一场pandas与SQL的巅峰大战（二）

上一篇中，我们对比了pandas与SQL常见的一些操作，我们的例子虽然是以MySQL为基础的，但换作其他的数据库软件，也一样适用。工作中除了MySQL，也经常会使用Hive SQL，相比之下，后者有更为强大和丰富的函数。本文将延续上一篇文章的风格和思路，继续对比Pandas与SQL，一方面是对上文的补充，另一方面也继续深入学习一下两种工具。方便起见，本文采用hive环境运行SQL，使用jupyter lab运行pandas。关于hive的安装和配置，我在之前的文章提到过，如果你觉得比较困难，可以考虑使用postgreSQL，它比MySQL支持更多的函数(不够代码可能需要进行一定的改动)。而jupyter lab和jupyter notebook功能相同，界面相似，完全可以用notebook代替。希望本文可以帮助各位读者在工作中进行pandas和Hive SQL的快速转换。

在公众号后台回复“**对比二**”可以获取本文的PDF版本以及全部的数据和代码。

数据概况

数据上，我们还是使用上一篇中虚拟的数据，在ts的格式上有些小改动，在使用之前同样需要先用read_csv的方式读取，具体可以参考上篇文章。本文不做这一步的演示。hive方面我们新建了一张表，并把同样的数据加载进了表中，直接使用即可。

id	ts	uid	orderid	amount
1	2019-08-01 09:15:00	10005	20190801091540	48.43
2	2019-08-01 10:00:00	10001	20190801100006	89.33
3	2019-08-01 10:04:00	10003	20190801091540	63.86
4	2019-08-01 12:17:00	10002	20190801121742	3.16
5	2019-08-01 14:05:00	10001	20190801140515	87.15
6	2019-08-01 14:05:00	10004	20190801140529	88.65
7	2019-08-02 08:13:00	10009	20190802081315	36.02
8	2019-08-02 11:14:00	10009	20190802111424	95.66
9	2019-08-02 13:18:00	10005	20190802131801	89.36
10	2019-08-02 15:18:00	10001	20190802151834	71.38
11	2019-08-02 16:00:00	10005	20190802160014	63.13
12	2019-08-02 17:03:00	10003	20190802170356	79.33
13	2019-08-02 17:11:00	10002	20190802171115	56.78
14	2019-08-02 19:05:00	10008	20190802190518	23.1
15	2019-08-02 20:07:00	10005	20190802200717	73.82
16	2019-08-02 20:08:00	10001	20190802200816	82.12
17	2019-08-02 20:10:00	10003	20190802201002	32.01
18	2019-08-03 09:02:00	10009	20190803090247	2.7
19	2019-08-03 10:08:00	10003	20190803100858	50.4
20	2019-08-03 12:08:00	10009	20190803120818	47.99

```
Time taken: 0.082 seconds
hive> CREATE TABLE `t_order`(
    >  `id` int comment "id",
    >  `ts` string  comment "订单时间",
    >  `uid` string comment "用户id",
    >  `orderid` string comment "订单id",
    >  `amount` float comment "订单金额"
    >  )
    >  ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
    >  STORED AS TEXTFILE;
OK
Time taken: 0.046 seconds
hive> load data local inpath 'order.csv' overwrite into table t_order;
Loading data to table test.t_order
Table test.t_order stats: [numFiles=1, numRows=0, totalSize=4208, rawDataSize=0]
OK
Time taken: 0.155 seconds
hive> select * from t_order;
OK
1       2019-08-01 09:15:00     10005   20190801091540  48.43
2       2019-08-01 10:00:00     10001   20190801100006  89.33
3       2019-08-01 10:04:00     10003   20190801091540  63.86
4       2019-08-01 12:17:00     10002   20190801121742  3.16
5       2019-08-01 14:05:00     10001   20190801140515  87.15
6       2019-08-01 14:05:00     10004   20190801140529  88.65
7       2019-08-02 08:13:00     10009   20190802081315  36.02
8       2019-08-02 11:14:00     10009   20190802111424  95.66
9       2019-08-02 13:18:00     10005   20190802131801  89.36
10      2019-08-02 15:18:00     10001   20190802151834  71.38
11      2019-08-02 16:00:00     10005   20190802160014  63.13
12      2019-08-02 17:03:00     10003   20190802170356  79.33
```

创建表

加载数据

查看数据

开始学习

一、字符串的截取

对于原始数据集中的一列，我们常常要截取其字串作为新的列来使用。例如我们想求出每一条订单对应的日期。需要从订单时间ts或者orderid中截取。在pandas中，我们可以将列转换为字符串，截取其子串，添加为新的列。代码如下图左侧所示，我们使用了`.str`将原字段视为字符串，从ts中截取了前10位，从orderid中截取了前8位。经验表明有时在`.str`之前需要加上astype，能够避免不必要的麻烦。两种写法供参考。

对于字符串截取的操作，Hive SQL中有substr函数，它在MySQL和Hive中的用法是一样的`substr(string A, int start, int len)`表示从字符串A中截取起始位置为start，长度为len的子串，其中起始位置从1开始算。实现上面效果的代码如下：

```
[4]: order['dt'] = order['ts'].str[:10]
     order.head()
```

[4]:

	id	ts	uid	orderid	amount	dt
0	1	2019-08-01 09:15:00	10005	20190801091540	48.43	2019-08-01
1	2	2019-08-01 10:00:00	10001	20190801100006	89.33	2019-08-01
2	3	2019-08-01 10:04:00	10003	20190801091540	63.86	2019-08-01
3	4	2019-08-01 12:17:00	10002	20190801121742	3.16	2019-08-01
4	5	2019-08-01 14:05:00	10001	20190801140515	87.15	2019-08-01

```
[6]: order['dt2'] = order['orderid'].astype(str).str[:8]
     order.head()
```

[6]:

	id	ts	uid	orderid	amount	dt	dt2
0	1	2019-08-01 09:15:00	10005	20190801091540	48.43	2019-08-01	20190801
1	2	2019-08-01 10:00:00	10001	20190801100006	89.33	2019-08-01	20190801
2	3	2019-08-01 10:04:00	10003	20190801091540	63.86	2019-08-01	20190801
3	4	2019-08-01 12:17:00	10002	20190801121742	3.16	2019-08-01	20190801
4	5	2019-08-01 14:05:00	10001	20190801140515	87.15	2019-08-01	20190801

pandas操作：截取字串

```
hive> select *, substr(ts, 1, 10) as dt, substring(orderid, 1, 8) as dt2
    > from t_order;
OK
1       2019-08-01 09:15:00     10005   20190801091540  48.43   2019-08-01      20190801
2       2019-08-01 10:00:00     10001   20190801100006  89.33   2019-08-01      20190801
3       2019-08-01 10:04:00     10003   20190801091540  63.86   2019-08-01      20190801
4       2019-08-01 12:17:00     10002   20190801121742  3.16    2019-08-01      20190801
5       2019-08-01 14:05:00     10001   20190801140515  87.15   2019-08-01      20190801
6       2019-08-01 14:05:00     10004   20190801140529  88.65   2019-08-01      20190801
7       2019-08-02 08:13:00     10009   20190802081315  36.02   2019-08-02      20190802
8       2019-08-02 11:14:00     10009   20190802111424  95.66   2019-08-02      20190802
9       2019-08-02 13:18:00     10005   20190802131801  89.36   2019-08-02      20190802
10      2019-08-02 15:18:00     10001   20190802151834  71.38   2019-08-02      20190802
11      2019-08-02 16:00:00     10005   20190802160014  63.13   2019-08-02      20190802
12      2019-08-02 17:03:00     10003   20190802170356  79.33   2019-08-02      20190802
13      2019-08-02 17:11:00     10002   20190802171115  56.78   2019-08-02      20190802
14      2019-08-02 19:05:00     10008   20190802190518  23.1    2019-08-02      20190802
15      2019-08-02 20:07:00     10005   20190802200717  73.82   2019-08-02      20190802
```

