Pandas基础

郑钦元

北京交通大学

目录

一. pandas库介绍

什么是pandas?

- pandas是python的第三方库，提供高性能，易用数据类型和分析工具
- pandas基于Numpy实现，常与Numpy和Matplotlib一同使用

pandas与Numpy区别

Numpy	pandas
作用：数据处理	作用：数值计算
基础数据类型	扩展数据类型
关注数据的结构表达	关注数据的应用表达
维度：数据间的关系	数据与索引间的关系

pandas引入

```
import pandas as pd
```

pandas主要数据结构

- pandas主要有两种数据结构：Series和DataFrame
- 引入: from pandas import Series , DataFrame
- Series是由一组数据及与之相关的数据索引组成，类似于一个带“标签”的数组，基本操作类似ndarray和字典
- DataFrame是由共用相同索引的一组列组成，主要用于表示二维数组
- 基于两种数据类型的操作：

  基本操作、运算操作、特征类操作、关联类操作

二. pandas数据结构——Series及常用操作

- 创建Series(映射、键值对)

  语法:

  ```
  Series(data,index=index,name=name)
  ```

- Series类型由如下类型创建
  - Python列表：index与列表元素个数一致
  - 标量值：index表达Series类型的尺寸
  - Python字典：键值对中的“键”是索引，index从字典中进行选择操作
  - ndarray:索引和数据都可以通过ndarray类型创建
  - 其他函数：range（）函数等

（1）使用python列表创建

例子

```
# 使用列表创建 Series

import numpy as np
import pandas as pd

series1 = pd.Series([1,3,8,10,123])
print(str.format("根据列表创建Series1:\n{0}",series1))

series2 = pd.Series([1,3,8,10,123],index=['a','b','c','d','e'])
print(str.format("根据列表创建Series2:\n{0}",series2))
```

执行结果

```
根据列表创建Series1:
0      1
1      3
2      8
3     10
4    123
dtype: int64
根据列表创建Series2:
a      1
b      3
c      8
d     10
e    123
dtype: int64
```

（2）使用标量值创建

- 第一个参数作为标量，index的值作为索引
- 每个索引对应的都是同一个标量的值

例子

```
# 使用标量值创建
import numpy as np
import pandas as pd

ser3=pd.Series(30,index=['a','b','c'])

print(str.format("使用标量值创建:\n{0}",ser3))
```

执行结果

```
使用标量值创建:
a    30
b    30
c    30
dtype: int64
```

（3）使用字典类型创建

字典的键作为Series对象的索引

例子

```
# 使用字典类型创建

import numpy as np
import pandas as pd


ser4=pd.Series({'a':10,'b':11,'c':12})
print(str.format("使用字典类型创建Series：\n{0}",ser4))
```

执行结果

```
使用字典类型创建Series：
a    10
b    11
c    12
dtype: int64
```

（4）使用ndarray创建

例子

```
# 使用ndarray创建
import numpy as np
import  pandas as pd

ser6 = pd.Series(np.arange(5))

print(str.format("使用ndarray创建Series:\n{0}",ser6))

ser6 = pd.Series(np.arange(5),index=['a','b','c','d','e'])

print(str.format("使用ndarray创建Series:\n{0}",ser6))
```

执行结果

```
使用ndarray创建Series:
0    0
1    1
2    2
3    3
4    4
dtype: int32
使用ndarray创建Series:
a    0
b    1
c    2
d    3
e    4
dtype: int32
```

(5) Series的访问

- Series对象的index和values属性
  - s.index获得索引列表
  - s.values获得数据列表
- Series对象本身及其索引的name属性
- 访问Series对象的数据
  - 位置索引访问
  - 标签索引访问
  - 切片索引访问
  - 布尔型访问（条件过滤访问）

