

## Python 全栈 400 之Pandas数据分析练习

### 288 Pandas 读取 URL 路径的文件

数据输入路径，可以是文件路径，也可以是 URL，或者实现 read 方法的任意对象。

如下经典的数据集 iris，直接通过 URL 获取。

```
In [160]: pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data')
Out[160]:
   5.1  3.5  1.4  0.2  Iris-setosa
0   4.9  3.0  1.4  0.2  Iris-setosa
1   4.7  3.2  1.3  0.2  Iris-setosa
2   4.6  3.1  1.5  0.2  Iris-setosa
3   5.0  3.6  1.4  0.2  Iris-setosa
4   5.4  3.9  1.7  0.4  Iris-setosa
..    ..    ..    ..    ..
144  6.7  3.0  5.2  2.3  Iris-virginica
145  6.3  2.5  5.0  1.9  Iris-virginica
146  6.5  3.0  5.2  2.0  Iris-virginica
147  6.2  3.4  5.4  2.3  Iris-virginica
148  5.9  3.0  5.1  1.8  Iris-virginica

[149 rows x 5 columns]
```

### 289 Pandas 读取文件之 sep 分隔符

默认为逗号，注意：如果分割字符长度大于1，且不是 \s+，启动 Python 引擎解析。

举例：test.csv 文件分割符为 \t，如果使用 sep 默认的逗号分隔符，读入后的数据混为一体。

```
# 创建并保存数据
In [1]: d = {'id':[1,2], 'name':['gz', 'lh'], 'age':[10,12]}
In [2]: df = pd.DataFrame(d)
In [3]: df.to_csv('test.csv', sep='\t')

# 读取数据
In [4]: df = pd.read_csv('test.csv')
In [5]: df
Out[5]:
   \tid\tname\tage
0      0\t1\tgz\t10
1      1\t2\tlh\t12
```

sep 必须设置为 '\t'，数据分割才会正常。

```
In [6]: df = pd.read_csv('test.csv', sep='\t')
In [6]: df
Out[6]:
   Unnamed: 0  id name age
0           0   1  gz  10
1           1   2  lh  12
```

### 290 Pandas 读取之列选择属性

参数用于选取数据文件的哪些列到 DataFrame 中，如下所示，只想使用源数据文件的 id 和 age 两列，那么可以为 usecols 参数赋值为 ['id','name']：

```
In [1]: df = pd.read_csv('test.csv', delim_whitespace=True, usecols=['id', 'name'])
In [2]: df
Out[2]:
   id name
0   1  gz
1   2  lh
```

### 291 Pandas 读取之空值处理

参数可以配置哪些值需要处理成 Na/NaN，类型为字典，键指明哪一列，值为看做 Na/NaN 的字符。

假设我们的数据文件如下，date 列中有一个 # 值，我们想把它处理成 NaN 值。

```
In [1]: d = {'id':[1,2], 'name':['gz', 'lh'], 'age':[10,12], 'date':['2020-03-10', '#']}
In [2]: df = pd.DataFrame(d)
In [3]: df.to_csv('test_date.csv', sep=' ', index=False)

In [4]: df = pd.read_csv('test_date.csv', sep='\s+')

# 复制
```

可以使用，na\_values 实现：

```
In [37]: df = pd.read_csv('test_date.csv', sep='\s+', na_values=['#'])

In [38]: df
Out[38]:
   id name age      date
0   1  gz  10  2020-03-10
1   2  lh  12         NaN
```

keepdefaultna 是和 navalues 搭配的，如果前者为 True，则 navalues 被解析为 Na/NaN 的字符除了用户设置外，还包括默认值。

### 292 Pandas 之分块读入数据

iterator 取值 boolean，默认为 False，返回一个 TextFileReader 对象，以便逐块处理文件。

这个在文件很大时，内存无法容纳所有数据文件，此时分批读入，依次处理。

具体操作演示如下，我们的文件数据域一共有 2 行。

先读入一行，get\_chunk 设置为 1 表示一次读入一行

[288 Pandas 读取 U...](#)[289 Pandas 读取文...](#)[290 Pandas 读取之...](#)[291 Pandas 读取之...](#)[292 Pandas 之分块...](#)[293 Pandas 之随机...](#)[294 Pandas 之一维...](#)[295 Pandas 之创建 ...](#)[296 Series 之增加...](#)[297 Series 之删除元素](#)[298 Series 之修改...](#)

```
In [64]: chunk = pd.read_csv('test.csv',sep='\\s+',iterator=True)

In [65]: chunk.get_chunk(1)
Out[65]:
   id name age
0    1  gz   10
```

再读入下一行，

```
In [66]: chunk.get_chunk(1)
Out[66]:
   id name age
1    2  lh   12
```

此时已到文件末尾，再次读入会报异常，

```
In [108]: chunk.get_chunk(1)

StopIteration
```

### 293 Pandas 之随机选取一部分数据案例

对于动辄就几十或几百个 G 的数据，在读取的这么大数据的时候，我们有没有办法随机选取一小部分数据，然后读入内存，快速了解数据和开展 EDA ？

使用 Pandas 的 skiprows 和 概率知识，就能做到。

下面解释具体怎么做。

如下所示，读取某 100 G 大小的 big\_data.csv 数据

- 1) 使用 skiprows 参数，
- 2) x > 0 确保首行读入，
- 3) np.random.rand() > 0.01 表示 99% 的数据都会被随机过滤掉

言外之意，只有全部数据的 1% 才有机会选入内存中。

```
import pandas as pd
import numpy as np

df = pd.read_csv("big_data.csv",
skiprows =
lambda x: x>0 and np.random.rand() > 0.01)

print("The shape of the df is {}.\nIt has been reduced 100 times!".format(df.shape))
```

使用这种方法，读取的数据量迅速缩减到原来的 1% ，对于迅速展开数据分析有一定的帮助。

### 294 Pandas 之一维数组 Series

Series 是 pandas 两大数据结构中(DataFrame, Series) 的一种，先从 Series 的定义说起，Series 是一种类似于 一维数组的对象，它由一组数据域以及一组与之相关联的数据标签和索引组成。

Series 对象也是一个 NumPy 的数组，因此 NumPy 的数组处理函数可以直接对 Series 进行处理。

与此同时，Series 除了可以使用位置索引作为下标存取元素之外，还可以使用 标签下标 存取元素，这一点和字典相似，每个 Series 对象都由两个数组组成：

1. index：它是从 NumPy 数组继承的 Index 对象，保存标签信息。
2. values：保存值的 NumPy 数组。

接下来，分别介绍 Series 内元素的增加、删除、修改、访问。

### 295 Pandas 之创建 Series

Series 的标准构造函数 列举常用的几个参数：

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

其中，data 为数据部分，index 为标签部分，省略下默认为自增整数索引，dtype 为 str, numpy.dtype, or ExtensionDtype。

创建一个 series，如下：

```
In [85]: ps = pd.Series(data=[-3,2,1],index=['a','f','b'],dtype=np.float32)

In [86]: ps

Out[86]:
a   -3.0
f    2.0
b    1.0
dtype: float32
```

### 296 Series 之增加元素

在 ps 基础上增加一个元素，使用 append，如下：

```
In [112]: ps.append(pd.Series(data=[-8.0],index=['f']))

Out[112]:
a    4.0
f    2.0
b    1.0
f   -8.0
dtype: float64
```