Hive SQL操作，使用substr截取字串

图片中的代码：

```
#python
import pandas as pd
order = pd.read_csv('order.csv', names=['id', 'ts', 'uid', 'orderid', 'amount'])
order.head()


order['dt'] = order['ts'].str[:10]
order.head()
```

```
order['dt2'] = order['orderid'].astype(str).str[:8]
order.head()

#Hive SQL
select *, substr(ts, 1, 10) as dt, substring(orderid, 1, 8) as dt2
from t_order;
```

二、字符串匹配

这一节我们来研究提取包含特定字符的字段。沿用上一节的写法，在pandas中我们可以使用字符串的contains，extract，replace方法，支持正则表达式。而在hive SQL中，既有简易的Like关键字匹配特定的字符，也可以使用regexp_extract，regexp_replace这两个函数更灵活地实现目标。接下来我们举例说明。

1. 假设要实现**筛选**订单时间中包含“08-01”的订单。pandas和SQL代码如下所示，注意使用like时，%是通配符，表示匹配任意长度的字符。

```
order_08_01 = order[order['ts'].astype(str).str.contains('08-01')]
order_08_01
```

	id	ts	uid	orderid	amount	dt	dt2
0	1	2019-08-01 09:15:00	10005	20190801091540	48.43	2019-08-01	20190801
1	2	2019-08-01 10:00:00	10001	20190801100006	89.33	2019-08-01	20190801
2	3	2019-08-01 10:04:00	10003	20190801091540	63.86	2019-08-01	20190801
3	4	2019-08-01 12:17:00	10002	20190801121742	3.16	2019-08-01	20190801
4	5	2019-08-01 14:05:00	10001	20190801140515	87.15	2019-08-01	20190801
5	6	2019-08-01 14:05:00	10004	20190801140529	88.65	2019-08-01	20190801

pandas操作：使用contains方法进行字符串匹配

```
    > select *
    > from t_order
    > where ts like "%08-01%";
OK
1       2019-08-01 09:15:00     10005   20190801091540  48.43
2       2019-08-01 10:00:00     10001   20190801100006  89.33
3       2019-08-01 10:04:00     10003   20190801091540  63.86
4       2019-08-01 12:17:00     10002   20190801121742  3.16
5       2019-08-01 14:05:00     10001   20190801140515  87.15
6       2019-08-01 14:05:00     10004   20190801140529  88.65
Time taken: 0.504 seconds, Fetched: 6 row(s)
```

Hive SQL操作，使用like关键字进行字符串匹配

图片中的代码：

```
#python
order_08_01 = order[order['ts'].astype(str).str.contains('08-01')]
order_08_01

#Hive SQL
select *
from t_order
where ts like "%08-01%";
```

2.假设要实现**提取**ts中的日期信息(前10位)，pandas里支持正则表达式的extract函数，而hive里除了前文提到的substr函数可以实现外，这里我们可以使用regexp_extract函数，通过正则表达式实现。

```
[12]: order['dt3'] = order['ts'].astype(str).str.extract('(\d{4}-\d{2}-\d{2}).*')
      #这个正则表达式表示4位数字横杠两位数字横杠两位数字，后面是任意字符，
      #我们提取的目标要放在小括号里
      order.head()
```

[12]:

	id	ts	uid	orderid	amount	dt	dt2	dt3
0	1	2019-08-01 09:15:00	10005	20190801091540	48.43	2019-08-01	20190801	2019-08-01
1	2	2019-08-01 10:00:00	10001	20190801100006	89.33	2019-08-01	20190801	2019-08-01
2	3	2019-08-01 10:04:00	10003	20190801091540	63.86	2019-08-01	20190801	2019-08-01
3	4	2019-08-01 12:17:00	10002	20190801121742	3.16	2019-08-01	20190801	2019-08-01
4	5	2019-08-01 14:05:00	10001	20190801140515	87.15	2019-08-01	20190801	2019-08-01

pandas操作：使用extract方法进行字符串提取

```
hive>
    > select *, regexp_extract(ts, '(\\d{4}-\\d{2}-\\d{2}).*', 1) as dt3
    > from t_order;
OK
1       2019-08-01 09:15:00     10005   20190801091540  48.43   2019-08-01
2       2019-08-01 10:00:00     10001   20190801100006  89.33   2019-08-01
3       2019-08-01 10:04:00     10003   20190801091540  63.86   2019-08-01
4       2019-08-01 12:17:00     10002   20190801121742  3.16    2019-08-01
5       2019-08-01 14:05:00     10001   20190801140515  87.15   2019-08-01
6       2019-08-01 14:05:00     10004   20190801140529  88.65   2019-08-01
7       2019-08-02 08:13:00     10009   20190802081315  36.02   2019-08-02
8       2019-08-02 11:14:00     10009   20190802111424  95.66   2019-08-02
9       2019-08-02 13:18:00     10005   20190802131801  89.36   2019-08-02
10      2019-08-02 15:18:00     10001   20190802151834  71.38   2019-08-02
```

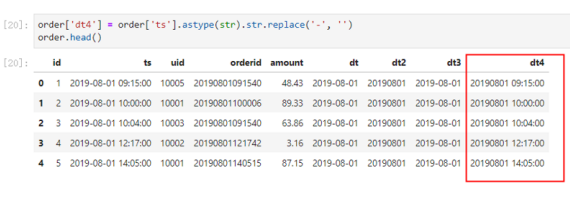
Hive SQL操作，使用regexp_extract函数进行字符串提取

图片中的代码

```
#python
order['dt3'] = order['ts'].astype(str).str.extract('(\d{4}-\d{2}-\d{2}).*')
#这个正则表达式表示"4位数字横杠两位数字横杠两位数字"，后面是任意字符，
#我们提取的目标要放在小括号里
order.head()

#Hive SQL
select *, regexp_extract(ts, '(\\d{4}-\\d{2}-\\d{2}).*', 1) as dt3
from t_order;
#我们的目标同样是在小括号里，1表示取第一个匹配的结果
```

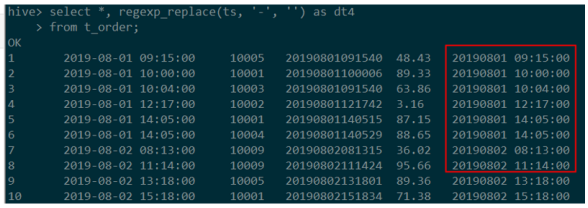
3.假设我们要去掉ts中的横杠，即**替换**ts中的"-"为空，在pandas中可以使用字符串的replace方法，hive中可以使用regexp_replace函数。代码如下：



```
[20]: order['dt4'] = order['ts'].astype(str).str.replace('-', '')
      order.head()
```

