第 3 章

pandas 数据分析库的数据结构

pandas 是一个用于数据分析的开源 Python 库，提供了高性能易用的数据类型，以及大量能使我们快速便捷地处理数据的函数和方法。pandas 的核心数据结构有两种，即二维表格型的 DataFrame 对象和一维数组的 Series 对象。本章主要介绍 Series 结构，DataFrame 结构，读取、修改和删除 DataFrame 的数据，以及检查 DataFrame 对象是否包含指定的值。

3.1 Series 结构

3.1.1 创建 Series

Series 结构（简称 Series 对象）是一维数组（表示 DataFrame 结构的对象的一列），类似 Python 的列表，区别是列表中的元素可以是不同类型的数据，而 Series 中则只允许存储同一数据类型的数据，因为这样可以更有效地使用内存，批量操作元素时速度更快。

Series 对象的内部结构如表 3-1 所示，由两个相互关联的数组组成，一个是数据（也称为元素，本章将这两个概念等同）数组 values，用来存放数据；数组 values 中的每个数据都有一个与之关联的索引（标签、序号、行号），这些索引存储在另外一个叫作 index 的索引数组中。

表 3-1 Series 对象的内部结构

index	values
0	'a'
1	'b'
2	'c'

创建 Series 对象最常用的方法是使用 pandas 的 Series()构造函数，其语法格式如下。

```
pandas.Series(data=None, index=None, dtype=None, name=None)
```

返回值：返回一个 Series 对象。

参数说明如下。

data：创建 Series 对象的数据，可以是 Python 的一个列表，如果列表是混合类型的列表，将会使用最常见的类型，通常数据类型 dtype 为对象（object）；可以是一个字典，将键值对中的“值”作为 Series 对象的数据，将“键”作为索引；也可是一个标量值，这种情况下必须设置索引 index，标量值会重复来匹配索引的长度；也可以是 NumPy 的一维数组 ndarray。

index：为 Series 对象的每个数据指定索引。创建 Series 对象时，若不用 index 明确指定索引，pandas 默认使用从 0 开始依次递增的整数值依次作为每个数据的索引。

dtype：为 Series 对象的数据指定数据类型。

name：为 Series 对象起个名字。

因此，根据 data 的数据类型不同，有如下创建 Series 对象的方式。

1. 用 NumPy 的一维 ndarray 数组创建 Series 对象

```
>>>import numpy as np   #导入 numpy 库
#本章中出现的 np 默认均是这个含义，numpy 是 Python 的一个科学计算库
>>>import pandas as pd  #本章中出现的 pd 默认均是这个含义
#arange()函数创建均匀间隔的一维数组，起始值为 0，结尾值为 5，间隔值为 2
>>>s1=pd.Series(np.arange(0,5,2),index=['a', 'b', 'c']) #通过 index 指定行索引
>>>s1
a    0
b    2
c    4
dtype: int32
```

如果想分别查看组成 Series 对象的两个数组，可通过调用它的两个属性 index(索引)和 values(数据)来得到。

```
>>>s1.values
array([0, 2, 4])
>>>s1.index
Index(['a', 'b', 'c'], dtype='object')
```

2. 用标量值创建 Series 对象

```
>>>s2 =pd.Series(25,index =['a', 'b','c'])
>>>s2
a    25
b    25
c    25
```

3. 用字典创建 Series 对象

用字典创建 Series 对象时，键值对中的“键”是用来作为 Series 对象的索引，键值对中的“值”作为 Series 对象的数据。

```
>>>dict1 ={'Alice':'2341', 'Beth':'9102','Cecil':'3258'}
>>>sd =pd.Series(dict1)
>>>sd
Alice    2341
Beth     9102
Cecil    3258
dtype: object
```

4. 用列表创建 Series 对象

```
>>>s3=pd.Series(data=[90,86,95],index=['Java','C','Python'])
```

```
>>>s3
Java      90
C         86
Python    95
dtype: int64
```

