     2    A
```

```
1    9    9    A
2    4    8    B
3    2    7    D
4    3    4    C
```

2. df.sort_values(by，axis＝0，ascending＝True)

作用：按指定的列或行进行值排序。

参数说明如下。

by：指定某些行或列作为排序的依据。

axis：axis＝0，对行进行排序；axis＝1，对列进行排序。

```
>>>df.sort_values(by=['col1']) #按 col1 进行排序
  col1  col2  col3
0    A     2     1
1    A     9     9
2    B     8     4
4    C     4     3
3    D     7     2
>>>df.sort_values(by=['col1','col2']) #按 col1、col2 进行排序
  col1  col2  col3
0    A     2     1
1    A     9     9
2    B     8     4
4    C     4     3
3    D     7     2
```

3. df.rank(axis＝0，method＝'average'，ascending＝True)

作用：沿着行计算元素值的排名，对于相同的两个元素值，沿着行顺序排在前面的数据排名高，返回各个位置上元素值从小到大排序对应的序号。

参数说明如下。

axis：为 0 时沿着行，为 1 时沿着列。

method：method＝'average'时，在相等分组中，为各个值分配平均排名；method＝ 'min'时，使用整个分组的最小排名；method＝'max'时，使用整个分组的最大排名；method＝ 'first'时，按值在原始数据中的出现顺序分配排名。

ascending：boolean，默认值为 True，True 为升序排名，False 为降序排名。

```
>>>df =pd.DataFrame({'a':[4,2,3,2],'b':[15,21,10,13]},index=['one','two',
'three','four'])
>>>df
       a   b
one    4  15
two    2  21
three  3  10
four   2  13
>>>df.rank(method ='first') #为每个位置分配从小到大排序后其元素对应的序号
         a    b
one    4.0  3.0
two    1.0  4.0
```

```
three  3.0  1.0
four   2.0  2.0
#将在两个 2 这一分组的排名 1 和 2 的最大排名 2 作为两个 2 的排名
>>>df.rank(method ='max')
        a    b
one    4.0  3.0
two    2.0  4.0
three  3.0  1.0
four   2.0  2.0
#将两个 2 这一分组的排名 1 和 2 的平均值 1.5 作为两个 2 的排名
>>>df.rank(method ='average')
        a    b
one    4.0  3.0
two    1.5  4.0
three  3.0  1.0
four   1.5  2.0
```

5.1.3　批量操作

DataFrame 对象的 apply()方法可对一行或一列做出操作，是对元素集合级别的操作，这里元素集合指的是以列为单位，或者以行为单位。

1. df.apply(func，axis＝0)

作用：对 df 的行或列应用函数 func。

参数说明如下。

func：函数名称，指定应用在每行或每列的函数的名称。

axis：axis＝0，对每一列应用 func 函数；axis＝1，对每一行应用 func 函数。

```
>>>import numpy as np
>>>df=pd.DataFrame(np.arange(16).reshape((4,4)),columns=list('abcd'),
index=['A','B','C','D'])
>>>df
    a   b   c   d
A   0   1   2   3
B   4   5   6   7
C   8   9  10  11
D  12  13  14  15
>>>def f(x):                    #计算数组元素的取值间隔
return x.max()-x.min()
>>>df.apply(f,axis=0)    #求每一列元素的取值间隔
a    12
b    12
c    12
d    12
dtype: int64
>>>df.apply(f,axis=1)
A    3
B    3
C    3
```

```
D    3
dtype: int64
```

apply()函数可一次执行多个函数,作用于行或列时,一次返回多个结果。

```
>>>def f1(x):
return pd.Series([x.max(),x.min()],index=['max','min'])
>>>df.apply(f1,axis=0)                    #操作列
      a   b   c   d
max  12  13  14  15
min   0   1   2   3
>>>df.apply(np.sum, axis=1)               #操作行
A     6
B    22
C    38
D    54
>>>df.apply(np.square)                    #操作元素
     a    b    c    d
A    0    1    4    9
B   16   25   36   49
C   64   81  100  121
D  144  169  196  225
>>>df['a'].apply(lambda x: 111 if x<8 else 0)    #Lambda 函数
A    111
B    111
C      0
D      0
```

2. df.applymap(func)

作用:将 func 函数应用到各个元素上。

```
>>>df
    a   b   c   d
A   0   1   2   3
B   4   5   6   7
C   8   9  10  11
D  12  13  14  15
>>>def f2(x):
    return 2*x+1
>>>df.applymap(f2)
    a   b   c   d
A   1   3   5   7
B   9  11  13  15
C  17  19  21  23
D  25  27  29  31
```

5.2 数据集成

5.2.1 两个 DataFrame 对象的连接

DataFrame 对象的 merge()方法用于连接两个 DataFrame 对象,语法格式如下。