[20]:	id	ts	uid	orderid	amount	dt	dt2	dt3	dt4
0	1	2019-08-01 09:15:00	10005	20190801091540	48.43	2019-08-01	20190801	2019-08-01	20190801 09:15:00
1	2	2019-08-01 10:00:00	10001	20190801100006	89.33	2019-08-01	20190801	2019-08-01	20190801 10:00:00
2	3	2019-08-01 10:04:00	10003	20190801091540	63.86	2019-08-01	20190801	2019-08-01	20190801 10:04:00
3	4	2019-08-01 12:17:00	10002	20190801121742	3.16	2019-08-01	20190801	2019-08-01	20190801 12:17:00
4	5	2019-08-01 14:05:00	10001	20190801140515	87.15	2019-08-01	20190801	2019-08-01	20190801 14:05:00

pandas操作：使用replace方法进行字符串替换



```
hive> select *, regexp_replace(ts, '-', '') as dt4
    > from t_order;
OK
1       2019-08-01 09:15:00     10005   20190801091540  48.43   20190801 09:15:00
2       2019-08-01 10:00:00     10001   20190801100006  89.33   20190801 10:00:00
3       2019-08-01 10:04:00     10003   20190801091540  63.86   20190801 10:04:00
4       2019-08-01 12:17:00     10002   20190801121742  3.16    20190801 12:17:00
5       2019-08-01 14:05:00     10001   20190801140515  87.15   20190801 14:05:00
6       2019-08-01 14:05:00     10004   20190801140529  88.65   20190801 14:05:00
7       2019-08-02 08:13:00     10009   20190802081315  36.02   20190802 08:13:00
8       2019-08-02 11:14:00     10009   20190802111424  95.66   20190802 11:14:00
9       2019-08-02 13:18:00     10005   20190802131801  89.36   20190802 13:18:00
10      2019-08-02 15:18:00     10001   20190802151834  71.38   20190802 15:18:00
```

Hive SQL操作，使用replace函数进行字符串替换

图片中代码：

```
#python
order['dt4'] = order['ts'].astype(str).str.replace('-', '')
order.head()

#Hive SQL
select *, regexp_replace(ts, '-', '') as dt4
from t_order;
```

三、带条件的计数：count（distinct case when ...end）

我们在上一篇文章中分别讨论过分组聚合和case操作。实际中，经常会遇到二者嵌套的情况，例如，我们想统计：ts中含有'2019-08-01'的不重复订单有多少，ts中含有'2019-08-02'的不重复订单有多少，这在Hive SQL中比较容易，代码和得到的结果为：

```
select count(distinct case when ts like '%2019-08-01%' then orderid end) as
0801_cnt,
count(distinct case when ts like '%2019-08-02%' then orderid end) as 0802_cnt
from t_order;
#运行结果：
5    11
```

你当然可以直接对日期进行分组，同时计算所有日期的订单数，此处我们仅仅是为了演示两种操作的结合。

pandas中实现这个问题可能比较麻烦，也可能有很多不同的写法。这里说一下我的思路和实现方式。

我定义了两个函数，第一个函数给原数据增加一列，标记我们的条件，第二个函数再增加一列，当满足条件时，给出对应的orderid，然后要对整个dataframe应用这两个函数。对于我们不关心的行，这两列的值都为nan。第三步再进行去重计数操作。代码和结果如下：

```
#第一步：构造一个辅助列
def func_1(x):
```

```
    if '2019-08-01' in x['ts']:
        return '2019-08-01'#这个地方可以返回其他标记
    elif '2019-08-02' in x['ts']:
        return '2019-08-02'
    else:
        return None

#第二步：将符合条件的order作为新的一列
def func_2(x):
    if '2019-08-01' in x['ts']:
        return str(x['orderid'])
    elif '2019-08-02' in x['ts']:
        return str(x['orderid'])
    else:
        return None

#应用两个函数，查看结果
#注意这里必须加上axis=1，你可以尝试下不加会怎样
order['cnt_condition'] = order.apply(func_1, axis=1)
order['cnt'] = order.apply(func_2, axis=1)
order[order['cnt'].notnull()]

#进行分组计数
order.groupby('cnt_condition').agg({'cnt': 'nunique'})
```

[74]:
```
#应用两个函数，查看结果
order['cnt_condition'] = order.apply(func_1, axis=1)
order['cnt'] = order.apply(func_2, axis=1)
order[order['cnt'].notnull()]
```

[74]:

	id	ts	uid	orderid	amount	dt	dt2	dt3	dt4	cnt	cnt_condition
0	1	2019-08-01 09:15:00	10005	20190801091540	48.43	2019-08-01	20190801	2019-08-01	20190801 09:15:00	20190801091540	2019-08-01
1	2	2019-08-01 10:00:00	10001	20190801100006	89.33	2019-08-01	20190801	2019-08-01	20190801 10:00:00	20190801100006	2019-08-01
2	3	2019-08-01 10:04:00	10003	20190801091540	63.86	2019-08-01	20190801	2019-08-01	20190801 10:04:00	20190801091540	2019-08-01
3	4	2019-08-01 12:17:00	10002	20190801121742	3.16	2019-08-01	20190801	2019-08-01	20190801 12:17:00	20190801121742	2019-08-01
4	5	2019-08-01 14:05:00	10001	20190801140515	87.15	2019-08-01	20190801	2019-08-01	20190801 14:05:00	20190801140515	2019-08-01
5	6	2019-08-01 14:05:00	10004	20190801140529	88.65	2019-08-01	20190801	2019-08-01	20190801 14:05:00	20190801140529	2019-08-01
6	7	2019-08-02 08:13:00	10009	20190802081315	36.02	2019-08-02	20190802	2019-08-02	20190802 08:13:00	20190802081315	2019-08-02
7	8	2019-08-02 11:14:00	10009	20190802111424	95.66	2019-08-02	20190802	2019-08-02	20190802 11:14:00	20190802111424	2019-08-02
8	9	2019-08-02 13:18:00	10005	20190802131801	89.36	2019-08-02	20190802	2019-08-02	20190802 13:18:00	20190802131801	2019-08-02
9	10	2019-08-02 15:18:00	10001	20190802151834	71.38	2019-08-02	20190802	2019-08-02	20190802 15:18:00	20190802151834	2019-08-02
10	11	2019-08-02 16:00:00	10005	20190802160014	63.13	2019-08-02	20190802	2019-08-02	20190802 16:00:00	20190802160014	2019-08-02
11	12	2019-08-02 17:03:00	10003	20190802170356	79.33	2019-08-02	20190802	2019-08-02	20190802 17:03:00	20190802170356	2019-08-02
12	13	2019-08-02 17:11:00	10002	20190802171115	56.78	2019-08-02	20190802	2019-08-02	20190802 17:11:00	20190802171115	2019-08-02
13	14	2019-08-02 19:05:00	10008	20190802190518	23.10	2019-08-02	20190802	2019-08-02	20190802 19:05:00	20190802190518	2019-08-02
14	15	2019-08-02 20:07:00	10005	20190802200717	73.82	2019-08-02	20190802	2019-08-02	20190802 20:07:00	20190802200717	2019-08-02
15	16	2019-08-02 20:08:00	10001	20190802200816	82.12	2019-08-02	20190802	2019-08-02	20190802 20:08:00	20190802200816	2019-08-02
16	17	2019-08-02 20:10:00	10003	20190802201002	32.01	2019-08-02	20190802	2019-08-02	20190802 20:10:00	20190802201002	2019-08-02

[75]:
```
#进行分组计数
order.groupby('cnt_condition').agg({'cnt': 'nunique'})
```

[75]:

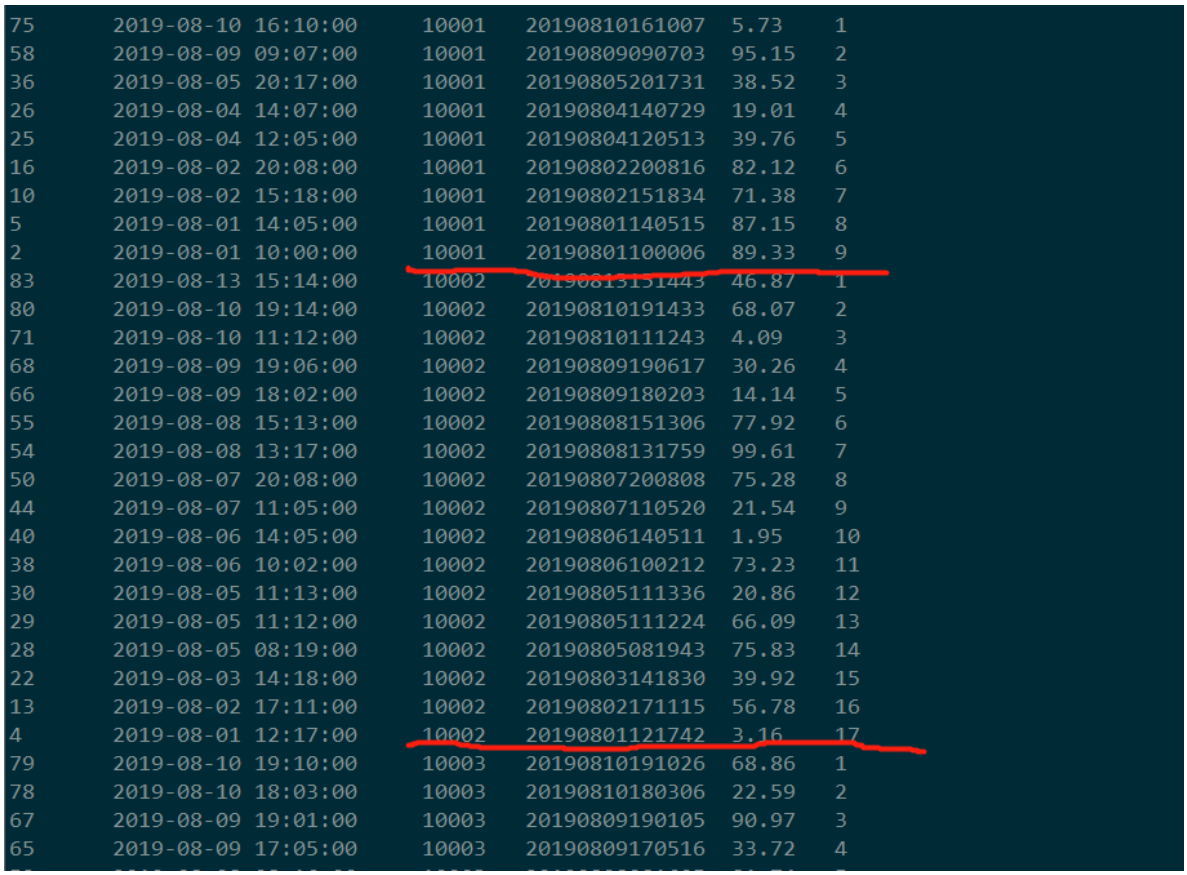
cnt_condition	cnt
2019-08-01	5
2019-08-02	11

可以看到，同样得到了5，11的结果。如果你有其他更好的实现方法，欢迎一起探讨交流。

四、窗口函数 row_number

hive中的row_number函数通常用来分组计数，每组内的序号从1开始增加，且没有重复值。比如我们对每个uid的订单按照订单时间倒序排列，获取其排序的序号。实现的Hive SQL代码如下，可以看到，每个uid都会有一个从1开始的计数，这个计数是按时间倒序排的。

```
select *, row_number() over (partition by uid order by ts desc) as rk
from t_order;
```



75	2019-08-10 16:10:00	10001	20190810161007	5.73	1
58	2019-08-09 09:07:00	10001	20190809090703	95.15	2
36	2019-08-05 20:17:00	10001	20190805201731	38.52	3
26	2019-08-04 14:07:00	10001	20190804140729	19.01	4
25	2019-08-04 12:05:00	10001	20190804120513	39.76	5
16	2019-08-02 20:08:00	10001	20190802200816	82.12	6
10	2019-08-02 15:18:00	10001	20190802151834	71.38	7
5	2019-08-01 14:05:00	10001	20190801140515	87.15	8
2	2019-08-01 10:00:00	10001	20190801100006	89.33	9
83	2019-08-13 15:14:00	10002	20190813151443	46.87	1
80	2019-08-10 19:14:00	10002	20190810191433	68.07	2
71	2019-08-10 11:12:00	10002	20190810111243	4.09	3
68	2019-08-09 19:06:00	10002	20190809190617	30.26	4
66	2019-08-09 18:02:00	10002	20190809180203	14.14	5
55	2019-08-08 15:13:00	10002	20190808151306	77.92	6
54	2019-08-08 13:17:00	10002	20190808131759	99.61	7
50	2019-08-07 20:08:00	10002	20190807200808	75.28	8
44	2019-08-07 11:05:00	10002	20190807110520	21.54	9
40	2019-08-06 14:05:00	10002	20190806140511	1.95	10
38	2019-08-06 10:02:00	10002	20190806100212	73.23	11
30	2019-08-05 11:13:00	10002	20190805111336	20.86	12
29	2019-08-05 11:12:00	10002	20190805111224	66.09	13
28	2019-08-05 08:19:00	10002	20190805081943	75.83	14
22	2019-08-03 14:18:00	10002	20190803141830	39.92	15
13	2019-08-02 17:11:00	10002	20190802171115	56.78	16
4	2019-08-01 12:17:00	10002	20190801121742	3.16	17
79	2019-08-10 19:10:00	10003	20190810191026	68.86	1
78	2019-08-10 18:03:00	10003	20190810180306	22.59	2
67	2019-08-09 19:01:00	10003	20190809190105	90.97	3
65	2019-08-09 17:05:00	10003	20190809170516	33.72	4

pandas中我们需要借助groupby和rank函数来实现同样的效果。改变rank中的method参数可以实现Hive中其他的排序，例如dense，rank等。

```
#由于我们的ts字段是字符串类型，先转换为datetime类型
order['ts2'] =  pd.to_datetime(order['ts'], format='%Y-%m-%d %H:%M:%S')

#进行分组排序，按照uid分组，按照ts2降序，序号默认为小数，需要转换为整数
#并添加为新的一列rk
order['rk'] = order.groupby(['uid'])['ts2'].rank(ascending=False,
method='first').astype(int)

#为了便于查看rk的效果,对原来的数据按照uid和时间进行排序，结果和SQL一致
order.sort_values(['uid','ts'], ascending=[True, False])
```