可以看到，Pandas 允许包含重复的标签

复制

```
In [114]: psn = ps.append(pd.Series(data=[-8.0],index=['f']))

In [115]: psn

Out[115]:
a    4.0
f    2.0
b    1.0
f   -8.0
dtype: float64

In [116]: psn['f']

Out[116]:
f    2.0
f   -8.0
dtype: float64
```

利用标签访问元素，返回所有带标签的数据。

#### 297 Series之删除元素

使用 drop 删除指定标签的数据，如下：

复制

```
In [119]: ps

Out[119]:
a    4.0
f    2.0
b    1.0
dtype: float32

In [120]: psd = ps.drop('f')

In [121]: psd

Out[121]:
a    4.0
b    1.0
dtype: float32
```

注意不管是 append 操作，还是 drop 操作，都是发生在原数据的副本上，不是原数据上。

#### 298 Series 之修改元素

通过标签修改对应数据，如下所示：

复制

```
In [123]: psn

Out[123]:
a    4.0
f    2.0
b    1.0
f   -8.0
dtype: float64

In [124]: psn['f'] = 10.0

In [125]: psn

Out[125]:
a    4.0
f   10.0
b    1.0
f   10.0
dtype: float64
```

标签相同的数据，都会被修改。

#### 299 Series 之访问元素

访问元素，Pandas 提供两种方法，

- 一种通过默认的整数索引，在 Series 对象未被显示的指定 label 时，都是通过索引访问；
- 另一种方式是通过标签访问。

复制

```
In [126]: ps

Out[126]:
a    4.0
f    2.0
b    1.0
dtype: float32

In [128]: ps[2] # 索引访问
Out[128]: 1.0

In [127]: ps['b'] # 标签访问
Out[127]: 1.0
```

#### 300 Pandas 之二维数组 DataFrame

DataFrame，Pandas 两个重要数据结构中的另一个，可以看做是 Series 的容器。

DataFrame 同时具有行、列标签，是二维的数组，行方向轴 axis 为 0，列方向 axis 为 1，如下：

复制

```
axis : {0 or 'index', 1 or 'columns'}
```

##### 创建 DataFrame

DataFrame 构造函数如下：

复制

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

参数意义与 Series 相似，不再赘述。

创建 DataFrame 的常用方法：

复制

```
In [134]: df = pd.DataFrame([[ 'gz',4.0,'2019-01-01'],[ 'lg',1.2,'2019-06-0
```

```
1']]),index = ['a','f'], columns = ['nm', 'y','da'])
```

```
In [135]: df

Out[135]:
   nm    y      da
a  gz  4.0  2019-01-01
f  lg  1.2  2019-06-01
```

也可以通过字典传入，得到一样的 DataFrame，如下：

```
In [136]: df2 = pd.DataFrame({'nm':['gz','lg'],'y':[4.0,1.2], 'da':['2019-01-01', '2019-06-01']},index = ['a','f'])

In [137]: df2

Out[137]:
   nm    y      da
a  gz  4.0  2019-01-01
f  lg  1.2  2019-06-01
```

### 301 DataFrame 之增加数据

通过增加一个 Series，扩充到 DataFrame 中，如下所示：

```
In [143]: dfn = df.append(pd.Series(data=['zx',3.6,'2019-05-01'],index=['nm','y','da'],name='b'))

In [144]: dfn

Out[144]:
   nm    y      da
a  gz  4.0  2019-01-01
f  lg  1.2  2019-06-01
b  zx  3.6  2019-05-01
```

Series 的 index 与 DataFrame 的 column 对齐，name 与 DataFrame 的 index 对齐。

### 302 DataFrame 之删除数据

与 Series 删除类似，也是使用 drop 删除指定索引或标签的数据。如下，注意删除仍然是在 dfn 的副本上进行，像下面这样删除对 dfn 没有任何影响。

```
In [145]: dfn.drop('b')

Out[145]:
   nm    y      da
a  gz  4.0  2019-01-01
f  lg  1.2  2019-06-01
```

如果要删除某列，需要设定 axis 为 1，如下所示：

```
In [147]: dfn.drop('y',axis=1)

Out[147]:
   nm      da
a  gz  2019-01-01
f  lg  2019-06-01
b  zx  2019-05-01
```

### 303 DataFrame 之修改数据

修改依然是先通过索引或标签定位到数据，然后修改，如下所示：

```
In [151]: dfn.loc['a','da']='2019-04-01'

In [152]: dfn

Out[152]:
   nm    y      da
a  gz  4.0  2019-04-01
f  lg  1.2  2019-06-01
b  zx  3.6  2019-05-01
```

作为入门或许理解到这里就够了，但要想进阶，还必须要了解一个关键点：链式赋值( chained assignment)。

使用 Pandas 链式赋值，经常会触发一个警告：SettingWithCopyWarning，类似下面一串警告：

```
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
See the caveats in the documentation
```

归根结底，是因为代码中出现链式操作，那么什么是链式操作？如下，就是一次链式赋值：

```
tmp = df[df.a<4]
tmp['c'] = 200
```

出现此 Warning，需要理会它吗？还是需要！

```
import pandas as pd

df = pd.DataFrame({'a':[1,3,5], 'b':[4,2,7], 'c':[0,3,1]})
print(df)

tmp = df[df.a<4]
tmp['c'] = 200
print('-----原 df 没有改变-----')
print(df)
```

输出结果：

```
   a  b  c
0  1  4  0
1  3  2  3
```

```
2 5 7 1
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
tmp['c'] = 200
-----原 df 没有改变-----
   a  b  c
0  1  4  0
1  3  2  3
2  5  7  1
```

以上发生链式赋值，本意想改变 df，但是 df 未发生任何变化。因为 tmp 是 df 的 copy，而非 view。

因此，如何创建 df 的 view，而确保不发生链式操作，建议使用 .loc，如下调用：

```
import pandas as pd

df = pd.DataFrame({'a':[1,3,5], 'b':[4,2,7], 'c':[0,3,1]})
print(df)

df.loc[df.a<4, 'c'] = 100
print(df)
```

输出结果：

```
   a  b  c
0  1  4  0
1  3  2  3
2  5  7  1
   a  b  c
0  1  4 100
1  3  2 100
2  5  7   1
```

### 304 Pandas 之访问数据

Pandas 推荐使用访问接口 iloc、loc 访问数据，详细使用如下：

```
In [153]: df

Out[153]:
   nm    y      da
a  gz  4.0  2019-01-01
f  lg  1.2  2019-06-01

In [154]: df.iloc[1,:]

Out[154]:
nm          lg
y          1.2
da  2019-06-01
Name: f, dtype: object

In [155]: df.loc['f',:]

Out[155]:
nm          lg
y          1.2
da  2019-06-01
Name: f, dtype: object

In [156]: df.loc['f', 'nm']

Out[156]: 'lg'
```

iloc 索引访问，loc 标签访问。

### 305 Pandas 强大的 [] 操作符

原生 Python 中，[] 操作符常见的是与 list 搭配使用，并且 [] 操作符支持的对象只能是：整型，切片，例如：

```
a= [1, 3, 5]
# 以下OK:
a[1]
a[True]
a[:2]
a[:6]
```

list 等可迭代对象是禁止的，如下：

```
# 不OK:
a[[1,2]]
```

为了更好地发挥 Python 的强大威力，Pandas 的 [] 操作符支持 list 对象。

为了演示，生成一个 DataFrame 实例 df1

```
import pandas as pd
import numpy as np

np.random.seed(1)
df1=pd.DataFrame(np.random.randint(1,10,(4,3)),
index=['r1','r2','r3','r4'],
columns=['c1','c2','c3'])

#结果
   c1  c2  c3
r1  6  9  6
r2  1  1  2
r3  8  7  3
r4  5  6  3
```

