一场Pandas与SQL的巅峰大战六

在之前的五篇系列文章中，我们对比了pandas和SQL在数据处理方面的多项操作。

具体来讲，第一篇文章涉及到数据查看，去重计数，条件选择，合并连接，分组排序等操作。

第二篇文章涉及字符串处理，窗口函数，行列转换，类型转换等操作。

第三篇文章围绕日期操作展开，主要讨论了日期获取，日期转换，日期计算等内容。

第四篇文章学习了在MySQL，Hive SQL和pandas中用多种方式计算日环比，周同比的方法。

第五篇文章我们用多种方案实现了分组和不分组情况下累计百分比的计算。

本篇我们主要来总结学习SQL和pandas中计算日活和多日留存的方法。

数据准备

先来看一下日活和留存的定义，对任何一款App而言，这两个指标都是很重要的。

日活(Daily Active User，即DAU)顾名思义即每天的活跃用户，至于如何定义就有多种口径了。一方面要约定何为“活跃”，可以是启动一次App，可以是到达某一个页面，可以是进入App后产生某一个行为等等。另一方面要约定计量的口径，可以是计算用户id的去重数，也可以是设备id的去重数。这两种口径统计结果会有差异，原因在于未登录的用户可能存在设备id，不存在用户id；并且设备id与用户id可能存在多对多的情况。因此对于运营来讲，确定合理有效的口径是很重要的。

留存是一个动态的概念，指的是某段时间使用了产品的用户，在一段时间之后仍然在使用产品的用户，二者相比可以求出留存率。常见的留存率有次日留存率，7日留存率，30日留存率等。次日留存是指今天活跃的用户，在明天还剩下多少仍然活跃的用户。留存率越高，说明产品的粘性越好。

我们的数据是一份用户登录数据，数据来源为。数据格式比较简单：id：自增id，uid：用户唯一id。ts：用户登录的时间(精确到秒)，数据样例如下图，在公众号后台回复“**对比六**”可以获得本文全部的数据和代码，方便进行实操。

本次我们只用到MySQL和pandas。MySQL可以直接运行我提供的`login.sql`文件加载数据，具体过程可以参考前面的文章。。pandas中直接使用read_excel的方式读取即可，可以参考后面的代码。

日活计算

这里我们约定日活是指每天登录的user_id去重数，从我们的数据来看，计算方式非常简单。

SQL计算日活

早在系列第一篇中我们就学习过group by聚合操作。只需要按天分组，将uid去重计数，即可得到答案。代码如下：

```
select substr(ts, 1, 10) as dt, count(distinct uid)
from t_login
group by substr(ts, 1, 10)
```

dt	dau
2017-01-07	3
2017-01-08	3
2017-01-09	9
2017-01-10	9
2017-01-11	8
2017-01-12	10
2017-01-13	8
2017-01-14	4
2017-01-15	4
2017-01-16	11
2017-01-17	7
2017-01-18	11

pandas计算日活

pandas计算日活也不难，同样是使用groupby，对uid进行去重计数。代码如下：

```
import pandas as pd
login_data = pd.read_csv('login_data.txt', sep='\t', parse_dates=['ts'])
login_data.head()

login_data['day'] = login_data['ts'].map(lambda x: x.strftime('%Y-%m-%d'))
uid_count = login_data.groupby('day').aggregate({'uid': lambda x: x.nunique()})
uid_count.reset_index(inplace=True)
uid_count
```

我们增加了一列精确到天的日期数据，便于后续分组。在聚合时，使用了nunique进行去重。在这里也纠正一下系列第一篇文章中第6部分中的写法，np.size 是不去重的，相当于count，但又不能直接写np.nunique，所以我们采用了lambda函数的形式。感谢热心读者的指出~最终uid_count的输出结果如下图所示，uid列就是我们要求的dau，结果和SQL算出来一样。可以再用rename对列进行重命名，此处略：

	day	uid
0	2017-01-07	3
1	2017-01-08	3
2	2017-01-09	9
3	2017-01-10	9
4	2017-01-11	8
5	2017-01-12	10
6	2017-01-13	8
7	2017-01-14	4
8	2017-01-15	4
9	2017-01-16	11
10	2017-01-17	7
11	2017-01-18	11