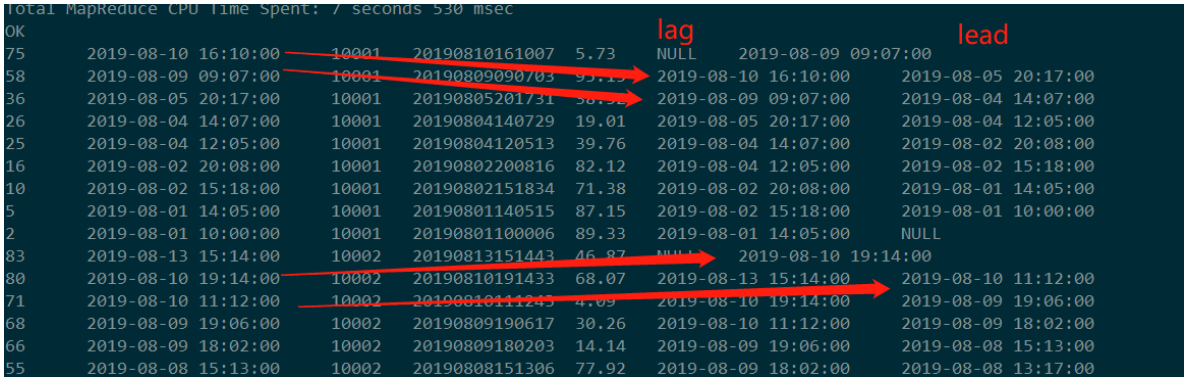
	id	ts	uid	orderid	amount	dt	dt2	dt3	dt4	rk	ts2
74	75	2019-08-10 16:10:00	10001	20190810161007	5.73	2019-08-10	20190810	2019-08-10	20190810 16:10:00	1	2019-08-10 16:10:00
57	58	2019-08-09 09:07:00	10001	20190809090703	95.15	2019-08-09	20190809	2019-08-09	20190809 09:07:00	2	2019-08-09 09:07:00
35	36	2019-08-05 20:17:00	10001	20190805201731	38.52	2019-08-05	20190805	2019-08-05	20190805 20:17:00	3	2019-08-05 20:17:00
25	26	2019-08-04 14:07:00	10001	20190804140729	19.01	2019-08-04	20190804	2019-08-04	20190804 14:07:00	4	2019-08-04 14:07:00
24	25	2019-08-04 12:05:00	10001	20190804120513	39.76	2019-08-04	20190804	2019-08-04	20190804 12:05:00	5	2019-08-04 12:05:00
15	16	2019-08-02 20:08:00	10001	20190802200816	82.12	2019-08-02	20190802	2019-08-02	20190802 20:08:00	6	2019-08-02 20:08:00
9	10	2019-08-02 15:18:00	10001	20190802151834	71.38	2019-08-02	20190802	2019-08-02	20190802 15:18:00	7	2019-08-02 15:18:00
4	5	2019-08-01 14:05:00	10001	20190801140515	87.15	2019-08-01	20190801	2019-08-01	20190801 14:05:00	8	2019-08-01 14:05:00
1	2	2019-08-01 10:00:00	10001	20190801100006	89.33	2019-08-01	20190801	2019-08-01	20190801 10:00:00	9	2019-08-01 10:00:00
82	83	2019-08-13 15:14:00	10002	20190813151443	46.87	2019-08-13	20190813	2019-08-13	20190813 15:14:00	1	2019-08-13 15:14:00
79	80	2019-08-10 19:14:00	10002	20190810191433	68.07	2019-08-10	20190810	2019-08-10	20190810 19:14:00	2	2019-08-10 19:14:00
70	71	2019-08-10 11:12:00	10002	20190810111243	4.09	2019-08-10	20190810	2019-08-10	20190810 11:12:00	3	2019-08-10 11:12:00
67	68	2019-08-09 19:06:00	10002	20190809190617	30.26	2019-08-09	20190809	2019-08-09	20190809 19:06:00	4	2019-08-09 19:06:00
65	66	2019-08-09 18:02:00	10002	20190809180203	14.14	2019-08-09	20190809	2019-08-09	20190809 18:02:00	5	2019-08-09 18:02:00
54	55	2019-08-08 15:13:00	10002	20190808151306	77.92	2019-08-08	20190808	2019-08-08	20190808 15:13:00	6	2019-08-08 15:13:00
53	54	2019-08-08 13:17:00	10002	20190808131759	99.61	2019-08-08	20190808	2019-08-08	20190808 13:17:00	7	2019-08-08 13:17:00
49	50	2019-08-07 20:08:00	10002	20190807200808	75.28	2019-08-07	20190807	2019-08-07	20190807 20:08:00	8	2019-08-07 20:08:00
43	44	2019-08-07 11:05:00	10002	20190807110520	21.54	2019-08-07	20190807	2019-08-07	20190807 11:05:00	9	2019-08-07 11:05:00
39	40	2019-08-06 14:05:00	10002	20190806140511	1.95	2019-08-06	20190806	2019-08-06	20190806 14:05:00	10	2019-08-06 14:05:00
37	38	2019-08-06 10:02:00	10002	20190806100212	73.23	2019-08-06	20190806	2019-08-06	20190806 10:02:00	11	2019-08-06 10:02:00
29	30	2019-08-05 11:13:00	10002	20190805111336	20.86	2019-08-05	20190805	2019-08-05	20190805 11:13:00	12	2019-08-05 11:13:00
28	29	2019-08-05 11:12:00	10002	20190805111224	66.09	2019-08-05	20190805	2019-08-05	20190805 11:12:00	13	2019-08-05 11:12:00
27	28	2019-08-05 08:19:00	10002	20190805081943	75.83	2019-08-05	20190805	2019-08-05	20190805 08:19:00	14	2019-08-05 08:19:00
21	22	2019-08-03 14:18:00	10002	20190803141830	39.92	2019-08-03	20190803	2019-08-03	20190803 14:18:00	15	2019-08-03 14:18:00
12	13	2019-08-02 17:11:00	10002	20190802171115	56.78	2019-08-02	20190802	2019-08-02	20190802 17:11:00	16	2019-08-02 17:11:00
3	4	2019-08-01 12:17:00	10002	20190801121742	3.16	2019-08-01	20190801	2019-08-01	20190801 12:17:00	17	2019-08-01 12:17:00

五、窗口函数 lag，lead

lag和lead函数也是Hive SQL中常用的窗口函数，他们的格式为：

```
lag(字段名,N) over(partition by 分组字段 order by 排序字段 排序方式)
lead(字段名,N) over(partition by 分组字段 order by 排序字段 排序方式)
```

lag函数表示，取分组排序之后比该条记录序号小N的对应记录的指定字段的值。lead刚好相反，是比当前记录大N的对应记录的指定字段值。我们来看例子。



例子中的lag表示分组排序后，前一条记录的ts，lead表示后一条记录的ts。不存在的用NULL填充。

对应的代码为：

```
select *,
lag(ts, 1) over (partition by uid order by ts desc) as lag,
lead(ts, 1) over (partition by uid order by ts desc) as lead
from t_order;
```

pandas中我们也有相应的shift函数来实现这样的需求。shift的参数为负数时，表示lag，为正数时，表示lead。

```
order['lag'] =  order.groupby(['uid'])['ts2'].shift(-1)
order['lead'] =  order.groupby(['uid'])['ts2'].shift(1)
```

```
order.sort_values(['uid','ts'], ascending=[True, False])
```