3.1.2　查看和修改 Series 对象的数据

查看 Series 对象中的数据有两种方式，一种是按位置索引查看数据，另一种是按指定索引（称为索引标签）查看。

1. 通过索引和切片查看 Series 对象的数据

可以使用“Series 对象[id]”方式访问 Series 对象的索引 id 对应的数据。

（1）通过索引查看对应的数据。

```
>>>s3['C']
86
```

可通过默认索引来读取。

```
>>>s3[1]
86
```

此外，还可以将索引作为 Series 对象的属性来获取对应的数据。

```
>>>s3.C
86
```

（2）通过截取（切片）索引的方式一次读取多个元素。

```
>>>s3[0:2]
科目
Java    90
C       86
Name: grade, dtype: int64
```

（3）通过索引列表一次读取多个元素。

使用多个数据对应的索引来一次读取多个元素，注意索引要放在一个列表中。

```
>>>s3[['Python','C','Java']]
科目
Python    95
C         86
Java      90
Name: grade, dtype: int64
```

（4）根据筛选条件读取数据。

```
>>>s3[s3>=90]          #获取数据值>=90 的元素
科目
Java      90
Python    95
Name: grade, dtype: int64
```

2. 修改 Series 对象的数据

可以“s3［id］＝新值”重新赋值的方式修改 s3 中索引 id 对应的数据。

```
>>>s3['C']=96
>>>s3['C']
96
```

3.1.3 Series 对象的常用属性

Series 对象的常用属性如表 3-2 所示。

表 3-2 Series 对象的常用属性

属性名称	说明
shape	获取 Series 对象的形状
axes	以列表的形式返回所有行索引标签
dtype	返回对象的数据类型
empty	返回一个布尔值,用于判断 Series 对象的数据是否为空
ndim	返回输入数据的维数
size	返回输入数据的元素数量
values	以 ndarray 的形式返回 Series 对象的所有数据
index	返回 Series 对象的所有索引

Series 对象的常用属性用法举例如下。

```
>>>s3.shape
(3,)
>>>s3.index
Index(['Java', 'C', 'Python'], dtype='object')
```

此外,可使用 name 属性为 Series 对象及索引命名。

```
>>>s3.name='grade'     #为 Series 对象 s3 命名'grade'
>>>s3.name
'grade'
>>>s3.index.name='科目' #为索引命名
>>>s3.index.name
'科目'
```

3.1.4 Series 对象的常用方法

Series 对象的常用方法如表 3-3 所示。

表 3-3 Series 对象的常用方法

方法名称	说明
sum()	求和

续表

方 法 名 称	说　　明
mean()	求均值
max()	求最大值
min()	求最小值
count()	求包含的元素个数
std()	求标准差
var()	求方差
median()	求中位数
describe()	获得描述性统计信息，如方差、标准差、中位数等
drop()	删除指定索引的元素
value_counts()	统计每个值重复的次数，返回一个 Series 对象
unique()	获得去重之后的 ndarray 对象
dropna()	用来删除空值
fillna(value＝填充的值)	填充空值
mask()	将满足条件的全部替换
where()	将不满足条件的替换
map()	不通过索引，直接通过内容修改数据
apply(func)	将原始值通过函数名为 func 的函数计算，得到想要的数组
isnull()和 notnull()	用于 Series 对象每个值是否是空值的判断
duplicated()	找出重复的数据，根据返回的布尔值显示是否重复
drop_duplicates()	删除重复的数据
sort_values()	按值从小到大排序
sort_values(ascending＝False)	按值从大到小排序
sort_index()	按索引从小到大排序
sort_index(ascending＝False)	按索引从大到小排序
nlargest(n)	找出元素中最大的前 n 个值
nsmallest(n)	找出元素中最小的前 n 个值