```
df1.merge(df2, how='inner', on=None, left_on=None, right_on=None, left_index=
False, right_index=False, sort=False, suffixes=('_x', '_y'))
```

作用：通过行索引或列索引进行两个 DataFrame 对象的连接。

参数说明如下。

df2：拟被合并的 DataFrame 对象。

how：{'left', 'right', 'outer', 'inner'}，连接的方式。how＝left 表示以左边 df1 的键为基准，右边出现匹配失败的用 NaN 代替；how＝right 表示以右边 df2 的键为基准；how＝outer 表示以左右两边 DataFrame 共有的键为基准，匹配成功的保留，匹配失败的键值对以 NaN 进行替换；how＝inner 表示以左右两边 DataFrame 共有的键为基准，匹配成功的保留，匹配失败所在的行全部删除，how 默认值为 inner。

on：设置连接的共有列名，默认为 None，以最多相同的共有列名作连接键。

left_on＝None 和 right_on＝None：以上 on 是在两个 df 有相同的列名的情况下使用，如果两个 df 没有相同的列名，使用 left_on 和 right_on 分别指明左边和右边的连接列名。

left_index 和 right_index：默认都为 False，如果需要按行索引进行合并，则需要使用 left_index 和 right_index 指明以 index 为键，on、left_on、right_on 都是指定以列为连接键。如果使用了 left_index 或者 right_index，必须指明另一方的参考键。如果是 left_on ＝'**' 和 right_index ＝ True 或者 left_index ＝True 和 right_on ＝ '**'，index 取 on 所在的一方，如果是 left_index ＝True 和 right_index ＝ True，取两者相同的。

sort＝False：合并结果是否按 on 指定的键进行排序。

suffixes：合并后如果有重复列，分别加上什么后缀，默认加上后缀_x、_y 进行区别。

1. 不同的合并方式

```
>>>df1 =pd.DataFrame({'a':['a1','a2','a3'], 'b':['b1','b2','b3'], 'key1':['a',
'b','c'], 'key2':['d','e','f']})
>>>df2 =pd.DataFrame({'c':['c1','c2','c3'], 'd':['d1','d2','d3'], 'key1':['a',
'b','a'], 'key2':['d','e','g']})
>>>print(df1)
    a   b key1 key2
0  a1  b1    a    d
1  a2  b2    b    e
2  a3  b3    c    f
>>>print(df2)
    c   d key1 key2
0  c1  d1    a    d
1  c2  d2    b    e
2  c3  d3    a    g
>>>df1.merge(df2,how ="left", on =['key1','key2'])  #how =left,以左边键为基准
    a   b key1 key2    c    d
0  a1  b1    a    d   c1   d1
1  a2  b2    b    e   c2   d2
2  a3  b3    c    f  NaN  NaN
>>>df1.merge(df2,how ="right", on =['key1','key2'])  #how =right,以右边键为基准
```

```
     a    b key1 key2   c   d
0   a1   b1    a    d  c1  d1
1   a2   b2    b    e  c2  d2
2  NaN  NaN   a    g  c3  d3
>>>df1.merge(df2,how ="inner", on =['key1','key2']) #how =inner,取左右交集
    a   b key1 key2   c   d
0  a1  b1    a    d  c1  d1
1  a2  b2    b    e  c2  d2
```

"outer"与"inner" 用法对应,以左右两边 DataFrame 共有的键为基准,匹配成功的保留,匹配失败的键对应的值以 NaN 进行替换。

```
>>>df1.merge(df2,how ="outer", on =['key1','key2'])  #how =outer,左右两边并集
     a    b key1 key2    c    d
0   a1   b1    a    d   c1   d1
1   a2   b2    b    e   c2   d2
2   a3   b3    c    f  NaN  NaN
3  NaN  NaN    a    g   c3   d3
```

2. 具有不同列名的合并