	id	ts	uid	orderid	amount	dt	dt2	dt3	dt4	rk	ts2	lag	lead
74	75	2019-08-10 16:10:00	10001	20190810161007	5.73	2019-08-10	20190810	2019-08-10	20190810 16:10:00	1	2019-08-10 16:10:00	NaT	2019-08-09 09:07:00
57	58	2019-08-09 09:07:00	10001	20190809090703	95.15	2019-08-09	20190809	2019-08-09	20190809 09:07:00	2	2019-08-09 09:07:00	2019-08-10 16:10:00	2019-08-05 20:17:00
35	36	2019-08-05 20:17:00	10001	20190805201731	38.52	2019-08-05	20190805	2019-08-05	20190805 20:17:00	3	2019-08-05 20:17:00	2019-08-09 09:07:00	2019-08-04 14:07:00
25	26	2019-08-04 14:07:00	10001	20190804140729	19.01	2019-08-04	20190804	2019-08-04	20190804 14:07:00	4	2019-08-04 14:07:00	2019-08-05 20:17:00	2019-08-04 12:05:00
24	25	2019-08-04 12:05:00	10001	20190804120513	39.76	2019-08-04	20190804	2019-08-04	20190804 12:05:00	5	2019-08-04 12:05:00	2019-08-04 14:07:00	2019-08-02 20:08:00
15	16	2019-08-02 20:08:00	10001	20190802200816	82.12	2019-08-02	20190802	2019-08-02	20190802 20:08:00	6	2019-08-02 20:08:00	2019-08-04 12:05:00	2019-08-02 15:18:00
9	10	2019-08-02 15:18:00	10001	20190802151834	71.38	2019-08-02	20190802	2019-08-02	20190802 15:18:00	7	2019-08-02 15:18:00	2019-08-02 20:08:00	2019-08-01 14:05:00
4	5	2019-08-01 14:05:00	10001	20190801140515	87.15	2019-08-01	20190801	2019-08-01	20190801 14:05:00	8	2019-08-01 14:05:00	2019-08-02 15:18:00	2019-08-01 10:00:00
1	2	2019-08-01 10:00:00	10001	20190801100006	89.33	2019-08-01	20190801	2019-08-01	20190801 10:00:00	9	2019-08-01 10:00:00	2019-08-01 14:05:00	NaT
82	83	2019-08-13 15:14:00	10002	20190813151443	46.87	2019-08-13	20190813	2019-08-13	20190813 15:14:00	1	2019-08-13 15:14:00	NaT	2019-08-10 19:14:00
79	80	2019-08-10 19:14:00	10002	20190810191433	68.07	2019-08-10	20190810	2019-08-10	20190810 19:14:00	2	2019-08-10 19:14:00	2019-08-13 15:14:00	2019-08-10 11:12:00
70	71	2019-08-10 11:12:00	10002	20190810111243	4.09	2019-08-10	20190810	2019-08-10	20190810 11:12:00	3	2019-08-10 11:12:00	2019-08-10 19:14:00	2019-08-09 19:06:00
67	68	2019-08-09 19:06:00	10002	20190809190617	30.26	2019-08-09	20190809	2019-08-09	20190809 19:06:00	4	2019-08-09 19:06:00	2019-08-10 11:12:00	2019-08-09 18:02:00

代码如下：

```
order['lag'] =  order.groupby(['uid'])['ts2'].shift(-1)
order['lead'] =  order.groupby(['uid'])['ts2'].shift(1)

#依然是为了看效果，对原来的数据按照uid和时间进行排序，结果和SQL一致
order.sort_values(['uid','ts'], ascending=[True, False])
```

六、列转行，collect_list

在我们的数据中，一个uid会对应多个订单，目前这多个订单id是分多行显示的。现在我们要做的是让多个订单id显示在同一行，用逗号分隔开。在pandas中，我们采用的做法是先把原来orderid列转为字符串形式，并在每一个id末尾添加一个逗号作为分割符，然后采用字符串相加的方式，将每个uid对应的字符串类型的订单id拼接到一起。代码和效果如下所示。为了减少干扰，我们将order数据重新读入，并设置了pandas的显示方式。

```
import pandas as pd
pd.set_option('display.max_columns', None)
#显示所有行
pd.set_option('display.max_rows', None)
#设置value的显示长度为100，默认为50
pd.set_option('max_colwidth',100)

order = pd.read_csv('order.csv', names=['id', 'ts', 'uid', 'orderid', 'amount'])
order.head()
```

	id	ts	uid	orderid	amount
0	1	2019-08-01 09:15:00	10005	20190801091540	48.43
1	2	2019-08-01 10:00:00	10001	20190801100006	89.33
2	3	2019-08-01 10:04:00	10003	20190801091540	63.86
3	4	2019-08-01 12:17:00	10002	20190801121742	3.16
4	5	2019-08-01 14:05:00	10001	20190801140515	87.15

```
order['orderid'] = order['orderid'].astype('str')
order['orderid'] = order['orderid'].apply(lambda x: x + ",")
order_group = order.groupby('uid').agg({'orderid':'sum'})
order_group
```

uid	orderid
10001	20190801100006,20190801140515,20190802151834,20190802200816,20190804120513,20190804140729,201908...
10002	20190801121742,20190802171115,20190803141830,20190805081943,20190805111224,20190805111336,201908...
10003	20190801091540,20190802170356,20190802201002,20190803100858,20190805150532,20190805191729,201908...
10004	20190801140529,20190804090709,20190804101308,20190805130103,20190805180710,20190809201337,201908...
10005	20190801091540,20190802131801,20190802160014,20190802200717,20190804141616,20190805161248,201908...
10006	20190809141227,

可以看到，同一个uid对应的订单id已经显示在同一行了，订单id之间以逗号分隔。

在Hive中实现同样的效果要方便多了，我们可以使用collect_set/collect_list函数，，二者的区别在于前者在聚合时会进行去重，别忘了加上group by。

```
select uid, collect_set(orderid) as order_list
from t_order
group by uid
;
```

```
OK
10001   ["20190801100006","20190801140515","20190802151834","20190802200816","20190804120513","20190804140729","20190805201731","20190809090703","20190810161007"]
10002   ["20190801121742","20190802171115","20190803141830","20190805081943","20190805111224","20190805111336","20190806100212","20190806140511","20190807110520","20190807200808
","20190808131759","20190808151306","20190809180203","20190809190617","20190810111243","20190810191433","20190813151443"]
10003   ["20190801091540","20190802170356","20190802201002","20190803100858","20190805150532","20190805191729","20190806170150","20190807201432","20190808081605","20190809170516
","20190809190105","20190810180306","20190810191026"]
10004   ["20190801140529","20190804090709","20190804101308","20190805130103","20190805180710","20190809201337","20190810131853","20190810151032"]
10005   ["20190801091540","20190802131801","20190802160014","20190802200717","20190804141616","20190805161248","20190806181523","20190807160957","20190808180638"]
10006   ["20190809141227"]
10007   ["20190808111203","20190808170710","20190809150204","20190809160206","20190810080224","20190811151111","20190812151638"]
10008   ["20190802190518","20190807080937","20190807130918","20190807150301","20190809121131","20190809161842","20190810150838","20190810170853"]
10009   ["20190802081315","20190802111424","20190803090247","20190803120818","20190803140705","20190806091123","20190806120531","20190807171717","20190807190318","20190809101113
","20190810170641"]
Time taken: 39.095 seconds, Fetched: 9 row(s)
```