[] 操作符支持 list 对象，对 c1 和 c3 列索引：

```
df1[['c1','c3']]
#测试支持列索引
   c1  c3
```

```
r1  6  6
r2  1  2
r3  8  3
r4  5  3
```

注意这种支持也仅仅是对列索引；对于行索引，依然不支持，如下：

```
df1[['r1', 'r4']]
#测试结果,会出现以下报错:
KeyError: "None of [Index(['r1', 'r4'], dtype='object')] are in the [columns]"
```

利用行切片整数索引，获取 DataFrame，Pandas 依然延续了原生 Python 的风格，如下：

```
In [7]: df1[1:3]
Out[7]:
   c1  c2  c3
r2  1  1  2
r3  8  7  3
```

Pandas 还对此做了增强，同时支持行切片标签索引获取 DataFrame，**注意包括终止标签**。

```
In [9]: df1['r1':'r3']
Out[9]:
   c1  c2  c3
r1  6  9  6
r2  1  1  2
r3  8  7  3
```

[] 操作符除了支持以上对象类型外，还支持哪些对象类型？

306 DataFrame 与标量结合

仅仅一行 `df1>4` 就能实现逐元素与 4 比较，大于 4 为 True，否则为 False。

然后返回一个新的 DataFrame，如下所示：

```
In [10]: df1 > 4
Out[10]:
   c1  c2  c3
r1  True  True  True
r2  False False False
r3  True  True  False
r4  True  True  False
```

结合 Pandas 的方括号 [] 后，便能实现元素级的操作，不大于 4 的元素被赋值为 NaN：

```
In [11]: df1[df1>4]
Out[11]:
   c1  c2  c3
r1  6.0  9.0  6.0
r2  NaN  NaN  NaN
r3  8.0  7.0  NaN
r4  5.0  6.0  NaN
```

`df1>4` 操作返回一个元素值为 True 或 False 的 DataFrame

`(df1>4).values.tolist()` 得到原生的 python list 对象：

```
In [15]: inner = ((df1>4).values).tolist()

In [16]: inner
Out[16]:
[[True, True, True],
 [False, False, False],
 [True, True, False],
 [True, True, False]]

In [17]: type(inner)
Out[17]: list
```

DataFrame, [] 和 list 实例结合会出现如下的结果：

```
In [18]: df1[inner]
Out[18]:
   c1  c2  c3
r1  6  9  6
r1  6  9  6
r1  6  9  6
r3  8  7  3
r3  8  7  3
r4  5  6  3
r4  5  6  3
```

仅仅改变为 `df1[python list]`，而不是原来的 `df1[DataFrame]` 后，返回的结果迥异。

307 iterrows 和 itertuples

尽管 Pandas 已经尽可能向量化，让使用者尽可能避免 for 循环，但是有时不得以，还得要遍历 DataFrame。Pandas 提供 iterrows, itertuples 两种行级遍历。

这两种遍历效率上有什么不同，下面使用 Kaggle 上泰坦尼克号数据集的训练部分，测试遍历时的效率。

```
import pandas as pd
df = pd.read_csv('dataset/titanic_train_data.csv', sep=',')
```

df 一共有 891 行。

使用 iterrows 遍历打印所有行，在 Ipython 里输入以下行：

```
def iterrows_time(df):
    for i,row in df.iterrows():
        print(row)
```

然后使用魔法指令 `%time`，直接打印出此行的运行时间：

```
In [77]: %time iterrows_time(df)
```

复制

实验 10 次，平均耗时：920 ms

然后，使用 `itertuples` 遍历打印每行：

```
def itertuples_time(df):
    for nt in df.itertuples():
        print(nt)
```

复制

操作平均耗时 132 ms

也就是说打印 891 行，`itertuples` 遍历能节省 6 倍时间，这对于处理几十万、几百万级别的数据时，效率优势会更加凸显。

分析 `iterrows` 慢的原因，通过分析源码，我们看到每次遍历时，Pandas 都会把 `v` 包装为一个 `klass` 对象，`klass(v, index=columns, name=k)` 操作耗时。

```
def iterrows(self):
    columns = self.columns
    klass = self._constructor_sliced
    for k, v in zip(self.index, self.values):
        s = klass(v, index=columns, name=k)
        yield k, s
```

复制

因此，平时涉及遍历时，建议使用 `itertuples`。

308 DataFrame 之列转 index

有时，我们想把现有的数据框的某些列转化为 `index`，为之后的更多操作做准备。

某列转为 `index` 的实现方法如下：

```
import pandas as pd
```

复制

```
In [19]: df1 = pd.DataFrame({'a':[1,3,5], 'b':[9,4,12]})

In [20]: df1
Out[20]:
   a  b
0  1  9
1  3  4
2  5 12
```

把 `a` 列转为 `index`，分别不保留、保留原来的 `a` 列：

```
In [21]: df1.set_index('a',drop=False)
Out[21]:
   a  b
1  1  9
3  3  4
5  5 12

In [22]: df1.set_index('a',drop=True)
Out[22]:
   b
1  9
3  4
5 12
```

复制

对于频繁使用的列，建议转化为索引，效率会更高。

309 DataFrame 之 `index` 转列

使用 `reset_index`，将 `index` 转化为 `DataFrame` 的列，操作如下：

```
In [92]: df1 = pd.DataFrame({'a':[1,3,5], 'b':[9,4,12]})

In [93]: df2 = df1.set_index('a',drop=True)

In [94]: df2.reset_index('a',drop=True)
Out[94]:
   b
0  9
1  4
2 12

In [95]: df2.reset_index('a',drop=False)
Out[95]:
   a  b
0  1  9
1  3  4
2  5 12
```

复制

310 DataFrame 之 `reindex`

如果想按照某种规则，按照行重新排序数据，靠 `reindex` 函数可以实现：

```
In [96]: df1 = pd.DataFrame({'a':[1,3,5], 'b':[9,4,12]})

In [97]: df1
Out[97]:
   a  b
0  1  9
1  3  4
2  5 12

In [26]: df1.reindex([0,3,2,1])
Out[26]:
   a  b
0  1  9
3 NaN NaN
2  5 12
1  3  4
```

复制

`df1` 原来的索引会重新按照最新的索引`[0,3,2,1]`重新对齐，原来没有的索引`3`，默认数据都

填充为 NaN.

