SQL与Pandas

作为一名数据分析师，平常用的最多的工具是SQL(包括MySQL和Hive SQL等)。对于存储在数据库中的数据，自然用SQL提取会比较方便，但有时我们会处理一些文本数据(txt,csv)，这个时候就不太好用SQL了。Python也是分析师常用的工具之一，尤其pandas更是一个数据分析的利器。虽然二者的语法，原理可能有很大差别，但在实现的功能上，他们有很多相通的地方，这里特进行一个总结，方便大家对比学习~

本次学习的数据是虚构的订单数据，和实际业务无关，目的只是为了学习。大概长下面这样子，分别表示，自增id，订单时间，用户id，订单id，订单金额。

	id	ts	uid	orderid	amount
2	1	2019/8/1 9:15	10005	20190801091540	48.43
3	2	2019/8/1 10:00	10001	20190801100006	89.33
4	3	2019/8/1 10:04	10003	20190801091540	63.86
5	4	2019/8/1 12:17	10002	20190801121742	3.16
6	5	2019/8/1 14:05	10001	20190801140515	87.15
7	6	2019/8/1 14:05	10004	20190801140529	88.65
8	7	2019/8/2 8:13	10009	20190802081315	36.02
9	8	2019/8/2 11:14	10009	20190802111424	95.66
10	9	2019/8/2 13:18	10005	20190802131801	89.36
11	10	2019/8/2 15:18	10001	20190802151834	71.38
12	11	2019/8/2 16:00	10005	20190802160014	63.13
13	12	2019/8/2 17:03	10003	20190802170356	79.33
14	13	2019/8/2 17:11	10002	20190802171115	56.78
15	14	2019/8/2 19:05	10008	20190802190518	23.1
16	15	2019/8/2 20:07	10005	20190802200717	73.82
17	16	2019/8/2 20:08	10001	20190802200816	82.12
18	17	2019/8/2 20:10	10003	20190802201002	32.01
19	18	2019/8/3 9:02	10009	20190803090247	2.7
20	19	2019/8/3 10:08	10003	20190803100858	50.4
21	20	2019/8/3 12:08	10009	20190803120818	47.99

我们将用pandas和SQL的实现同样的目标，以此来联系二者，达到共同学习的目的。数据可以再公众号后台回复“数据”获取，你将得到本文所有的excel数据和SQL脚本数据，便于实操。

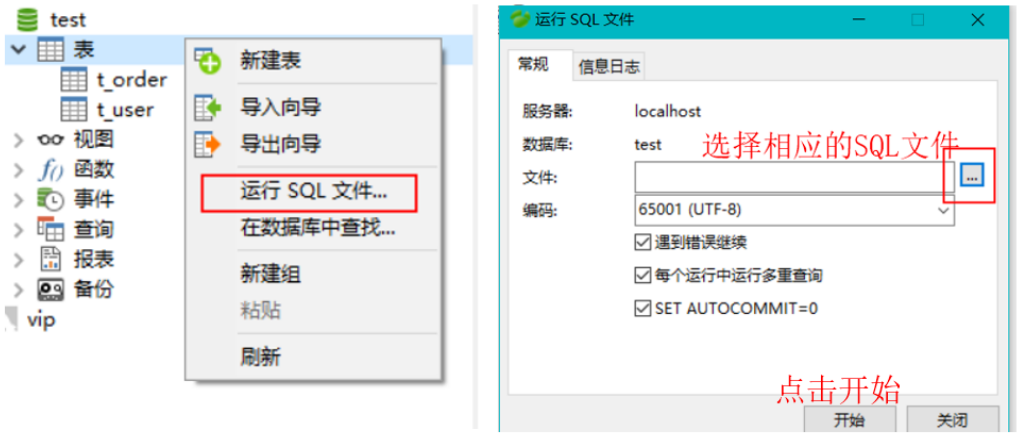
准备工作：

- pandas准备，我们本次采用jupyter notebook进行演示。

```
import pandas as pd
order_data = pd.read_csv('order.csv')
```