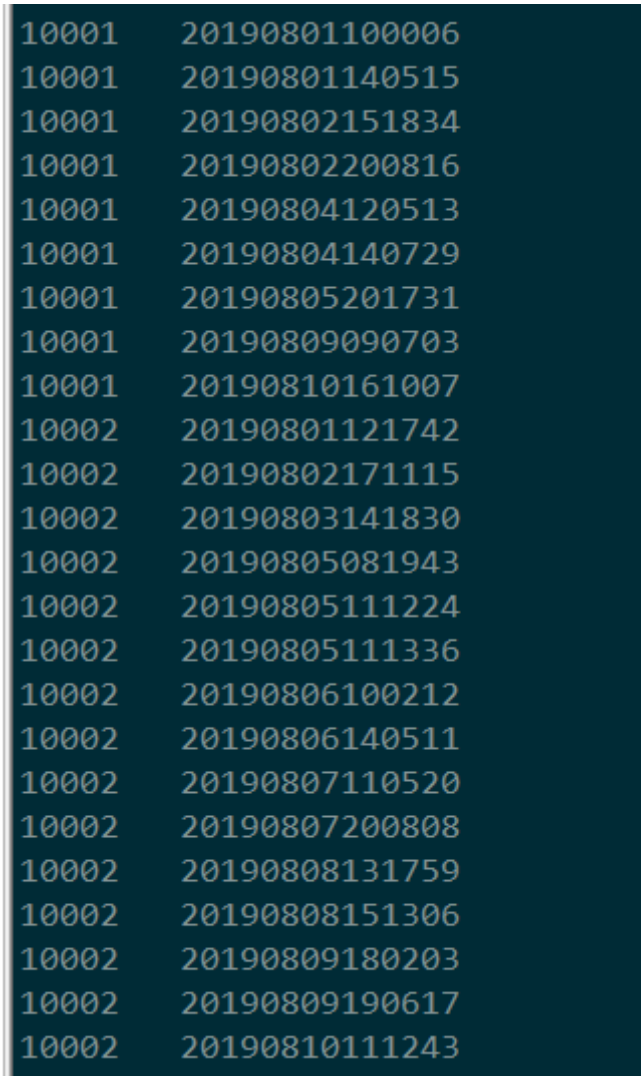
可以看出hive实现的效果中，将同一个uid的orderid作为一个“数组”显示出来。虽然和pandas实现的效果不完全一样，但表达的含义是一致的。我没有找到pandas实现这样数组形式比较好的方法，如果你知道，欢迎一起交流.另外，pandas在聚合时，如何去重，也是一个待解决的问题。

七 行转列 later view explode

行转列的操作在Hive SQL中有时会遇到，可以理解为将上一小节的结果还原为每个orderid显示一行的形式。hive中有比较方便的explode函数，结合lateral view，可以很容易实现。代码和效果如下：

```
-- 使用上一节的结果，定义为tmp表，后面可以直接用
with tmp as
(
select uid, collect_set(orderid) as order_list
from t_order
group by uid
)

select uid, o_list
from tmp lateral view explode(order_list) t as o_list;
```



我们来看在pandas中的实现。目标是把上一节合并起来的用逗号分隔的数组拆分开。这里给出一个参考链接：https://blog.csdn.net/sscc_learning/article/details/89473151。

首先我们要把groupby的结果索引重置一下，然后再进行遍历，和赋值，最后将每一个series拼接起来。我采用的是链接中的第一种方式。由于是遍历，效率可能比较低下，读者可以尝试下链接里的另一种方式。我先给出我的代码：

```
order_group = order_group.reset_index()
order_group

order_group1 = pd.concat([pd.Series(row['uid'], row['orderid'].split(',')) for _
, row in order_group.iterrows()]).reset_index()
order_group1
```

这样的结果中会有一个空行，这是因为用逗号分隔的时候，最后一个元素为空。后续可以使用我们之前学习的方法进行过滤或删除。这里省略这一步骤。

	index	0
0	20190801100006	10001
1	20190801140515	10001
2	20190802151834	10001
3	20190802200816	10001
4	20190804120513	10001
5	20190804140729	10001
6	20190805201731	10001
7	20190809090703	10001
8	20190810161007	10001
9		10001
10	20190801121742	10002
11	20190802171115	10002
12	20190803141830	10002
13	20190805081943	10002
14	20190805111224	10002
15	20190805111336	10002
16	20190806100212	10002
17	20190806140511	10002
18	20190807110520	10002
19	20190807200808	10002

八、数组元素解析

这一小节我们引入一个新的数据集，原因是我想分享的内容，目前的数据集不能够体现，哈哈。下面是在Hive和pandas中查看数据样例的方式。我们的目标是将原始以字符串形式存储的数组元素解析出来。

```
hive> CREATE TABLE `new_data`(
    > `id` int comment "id",
    > `arr` string  comment "数组"
    > )
    > ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
    > STORED AS TEXTFILE;
OK
Time taken: 0.175 seconds
hive> load data local inpath 'new_data.txt' overwrite into table new_data;
Loading data to table test.new_data
Table test.new_data stats: [numFiles=1, numRows=0, totalSize=291, rawDataSize=0]
OK
Time taken: 0.345 seconds
hive> select * from new_data;
OK
1       ["1","1","1","1","1","1"]
2       ["1","1","1","1","1","1"]
3       ["1","1","1","1","1","1"]
4       ["1","1","1","1","1","1"]
5       ["1","1","1","1","1","1"]
6       ["1","1","1","1","1","1"]
7       ["1","0","0","0","0","0"]
8       ["1","1","1","1","1","1"]
9       ["1","1","0","0","0","0"]
10      ["1","1","1","1","1","0"]
Time taken: 0.036 seconds, Fetched: 10 row(s)
```

```
import pandas as pd
new_data = pd.read_excel('new_data.xlsx')
new_data
```

	uid	arr
0	1	["1","1","1","1","1","1"]
1	2	["1","1","1","1","1","1"]
2	3	["1","1","1","1","1","1"]
3	4	["1","1","1","1","1","1"]
4	5	["1","1","1","1","1","1"]
5	6	["1","1","1","1","1","1"]
6	7	["1","0","0","0","0","0"]
7	8	["1","1","1","1","1","1"]
8	9	["1","1","0","0","0","0"]
9	10	["1","1","1","1","1","0"]

```
new_data.dtypes
```

```
uid     int64
arr    object
dtype: object
```

先来看pandas中如何实现，这里我们需要用到literal_eval这个包，能够自动识别以字符串形式存储的数组。我定义了一个解析函数，将arr列应用该函数多次，解析出的结果作为新的列，代码如下：

[157]:
```
def extract_num(x, i):
    from ast import literal_eval
    return literal_eval(x)[i]
```

[162]:
```
new_data['arr_1'] =  new_data.arr.apply(extract_num, args=(0,))#0代表第一个数字
new_data['arr_2'] =  new_data.arr.apply(extract_num, args=(1,))#1代表第二个数字
new_data['arr_3'] =  new_data.arr.apply(extract_num, args=(2,))#2代表第三个数字
new_data['arr_4'] =  new_data.arr.apply(extract_num, args=(3,))#3代表i第四个数字
new_data['arr_5'] =  new_data.arr.apply(extract_num, args=(4,))#4代表第五个数字
new_data['arr_6'] =  new_data.arr.apply(extract_num, args=(5,))#5代表第六个数字
new_data
```

[162]:

	uid	arr	arr_1	arr_2	arr_3	arr_4	arr_5	arr_6
0	1	["1","1","1","1","1","1"]	1	1	1	1	1	1
1	2	["1","1","1","1","1","1"]	1	1	1	1	1	1
2	3	["1","1","1","1","1","1"]	1	1	1	1	1	1
3	4	["1","1","1","1","1","1"]	1	1	1	1	1	1
4	5	["1","1","1","1","1","1"]	1	1	1	1	1	1
5	6	["1","1","1","1","1","1"]	1	1	1	1	1	1
6	7	["1","0","0","0","0","0"]	1	0	0	0	0	0
7	8	["1","1","1","1","1","1"]	1	1	1	1	1	1
8	9	["1","1","0","0","0","0"]	1	1	0	0	0	0
9	10	["1","1","1","1","1","0"]	1	1	1	1	1	0