列数据的调整，也一样通过 `reindex` 实现，如下：

```
In [27]: df1.reindex(columns=['b','a','c'])
Out[27]:
      b  a  c
0     9  1 NaN
1     4  3 NaN
2    12  5 NaN
```

以上是关于 `index` 调整的方法，在实际中很有用，希望读者们能掌握。

### 311 数据清洗的主要几个步骤？

数据清洗 (data cleaning) 是机器学习和深度学习进入算法步前的一项重要任务，总结为下面几个步骤。

步骤 1：读入 `csv` 数据；

步骤 2：预览数据；

步骤 3：统计每一列的空值；

步骤 4：填充空值

步骤 5：特征工程，子步骤包括：删除一些特征列；创建新的特征列；创建数据分箱；

步骤 6：对分类编码，常用的包括，调用 `Sklearn` 中 `LabelEncode` 编码；`Pandas` 中哑编码；

步骤 7：再验证核实

今天使用泰坦尼克数据集，完整介绍以上步骤的具体操作过程。

### 312 Pandas 读入数据后预览技巧

使用 `Pandas`, 读入 `csv` 训练数据，然后了解每个字段的含义，数据有多少行和多少列等。

```
import pandas as pd

data_raw = pd.read_csv('train.csv')
data_raw
```

结果如下，一共训练集有 891 行数据，12 列

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cummings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Hekkinen, Miss. Laina	female	26.0	0	0	STON/OZ 3107282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Aller, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carnie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows x 12 columns

`PassengerId`: 乘客的 `Id`；

`Survived`：乘客生还情况，取值 1,2；

`Pclass`：乘客等级，取值：1,2,3；

`SibSp`：乘客的兄弟姐妹和配偶在船上的人数；

`Parch`：乘客的父母和孩子在船上的人数；

`Fare`：乘船的费用；

`Cabin`：舱的编号；

`Embarked`：分类变量，取值 S, C, Q；

其他几个特征比较好辨别，不再解释。

`Pandas` 提供 2 个好用的方法：`info`，`describe`

`info` 统计出数据的每一列类型、是否为 `null` 和个数；

`describe` 统计出数据每一列的统计学属性信息，平均值，方差，中位数，分位数等。

```
data_raw.info()
data_raw.describe(include='all')
```

结果：

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId      891 non-null int64
Survived         891 non-null int64
Pclass           891 non-null int64
Name             891 non-null object
Sex              891 non-null object
Age              714 non-null float64
SibSp            891 non-null int64
Parch           891 non-null int64
Ticket           891 non-null object
Fare             891 non-null float64
Cabin            284 non-null object
Embarked         889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
count	891.000000	891.000000	891.000000	891	891	714.000000	891.000000	891.000000	891	891.000000	284	889
unique	NaN	NaN	NaN	891	2	NaN	NaN	NaN	681	NaN	147	3
top	NaN	NaN	NaN	Richards, Master. William Rowe	male	NaN	NaN	NaN	CA. 2343	NaN	C23 C25 C27	S
freq	NaN	NaN	NaN	1	577	NaN	NaN	7	NaN	4	644	NaN
mean	448.000000	0.383838	2.308642	NaN	NaN	29.699118	0.521008	0.381594	NaN	32.204208	NaN	NaN
std	257.303842	0.486592	0.830071	NaN	NaN	14.526487	1.102743	0.806057	NaN	49.693429	NaN	NaN
min	1.000000	0.000000	1.000000	NaN	NaN	0.420000	0.000000	0.000000	NaN	0.000000	NaN	NaN



25%	223.500000	0.000000	2.000000	NaN	NaN	20.125000	0.000000	0.000000	NaN	7.910400	NaN	NaN
50%	448.000000	0.000000	3.000000	NaN	NaN	28.000000	0.000000	0.000000	NaN	14.454200	NaN	NaN
75%	668.500000	1.000000	3.000000	NaN	NaN	38.000000	1.000000	0.000000	NaN	31.000000	NaN	NaN
max	891.000000	1.000000	3.000000	NaN	NaN	80.000000	8.000000	6.000000	NaN	512.329200	NaN	NaN

```
Parch      0
Ticket      0
Fare        0
Cabin     687
Embarked    0
dtype: int64
```

### 315 Pandas 特征工程之删除特征列

因为 Cabin 缺失率较大，所以直接删除此列。

使用 Pandas 删除 列 Cabin ， axis 参数设置为 1，表示轴为列方向， inplace 为 True 表示就地删除。

```
data1.drop('Cabin', axis=1, inplace = True)
```

另外两列，PassengerId, Ticket 都是 ID 类型的，对预测乘客能否逃离没有关系，也直接删除。

```
drop_column = ['PassengerId','Ticket']
data1.drop(drop_column, axis=1, inplace = True)
```

### 316 Pandas 特征工程之增加特征列

增加一列 FamilySize ，计算公式如下：

```
data1['FamilySize'] = data1 ['SibSp'] + data1['Parch'] + 1
data1.head(3)
```

再创建一列 IsAlone ，如果 FamilySize 为 0，则表示只有一个人， IsAlone 为 True。

应用前面介绍的 where 函数，非常简洁地实现 IsAlone 列的赋值。

```
data1['IsAlone'] = np.where(data1['FamilySize'] > 1,0,1)
```

再创建一列 Title ，它是从 Name 列中提取出头衔或称谓。

Name 列的前三行，如下：

```
0      Braund, Mr. Owen Harris
1  Cumings, Mrs. John Bradley (Florence Briggs Th...
2  Heikkinen, Miss. Laina
Name: Name, dtype: object
```

Pandas 中使用 str 属性，直接拿到整个此列的字符串值，然后使用前面介绍的字符串分隔方法 split

```
data1['Title'] = data1['Name'].str.split(" ", expand=True)[1].str.split(
" ", expand=True)[0]
data1
```

数据前三行使用上面代码，提取后结果如下：

```
0      Mr
1      Mrs
2      Miss
Name: 0, dtype: object
```

### 317 Pandas 特征工程之分箱

Pandas 提供两种数据分箱方法：qcut, cut。

qcut 方法是基于分位数的分箱技术，cut 基于区间长度切分为若干。使用方法如下：

```
a = [3,1,5, 7,6, 5, 4, 6, 3]
pd.qcut(a,3)
```

结果如下，共划分为 3 个分类：

```
[(0.999, 3.667], (0.999, 3.667], (3.667, 5.333], (5.333, 7.0], (5.333, 7.0], (3.667, 5.333], (3.667, 5.333], (5.333, 7.0], (0.999, 3.667]]
Categories (3, interval[float64]): [(0.999, 3.667] < (3.667, 5.333] < (5.333, 7.0]]
```

a 元素这 3 个分类中的个数相等：

```
dfa = pd.DataFrame(a)
len1 = dfa[(0.999 < dfa[0]) & (dfa[0] <= 3.667)].shape
len2 = dfa[(3.667 < dfa[0]) & (dfa[0] <= 5.333)].shape
len3 = dfa[(5.333 < dfa[0]) & (dfa[0] <= 7.0)].shape
len1,len2,len3
```

结果如下，每个区间内都有 3 个元素

```
((3, 1), (3, 1), (3, 1))
```

cut 方法：

```
a = [3,1,5, 7,6, 5, 4, 6, 3]
pd.cut(a,3)
```

得到结果，与 qcut 划分出的 3 个区间不同，cut 根据 a 列表中最大与最小值间隔，均分，第一个左区间做一定偏移。

```
[(0.994, 3.0], (0.994, 3.0], (3.0, 5.0], (5.0, 7.0], (5.0, 7.0], (3.0, 5.0], (3.0, 5.0], (5.0, 7.0], (0.994, 3.0]]
Categories (3, interval[float64]): [(0.994, 3.0] < (3.0, 5.0] < (5.0, 7.0]]
```

除此之外，1992 年 Kerber 在论文中提出 ChiMerge 算法，自底向上的先分割再合并的分箱思想，具体算法步骤：

1) 设置 step 初始值

2) while 相邻区间的 merge 操作：

- 计算相邻区间的卡方值
- 合并卡方值最小的相邻区间
- 判断：是否所有相邻区间的卡方值都大于阈值，若是 break，否则继续 merge.