- SQL 准备

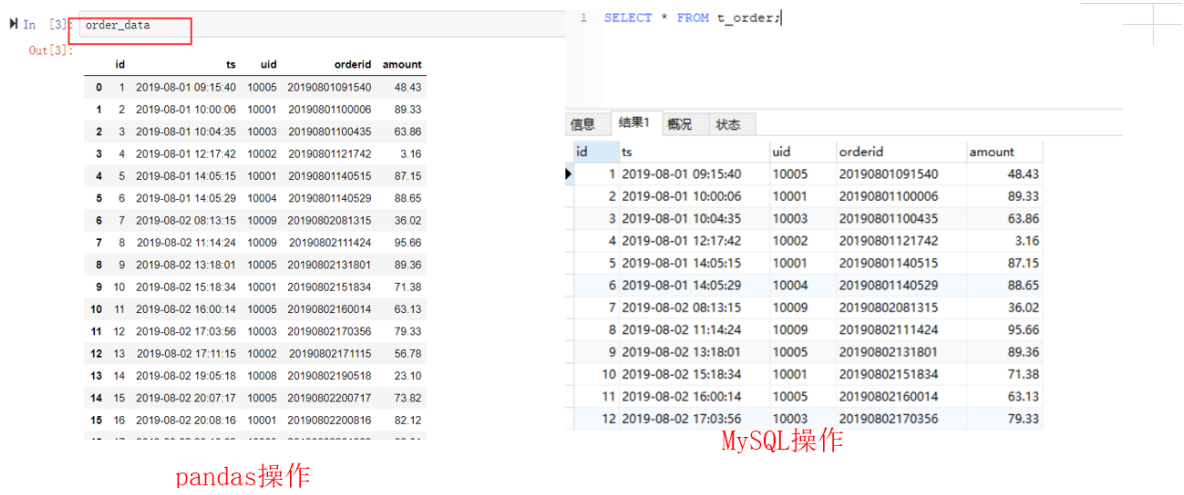
  只需将我提供的SQL文件运行一下即可将数据插入数据库表中。推荐使用navicate客户端连接数据库。



开始学习

1.查看全部数据或者前n行数据

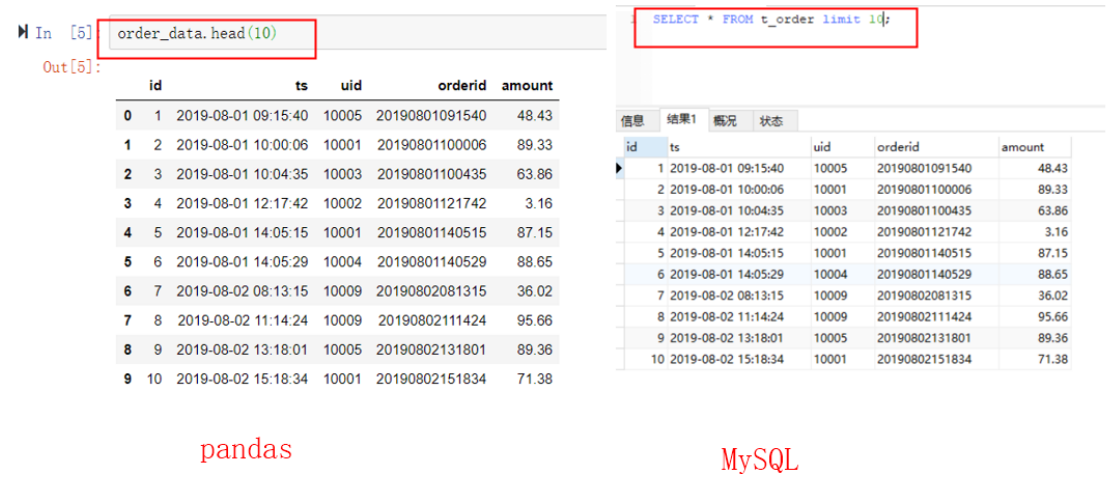
查看全部数据，pandas中直接打印dataframe对象即可，此处是order_data。而在SQL中，需要执行的语句是`select * from t_order;`表示从t_order表中查询全部的数据，*号表示查询所有的字段。结果如下：