[163]:
```
new_data.dtypes
```

[163]:
```
uid       int64
arr      object
arr_1    object
arr_2    object
arr_3    object
arr_4    object
arr_5    object
arr_6    object
dtype: object
```

这里需要注意解析出的结果是object类型的，如果想让它们参与数值计算，需要再转换为int类型，可以在解析的时候增加转换的代码。

```
new_data['arr_1'] =  new_data.arr.apply(extract_num, args=(0,)).astype(int)
```

回到Hive SQL，实现起来比较容易。我们可以通过split函数将原来的字符串形式变为数组，然后依次取数组的元素即可，但是要注意使用substr函数处理好前后的中括号，代码如下：

```
hive> select id, arr,
    > split(substr(arr, 2, length(arr)-2), ',')[0] arr_1,
    > split(substr(arr, 2, length(arr)-2), ',')[1] arr_2,
    > split(substr(arr, 2, length(arr)-2), ',')[2] arr_3,
    > split(substr(arr, 2, length(arr)-2), ',')[3] arr_4,
    > split(substr(arr, 2, length(arr)-2), ',')[4] arr_5,
    > split(substr(arr, 2, length(arr)-2), ',')[5] arr_6
    > from new_data;
OK
1       ["1","1","1","1","1","1"]       "1"     "1"     "1"     "1"     "1"     "1"
2       ["1","1","1","1","1","1"]       "1"     "1"     "1"     "1"     "1"     "1"
3       ["1","1","1","1","1","1"]       "1"     "1"     "1"     "1"     "1"     "1"
4       ["1","1","1","1","1","1"]       "1"     "1"     "1"     "1"     "1"     "1"
5       ["1","1","1","1","1","1"]       "1"     "1"     "1"     "1"     "1"     "1"
6       ["1","1","1","1","1","1"]       "1"     "1"     "1"     "1"     "1"     "1"
7       ["1","0","0","0","0","0"]       "1"     "0"     "0"     "0"     "0"     "0"
8       ["1","1","1","1","1","1"]       "1"     "1"     "1"     "1"     "1"     "1"
9       ["1","1","0","0","0","0"]       "1"     "1"     "0"     "0"     "0"     "0"
10      ["1","1","1","1","1","0"]       "1"     "1"     "1"     "1"     "1"     "0"
Time taken: 0.293 seconds, Fetched: 10 row(s)
```

可以看到最终我们得到的结果是字符串的形式，如果想要得到数值，可以再进行一步截取。

```
hive> select id, arr,
    > substr(split(substr(arr, 2, length(arr)-2), ',')[0], 2, 1) arr_1,
    > substr(split(substr(arr, 2, length(arr)-2), ',')[1], 2, 1) arr_2,
    > substr(split(substr(arr, 2, length(arr)-2), ',')[2], 2, 1) arr_3,
    > substr(split(substr(arr, 2, length(arr)-2), ',')[3], 2, 1) arr_4,
    > substr(split(substr(arr, 2, length(arr)-2), ',')[4], 2, 1) arr_5,
    > substr(split(substr(arr, 2, length(arr)-2), ',')[5], 2, 1) arr_6
    > from new_data;
OK
1       ["1","1","1","1","1","1"]       1       1       1       1       1       1
2       ["1","1","1","1","1","1"]       1       1       1       1       1       1
3       ["1","1","1","1","1","1"]       1       1       1       1       1       1
4       ["1","1","1","1","1","1"]       1       1       1       1       1       1
5       ["1","1","1","1","1","1"]       1       1       1       1       1       1
6       ["1","1","1","1","1","1"]       1       1       1       1       1       1
7       ["1","0","0","0","0","0"]       1       0       0       0       0       0
8       ["1","1","1","1","1","1"]       1       1       1       1       1       1
9       ["1","1","0","0","0","0"]       1       1       0       0       0       0
10      ["1","1","1","1","1","0"]       1       1       1       1       1       0
Time taken: 0.245 seconds, Fetched: 10 row(s)
```

可以看到，我们这里得到的依然是字符串类型，和pandas中的强制转换类似，hive SQL中也有类型转换的函数cast，使用它可以强制将字符串转为整数，使用方法如下面代码所示。

```
hive> select id, arr,
    > cast(substr(split(substr(arr, 2, length(arr)-2), ',')[0], 2, 1) as int) arr_1,
    > cast(substr(split(substr(arr, 2, length(arr)-2), ',')[1], 2, 1) as int) arr_2,
    > cast(substr(split(substr(arr, 2, length(arr)-2), ',')[2], 2, 1) as int) arr_3,
    > cast(substr(split(substr(arr, 2, length(arr)-2), ',')[3], 2, 1) as int) arr_4,
    > cast(substr(split(substr(arr, 2, length(arr)-2), ',')[4], 2, 1) as int) arr_5,
    > cast(substr(split(substr(arr, 2, length(arr)-2), ',')[5], 2, 1) as int) arr_6
    > from new_data;
OK
1       ["1","1","1","1","1","1"]       1       1       1       1       1       1
2       ["1","1","1","1","1","1"]       1       1       1       1       1       1
3       ["1","1","1","1","1","1"]       1       1       1       1       1       1
4       ["1","1","1","1","1","1"]       1       1       1       1       1       1
5       ["1","1","1","1","1","1"]       1       1       1       1       1       1
6       ["1","1","1","1","1","1"]       1       1       1       1       1       1
7       ["1","0","0","0","0","0"]       1       0       0       0       0       0
8       ["1","1","1","1","1","1"]       1       1       1       1       1       1
9       ["1","1","0","0","0","0"]       1       1       0       0       0       0
10      ["1","1","1","1","1","0"]       1       1       1       1       1       0
Time taken: 0.059 seconds, Fetched: 10 row(s)
```

小结

本文涉及的操作概括如下表所示，虽然内容没有上篇文章多，但相对难度还是比上篇高一些。

序号	操作	pandas	Hive SQL
1	字符串截取	astype转为str之后用切片方式截取	substr,substring函数
2	字符串匹配-筛选	str.contains()	like 关键字
3	字符串匹配-提取	str.extract()	regexp_extract函数
4	字符串匹配-替换	str.replace()	regexp_replace函数
5	带条件的计数	构造两个辅助列再进行计数	count(distinct case when … end)
6	窗口函数-排序	groupby + rank函数	row_number() over (partition by)
7	窗口函数-前后记录	shift函数，负数为lag，正数为lead	lag/lead(col, n) over (partition by)
8	列转行	转换为字符串的聚合相加	collect_list/collect_set函数
9	行转列	遍历重新构造dataframe	lateral view + explode 函数
10	数组元素解析	literal包的literal_eval()	split + substr函数
11	类型转换	astype()	cast(XXX as YYY) 函数

如果你认真读了本文，会发现有一些情况下，Hive SQL比pandas更方便，为了达到同样的效果，pandas可能要用一种全新的方式来实现。实际工作中，如果数据存在数据库中，使用SQL语句来处理还是方便不少的，尤其是如果数据量大了，pandas可能会显得有点吃力。本文的出发点仅仅是对比两者的操作，方便从两个角度理解常见的数据处理手段，也方便工作中的转换查阅，不强调孰优孰劣。对于文中遗留的不是很完美的地方，如果您想到了好的方案，欢迎一起探讨交流~文中用到的数据和代码我已经打包整理好，在公众号后台回复“**对比二**”即可获得，祝您练习愉快！