论文中 m 取值为 2，即计算 2 个相邻区间的卡方值，计算方法如下：

k : 类别个数

R<sub>i</sub> : 第 i 个分箱内样本总数

C<sub>j</sub> : 第 j 类别的样本总数

$$\chi^2 = \sum_{i=1}^2 \sum_{j=1}^k \frac{(A_{ij} - E_{ij})^2}{E_{ij}}$$
$$E_{ij} = R_i * C_j / N$$
$$R_i = \sum_{j=1}^k A_{ij} \text{ (在 } i \text{ 区间的 } j \text{ 类别数)}$$
$$C_j = \sum_{i=1}^2 A_{ij} \text{ ( } j \text{ 类别样本个数)}$$
$$N = \sum_{i=1}^2 R_i \text{ (总样本数)}$$

分别对 Fare 和 Age 列使用 qcut, cut 完成分箱，分箱数分别为 4 份，6 份。

```
datal['FareCut'] = pd.qcut(datal['Fare'], 4)
datal['AgeCut'] = pd.cut(datal['Age'].astype(int), 6)
datal.head(3)
```

复制

结果：

	Survived	Pclass		Name	Sex	Age	SibSp	Parch	Fare	Embarked	FamilySize	IsAlone	Title	FareCut	AgeCut
0	0	3		Braund, Mr. Owen Harris	male	22.0	1	0	7.2500	S	2	0	Mr	(-0.001, 7.91]	(13.333, 26.667]
1	1	1		Cummings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	71.2833	C	2	0	Mrs	(31.0, 512.329]	(26.667, 40.0]
2	1	3		Heikkinen, Miss. Laina	female	26.0	0	0	7.9250	S	1	1	Miss	(7.01, 14.454]	(13.333, 26.667]

318 数值编码常用方法：LabelEncoder 和 get\_dummies

本节介绍 2 种常用的分类型变量编码方法，一种是分类变量直接编码，LabelEncoder；另一种对分类变量创建哑变量( dummy variables).

LabelEncoder 方法

使用 Sklearn 的 LabelEncoder 方法，对分类型变量完成编码。

```
from sklearn.preprocessing import LabelEncoder
```

复制

泰坦尼克预测数据集中涉及的分类型变量有：Sex, Embarked, Title，还有我们新创建的2 个分箱列：AgeCut, FareCut。

```
label = LabelEncoder()
datal['Sex_Code'] = label.fit_transform(datal['Sex'])
datal['Embarked_Code'] = label.fit_transform(datal['Embarked'])
datal['Title_Code'] = label.fit_transform(datal['Title'])
datal['AgeBin_Code'] = label.fit_transform(datal['AgeCut'])
datal['FareBin_Code'] = label.fit_transform(datal['FareCut'])
datal.head(3)
```

复制

使用 LabelEncoder 完成编码后，数据的前三行打印显示：

	Survived	Pclass		Name	Sex	Age	SibSp	Parch	Fare	Embarked	FamilySize	IsAlone	Title	FareCut	AgeCut	Sex_Code	Embarked_Code	Title_Code	AgeBin_Code	FareBin_Code
0	0	3		Braund, Mr. Owen Harris	male	22.0	1	0	7.2500	S	2	0	Mr	(-0.001, 7.91]	(13.333, 26.667]	0	0	1	1	0
1	1	1		Cummings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	71.2833	C	2	0	Mrs	(31.0, 512.329]	(26.667, 40.0]	1	2	2	2	3
2	1	3		Heikkinen, Miss. Laina	female	26.0	0	0	7.9250	S	1	1	Miss	(7.01, 14.454]	(13.333, 26.667]	0	2	0	1	1

get\_dummies 方法

Pandas 的 get\_dummies 方法，也能实现对分类型变量实现哑编码， 将长 DataFrame 变为宽 DataFrame.

数据集中 Sex 分类型列取值有 2 种：female, male DataFrame:

```
pd.get_dummies(datal['Sex'])
```

复制

使用 get\_dummies，返回 2 列，分别为 female, male 列，结果如下：

```
female    male
0         0         1
1         1         0
2         1         0
3         1         0
4         0         1
...
886        0         1
887         1         0
888         1         0
889         0         1
890         0         1
891 rows x 2 columns
```

复制

而 LabelEncoder 编码后，仅仅是把 Female 编码为 0，male 编码为 1.

```
label.fit_transform(datal['Sex'])
0         1
```

复制

```
2      0
3      0
4      1
..
886    1
887      0
888      0
889      1
890      1
Name: Sex_Code, Length: 891, dtype: int64
```

对以下变量实现哑编码：

```
data1_dummy = pd.get_dummies(data1[['Sex', 'Embarked', 'Title', 'AgeCut', 'FareCut']])
data1_dummy
```

结果：

Sex_Male	Sex_Female	Embarked_C	Embarked_Q	Embarked_S	Title_Capt	Title_Col	Title_Don	Title_Mr	Title_Ms	Title_Miss	AgeCut_10-18	AgeCut_18-25	AgeCut_25-35	AgeCut_35-45	AgeCut_45-55	AgeCut_55-65	AgeCut_65-75	AgeCut_75-85	AgeCut_85-95	AgeCut_95-105
0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
886	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
887	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
888	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
889	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
890	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

### 319 Pandas 数据分析之 melt 透视

melt 函数，将宽 DataFrame 透视为长 DataFrame。

构造一个 DataFrame：

```
d = {\n    "district_code": [12345, 56789, 101112, 131415],\n    "apple": [5.2, 2.4, 4.2, 3.6],\n    "banana": [3.5, 1.9, 4.0, 2.3],\n    "orange": [8.0, 7.5, 6.4, 3.9]\n}\n\ndf = pd.DataFrame(d)\ndf
```

打印结果：

	district_code	apple	banana	orange
0	12345	5.2	3.5	8.0
1	56789	2.4	1.9	7.5
2	101112	4.2	4.0	6.4
3	131415	3.6	2.3	3.9

5.2 表示 12345 区域的 apple 价格，并且 apple, banana, orange，这三列都是一种水果，使用 melt 把这三列合并为一列。方法如下：

```
dfrm = df.melt(\n    id_vars = "district_code",\n    var_name = "fruit_name",\n    value_name = "price")\ndfrm
```

打印结果：

	district_code	fruit_name	price
0	12345	apple	5.2
1	56789	apple	2.4
2	101112	apple	4.2
3	131415	apple	3.6
4	12345	banana	3.5
5	56789	banana	1.9
6	101112	banana	4.0
7	131415	banana	2.3
8	12345	orange	8.0
9	56789	orange	7.5
10	101112	orange	6.4
11	131415	orange	3.9