	id	ts	uid	orderid	amount
0	1	2019-08-01 09:15:40	10005	20190801091540	48.43
1	2	2019-08-01 10:00:06	10001	20190801100006	89.33
2	3	2019-08-01 10:04:35	10003	20190801100435	63.86
3	4	2019-08-01 12:17:42	10002	20190801121742	3.16
4	5	2019-08-01 14:05:15	10001	20190801140515	87.15
5	6	2019-08-01 14:05:29	10004	20190801140529	88.65
6	7	2019-08-02 08:13:15	10009	20190802081315	36.02
7	8	2019-08-02 11:14:24	10009	20190802111424	95.66
8	9	2019-08-02 13:18:01	10005	20190802131801	89.36
9	10	2019-08-02 15:18:34	10001	20190802151834	71.38
10	11	2019-08-02 16:00:14	10005	20190802160014	63.13
11	12	2019-08-02 17:03:56	10003	20190802170356	79.33
12	13	2019-08-02 17:11:15	10002	20190802171115	56.78
13	14	2019-08-02 19:05:18	10008	20190802190518	23.10
14	15	2019-08-02 20:07:17	10005	20190802200717	73.82
15	16	2019-08-02 20:08:16	10001	20190802200816	82.12

id	ts	uid	orderid	amount
1	2019-08-01 09:15:40	10005	20190801091540	48.43
2	2019-08-01 10:00:06	10001	20190801100006	89.33
3	2019-08-01 10:04:35	10003	20190801100435	63.86
4	2019-08-01 12:17:42	10002	20190801121742	3.16
5	2019-08-01 14:05:15	10001	20190801140515	87.15
6	2019-08-01 14:05:29	10004	20190801140529	88.65
7	2019-08-02 08:13:15	10009	20190802081315	36.02
8	2019-08-02 11:14:24	10009	20190802111424	95.66
9	2019-08-02 13:18:01	10005	20190802131801	89.36
10	2019-08-02 15:18:34	10001	20190802151834	71.38
11	2019-08-02 16:00:14	10005	20190802160014	63.13
12	2019-08-02 17:03:56	10003	20190802170356	79.33

如果只想查看前10行数据呢。pandas可以调用head(n)方法，n是行数。MySQL可以使用limit n，n同样表示行数。

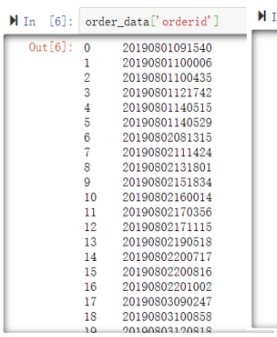


	id	ts	uid	orderid	amount
0	1	2019-08-01 09:15:40	10005	20190801091540	48.43
1	2	2019-08-01 10:00:06	10001	20190801100006	89.33
2	3	2019-08-01 10:04:35	10003	20190801100435	63.86
3	4	2019-08-01 12:17:42	10002	20190801121742	3.16
4	5	2019-08-01 14:05:15	10001	20190801140515	87.15
5	6	2019-08-01 14:05:29	10004	20190801140529	88.65
6	7	2019-08-02 08:13:15	10009	20190802081315	36.02
7	8	2019-08-02 11:14:24	10009	20190802111424	95.66
8	9	2019-08-02 13:18:01	10005	20190802131801	89.36
9	10	2019-08-02 15:18:34	10001	20190802151834	71.38

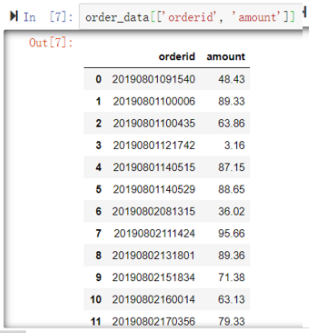
id	ts	uid	orderid	amount
1	2019-08-01 09:15:40	10005	20190801091540	48.43
2	2019-08-01 10:00:06	10001	20190801100006	89.33
3	2019-08-01 10:04:35	10003	20190801100435	63.86
4	2019-08-01 12:17:42	10002	20190801121742	3.16
5	2019-08-01 14:05:15	10001	20190801140515	87.15
6	2019-08-01 14:05:29	10004	20190801140529	88.65
7	2019-08-02 08:13:15	10009	20190802081315	36.02
8	2019-08-02 11:14:24	10009	20190802111424	95.66
9	2019-08-02 13:18:01	10005	20190802131801	89.36
10	2019-08-02 15:18:34	10001	20190802151834	71.38