留存率计算

如前文所示，这里我们定义，留存率是指一段时间后仍然登录的用户占第一天登录用户的比例，由于2017-01-07登录的用户太少，我们选择2017-01-12作为第一天。分别计算次日留存率，7日，14日留存率。

SQL方式

次日留存计算

同前面计算日环比周同比一样，我们可以采用自连接的方式，但连接的条件除了日期外，还需要加上uid，这是一个更加严格的限制。左表计数求出初始活跃用户，右表计数求出留存用户，之后就可以求出留存率。代码如下：

```
SELECT substr(a.ts, 1, 10) as dt,
count(distinct a.uid), count(distinct b.uid),
concat(round((count(distinct b.uid) / count(distinct a.uid)) * 100, 2) ,'%') as
1_day_remain
from t_login a
left join t_login b
on a.uid = b.uid
and date_add(substr(a.ts, 1, 10), INTERVAL 1 day) = substr(b.ts, 1, 10)
group by substr(a.ts, 1, 10)
```

得到的结果如下：

dt	count(distinct a.uid)	count(distinct b.uid)	1_day_remain
2017-01-12	10	5	50.00%
2017-01-13	8	2	25.00%
2017-01-14	4	2	50.00%
2017-01-15	4	3	75.00%
2017-01-16	11	5	45.45%
2017-01-17	7	5	71.43%
2017-01-18	11	3	27.27%
2017-01-19	7	4	57.14%
2017-01-20	7	3	42.86%

多日留存计算

上面自连接的方法固然可行，但是如果要同时计算次日，7日，14日留存，还需要在此基础上进行关联两次，关联条件分别为日期差为6和13。读者可以试试写一下代码。当数据量比较大时，多次关联在执行效率上会有瓶颈。因此我们可以考虑新的思路。在确定要求固定日留存时，我们使用了日期关联，那么如果不确定求第几日留存的情况下，是不是可以不写日期关联的条件呢，答案是肯定的。来看代码

```
select substr(a.ts, 1, 10) as dt,
count(distinct a.uid),
count(distinct if(datediff(substr(b.ts, 1, 10), substr(a.ts, 1, 10))=1, b.uid,
null)) as 1_day_remain_uid,
count(distinct if(datediff(substr(b.ts, 1, 10), substr(a.ts, 1, 10))=6, b.uid,
null)) as 7_day_remain_uid,
count(distinct if(datediff(substr(b.ts, 1, 10), substr(a.ts, 1, 10))=13, b.uid,
null)) as 14_day_remain_uid
from t_login a
left join t_login b
on a.uid = b.uid
group by
substr(a.ts, 1, 10)
```

如代码所示，在关联时先不限制日期，最外层查询时根据自己的目标限定日期差，可以算出相应的留存用户数，第一天的活跃用户也可以看作是日期差为0时的情况。这样就可以一次性计算多日留存了。结果如下，如果要计算留存率，只需转换为对应的百分比即可，参考前面的代码，此处略。

dt	count(distinct a.uid)	1_day_remain_uid	7_day_remain_uid	14_day_remain_uid
2017-01-12	10	5	7	3
2017-01-13	8	2	4	1
2017-01-14	4	2	3	2
2017-01-15	4	3	2	1
2017-01-16	11	5	1	0
2017-01-17	7	5	5	2
2017-01-18	11	3	2	3
2017-01-19	7	4	2	2
2017-01-20	7	3	2	2