Series 对象的常用方法用法举例如下。

```
>>>import numpy as np
>>>import pandas as pd
>>>s =pd.Series(data=[90,86,np.NaN,95],index=['Java','C','Scala', 'Python'])
>>>s.sum()              #求 s 中元素的和
271.0
```

```
>>>s.var()                          #求 s 中元素的方差
20.333333333333332
>>>s.drop('Scala')                  #删除索引为'Scala'的元素
Java      90.0
C         86.0
Python    95.0
>>>s.dropna()                       #删除空值
Java      90.0
C         86.0
Python    95.0
>>>s.fillna(value=100)              #填充空值
Java      90.0
C         86.0
Scala    100.0
Python    95.0
>>>s.mask(s>88,99)                  #将大于 88 的值替换为 99
Java      99.0
C         86.0
Scala      NaN
Python    99.0
s.map({})
>>>import math
>>>s.apply(math.sqrt)
Java      9.486833
C         9.273618
Scala          NaN
Python    9.746794
>>>s.sort_values()                  #按值从小到大排序
C         86.0
Java      90.0
Python    95.0
Scala      NaN
>>>s.sort_index()                   #按索引从小到大排序
C         86.0
Java      90.0
Python    95.0
Scala      NaN
>>>s.nlargest(2)                    #找出最大的前两个值
Python    95.0
Java      90.0
>>> ser3 = pd.Series(data= ['apple', 'banana', 'apple', 'orange', 'apple',
'orange', 'orange'])
>>>ser3.value_counts()               #统计每个值重复的次数
apple     3
orange    3
banana    1
>>>ser3.unique()                    #获得去重之后的 ndarray 对象
array(['apple', 'banana', 'orange'], dtype=object)
>>>ser3.duplicated()                 #找出重复的数据
0    False
```

```
1    False
2     True
3    False
4     True
5     True
6     True
>>>ser3.drop_duplicates()                                    #删除重复的数据
0     apple
1    banana
3    orange
>>>ser3.map({'apple':'Apple','orange':'Orange'})             #修改元素值
0     Apple
1       NaN
2     Apple
3    Orange
4     Apple
5    Orange
6    Orange
```

3.1.5　Series 对象的运算

1. 算术运算

适用于 NumPy 数组的运算符(+、-、*、/)或其他数学函数，也适用于 Series 对象。可以将 Series 对象的数据数组与标量进行+、-、*、/等算术运算。

```
>>>s3 +2
科目
Java      92
C         98
Python    97
```

2. Series 对象之间的运算

Series 对象之间也可进行+、-、*、/等运算，不同 Series 对象运算的时候，能够通过识别索引进行匹配计算，即只有索引相同的元素才会进行相应的运算操作。

```
>>>s5=pd.Series([10,20],index=['c','d'])
>>>s6=pd.Series([2,4,6,8],index=['a','b','c','d'])
>>>s5+s6                #索引相同的元素相加
a     NaN
b     NaN
c    16.0
d    28.0
dtype: float64
```

上述运算得到一个新 Series 对象，其中只有索引相同的元素才求和，其他只属于任何一个 Series 对象的索引也被添加到新对象中，只不过它们的值为 NaN。NaN(Not a Number，非数值)，pandas 数据结构中若字段为空或者不符合数字的定义时，就用这个特定的值来表示。通常来讲，NaN 表示的数据是有问题的，如从某些数据源抽取数据时遇到了问题，数据源缺失也会产生 NaN 值这类数据。此外，对一个负数求平方根、求对数时，也可能产生这样

的结果。

3.2 DataFrame 结构

3.2.1 创建 DataFrame

DataFrame 是一个表格型的数据结构(可看作一个二维表格),DataFrame 对象既有行索引(行标签,保存在 index 中)又有列索引(列名、列标签,保存在 columns 中),是 Series 结构从一维到多维的扩展。DataFrame 对象每列相同位置处的元素共用一个行索引,每行相同位置处的元素共用一个列索引。DataFrame 对象各列的数据类型可以不相同。

DataFrame 对象的内部组成如图 3-1 所示。

columns

index	course	scores
0	'C'	80
1	'Java'	96
2	'Python'	90
3	'Hadoop'	88

图 3-1 DataFrame 对象的内部组成

DataFrame 对象可以理解为一个由多个 Series 对象组成的字典,其中每一列的列名作为字典的键,形成 DataFrame 列的 Series 对象的数据数组作为字典的值。

创建 DataFrame 对象最常用的方法是使用 pandas 的 DataFrame()构造函数,其语法格式如下。

```
DataFrame(data=None, index=None, columns=None, dtype=None)
```