### 320 Pandas 数据分析之 pivot 和 pivot\_table 透视

pivot 将长 DataFrame 透视为宽 DataFrame，与 melt 函数透视方向相反。函数的主要参数说明：

1). index 指明哪个列变为新 DataFrame 的 index；2). columns 指明哪些列变为 columns；3). values 指明哪些列变为新 DataFrame 的数据域，如果不指明，则默认除了被指明 index 和 columns 的其他列。

对上面使用 melt 透视后的 dfrm，使用 pivot 函数，将长 DataFrame 变形为宽 DataFrame：

 回到主页

 目录

Python 全栈 450 道常见问题全解析（配套教学） 16/26Python 全栈 400 之Pandas数据分析练习

```
dfp
```

结果：

	price	fruit_name	apple	banana	orange
district_code					
12345	5.2	3.5	8.0		
56789	2.4	1.9	7.5		
101112	4.2	4.0	6.4		
131415	3.6	2.3	3.9		

打印透视后的 dfp 的列：

```
dfp.columns
```



存

1

评论

<

>

288 Pandas 读取 U...

289 Pandas 读取文...

290 Pandas 读取之...

291 Pandas 读取之...

292 Pandas 之分块...

293 Pandas 之随机...

294 Pandas 之一维...

295 Pandas 之创建...

296 Series 之增加...

297 Series 之删除元素

298 Series 之修改...

结果如下，为多索引列：

```
MultiIndex([('price', 'apple'),
            ('price', 'banana'),
            ('price', 'orange')],
           names=[None, 'fruit_name'])
```

因此，透视后访问某列，就得使用多索引列：

```
dfp[['price', 'apple']]
```

结果：

```
district_code
12345         5.2
56789         2.4
101112        4.2
131415        3.6
Name: (price, apple), dtype: float64
```

pivot 函数是没有聚合功能的。

但是，pandas 中提供的 pivot\_table() 提供聚合功能。因此，pivot\_table 可看是 pivot 函数的升级版。

**pivot\_table**

为了演示，pivot\_table 的聚合透视功能，先生成一个 DataFrame:

```
d = {\n  "district_code": [12345, 12345, 56789, 101112, 131415,12345, 12345, 56789\n    , 101112, 131415],\n  "fruit_name": ['apple', 'apple', 'orange', 'banana', 'orange','apple', 'a\n    pple', 'orange', 'banana', 'orange'],\n  "price":      [3.5, 3.7, 1.9, 4.0, 2.3,4.5, 4.7, 2.9, 5.0, 3.3]\n}\n\ndf2 = pd.DataFrame(d)\ndf2
```

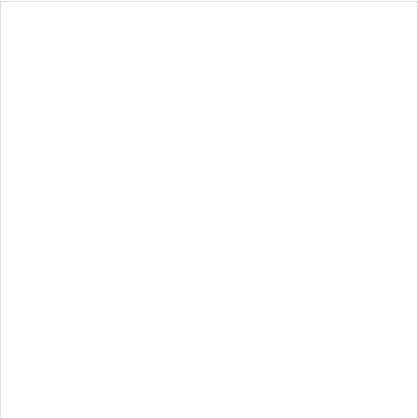
结果：

```
district_code  fruit_name  price\n0      12345    apple      3.5\n1      12345    apple      3.7\n2      56789    orange      1.9\n3      101112   banana      4.0\n4      131415    orange      2.3\n5      12345    apple      4.5\n6      12345    apple      4.7\n7      56789    orange      2.9\n8      101112   banana      5.0\n9      131415    orange      3.3
```

pivot\_table 函数，不仅具有 pivot 函数将长 DataFrame 透视为宽 DataFrame,同时还具有 sum 聚合功能：

```
dfp2 = df2.pivot_table(index='district_code',columns='fruit_name',values=[\n  'price'],aggfunc=[np.sum])\ndfp2
```

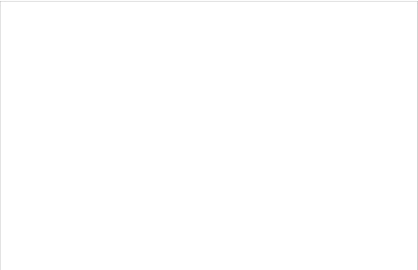
结果如下，district\_code 为 12345 的区域，apple 价格求和已经聚合为 16.4



一次可以使用多个聚合方法，如下，使用求和、求平均值聚合：

```
dfp3 = df2.pivot_table(index='district_code',columns='fruit_name',values=[\n  'price'],aggfunc=[np.sum,np.mean])\ndfp3
```

结果：



Pandas 还提供一个透视功能更加专一的函数，按照频次透视函数：crosstab

321 Pandas 数据分析之 crosstab 频次透视

有 2 个必传递的参数 index, columns，分别为透视后的行、列索引。

crosstab 与 pivot\_table 很相似，看一个例子。

```
a = np.array(['apple','apple', 'orange', 'banana', 'orange'], dtype=object)
b = np.array(['china','china', 'ameri', 'ameri', 'korea'], dtype=object)
c = np.array(['good','good','good','good','better'],dtype=object)
pd.crosstab(a,[b,c])
```

结果为：



以上，实质等价于：

```
for it in zip(a,b,c):
    print(it)
```

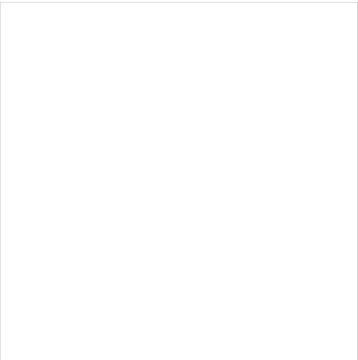
可以看到 apple, china, good 这项出现的频次为 2，其他频次都是 1。

```
('apple', 'china', 'good')
('apple', 'china', 'good')
('orange', 'ameri', 'good')
('banana', 'ameri', 'good')
('orange', 'korea', 'better')
```

同理，

```
pd.crosstab([a,b],[c])
```

结果为：



还是只有一项(apple, china, good)频次为2，和上面的原理一样。

例子

```
df = pd.DataFrame({'类别':['水果','水果','水果','蔬菜','蔬菜','肉类','肉类'],
                  '产地':['美国','中国','中国','中国','新西兰','新西兰','美国'],
                  '水果':['苹果','梨','草莓','番茄','黄瓜','羊肉','牛肉'],
                  '数量':[5,5,9,3,2,10,8],
                  '价格':[5,5,10,3,3,13,20]})
df #显示df
```

结果：

	类别	产地	水果	数量	价格
0	水果	美国	苹果	5	5
1	水果	中国	梨	5	5
2	水果	中国	草莓	9	10
3	蔬菜	中国	番茄	3	3
4	蔬菜	新西兰	黄瓜	2	3
5	肉类	新西兰	羊肉	10	13
6	肉类	美国	牛肉	8	20

类别 列设置为 index, 产地 列设置为 columns，统计词条出现频次：

```
pd.crosstab(df['类别'],df['产地'],margins=True)
```