例子

```
import numpy as np
import  pandas as pd

ser = pd.Series(np.arange(4),index=['a','b','c','d'])

print(str.format("通过index打印索引: \n {0}",ser.index))
print(str.format("通过value打印值: \n {0}",ser.values))
```

```
print(str.format("值的类型为:\n{0}",type(ser.values)))

print("#####################################")

# Series对象本身及其索引的name属性

ser.name="序列对象"
ser.index.name="索引"

print(ser)

print("#####################################")

# 通过位置访问
print("通过位置访问: ",ser[2])
# 通过标签值访问
print("通过标签值访问:",ser['d'])
# 通过切片访问
print("通过切片访问: ",ser[:2])
```

执行结果

```
通过index打印索引:
 Index(['a', 'b', 'c', 'd'], dtype='object')
通过value打印值:
 [0 1 2 3]
值的类型为:
<class 'numpy.ndarray'>
#####################################
索引
a    0
b    1
c    2
d    3
Name: 序列对象, dtype: int32
#####################################
通过位置访问:  2
通过标签值访问: 3
通过切片访问:  索引
a    0
b    1
Name: 序列对象, dtype: int32
```

---

(6) 删除Series中的数据

- drop函数：返回删除后的副本，但源数据不变
- pop函数：返回删除的数据，源数据改变

三. pandas数据结构——DataFrame及其常用操作

1. DataFrame创建

- DataFrame类型由共同索引的一组列组成
- DataFrame是一个表格型的数据类型，每列值类型可以不同。
- DataFrame常用于表达二维数据，也可以表达多维数据
- DataFrame既有行索引(index)，也有列索引(column)

（1）二维ndarray对象创建

例子

```
# 创建DataFrame
import numpy as np
import pandas as pd

# 二维ndarray对象创建
# 没有指定行和列索引，自动索引

df1 = pd.DataFrame(np.random.randint(10,size=(2,5)))
print(str.format("自动行列索引:\n{0}",df1))

# 指定行列索引
df2 = pd.DataFrame(np.random.randint(10,size=
(2,5)),index=list('ab'),columns=list('ABCDE'))
print(str.format("指定行列索引:\n{0}",df2))
```

执行结果

```
自动行列索引:
   0  1  2  3  4
0  4  8  1  9  6
1  1  0  6  4  7
指定行列索引:
   A  B  C  D  E
a  5  9  7  4  6
b  9  4  9  4  0
```

（2）使用列表创建

例子

```
import numpy as np
import pandas as pd

# 列表创建
df3 = pd.DataFrame([0,1,2,3,4])
print(str.format("一维列表创建:\n{0}",df3))

df4 = pd.DataFrame([ [1,2,3], [4,5,6] ])
print(str.format("二维列表创建:\n{0}",df4))
```

执行结果

```
一维列表创建:
   0
0  0
1  1
2  2
3  3
4  4
二维列表创建:
   0  1  2
0  1  2  3
1  4  5  6
```

（3）使用字典创建

- 字典的key成为列索引，行索引是字典中指定的index或自动索引
- 在创建的时候，也可以显示指定的index和columns

例子

```
import numpy as np
import pandas as pd

# 使用字典创建，字典的键作为列索引，值的长度相同
data = {'name':['张三','李四','王五'],'age':[20,25,35]}

df5 = pd.DataFrame(data)
print(str.format('使用字典创建DataFrame:\n{0}',df5))
```

执行结果

```
使用字典创建DataFrame:
  name  age
0   张三   20
1   李四   25
2   王五   35
```

---

2. DataFrame基本属性操作

- index: 行索引列表
- columns: 列索引列表
- values: 元素
- dtypes: 元素类型
- size: 元素的个数
- ndim: 维度
- shape: 形状

例子

```
import numpy as np
import pandas as pd

# 二维ndarray对象创建
# 没有指定行列索引时、自动索引

df1 = pd.DataFrame(np.random.randint(10,size=(2,5)))
print(str.format("DataFrame对象:\n{0}",df1))

print('行索引: ',df1.index)
print('列索引: ',df1.columns)
print('元素: ',df1.values)
print('元素类型: ',df1.dtypes)
print('维度: ',df1.ndim)
print('形状: ',df1.shape)
print('元素个数: ',df1.size)
```