```
>>>df3 =pd.DataFrame({'lkey': ['Zhang', 'Wang', 'Li', 'Zhao'], 'value': [70, 80,
85, 95]})
>>>df4 =pd.DataFrame({'rkey': ['Zhang', 'Wang', 'Li', 'Zhao'], 'value': [7, 8,
8.5, 9.5]})
>>>print(df3)
    lkey  value
0  Zhang     70
1   Wang     80
2     Li     85
3   Zhao     95
>>>print(df4)
    rkey  value
0  Zhang    7.0
1   Wang    8.0
2     Li    8.5
3   Zhao    9.5
>>>df3.merge(df4, left_on ='lkey',right_on ='rkey')
    lkey  value_x   rkey  value_y
0  Zhang       70  Zhang      7.0
1   Wang       80   Wang      8.0
2     Li       85     Li      8.5
3   Zhao       95   Zhao      9.5
#设置 suffixes 参数
>>>df3.merge(df4, left_on ='lkey',right_on ='rkey',suffixes=("_lf","_rf"))
    lkey  value_lf   rkey  value_rf
0  Zhang       70  Zhang      7.0
1   Wang       80   Wang      8.0
2     Li       85     Li      8.5
3   Zhao       95   Zhao      9.5
```

3. 指定是否按索引进行合并

```
>>>df5 =pd.DataFrame({'key1':['a','b','c','d'],'key2':['e','f','g','h']},
index=['k','l','m','n'])
>>>df6 =pd.DataFrame({'key1':['k',6,'l']},index=['a','m','c'])
>>>print(df5)
  key1 key2
k    a    e
l    b    f
m    c    g
n    d    h
>>>print(df6)
  key1
a    k
m    6
c    l
>>>df5.merge(df6, left_index=True, right_index=True) #根据行索引进行合并
  key1_x key2 key1_y
m      c    g      6
>>>df5.merge(df6,left_on='key1', right_index=True)
  key1 key1_x key2 key1_y
k    a      a    e      k
m    c      c    g      l
```

5.2.2　多个 DataFrame 对象的堆叠

pandas 的 concat()函数可一次实现对多个 DataFrame 对象的堆叠，默认是对所有列在垂直方向进行堆叠，index 为原来各自的索引。

```
pandas.concat(objs, axis=0, join='outer', ignore_index=False, keys=None, sort=None)
```

作用：以指定的轴将多个对象堆叠到一起，concat()不会去掉重复的记录。

参数说明如下。

objs：需要连接的对象集合，其类型可以是 Series、DataFrame、列表或字典。

axis：连接方向，axis＝0，对行操作，纵向连接；axis＝1，对列操作，横向连接。

join：表示堆叠方式，只有 outer 和 inner 方式，默认为 outer，inner 表示只对相同列进行堆叠。

keys：创建层次化索引，以标识数据来自不同的连接对象。

ignore_index：ignore_index＝False，保留原索引；ignore_index＝True，忽略原索引，重建索引。

1. join 指定堆叠方式

```
>>>s1 =pd.DataFrame([[1,2,3],[4,5,6]],columns=['a','b','c'])
>>>s2 =pd.DataFrame([[3,4,'w'],[5,6,'x']],columns=['a','b','d'])
>>>print(s1)
   a  b  c
0  1  2  3
1  4  5  6
```

```
>>>print(s2)
   a  b  d
0  3  4  w
1  5  6  x
#join为'inner'时得到的是两表的交集,是'outer'时得到的是两表的并集
>>>print(pd.concat([s1,s2]))
   a  b    c    d
0  1  2  3.0  NaN
1  4  5  6.0  NaN
0  3  4  NaN    w
1  5  6  NaN    x
>>>print(pd.concat([s1,s2],join="inner"))
   a  b
0  1  2
1  4  5
0  3  4
1  5  6
```

2. 在水平方向进行堆叠

如果是在水平方向进行堆叠,需要指定 axis=1,如果合并后有相同列名称,可通过 add_prefix 或者 add_suffix 为其中一个添加名称前缀或后缀。

```
>>>print(pd.concat([s1,s2],axis=1,sort=False))
   a  b  c  a  b  d
0  1  2  3  3  4  w
1  4  5  6  5  6  x
```

3. ignore_index 指定是否保留原对象的 index

ignore_index=False,保留原索引;ignore_index=True,忽略原索引,重建索引。

```
>>>print(pd.concat([s1,s2],ignore_index=True))
   a  b    c    d
0  1  2  3.0  NaN
1  4  5  6.0  NaN
2  3  4  NaN    w
3  5  6  NaN    x
```

4. 参数 key 用于创建层次索引以标识数据源自于哪张表

```
>>>pd.concat([s1,s2],keys=['s1','s2'])
      a  b    c    d
s1 0  1  2  3.0  NaN
   1  4  5  6.0  NaN
s2 0  3  4  NaN    w
   1  5  6  NaN    x
```

5.2.3 DataFrame 对象的行列旋转

1. 列旋转为行

```
df.stack(level=-1, dropna=True)
```