pandas方式

次日留存计算

pandas计算留存也是仅仅围绕我们的目标进行：同时求出第一日和次日的活跃用户数，然后求比值。同样也可以采用自连接的方式。代码如下(这里的步骤比较多)：

1.导入数据并添加两列日期，分别是字符串格式和datetime64格式，便于后续日期计算

```
import pandas as pd
from datetime import timedelta
login_data = pd.read_csv('login_data.txt', sep='\t', parse_dates=['ts'])
login_data['day'] = login_data['ts'].map(lambda x: x.strftime('%Y-%m-%d'))
login_data['dt_ts'] = pd.to_datetime(login_data['day'], format='%Y-%m-%d')
login_data.head()
```

```
login_data = pd.read_csv('login_data.txt', sep='\t', parse_dates=['ts'])
login_data['day'] = login_data['ts'].map(lambda x: x.strftime('%Y-%m-%d'))
login_data['dt_ts'] = pd.to_datetime(login_data['day'], format='%Y-%m-%d')
login_data.head()
```

	id	uid	ts	day	dt_ts
0	1	466	2017-01-07 18:24:55	2017-01-07	2017-01-07
1	2	458	2017-01-07 18:25:18	2017-01-07	2017-01-07
2	3	458	2017-01-07 18:26:21	2017-01-07	2017-01-07
3	4	592	2017-01-07 19:09:59	2017-01-07	2017-01-07
4	5	393	2017-01-08 00:41:19	2017-01-08	2017-01-08

2.构造新的dataframe，计算日期，之后与原数据进行连接

```
data_1 = login_data.copy()
data_1['dt_ts_1'] = data_1['dt_ts'] + timedelta(-1)
data_1.head()
```

:

```
data_1 = login_data.copy()
data_1['dt_ts_1'] = data_1['dt_ts'] + timedelta(-1)
data_1.head()
```

:

	id	uid	ts	day	dt_ts	dt_ts_1
0	1	466	2017-01-07 18:24:55	2017-01-07	2017-01-07	2017-01-06
1	2	458	2017-01-07 18:25:18	2017-01-07	2017-01-07	2017-01-06
2	3	458	2017-01-07 18:26:21	2017-01-07	2017-01-07	2017-01-06
3	4	592	2017-01-07 19:09:59	2017-01-07	2017-01-07	2017-01-06
4	5	393	2017-01-08 00:41:19	2017-01-08	2017-01-08	2017-01-07

3.合并前面的两个数据，使用uid和dt_ts 关联，dt_ts_1是当前日期减一天，左边是第一天活跃的用户，右边是第二天活跃的用户

```
merge_1 = pd.merge(login_data, data_1, left_on=['uid', 'dt_ts'], right_on=
['uid', 'dt_ts_1'], how='left')
merge_1.head(10)
```

```
#dt_ts_1是当前日期减一天，使用uid和dt_ts 关联，左边是第一天活跃的用户，右边是第二天活跃的用户
merge_1 = pd.merge(login_data, data_1, left_on=['uid', 'dt_ts'], right_on=['uid', 'dt_ts_1'], how='left')
merge_1.head(10)
```