执行结果

```
DataFrame对象:
   0  1  2  3  4
0  7  8  6  4  9
1  9  7  5  9  2
行索引:  RangeIndex(start=0, stop=2, step=1)
列索引:  RangeIndex(start=0, stop=5, step=1)
元素:  [[7 8 6 4 9]
 [9 7 5 9 2]]
元素类型:  0    int32
1    int32
2    int32
3    int32
4    int32
dtype: object
维度:  2
形状:  (2, 5)
元素个数:  10
```

---

3. 查询访问DataFrame中的数据

- 访问单列数据
- 访问多列数据
  - DataFrame中的每一列都是一个Series对象
- 访问单列多行数据
  - 类似操作Series对象
- 访问某个数据
- 访问多列多行数据
- 访问多行数据
  - `df[:][:5]`
  - `head(rows=5)`:访问前几行，默认5行
  - `tail(rows=5)`:访问后几行，默认5行

例子

```
import numpy as np
import pandas as pd

# 二维ndarray对象创建
# 没有指定行列索引时、自动索引
data = {'goods':['apple','milk','pig','chicken'],'quantity':[2,4,9,10],'price':
[20.0,19.9,11,24.3]}
df1 = pd.DataFrame(data)
print(str.format("DataFrame对象:\n{0}",df1))

print(str.format('访问单列:\n{0}',df1['goods']))

print(str.format('访问多列:\n{0}',df1[ ['goods','quantity'] ]))

print(str.format('访问某个数据:\n{0}',df1['goods'][0]))

print(str.format('访问多列多行数据:\n{0}',df1[ ['goods','quantity'] ][:2]))

print(str.format('访问全部列多行数据:\n{0}',df1[:][:2]))
```

执行结果

```
DataFrame对象:
     goods  quantity  price
0    apple         2   20.0
1     milk         4   19.9
2      pig         9   11.0
3  chicken        10   24.3
访问单列:
0      apple
1       milk
2        pig
3    chicken
Name: goods, dtype: object
访问多列:
     goods  quantity
0    apple         2
1     milk         4
2      pig         9
```

```
3  chicken        10
访问某个数据:
apple
访问多列多行数据:
   goods  quantity
0  apple         2
1   milk         4
访问全部列多行数据:
   goods  quantity  price
0  apple         2   20.0
1   milk         4   19.9
```

---

4. 访问DataFrame中的任意数据

- 按照条件筛选数据
  - and(&与)，or(|或), not(~非),xor(^异或)
- 使用loc切片方法
  - DataFrame.loc(行索引|名称或条件，列索引|名称)
  - 行索引为区间时，前后都闭合
- 使用iloc切片方法
  - DataFrame.ilic[行索引位置，列索引位置]
  - 行索引为区间时，前闭后开

注意: loc更加灵活多变，代码的可读性更高，iloc的代码简洁，但可读性不高，建议多使用loc方法

例子

```
import numpy as np
import pandas as pd

# 二维ndarray对象创建
# 没有指定行列索引时、自动索引
data = {'goods':['apple','milk','pig','chicken'],'quantity':[2,4,9,10],'price':
[20.0,19.9,11,24.3]}
df1 = pd.DataFrame(data)
print(str.format("DataFrame对象:\n{0}",df1))

print(str.format('goods为milk的行数据：\n{0}',df1[ df1.goods=='milk' ]))

# loc切片方法
# 单列
print(df1.loc[:,'goods'])
```

执行结果

```
DataFrame对象:
     goods  quantity  price
0    apple         2   20.0
1     milk         4   19.9
```

```
2      pig         9   11.0
3  chicken        10   24.3
goods为milk的行数据:
  goods  quantity  price
1  milk         4   19.9
0      apple
1       milk
2        pig
3    chicken
Name: goods, dtype: object
```

---

5. 添加数据

- 添加一列
  - 新建一个列索引，并赋予固定或非固定的值，添加到最后一列
- 插入一列
  - df.insert(int=位置，column=列名，value=插入的值)
  - 插入的值可以是固定值、列表、Series
- 添加一行
  - loc[rowIndex]方法添加行到末尾，直接修改源数据
  - append在末尾添加数据，返回副本，源数据不变

