Q Search the docs ...

前言

第一章预备知识

第二章 Pandas基础

第三章索引

第四章分组

第五章变形

第六章合并

第七章缺失数据

第八章 文本数据

第九章 分类数据

第十章时序数据

第十一章数据清洗

第十二章 特征工程

第十三章 性能优化

第十四章案例分析

参考答案

第九章 分类数据¶

```
In [1]: import numpy as np
In [2]: import pandas as pd
```

一、cat对象

1. cat对象的属性

在 pandas 中提供了 category 类型,使用户能够处理分类类型的变量,将一个普通序列转换成分类变量可以使用 astype 方法。

```
In [3]: df = pd.read_csv('data/learn_pandas.csv',
             usecols = ['Grade', 'Name', 'Gender', 'Height', 'Weight'])
   . . . :
In [4]: s = df.Grade.astype('category')
In [5]: s.head()
Out[5]:
      Freshman
1
      Freshman
2
        Senior
3
     Sophomore
     Sophomore
Name: Grade, dtype: category
Categories (4, object): ['Freshman', 'Junior', 'Senior', 'Sophomore']
```

在一个分类类型的 Series 中定义了 cat 对象,它和上一章中介绍的 str 对象类似,定义了一些属性和方法来进行分类类别的操作。

```
In [6]: s.cat
Out[6]: <pandas.core.arrays.categorical.CategoricalAccessor object at 0x0000018B64F6F4C8>
```

对于一个具体的分类,有两个组成部分,其一为类别的 Index ,其二为是否有序,它们可以通过 cat 的属性被访问:

```
In [7]: s.cat.categories
Out[7]: Index(['Freshman', 'Junior', 'Senior', 'Sophomore'], dtype='object')
In [8]: s.cat.ordered
Out[8]: False
```

另外,每一个序列的类别会被赋予唯一的整数编号,它们的编号取决于 cat.categories 中的顺序,该属性可以通过 codes 访问:

```
In [9]: s.cat.codes.head()
Out[9]:
0    0
1    0
2    2
3    3
4    3
dtype: int8
```

2. 类别的增加、删除和修改

通过 cat 对象的 categories 属性能够完成对类别的查询,那么应该如何进行"增改查删"的其他三个操作呢?

类别不得直接修改

在第三章中曾提到,索引 Index 类型是无法用 index_obj[0] = item 来修改的,而 categories 被存储在 Index 中,因此 pandas 在 cat 属性上定义了若干方法来达到相同的目的。

首先,对于类别的增加可以使用 add_categories:

```
In [10]: s = s.cat.add_categories('Graduate') # 增加一个毕业生类别
In [11]: s.cat.categories
Out[11]: Index(['Freshman', 'Junior', 'Senior', 'Sophomore', 'Graduate'], dtype='object')
```

若要删除某一个类别可以使用 remove_categories ,同时所有原来序列中的该类会被设置为缺失。例如,删除大一的类别:

```
In [12]: s = s.cat.remove_categories('Freshman')
In [13]: s.cat.categories
Out[13]: Index(['Junior', 'Senior', 'Sophomore', 'Graduate'], dtype='object')
In [14]: s.head()
Out[14]:
0
           NaN
1
           NaN
2
        Senior
3
     Sophomore
    Sophomore
Name: Grade, dtype: category
Categories (4, object): ['Junior', 'Senior', 'Sophomore', 'Graduate']
```

此外可以使用 set_categories 直接设置序列的新类别,原来的类别中如果存在元素不属于新类别,那么会被设置为缺失。

如果想要删除未出现在序列中的类别,可以使用 remove_unused_categories 来实现:

```
In [18]: s = s.cat.remove_unused_categories() # 移除了未出现的博士生类别
In [19]: s.cat.categories
Out[19]: Index(['Sophomore'], dtype='object')
```

最后,"增改查删"中还剩下修改的操作,这可以通过 rename_categories 方法完成,同时需要注意的是,这个方法会对原序列的对应值也进行相应修改。例如,现在把 Sophomore 改成中文的 本科二年级学生:

二、有序分类

1. 序的建立

有序类别和无序类别可以通过 as_unordered 和 reorder_categories 互相转化,需要注意的是后者传入的参数必须是由当前序列的无需类别构成的列表,不能够增加新的类别,也不能缺少原来的类别,并且必须指定参数 ordered=True ,否则方法无效。例如,对年级高低进行相对大小的类别划分,然后再恢复无序状态:

```
In [22]: s = df.Grade.astype('category')
In [23]: s = s.cat.reorder_categories(['Freshman', 'Sophomore',
                                        'Junior', 'Senior'],ordered=True)
   . . . . :
In [24]: s.head()
Out[24]:
      Freshman
1
     Freshman
        Senior
2
3
     Sophomore
     Sophomore
Name: Grade, dtype: category
Categories (4, object): ['Freshman' < 'Sophomore' < 'Junior' < 'Senior']</pre>
In [25]: s.cat.as_unordered().head()
Out[25]:
0
      Freshman
1
      Freshman
2
        Senior
3
     Sophomore
4
    Sophomore
Name: Grade, dtype: category
Categories (4, object): ['Freshman', 'Sophomore', 'Junior', 'Senior']
```

1 类别不得直接修改

如果不想指定 ordered=True 参数,那么可以先用 s.cat.as_ordered() 转化为有序类别,再利用 reorder_categories 进行具体的相对大小调整。

2. 排序和比较

在第二章中,曾提到了字符串和数值类型序列的排序,此时就要说明分类变量的排序:只需把列的类型修改为 category 后,再赋予相应的大小关系,就能正常地使用 sort_index 和 sort_values 。例如,对年级进行排序:

```
In [26]: df.Grade = df.Grade.astype('category')
In [27]: df.Grade = df.Grade.cat.reorder_categories(['Freshman',
                                                   ر'Sophomore'
                                                   'Junior',
                                                   'Senior'], ordered=True)
   . . . . :
   . . . . :
In [28]: df.sort_values('Grade').head() # 值排序
Out[28]:
       Grade
                      Name Gender Height Weight
   Freshman Gaopeng Yang Female 158.9
                                             46.0
                  Qiang Shi Female 164.5
105 Freshman
                                              52.0
   Freshman Changmei Feng Female
    Freshman Xiaopeng Han Female 164.1
                                              53.0
                Yanli Zhang Female 165.1
    Freshman
                                              52.0
In [29]: df.set_index('Grade').sort_index().head() # 索引排序
Out[29]:
                  Name Gender Height Weight
Grade
Freshman
         Gaopeng Yang Female
                                158.9
                                         46.0
Freshman
             Qiang Shi Female
                              164.5
                                         52.0
                                         56.0
Freshman Changmei Feng Female 163.8
         Xiaopeng Han Female
                                         53.0
Freshman
                                164.1
Freshman
           Yanli Zhang Female
                                165.1
                                         52.0
```

由于序的建立,因此就可以进行比较操作。分类变量的比较操作分为两类,第一种是 == 或 != 关系的比较,比较的对象可以是标量或者同长度的 Series(或 list),第二种是 >,>=,<,<= 四类大小关系的比较,比较的对象和第一种类似,但是所有参与比较的元素必须属于原序列的 categories ,同时要和原序列具有相同的索引。

```
In [30]: res1 = df.Grade == 'Sophomore'
In [31]: res1.head()
Out[31]:
0
     False
     False
1
2
    False
3
      True
4
      True
Name: Grade, dtype: bool
In [32]: res2 = df.Grade == ['PhD']*df.shape[0]
In [33]: res2.head()
Out[33]:
0
    False
1
    False
2
    False
3
    False
4
    False
Name: Grade, dtype: bool
In [34]: res3 = df.Grade <= 'Sophomore'</pre>
In [35]: res3.head()
Out[35]:
0
      True
1
      True
2
     False
3
      True
4
      True
Name: Grade, dtype: bool
In [36]: res4 = df.Grade <= df.Grade.sample(</pre>
                                      frac=1).reset_index(
   . . . . :
                                                 drop=True) # 打乱后比较
   . . . . :
   . . . . :
In [37]: res4.head()
Out[37]:
    True
1
     True
2
     True
3
     True
4
     True
Name: Grade, dtype: bool
```

三、区间类别

1. 区间的构造

区间是一种特殊的类别,在实际数据分析中,区间序列往往是通过 cut 和 qcut 方法进行构造的,这两个函数能够把原序列的数值特征进行装箱,即用区间位置来代替原来的具体数值。

首先介绍 cut 的常见用法:

其中,最重要的参数是 bin ,如果传入整数 n ,则代表把整个传入数组的按照最大和最小值等间距地分为 n 段。由于区间默认是左开右闭,需要进行调整把最小值包含进去,在 pandas 中的解决方案是在值最小的区间左端点再减去 0.001*(max-min) ,因此如果对序列 [1,2] 划分为2个箱子时,第一个箱子的范围 (0.999,1.5] ,第二个箱子的范围是 (1.5,2] 。如果需要指定左闭右开时,需要把 right 参数设置为 False ,相应的区间调整方法是在值最大的区间右端点再加上 0.001*(max-min) 。

```
In [38]: s = pd.Series([1,2])
In [39]: pd.cut(s, bins=2)
Out[39]:
0    (0.999, 1.5]
1    (1.5, 2.0]
dtype: category
Categories (2, interval[float64]): [(0.999, 1.5] < (1.5, 2.0]]

In [40]: pd.cut(s, bins=2, right=False)
Out[40]:
0    [1.0, 1.5)
1    [1.5, 2.001)
dtype: category
Categories (2, interval[float64]): [[1.0, 1.5) < [1.5, 2.001)]</pre>
```

bins 的另一个常见用法是指定区间分割点的列表(使用 np.infty 可以表示无穷大):

```
In [41]: pd.cut(s, bins=[-np.infty, 1.2, 1.8, 2.2, np.infty])
Out[41]:
0   (-inf, 1.2]
1   (1.8, 2.2]
dtype: category
Categories (4, interval[float64]): [(-inf, 1.2] < (1.2, 1.8] < (1.8, 2.2] < (2.2, inf]]</pre>
```

另外两个常用参数为 labels 和 retbins,分别代表了区间的名字和是否返回分割点(默认不返回):

从用法上来说,qcut 和 cut 几乎没有差别,只是把 bins 参数变成的 q 参数,qcut 中的 q 是指 quantile 。这里的 q 为整数 n 时,指按照 n 等分位数把数据分箱,还可以传入浮点列表指代相应的分位数分割点。

```
In [46]: s = df.Weight
In [47]: pd.qcut(s, q=3).head()
Out[47]:
     (33.999, 48.0]
1
       (55.0, 97.0]
2
       (55.0, 97.0]
     (33.999, 48.0]
3
       (55.0, 97.0]
Name: Weight, dtype: category
Categories (3, interval[float64]): [(33.999, 48.0] < (48.0, 55.0] < (55.0, 97.0]]
In [48]: pd.qcut(s, q=[0,0.2,0.8,1]).head()
Out[48]:
       (44.0, 69.4]
1
       (69.4, 97.0]
2
       (69.4, 97.0]
     (33.999, 44.0]
       (69.4, 97.0]
Name: Weight, dtype: category
Categories (3, interval[float64]): [(33.999, 44.0] < (44.0, 69.4] < (69.4, 97.0]]
```

2. 区间的属性与方法

IntervalIndex 上定义了一些有用的属性和方法,在此之前需要先转为该种索引类型:

```
In [49]: id_interval = pd.IntervalIndex(pd.cut(s, 3))
```

有四个常用属性: left, right, mid, length, 分别表示左右端点、两端点均值和区间长度。

```
In [50]: id_demo = id_interval[:5] # 选出全5个展示
In [51]: id_demo
Out[51]:
IntervalIndex([(33.937, 55.0], (55.0, 76.0], (76.0, 97.0], (33.937, 55.0], (55.0, 76.0]],
             closed='right',
             name='Weight',
             dtype='interval[float64]')
In [52]: id_demo.left
Out[52]: Float64Index([33.937, 55.0, 76.0, 33.937, 55.0], dtype='float64')
In [53]: id_demo.right
Out[53]: Float64Index([55.0, 76.0, 97.0, 55.0, 76.0], dtype='float64')
In [54]: id_demo.mid
Out[54]: Float64Index([44.4685, 65.5, 86.5, 44.4685, 65.5], dtype='float64')
In [55]: id_demo.length
Out[55]: Float64Index([21.063000000000002, 21.0, 21.0, 21.063000000000002, 21.0],
dtype='float64')
```

IntervalIndex 还有两个常用方法,包括 contains 和 overlaps ,分别指逐个判断每个区间是否包含某元素,以及是否和一个 pd.Interval 对象有交集。

```
In [56]: id_demo.contains(4)
Out[56]: array([False, False, False, False])
In [57]: id_demo.overlaps(pd.Interval(40,60))
Out[57]: array([ True, True, False, True, True])
```

© Copyright 2020, Datawhale, 耿远昊. Created using <u>Sphinx</u> 3.2.1.