	id_x	uid	ts_x	day_x	dt_ts_x	id_y	ts_y	day_y	dt_ts_y	dt_ts_1
0	1	466	2017-01-07 18:24:55	2017-01-07	2017-01-07	NaN	NaT	NaN	NaT	NaT
1	2	458	2017-01-07 18:25:18	2017-01-07	2017-01-07	NaN	NaT	NaN	NaT	NaT
2	3	458	2017-01-07 18:26:21	2017-01-07	2017-01-07	NaN	NaT	NaN	NaT	NaT
3	4	592	2017-01-07 19:09:59	2017-01-07	2017-01-07	6.0	2017-01-08 09:14:27	2017-01-08	2017-01-08	2017-01-07
4	4	592	2017-01-07 19:09:59	2017-01-07	2017-01-07	7.0	2017-01-08 11:18:08	2017-01-08	2017-01-08	2017-01-07
5	5	393	2017-01-08 00:41:19	2017-01-08	2017-01-08	19.0	2017-01-09 16:22:48	2017-01-09	2017-01-09	2017-01-08
6	6	592	2017-01-08 09:14:27	2017-01-08	2017-01-08	13.0	2017-01-09 12:47:22	2017-01-09	2017-01-09	2017-01-08
7	6	592	2017-01-08 09:14:27	2017-01-08	2017-01-08	20.0	2017-01-09 16:31:07	2017-01-09	2017-01-09	2017-01-08
8	6	592	2017-01-08 09:14:27	2017-01-08	2017-01-08	21.0	2017-01-09 17:03:37	2017-01-09	2017-01-09	2017-01-08
9	6	592	2017-01-08 09:14:27	2017-01-08	2017-01-08	22.0	2017-01-09 20:06:54	2017-01-09	2017-01-09	2017-01-08

4.计算第一天活跃的用户数

```
init_user = merge_1.groupby('day_x').aggregate({'uid': lambda x: x.nunique()})
init_user.reset_index(inplace=True)
init_user.head()
```

```
init_user = merge_1.groupby('day_x').aggregate({'uid': lambda x: x.nunique()})
init_user.reset_index(inplace=True)
init_user.head()
```

	day_x	uid
0	2017-01-07	3
1	2017-01-08	3
2	2017-01-09	9
3	2017-01-10	9
4	2017-01-11	8

5.计算次日活跃的用户数

```
one_day_remain_user =
merge_1[merge_1['day_y'].notnull()].groupby('day_x').aggregate({'uid': lambda x:
x.nunique()})
one_day_remain_user.reset_index(inplace=True)
one_day_remain_user.head()
```

```
one_day_remain_user = merge_1[merge_1['day_y'].notnull()].groupby('day_x').aggregate({'uid': lambda x: x.nunique()})
one_day_remain_user.reset_index(inplace=True)
one_day_remain_user.head()
```

	day_x	uid
0	2017-01-07	1
1	2017-01-08	2
2	2017-01-09	5
3	2017-01-10	4
4	2017-01-11	6

6.合并前面两步的结果，计算最终留存

```
merge_one_day = pd.merge(init_user, one_day_remain_user, on=['day_x'])
merge_one_day['one_remain_rate'] = merge_one_day['uid_y'] /
merge_one_day['uid_x']
merge_one_day['one_remain_rate'] = merge_one_day['one_remain_rate'].apply(lambda
x: format(x, '.2%'))
merge_one_day.head(20)
```

	day_x	uid_x	uid_y	one_remain_rate
0	2017-01-07	3	1	33.33%
1	2017-01-08	3	2	66.67%
2	2017-01-09	9	5	55.56%
3	2017-01-10	9	4	44.44%
4	2017-01-11	8	6	75.00%
5	2017-01-12	10	5	50.00%
6	2017-01-13	8	2	25.00%
7	2017-01-14	4	2	50.00%
8	2017-01-15	4	3	75.00%
9	2017-01-16	11	5	45.45%
10	2017-01-17	7	5	71.43%
11	2017-01-18	11	3	27.27%
12	2017-01-19	7	4	57.14%
13	2017-01-20	7	3	42.86%
14	2017-01-21	5	1	20.00%
15	2017-01-22	2	1	50.00%
16	2017-01-23	7	2	28.57%
17	2017-01-24	6	2	33.33%
18	2017-01-26	4	2	50.00%
19	2017-01-27	5	4	80.00%