例子

```
import numpy as np
import pandas as pd

# 二维ndarray对象创建
# 没有指定行列索引时、自动索引
data = {'goods':['apple','milk','pig','chicken'],'quantity':[2,4,9,10],'price':
[20.0,19.9,11,24.3]}
df1 = pd.DataFrame(data)
print(str.format("DataFrame对象:\n{0}",df1))

df1['total'] = df1['quantity'] * df1['price']

print(str.format('添加新一列:\n{0}',df1))

# 添加新一列
df1['isQualified']=True

print(str.format('添加新一列:\n{0}',df1))

# 插入一列
df1.insert(2,'allQuantity',[123,100,110,150])

print(str.format('插入一列:\n{0}',df1))

# 添加一行

df1.loc[4]=['Bird',11,200,24.5,1223,False]
print(str.format('插入一行:\n{0}',df1))
```

执行结果

```
DataFrame对象:
     goods  quantity  price
0    apple         2   20.0
1     milk         4   19.9
2      pig         9   11.0
3  chicken        10   24.3
添加新一列:
     goods  quantity  price  total
0    apple         2   20.0   40.0
1     milk         4   19.9   79.6
2      pig         9   11.0   99.0
3  chicken        10   24.3  243.0
添加新一列:
     goods  quantity  price  total  isQualified
0    apple         2   20.0   40.0         True
1     milk         4   19.9   79.6         True
2      pig         9   11.0   99.0         True
3  chicken        10   24.3  243.0         True
插入一列:
     goods  quantity  allQuantity  price  total  isQualified
0    apple         2          123   20.0   40.0         True
1     milk         4          100   19.9   79.6         True
2      pig         9          110   11.0   99.0         True
3  chicken        10          150   24.3  243.0         True
插入一行:
     goods  quantity  allQuantity  price   total  isQualified
0    apple         2          123   20.0    40.0         True
1     milk         4          100   19.9    79.6         True
2      pig         9          110   11.0    99.0         True
3  chicken        10          150   24.3   243.0         True
4     Bird        11          200   24.5  1223.0        False
```

---

6. 删除DataFrame中的数据

- 删除列
  - df.drop(labels=列名，axis=1,inplace=True)
  - axis=1 表示列，inplace=True表示改变源数据
- 删除行
  - df.drop[labels=索引，axis=0,inplace=True]
  - axis=0表示行，inplace=True表示改变源数据
- 修改DataFrame中的数据
  - 使用loc方法获取数据并赋值修改

例子

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

```
# 二维ndarray对象创建
# 没有指定行列索引时、自动索引
data = {'goods':['apple','milk','pig','chicken'],'quantity':[2,4,9,10],'price':
[20.0,19.9,11,24.3]}
df1 = pd.DataFrame(data)
df1['total'] = df1['quantity'] * df1['price']
# 添加新一列
df1['isQualified']=True
# 插入一列
df1.insert(2,'allQuantity',[123,100,110,150])
# 添加一行
df1.loc[4]=['Bird',11,200,24.5,1223,False]
##########################################################
print(str.format("DataFrame对象:\n{0}",df1))
# axis=0表示行
df1.drop(labels=4,axis=0,inplace=True)

print('删除一行:\n',df1)

# axis = 1表示一列

df1.drop(labels='isQualified',axis=1,inplace=True)
print('删除一列:\n',df1)
```

执行结果

```
DataFrame对象:
     goods  quantity  allQuantity  price   total  isQualified
0    apple         2          123   20.0    40.0         True
1     milk         4          100   19.9    79.6         True
2      pig         9          110   11.0    99.0         True
3  chicken        10          150   24.3   243.0         True
4     Bird        11          200   24.5  1223.0        False
删除一行:
      goods  quantity  allQuantity  price  total  isQualified
0    apple         2          123   20.0   40.0         True
1     milk         4          100   19.9   79.6         True
2      pig         9          110   11.0   99.0         True
3  chicken        10          150   24.3  243.0         True
删除一列:
      goods  quantity  allQuantity  price  total
0    apple         2          123   20.0   40.0
1     milk         4          100   19.9   79.6
2      pig         9          110   11.0   99.0
3  chicken        10          150   24.3  243.0
```

