昵称：Treant
园龄：5年4个月
粉丝：197
关注：78
+加关注

【Python实战】Pandas：让你像写SQL一样做数据分析（一）

1. 引言

Pandas是一个开源的Python数据分析库。Pandas把结构化数据分为了三类：

- Series，1维序列，可视作为没有column名的、只有一个column的DataFrame；
- DataFrame，同Spark SQL中的DataFrame一样，其概念来自于R语言，为多column并schema化的2维结构化数据，可视作为Series的容器（container）；
- Panel，为3维的结构化数据，可视作为DataFrame的容器；

DataFrame较为常见，因此本文主要讨论内容将为DataFrame。DataFrame的生成可通过读取纯文本、Json等数据来生成，亦可以通过Python对象来生成：

```
import pandas as pd
import numpy as np


df = pd.DataFrame({'total_bill': [16.99, 10.34, 23.68, 23.68,
24.59],
                   'tip': [1.01, 1.66, 3.50, 3.31, 3.61],
                   'sex': ['Female', 'Male', 'Male', 'Male',
'Female']})
```

对于DataFrame，我们可以看到其固有属性：

```
# data type of columns
df.dtypes
# indexes
df.index
# return pandas.Index
df.columns
# each row, return array[array]
df.values
# a tuple representing the dimensionality of df
df.shape
```

- .index，为行索引
- .columns，为列名称（label）
- .dtype，为列数据类型

2. SQL操作

官方Doc给出了部分SQL的Pandas实现，在此基础上本文给出了一些扩充说明。以下内容基于Python 2.7 + Pandas 0.18.1的版本。

select

名称选取，还可以根据列所在的position选取。相关函数如下：

- loc，基于列label，可选取特定行（根据行index）；
- iloc，基于行/列的position；

```
print df.loc[1:3, ['total_bill', 'tip']]
print df.loc[1:3, 'tip': 'total_bill']
print df.iloc[1:3, [1, 2]]
print df.iloc[1:3, 1: 3]
```

- at，根据指定行index及列label，快速定位DataFrame的元素；
- iat，与at类似，不同的是根据position来定位的；

```
print df.at[3, 'tip']
print df.iat[3, 1]
```

- ix，为loc与iloc的混合体，既支持label也支持position；

```
print df.ix[1:3, [1, 2]]
print df.ix[1:3, ['total_bill', 'tip']]
```

此外，有更为简洁的行/列选取方式：

```
print df[1: 3]
print df[['total_bill', 'tip']]
# print df[1:2, ['total_bill', 'tip']]  # TypeError: unhashable type
```

where

Pandas实现where filter，较为常用的办法为 `df[df[colunm] boolean expr]` ，比如：

```
print df[df['sex'] == 'Female']
print df[df['total_bill'] > 20]

# or
print df.query('total_bill > 20')
```

在where子句中常常会搭配and, or, in, not关键词，Pandas中也有对应的实现：

```
# and
print df[(df['sex'] == 'Female') & (df['total_bill'] > 20)]
# or
print df[(df['sex'] == 'Female') | (df['total_bill'] > 20)]
# in
print df[df['total_bill'].isin([21.01, 23.68, 24.59])]
# not
print df[-(df['sex'] == 'Male')]
print df[-df['total_bill'].isin([21.01, 23.68, 24.59])]
# string function
print df = df[(-df['app'].isin(sys_app)) & (-df.app.str.contains('^微信\d+$'))]
```

对where条件筛选后只有一行的dataframe取其中某一列的值，其两种实现方式如下：

```
total = df.loc[df['tip'] == 1.66, 'total_bill'].values[0]
total = df.get_value(df.loc[df['tip'] == 1.66].index.values[0], 'total_bill')
```

distinct

Coder 无他，但手熟尔

```
df.drop_duplicates(subset=['sex'], keep='first', inplace=True)
```

包含参数：

- subset，为选定的列做distinct，默认为所有列；
- keep，值选项{'first', 'last', False}，保留重复元素中的第一个、最后一个，或全部删除；
- inplace ，默认为False，返回一个新的dataframe；若为True，则返回去重后的原dataframe

group

group一般会配合**合计函数**（Aggregate functions）使用，比如：count、avg等。Pandas对合计函数的支持有限，有count和size函数实现SQL的count：

```
print df.groupby('sex').size()
print df.groupby('sex').count()
print df.groupby('sex')['tip'].count()
```

对于多合计函数，

```
select sex, max(tip), sum(total_bill) as total
from tips_tb
group by sex;
```

实现在agg()中指定dict：

```
print df.groupby('sex').agg({'tip': np.max, 'total_bill': np.sum})

# count(distinct **)
print df.groupby('tip').agg({'sex': pd.Series.nunique})
```

as

SQL中使用as修改列的别名，Pandas也支持这种修改：

```
# first implementation
df.columns = ['total', 'pit', 'xes']
# second implementation
df.rename(columns={'total_bill': 'total', 'tip': 'pit', 'sex':
'xes'}, inplace=True)
```