返回值:DataFrame 对象。

参数说明如下。

data:创建 DataFrame 对象的数据,其类型可以是字典、嵌套列表、元组列表、NumPy 的 ndarray 对象、其他 DataFrame 对象。

index:行索引,创建 DataFrame 对象的数据时,如果没有提供行索引,默认使用从 0 开始依次递增的整数值作为索引。

columns:列索引,若没有提供索引列,默认使用从 0 开始依次递增的整数值作为索引。

dtype:用来指定元素的数据类型,如果为空,自动推断类型。

1. 使用值为列表的字典创建 DataFrame 对象

可将一个值为列表的字典对象传递给 DataFrame()函数来生成一个 DataFrame 对象,字典的键作为 DataFrame 对象的列索引,字典的值作为列索引对应的列值,pandas 也会自动为其添加一列从 0 开始递增的整数值作为行索引,也可以指定行索引。

```
>>>import pandas as pd
```

```
>>>data={'C':[86,90,87,95],'Python':[92,89,89,96],'DataMining': [90,91,89,86]}
>>>studentDF=pd.DataFrame(data,index=['LiQian','WangLi','YangXue', 'LiuTao'])
>>>studentDF          #查看 studentDF 中的数据
          C  DataMining  Python
LiQian   86          90      92
WangLi   90          91      89
YangXue  87          89      89
LiuTao   95          86      96
```

可以只选择字典对象的一部分数据来创建 DataFrame 对象，只需在 DataFrame 构造函数中，用 columns 选项指定需要的字典的键（也就是列）即可，新建的 DataFrame 对象各列顺序与指定的键的顺序一致。

```
>>>studentDF1=pd.DataFrame (data,columns=['C','DataMining'])   #指定需要的列
>>>studentDF1
    C  DataMining
0  86          90
1  90          91
2  87          89
3  95          86
```

2. 使用嵌套列表对象创建 DataFrame 对象

创建 DataFrame 对象时，可以同时指定行索引和列索引，这时候就需要传递三个参数给 DataFrame()构造函数，三个参数的顺序是：数据、index 选项和 columns 选项。或将存放数据的对象赋给 data 选项，将存放行索引的列表赋给 index 选项，将存放列索引的列表赋给 columns 选项。使用嵌套列表对象创建 DataFrame 对象，嵌套列表的每个子列表作为创建的 DataFrame 对象的一列。

```
>>>studentDF2=pd.DataFrame ([[78,82,93],[85,86,97],[90,92,81]], ['ZhangSan',
'LiHua','WangQiang'],['Python','Java','C'])
>>>studentDF2      #查看 studentDF2 中的数据
           Python  Java   C
ZhangSan       78    82  93
LiHua          85    86  97
WangQiang      90    92  81
>>>studentDF3=pd.DataFrame([[78,82,93],[85,86,97],[90,92,81]])#默认行列索引
>>>studentDF3
    0   1   2
0  78  82  93
1  85  86  97
2  90  92  81
```

3. 使用字典的字典或 Series 的字典创建 DataFrame 对象

以字典的字典或 Series 的字典创建 DataFrame 对象，pandas 会将外边的键解释成列名称，将里面的键解释成行索引。

```
>>>data={ "name":{'one':"Jack",'two':"Mary",'three':"John",'four':"Alice"},
          "age":{'one':10,'two':20,'three':30,'four':40},
          "weight":{'one':30,'two':40,'three':50,'four':65}}
```

```
>>>df2=pd.DataFrame(data)
>>>df2
        name  age  weight
one     Jack   10      30
two     Mary   20      40
three   John   30      50
four   Alice   40      65
```

4. 使用字典列表创建 DataFrame 对象

使用字典列表创建 DataFrame 对象，每个字典代表一条记录(DataFrame 中的一行)，字典中各个键对应的值对应这条记录的字段值。

```
>>>data2 =[{'Name' :'李华','Age':18},{'Name' : '李强','Age':19},{'Name' : '李涛','Age':20}]
>>>df5 =pd.DataFrame(data2,index=['ID1','ID2','ID3'])
>>>df5
     Age   Name
ID1   18   李华
ID2   19   李强
ID3   20   李涛
```