四. 汇总和计算描述统计

1.数值型数据统计

（1）使用基于Numpy的统计函数

- pandas库基于Numpy,可以使用Numpy提供的函数对数据进行描述性统计
- 例如: np.mean(df['colName'])

方法	说明
sum	对数组全部或某轴向的元素求和 横轴求和: axis=1 纵轴求和：axis=0
mean、median	求算数平均数、中位数
std、var、ptp、cov	求标准差、方差、极差、协方差
min、max	求数组最大值、最小值
argmin、argmax	求最大、最小元素的索引
cumsum	求所有元素的累计和
cumprod	求所有元素的累计积

(2) 使用基于pandas描述性统计方法

- pandas提供了更加便利的统计方法，如detail['列名"].mean()
- describe方法，能够一次性得出DataFrame所有数值型特征的非空值数目、均值、四分位数、标准差

方法名称	说明	方法名称	说明
min	最小值	max	最大值
mean	均值	ptp	极差
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
describe	描述统计	mad	平均绝对离差

例子

```
import numpy as np
import pandas as pd

# 二维ndarray对象创建
# 没有指定行列索引时、自动索引
data = {'goods':['apple','milk','pig','chicken'],'quantity':[2,4,9,10],'price':
[20.0,19.9,11,24.3]}
df1 = pd.DataFrame(data)
print('DataFrame对象: \n',df1)
```

```
# 使用Numpy中的方法统计DataFrame的数据
print('质量总数为(使用Numpy方法): ',np.sum(df1['quantity']))
# 使用Pandas中的方法
print('总质量为(使用Pandas方法):',df1['quantity'].sum())

# 使用pandas的describe方法

print('使用pandas的describe方法: \n',df1[['quantity','price'] ].describe())
```

执行结果

```
DataFrame对象:
      goods  quantity  price
0    apple         2   20.0
1     milk         4   19.9
2      pig         9   11.0
3  chicken        10   24.3
质量总数为(使用Numpy方法):  25
总质量为(使用Pandas方法): 25
使用pandas的describe方法:
        quantity      price
count   4.00000   4.000000
mean    6.25000  18.800000
std     3.86221   5.589872
min     2.00000  11.000000
25%     3.50000  17.675000
50%     6.50000  19.950000
75%     9.25000  21.075000
max    10.00000  24.300000
```

---

2. 类别型数据统计

- 类别型数据：例如男、女

类别型数据统计

- 描述类别型特征的分布状态，可以使用频数统计表
  - pandas库中实现频数统计的方法为value_counts
- describe方法能够支持对category类型的数据进行描述性统计
  - 返回四个统计量，分别为列非空元素的数目、类别的数目、数目最多的类别、数目最多类别的数目
  - pandas提供了category类，可以使用astype方法将目标特征的数据类型转换为category类别

例子

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

```
df = pd.DataFrame({'id':range(4),'name':['jack','tom','mike','jack'],'sex':
['F','F','M','F'],'age':[21,21,23,24]})

print('DataFrame对象:\n',df)

print('实现词频统计:\n',df['name'].value_counts())

# 将数组类型转换为category

df['sex']=df['sex'].astype('category')
print('修改后的数组类型为:\n',df['sex'].dtypes)

# 使用describe实现描述性统计

print('使用pandas的describe视线描述性统计:\n',df['sex'].describe())
```

执行结果

```
DataFrame对象:
    id  name sex  age
0   0  jack   F   21
1   1   tom   F   21
2   2  mike   M   23
3   3  jack   F   24
实现词频统计:
 jack    2
tom     1
mike    1
Name: name, dtype: int64
修改后的数组类型为:
 category
使用pandas的describe视线描述性统计:
 count     4
unique    2
top       F
freq      3
Name: sex, dtype: object
```