多日留存计算

方法一：

多日留存的计算可以沿用SQL中的思路，关联时先不用带日期条件

1.计算日期差，为后续做准备

```
merge_all = pd.merge(login_data, login_data, on=['uid'], how='left')
merge_all['diff'] = (merge_all['dt_ts_y'] - merge_all['dt_ts_x']).map(lambda x:
x.days)#使用map取的具体数字
merge_all.head()
```

```
merge_all = pd.merge(login_data, login_data, on=['uid'], how='left')
merge_all['diff'] = (merge_all['dt_ts_y'] - merge_all['dt_ts_x']).map(lambda x: x.days)#使用map取的具体数字
merge_all.head()
```

	id_x	uid	ts_x	day_x	dt_ts_x	id_y	ts_y	day_y	dt_ts_y	diff
0	1	466	2017-01-07 18:24:55	2017-01-07	2017-01-07	1	2017-01-07 18:24:55	2017-01-07	2017-01-07	0
1	1	466	2017-01-07 18:24:55	2017-01-07	2017-01-07	42	2017-01-10 19:12:47	2017-01-10	2017-01-10	3
2	1	466	2017-01-07 18:24:55	2017-01-07	2017-01-07	111	2017-01-16 15:21:20	2017-01-16	2017-01-16	9
3	1	466	2017-01-07 18:24:55	2017-01-07	2017-01-07	120	2017-01-17 14:06:57	2017-01-17	2017-01-17	10
4	1	466	2017-01-07 18:24:55	2017-01-07	2017-01-07	125	2017-01-17 17:47:23	2017-01-17	2017-01-17	10

2.计算第n天的留存人数，n=0,1,6,13。需要先进行筛选再进行计数，仍然使用nunique

```
diff_0 = merge_all[merge_all['diff'] == 0].groupby('day_x')['uid'].nunique()
diff_1 = merge_all[merge_all['diff'] == 1].groupby('day_x')['uid'].nunique()
diff_6 = merge_all[merge_all['diff'] == 6].groupby('day_x')['uid'].nunique()
diff_13 = merge_all[merge_all['diff'] == 13].groupby('day_x')['uid'].nunique()
diff_0 = diff_0.reset_index()#groupby计数后得到的是series格式，reset得到dataframe
diff_1 = diff_1.reset_index()
diff_6 = diff_6.reset_index()
diff_13 = diff_13.reset_index()
```

3.对多个dataframe进行一次合并

```
liucun = pd.merge(pd.merge(pd.merge(diff_0, diff_1, on=['day_x'], how='left'),
diff_6, on=['day_x'], how='left'), diff_13, on=['day_x'], how='left')
liucun.head()
```

```
liucun = pd.merge(pd.merge(pd.merge(diff_0, diff_1, on=['day_x']), diff_6, on=['day_x']), diff_13, on=['day_x'])
liucun.head()
```

	day_x	uid_x	uid_y	uid_x	uid_y
0	2017-01-07	3	1	2	2
1	2017-01-08	3	2	1	1
2	2017-01-10	9	4	5	4
3	2017-01-11	8	6	5	1
4	2017-01-12	10	5	7	3

4.对结果重命名，并用0填充na值

```
liucun.columns=['day', 'init', 'one_day_remain', 'seven_day_remain',
'fifteen_day_remain']#后来发现英文写错了，将就看，懒得改了
liucun.fillna(0, inplace=True)
liucun.head(20)
```

```
liucun.columns=['day', 'init', 'one_day_remain', 'seven_day_remain', 'fifteen_day_remain']
liucun.fillna(0, inplace=True)
liucun.head(20)
```

	day	init	one_day_remain	seven_day_remain	fifteen_day_remain
0	2017-01-07	3	1.0	2.0	2.0
1	2017-01-08	3	2.0	1.0	1.0
2	2017-01-09	9	5.0	3.0	0.0
3	2017-01-10	9	4.0	5.0	4.0
4	2017-01-11	8	6.0	5.0	1.0
5	2017-01-12	10	5.0	7.0	3.0
6	2017-01-13	8	2.0	4.0	1.0
7	2017-01-14	4	2.0	3.0	2.0
8	2017-01-15	4	3.0	2.0	1.0
9	2017-01-16	11	5.0	1.0	0.0
10	2017-01-17	7	5.0	5.0	2.0
11	2017-01-18	11	3.0	2.0	3.0
12	2017-01-19	7	4.0	2.0	2.0
13	2017-01-20	7	3.0	2.0	2.0
14	2017-01-21	5	1.0	2.0	0.0
15	2017-01-22	2	1.0	0.0	0.0
16	2017-01-23	7	2.0	0.0	1.0