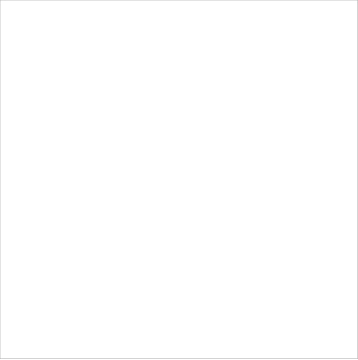
结果：



如果想使用聚合函数，aggfun 参数，必须指明 values 列，如下：

```
pd.crosstab(df['类别'],df['产地'],values=df['价格'],aggfunc=np.max, margins=True)
```

结果如下：



crosstab本质：按照指定的 index 和 columns 统计 DataFrame 中出现(index, columns)的频次。  
值得注意，这些透视函数，melt, pivot, pivot\_table, crosstab，都基于 groupby 分组基础上，而分组大家是更容易理解的，所以在理解这些透视函数时，可以结合分组思想。

322 4 种表连接方法详解

给定两个 DataFrame，它们至少存在一个名称相同的列，如何连接两个表？使用merge 函数连接两个 DataFrame，连接方式共有 4 种，分别为：left, right, inner,outer. 如何区分这 4 种连接关系

2 个 DataFrame 分别为 left、right，名称相同的列为 key，left 的 key 取值为：k0, k1, k2；right 表的 key 取值为：k0, k0, k1

1) 如果 left 的 key 指向 right，此连接方式为: left：关系图表达为：

left	right
k0	k0
	k0
k1	k1
k2	NaN

2) 如果 right 的 key 指向 left，则称此连接方式为: right

right	left
k0	k0
k0	
k1	k1

3) 如果只拿 left 和 right 公有 key（也就是交集）建立关系，称此连接方式为: inner

left	right
k0	k0
k0	k0
k1	k1

4) 如果 left 和 right 的 key合并（也就是并集）再建立关系后，称此连接方式为: outer

left	right
k0	k0
	k0
k1	k1
k2	NaN

以上就是 merge 连接 2 个DataFrame 时，根据 key 节点建立关系的 4 种方法。

下面举例说明：

left 和 right 分别为：

left

	age1	key
0	10	k0
1	20	k1
2	30	k2

right

	age2	key
--	------	-----

	age2	key
0	40	k0
1	50	k0
2	60	k1

如果连接方法参数 `how` 取值为 'left'

```
pd.merge(left, right, how='left', on='key')
```

复制

结果：

	age1	key	age2
0	10	k0	40.0
1	10	k0	50.0
2	20	k1	60.0
3	30	k2	NaN

复制

如果连接方法参数 `how` 取值为 'right'

```
pd.merge(left, right, how='right', on='key')
```

复制

结果：

	age1	key	age2
0	10	k0	40
1	10	k0	50
2	20	k1	60

复制

如果连接方法参数 `how` 取值为 'inner'

```
pd.merge(left, right, how='inner', on='key')
```

复制

结果：

	age1	key	age2
0	10	k0	40
1	10	k0	50
2	20	k1	60

复制

如果连接方法参数 `how` 取值为 'outer'

```
pd.merge(left, right, how='outer', on='key')
```

复制

结果：

	age1	key	age2
0	10	k0	40.0
1	10	k0	50.0
2	20	k1	60.0
3	30	k2	NaN

复制

### 323 Pandas 使用技巧：生成时间序列的数据集

与时间序列相关的问题，平时挺常见。如何用 Pandas 快速生成时间序列数据？使用

```
pd.util.testing.makeTimeDataFrame
```

只需要一行代码，便能生成一个 `index` 为时间序列的 `DataFrame`：

```
import pandas as pd

pd.util.testing.makeTimeDataFrame(10)
```

复制

结果：

A	B	C	D
2000-01-03	0.932776	-1.509302	0.285825
2000-01-04	0.565230	-1.598449	-0.786274
2000-01-05	-0.152743	-0.392053	-0.127415
2000-01-06	1.321998	-0.927537	0.205666
2000-01-07	0.324359	1.512743	0.553633
2000-01-10	-0.566780	0.201565	-0.801172
2000-01-11	-0.259348	-0.035893	-1.363496
2000-01-12	-0.341700	-1.438874	-0.260598
2000-01-13	-1.085183	0.286239	2.475605
2000-01-14	-0.057128	-0.602625	0.461550

复制

时间序列的间隔还能配置，默认的 A B C D 四列也支持配置。

```
import numpy as np

df = pd.DataFrame(np.random.randint(1,1000,size=(10,3)), columns = ['商品编码', '商品销量', '商品库存'])
df.index = pd.util.testing.makeDateIndex(10, freq='H')
```

复制

结果：

商品编码	商品销量	商品库存
2000-01-01 00:00:00	99	264
2000-01-01 01:00:00	294	406
2000-01-01 02:00:00	89	221
2000-01-01 03:00:00	962	153
2000-01-01 04:00:00	538	46
2000-01-01 05:00:00	226	973
2000-01-01 06:00:00	193	866
2000-01-01 07:00:00	300	129
2000-01-01 08:00:00	966	372
2000-01-01 09:00:00	687	493

复制

### 324 Pandas 使用技巧：apply(type) 做类型检查

有时肉眼所见的列类型，未必就是你预期的类型，如下 `DataFrame` 销量这一列，看似为浮点



型。

□

实际上，我们是这样生成 DataFrame 的，

```
d = {"商品":["A", "B", "C", "D", "E"], "销量":[100, "100", 50, 550.20, "375.25"]}
df = pd.DataFrame(d)
df
```

所以直接在销量这一列上求平均值，就会报错。

```
df['销量'].sum()
```

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

所以在计算前，检查此列每个单元格的取值类型就很重要。

```
df['销量'].apply(type)
```

```
0    <class 'int'>
1    <class 'str'>
2    <class 'int'>
3    <class 'float'>
4    <class 'str'>
Name: 销量, dtype: object
```

上面是打印结果，看到取值有 int, str, float. 有的读者就问了，这才几行，如果上千上百行，也有一个一个歇吗，当然不能，使用 value\_counts 统计下各种值的频次。

```
df['销量'].apply(type).value_counts()
```

```
<class 'int'>      2
<class 'str'>      2
<class 'float'>    1
Name: 销量, dtype: int64
```

325 Pandas使用技巧： 标签和位置选择数据

读入数据，本专栏使用到的数据集都会打包发给各位读者。

```
df = pd.read_csv("drinksbycountry.csv", index_col="country")
df
```

□

使用位置，`iloc` 选择前 10 行数据：

```
df.iloc[:10, :]
```

□

标签和 `loc` 选择两列数据

```
df.iloc[:10, :].loc[:, "spirit_servings":"wine_servings"]
```



也可以直接使用 `iloc`，得到结果如上面一样。

```
df.iloc[:10, 1:3]
```

326 Pandas 使用技巧： 空值检查

实际使用的数据，null 值在所难免。如何快速找出 DataFrame 所有列的 null 值个数？

使用 Pandas 能非常方便实现，只需下面一行代码：

```
data.isnull().sum()
```

`data.isnull()`: 逐行逐元素查找元素值是否为 null.