3.2.2 DataFrame 对象的属性

DataFrame 对象的常用属性如表 3-4 所示。

表 3-4 DataFrame 对象的常用属性

属 性 名	功 能 描 述
T	DataFrame 的行列转置
columns	得到 DataFrame 的列标签，得到各列的名称
dtypes	查看各列的数据类型
index	得到 DataFrame 的行索引(标签)
shape	返回 DataFrame 的形状(维度)
size	返回 DataFrame 的元素个数，为行数、列数大小的乘积
values	以 NumPy 的 ndarray 数组形式返回 DataFrame 中的实际数
index.name	返回 DataFrame 的行索引的名称
columns.name	返回 DataFrame 的列索引的名称
loc	按标签(列名和行索引)取值，还可以指定返回的列
iloc	按数字索引访问，均支持单值访问或切片查询

```
>>>studentDF       #查看 studentDF 中的数据,该对象在前面已经创建
          C  DataMining  Python
LiQian   86          90      92
WangLi   90          91      89
YangXue  87          89      89
```

```
LiuTao   95         86      96
>>>studentDF.values                #获取存储在 studentDF 中的数据
array([[86, 90, 92],
       [90, 91, 89],
       [87, 89, 89],
       [95, 86, 96]], dtype=int64)
#获取 DataFrame 对象某一列的内容
>>>studentDF['Python']
LiQian     92
WangLi     89
YangXue    89
LiuTao     96
Name: Python, dtype: int64
#也可以通过将列名称作为 DataFrame 对象的属性来获取该列的内容
>>>studentDF.Python
LiQian     92
WangLi     89
YangXue    89
LiuTao     96
Name: Python, dtype: int64
>>>studentDF.loc['YangXue']     #通过行索引获取行数据
C             87
DataMining    89
Python        89
Name: YangXue, dtype: int64
>>>studentDF.loc[['YangXue','LiuTao']] #通过行索引列表查看两行数据
          C  DataMining  Python
YangXue  87          89      89
LiuTao   95          86      96
>>>studentDF.iloc[0:2,:2]    #获取前 2 行前 2 列的数据
         C  DataMining
LiQian  86          90
WangLi  90          91
```

3.3　读取、修改和删除 DataFrame 的数据

3.3.1　读取 DataFrame 对象中的数据

要想获取存储在 DataFrame 对象中的一个数据，需要依次指定数据所在的列名称、行名称(索引)。

```
>>>import pandas as pd
>>>data={ "name":["Jack","Mary","John","Alice"],
        "age":[10,20,30,40],
        "weight":[30,40,55,65] }
>>>df=pd.DataFrame(data)
>>>df
   age   name  weight
```

```
0   10   Jack      30
1   20   Mary      40
2   30   John      55
3   40  Alice      65
>>>df['age'][1]
20
```

可以通过指定条件筛选 DataFrame 对象的元素。

```
>>>df[df.weight>35]     #获取 weight 大于 35 的行
   age   name  weight
1   20   Mary      40
2   30   John      55
3   40  Alice      65
>>>df[df>35]           #获取 DataFrame 对象中数值大于 35 的所有元素
    age   name  weight
0   NaN   Jack     NaN
1   NaN   Mary    40.0
2   NaN   John    55.0
3  40.0  Alice    65.0
```

返回的 DataFrame 对象只包含所有大于 35 的数值,各元素的位置保持不变,不符合条件的数值元素被替换为 NaN。

3.3.2 修改 DataFrame 对象中的数据

可以用 DataFrame 对象的 name 属性为 DataFrame 对象的列索引 columns 和行索引 index 指定别的名称,以便于识别。

```
>>>df.index.name='id'
>>>df.columns.name='item'
>>>df
item  age   name  weight
id
0      10   Jack      30
1      20   Mary      40
2      30   John      55
3      40  Alice      65
```

可以为 DataFrame 对象添加新的列,指定新列的名称,以及为新列赋值。

```
>>>df['new']=10    #添加新列,并将新列的所有元素都赋值为 10
>>>df
item  age   name  weight  new
id
0      10   Jack      30   10
1      20   Mary      40   10
2      30   John      55   10
3      40  Alice      65   10
```

从显示结果可以看出,DataFrame 对象新增了名称为“new”的列,它的各个元素都为 10。

如果想更新一列的内容，需要把一列数赋给这一列。

```
>>>df['new']=[11,12,13,14]
>>>df
item  age   name  weight  new
id
0      10   Jack      30   11
1      20   Mary      40   12
2      30   John      55   13
3      40  Alice      65   14
```

修改单个元素的方法是：选择元素，为其赋新值即可。

```
>>>df['weight'][0]=25
>>>df
item  age   name  weight  new
id
0      10   Jack      25   11
1      20   Mary      40   12
2      30   John      55   13
3      40  Alice      65   15
```

使用 append()函数将新行添加到 DataFrame 对象中。

```
>>>df1 =pd.DataFrame([[1, 2], [3, 4]], columns =['a','b'])
>>>df2 =pd.DataFrame([[5, 6], [7, 8]], columns =['a','b'])
>>>df3 =df1.append(df2)
>>>df3
   a  b
0  1  2
1  3  4
0  5  6
1  7  8
```