得到的结果和SQL计算的一致。

方法二：

这种方法是从网上看到的，也放在这里供大家学习。它没有用自关联，而是对日期进行循环，计算当日的活跃用户数和n天后的活跃用户数。把n作为参数传入封装好的函数中。参考下面代码：

```
def cal_n_day_remain(df, n):
    dates = pd.Series(login_data.dt_ts.unique()).sort_values()[:-n]#取截止到n天的日期，保证有n日留存
    users = [] #定义列表存放初始用户数
    remains = []#定义列表存放留存用户数
    for d in dates:
        user = login_data[login_data['dt_ts'] == d]['uid'].unique()#当日活跃用户
        user_n_day = login_data[login_data['dt_ts']==d+timedelta(n)]['uid'].unique()#n日后活跃用户
        remain = [x for x in user_n_day if x in user]#取交集
        users.append(len(user))
        remains.append(len(remain))
    #一次循环计算一天的n日留存
    #循环结束后构造dataframe并返回
    remain_df = pd.DataFrame({'days': dates, 'user': users, 'remain': remains})
    return remain_df
```

代码的逻辑整体比较简单，必要的部分我做了注释。但需要一次一次调用，最后再merge起来。最后结果如下所示，从左到右依次是次日，7日，14日留存，和前面结果一样(可以再重命名一下)。

```
one_day_remain = cal_n_day_remain(login_data, 1)
seven_day_remain = cal_n_day_remain(login_data, 6)
fifteen_day_remain = cal_n_day_remain(login_data, 13)

liucun2 = pd.merge(pd.merge(one_day_remain, seven_day_remain[['days',
'remain']], on=['days'], how='left'), fifteen_day_remain[['days', 'remain']],
on=['days'], how='left')
liucun2.head(20)
```

:

	days	user	remain_x	remain_y	remain
0	2017-01-07	3	1	2.0	2.0
1	2017-01-08	3	2	1.0	1.0
2	2017-01-09	9	5	3.0	0.0
3	2017-01-10	9	4	5.0	4.0
4	2017-01-11	8	6	5.0	1.0
5	2017-01-12	10	5	7.0	3.0
6	2017-01-13	8	2	4.0	1.0
7	2017-01-14	4	2	3.0	2.0
8	2017-01-15	4	3	2.0	1.0
9	2017-01-16	11	5	1.0	0.0
10	2017-01-17	7	5	5.0	2.0
11	2017-01-18	11	3	2.0	3.0
12	2017-01-19	7	4	2.0	2.0
13	2017-01-20	7	3	2.0	2.0
14	2017-01-21	5	1	2.0	0.0
15	2017-01-22	2	1	0.0	0.0
16	2017-01-23	7	2	0.0	1.0
17	2017-01-24	6	2	1.0	NaN
18	2017-01-25	6	0	1.0	NaN
19	2017-01-26	4	2	0.0	NaN

至此，我们完成了SQL和pandas对日活和留存的计算。

小结

本篇文章我们研究了非常重要的两个概念，日活和留存。探讨了如何用SQL和pandas进行计算。日活计算比较简单。留存计算可以有多种思路。pandas计算这两个指标没有特别之处，但是用到了前面文章中的分组聚合，日期处理的部分。后台回复“**对比六**”可以获取本文pdf版本、数据和代码~希望对你有所帮助！

reference:

https://blog.csdn.net/msspark/article/details/86727058