`.sum()`: 默认在 axis 为 0 上完成一次 reduce 求和。

如下 DataFrame:

□

检查 null 值:

```
data.isnull().sum()

结果：
```

```
PassengerId      0
Survived          0
Pclass           0
Name             0
Sex              0
Age            177
SibSp           0
Parch           0
Ticket           0
Fare            0
Cabin          687
Embarked         2
dtype: int64
```

Age 列 177 个 null 值

Cabin 列 687 个 null 值

Embarked 列 2 个 null 值

327 Pandas 使用技巧：replace 做清洗

Pandas 的强项在于数据分析，自然就少不了对数据清洗的支持。一个快速清洗数据的小技巧，在某列上使用 replace 方法和正则，快速完成值的清洗。

源数据：

```
d = {"customer": ["A", "B", "C", "D"],
     "sales": [1100, "950.5", "$400", "$1250.75"]}

df = pd.DataFrame(d)
df
```

打印结果：

```
customer  sales
0      A    1100
1      B    950.5
2      C     400
3      D  1250.75
```

看到 sales 列的值，有整型，还有美元+整型，美元+浮点型。

我们的目标：清洗掉 \$ 符号，字符串型转化为浮点型。

一行代码搞定：（ 点击代码区域，向右滑动，查看完整代码 ）

```
df["sales"] = df["sales"].replace("[\$]", "", regex = True).astype("float")
```

使用正则替换，将要替换的字符放到列表中 [\$]，替换为空字符，即 "" ；

最后使用 astype 转为 float

打印结果：

```
customer  sales
0      A   1100.00
1      B   950.50
2      C   400.00
3      D  1250.75
```

如果不放心，再检查下值的类型：

```
df["sales"].apply(type)
```

打印结果：

```
0    <class 'float'>
1    <class 'float'>
2    <class 'float'>
3    <class 'float'>
```

328 Pandas 使用技巧：转 datetime

已知年和 dayofyear，怎么转 datetime?

原 DataFrame

```
d = {
    "year": [2019, 2019, 2020],
    "day_of_year": [350, 365, 1]
}
df = pd.DataFrame(d)
df
```

打印结果：

```
year  day_of_year
0   2019         350
1   2019         365
2   2020          1
```

下面介绍如何转 datetime 的 trick

Step 1: 创建整数

```
df["int_number"] = df["year"]*1000 + df["day_of_year"]
df
```

打印结果：

复制

	year	day_of_year	int_number
0	2019	350	2019350
1	2019	365	2019365
2	2020	1	2020001

Step 2: to\_datetime

复制

df["date"] = pd.to_datetime(df["int_number"], format = "%Y%j")
df

注意 "%Y%j" 中转化格式 j，正是对应 dayofyear 参数

打印结果：

复制

	year	day_of_year	int_number	date
0	2019	350	2019350	2019-12-16
1	2019	365	2019365	2019-12-31
2	2020	1	2020001	2020-01-01

329 Pandas 使用技巧：小分类值的替换

分类中出现次数较少的值，如何统一归为 others，该怎么做到？

这也是我们在数据清洗、特征构造中经常会面临的一个任务。

如下 DataFrame:

复制

d = {"name":["Jone",'Alica','Emily','Robert','Tomas','Zhang','Liu','Wang', 'Jack','Wsx','Guo'], "categories": ["A", "C", "A", "D", "A", "B", "B", "C", "A", "E", "F"] }
df = pd.DataFrame(d)
df

结果：

复制

	name	categories
0	Jone	A
1	Alica	C
2	Emily	A
3	Robert	D
4	Tomas	A
5	Zhang	B
6	Liu	B
7	Wang	C
8	Jack	A
9	Wsx	E
10	Guo	F

D、E、F 仅在分类中出现一次，A 出现次数较多。

步骤 1：统计频次，并归一化

复制

frequencies = df["categories"].value_counts(normalize = True)
frequencies

结果：

复制

A	0.363636
B	0.181818
C	0.181818
F	0.090909
E	0.090909
D	0.090909
Name: categories, dtype: float64	

步骤 2：设定阈值，过滤出频次较少的值

复制

threshold = 0.1
small_categories = frequencies[frequencies < threshold].index
small_categories

结果：

复制

Index(['F', 'E', 'D'], dtype='object')
----------------------------------------

步骤 3：替换值

复制

df["categories"] = df["categories"].replace(small_categories, "Others")
-------------------------------------------------------------------------

替换后的 DataFrame:

复制

	name	categories
0	Jone	A
1	Alica	C
2	Emily	A
3	Robert	Others
4	Tomas	A
5	Zhang	B
6	Liu	B
7	Wang	C
8	Jack	A
9	Wsx	Others
10	Guo	Others

330 Pandas 使用技巧：重新排序列

某些场景需要重新排序 DataFrame 的列，如下 DataFrame:

如何将列快速变为：

□

先构造数据：

```
df = pd.DataFrame(np.random.randint(0,20,size=(5,7)),columns=list('ABCDEFGG'))
df
```

□

方法1，直接了当：

```
df2 = df[["A", "C", "D", "F", "E", "G", "B"]]
df2
```

结果：

□

方法2，也了解下：

```
cols = df.columns[[0, 2, 3, 5, 4, 6, 1]]
df3 = df[cols]
df3
```

也能得到方法1的结果。

331 Pandas 使用技巧：时间数据下采样

步长为小时的时间序列数据，有没有小技巧，快速完成下采样，采集成按天的数据呢？

先生成测试数据：

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randint(1,10,
size=(240,3)),
columns = ['商品编码','商品销量','商品库存'])

df.index = pd.util.testing.makeDateIndex(240,freq='H')
df
```

生成 240 行步长为小时间隔的数据：

□

使用 `resample` 方法，合并为天(D)

```
day_df = df.resample("D")["商品销量"].sum().to_frame()
day_df
```

结果如下，10 行，240 小时，正好为 10 天：

□

332 Pandas 使用技巧：map 做特征工程

DataFrame 上快速对某些列展开特征工程，如何做到？

先生成数据：

```
d = {
"gender":["male", "female", "male","female"],
"color":["red", "green", "blue","green"],
"age":[25, 30, 15, 32]
}

df = pd.DataFrame(d)
df
```

□

在 `gender` 列上做如下映射：

```
d = {"male": 0, "female": 1}
df["gender2"] = df["gender"].map(d)
```

□

333 Pandas 使用技巧：结合使用 where 和 isin

如下 DataFrame:

```
d = {"genre": ["A", "C", "A", "A", "A", "B", "B", "C", "D", "E", "F"]}
df = pd.DataFrame(d)
df
```

```
genre
0    A
1    C
2    A
3    A
4    A
5    B
6    B
7    C
8    D
9    E
10   F
```

除了 3 个出现频率最高的值外，统一替换其他值为 others.

首先找到出现频率最高的 3 个值：

```
top3 = df["genre"].value_counts().nlargest(3).index
top3
```

复制

结果：

```
Index(['A', 'C', 'B'], dtype='object')
```

复制

使用 isin:

```
df_update = df.where(df["genre"].isin(top3), other = "others")
df_update
```

复制

使用 where, 不满足第一个参数条件的，都会被替换为 others.

结果：


```
genre
0    A
1    C
2    A
3    A
4    A
5    B
6    B
7    C
8  others
9  others
10  others
```

复制

结合使用 where, isin 做数据工程，特征清洗非常方便。

下一章

互动评论

 说点什么

评论

 The Scrapper

1 个月前

完成了pandas

 鼓掌



存

评论

<

>