其中，第一种方法的修改是有问题的，因为其是按照列position逐一替换的。因此，我推荐第二种方法。

join

Pandas中join的实现也有两种：

```
# 1.
df.join(df2, how='left'...)

# 2.
pd.merge(df1, df2, how='left', left_on='app', right_on='app')
```

第一种方法是按DataFrame的index进行join的，而第二种方法才是按on指定的列做join。Pandas满足left、right、inner、full outer四种join方式。

order

Coder 无他，但手熟尔

出度：

```
print df.sort_values(['total_bill', 'tip'], ascending=[False, True])
```

top

对于全局的top：

```
print df.nlargest(3, columns=['total_bill'])
```

对于分组top，MySQL的实现（采用自join的方式）：

```
select a.sex, a.tip
from tips_tb a
where (
    select count(*)
    from tips_tb b
    where b.sex = a.sex and b.tip > a.tip
) < 2
order by a.sex, a.tip desc;
```

Pandas的等价实现，思路与上类似：

```
# 1.
df.assign(rn=df.sort_values(['total_bill'], ascending=False)
          .groupby('sex')
          .cumcount()+1)\
    .query('rn < 3')\
    .sort_values(['sex', 'rn'])

# 2.
df.assign(rn=df.groupby('sex')['total_bill']
          .rank(method='first', ascending=False)) \
    .query('rn < 3') \
    .sort_values(['sex', 'rn'])
```


replace

replace函数提供对dataframe全局修改，亦可通过where条件进行过滤修改（搭配loc）：

```
# overall replace
df.replace(to_replace='Female', value='Sansa', inplace=True)

# dict replace
df.replace({'sex': {'Female': 'Sansa', 'Male': 'Leone'}},
inplace=True)

# replace on where condition
df.loc[df.sex == 'Male', 'sex'] = 'Leone'
```

自定义

除了上述SQL操作外，Pandas提供对每列/每一元素做自定义操作，为此而设计以下三个函数：

- map(func)，为Series的函数，DataFrame不能直接调用，需取列后再调用；
- apply(func)，对DataFrame中的某一行/列进行func操作；
- applymap(func)，为element-wise函数，对每一个元素做func操作

```
print df['tip'].map(lambda x: x - 1)
print df[['total_bill', 'tip']].apply(sum)
```

Coder 无他，但手熟尔

3. 实战

环比增长

现有两个月APP的UV数据，要得到月UV环比增长；该操作等价于两个Dataframe left join后按指定列做减操作：

```
def chain(current, last):
    df1 = pd.read_csv(current, names=['app', 'tag', 'uv'], sep='\t')
    df2 = pd.read_csv(last, names=['app', 'tag', 'uv'], sep='\t')
    df3 = pd.merge(df1, df2, how='left', on='app')
    df3['uv_y'] = df3['uv_y'].map(lambda x: 0.0 if pd.isnull(x) else
x)
    df3['growth'] = df3['uv_x'] - df3['uv_y']
    return df3[['app', 'growth', 'uv_x',
'uv_y']].sort_values(by='growth', ascending=False)
```

差集

对于给定的列，一个Dataframe过滤另一个Dataframe该列的值；相当于集合的差集操作：

```
def difference(left, right, on):
    """
    difference of two dataframes
    :param left: left dataframe
    :param right: right dataframe
    :param on: join key
    :return: difference dataframe
    """
    df = pd.merge(left, right, how='left', on=on)
    left_columns = left.columns
    col_y = df.columns[left_columns.size]
    df = df[df[col_y].isnull()]
    df = df.ix[:, 0:left_columns.size]
    df.columns = left_columns
    return df
```


分类: Python

Treant
关注 - 78
粉丝 - 197
+加关注

2 0

« 上一篇：【Python实战】Scrapy豌豆荚应用市场爬虫
» 下一篇：Scala比较器：Ordered与Ordering

posted @ 2016-06-30 18:08 Treant 阅读(20330) 评论(4) 编辑 收藏

评论列表

#1楼 2016-06-30 18:22 会长

您是做大数据分析工作的吗？我也打算学一些呢

支持(0) 反对(0)

#2楼[楼主] 2016-06-30 19:04 Treant

Coder 无他，但手熟尔

> 您是做大数据分析工作的吗？我也打算学一些呢

是的

支持(0) 反对(0)

#3楼 2016-07-01 10:11 DelphiSeattle

后面学的东西太多了，数据分析这个门槛太高了！

支持(1) 反对(0)

#4楼 2016-07-20 07:53 智浪淘沙

不错的笔记，非常感谢

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 登录 或 注册，访问网站首页。