3.3.3　删除 DataFrame 对象中的数据

调用 DataFrame 对象的 drop()删除 DataFrame 对象的行或列，具体语法格式如下。

```
df.drop(labels=None, axis=0, index=None, columns=None, inplace=False)
```

参数说明如下。

labels：要删除的行、列的名字。

axis：默认为 0，指删除行，因此删除 columns 时要指定 axis=1。

index：直接指定要删除的行。

columns：直接指定要删除的列。

inplace=False，默认该删除操作不改变原数据，而是返回一个执行删除操作后的新 DataFrame 对象；inplace=True，则会直接在原数据上进行删除操作。

1. 删除指定的行

```
>>>df.drop(labels=0, axis=0)      #删除行索引为 0 的行
    name  age  weight
1   Mary   20      40
```

```
2   John   30      55
3  Alice   40      65
>>>df.drop(index=0)              #删除行索引为 0 的行
    name  age  weight
1   Mary   20      40
2   John   30      55
3  Alice   40      65
>>>df.drop(index=[0,1])          #删除行索引为 0、1 的行
    name  age  weight
2   John   30      55
3  Alice   40      65
>>>df.drop(0)                    #删除行索引为 0 的行
    name  age  weight
1   Mary   20      40
2   John   30      55
3  Alice   40      65
>>>df.drop([0,1,2])              #删除行索引为 0、1、2 的行
    name  age  weight
3  Alice   40      65
>>>df.drop(df [df ['age']==40].index)        #删除 age 值为 40 的行
   name  age  weight
0  Jack   10      30
1  Mary   20      40
2  John   30      55
```

2. 删除指定的列

```
>>>df.drop(columns='age')           #删除列名为 age 的列
    name  weight
0   Jack      30
1   Mary      40
2   John      55
3  Alice      65
>>>df.drop(labels='age', axis=1)    #删除列名为 age 的列
    name  weight
0   Jack      30
1   Mary      40
2   John      55
3  Alice      65
```

通过 del 命令直接在 DataFrame 对象上进行删除操作，删除一列的命令如下。

```
>>>del df['age']                   #删除 df 的 age 列
>>>df
    name  weight
0   Jack      30
1   Mary      40
2   John      55
3  Alice      65
```

3.4　检查 DataFrame 对象是否包含指定的值

可通过 DataFrame 对象的方法 isin()检查 DataFrame 是否包含指定的值。isin()的语法格式如下。

```
df.isin(values)
```

参数说明如下。

values：指定要检查的值，可以是一个列表、字典、Series、DataFrame 对象。

返回值：返回与原始 DataFrame 表格结构相同的值为布尔值的 DataFrame 对象，但如果该值是指定值之一，则原始值已替换为 True，否则为 False。可以使用布尔值的 DataFrame 对象对 DataFrame 对象进行筛选，获取 DataFrame 对象中布尔值为 True 的数据。

```
>>>df
    name  age  weight
0   Jack   10      30
1   Mary   20      40
2   John   30      55
3  Alice   40      65
>>>df.isin(['Jack',30])                    #检查 df 是否包含 ['Jack',30]中的值
    name    age  weight
0   True  False    True
1  False  False   False
2  False   True   False
3  False  False   False
>>>df[df.isin(['Jack',30])]                #使用 df.isin(['Jack',30])筛选数据
   name   age  weight
0  Jack   NaN    30.0
1   NaN   NaN     NaN
2   NaN  30.0     NaN
3   NaN   NaN     NaN
#检查 df 的 name 列是否包含 ['Jack','John']中的值
>>>df['name'].isin(['Jack','John'])
0     True
1    False
2     True
3    False
Name: name, dtype: bool
>>>df[df['name'].isin(['Jack','John'])]   #获取 df 满足 name 条件的行
   name  age  weight
0  Jack   10      30
2  John   30      55
>>>filt =~df['name'].isin(['Jack','John'])     #通过~取反操作获取不满足条件的行
>>>df[filt]
    name  age  weight
1   Mary   20      40
3  Alice   40      65
```

习　　题

1. 概述 DataFrame 对象的属性。
2. 概述删除 DataFrame 对象中的数据的方法。