2.查询特定列的数据

有的时候我们只想查看某几列的数据。在pandas里可以使用中括号或者loc，iloc等多种方式进行列选择，可以选择一列或多列。loc方式可以直接写列名，iloc方式需要指定索引，即第几列。SQl里只需写相应的列名即可，举例如下，实际操作一下更容易理解，选择一种自己习惯的即可。

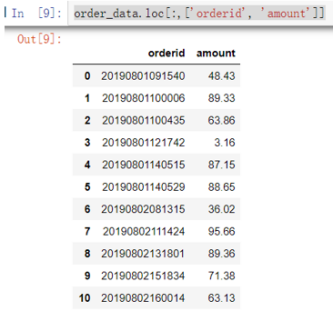


pandas中括号方式取1列

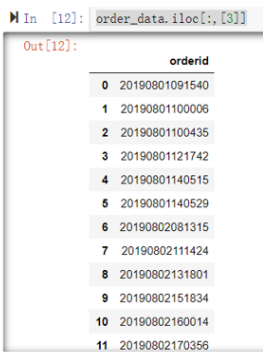


pandas中括号方式取多列

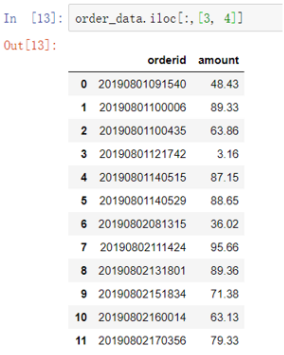
pandas loc方式取1列



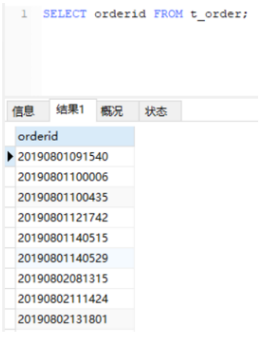
pandas loc方式取多列



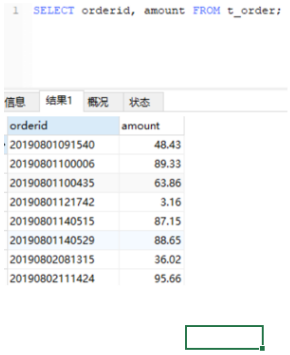
pandas iloc方式取1列



pandas iloc方式取多列



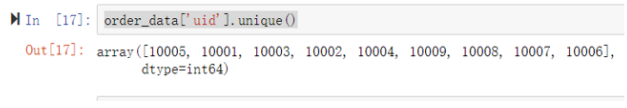
SQL 方式取1列



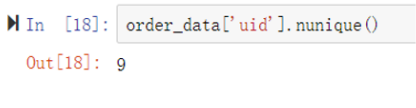
SQL 方式取多列

3.查询特定列去重后的数据

例如我们想查看一共有多少人(去重过的)下过单。pandas里有unique方法，SQL里有distinct关键字。如下面图左侧代码所示。两种方式输出的结果都含有9个uid，并且知道是哪9个。如果仅仅想知道有多少个uid，不关注具体值的话，可以参考右边的SQL，pandas用nunique()方法实现，而SQL里就需要用到一个count聚合函数与distinct组合的方式，表示去重并计数。

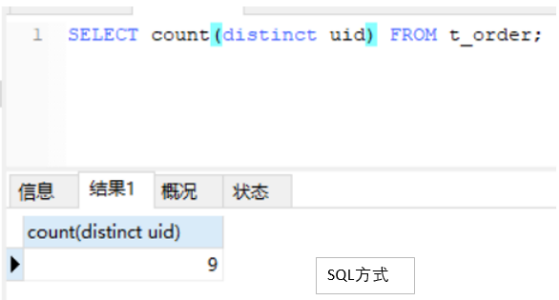


pandas 方式



pandas方式

SQL方式



SQL方式

4.查询带有1个条件的数据

例如我们要查询uid为10003的所有记录。pandas需要使用布尔索引的方式，而SQL中需要使用where关键字。指定条件时，可以指定等值条件，也可以使用不等值条件，如大于小于等。但一定要注意数据类型。例如如果uid是字符串类型，就需要将10003加引号，这里是整数类型所以不用加。代码如下：

```
]: order_data[order_data['uid']==10003]
:
```

	id	ts	uid	orderid	amount
2	3	2019-08-01 10:04:35	10003	20190801100435	63.86
11	12	2019-08-02 17:03:56	10003	20190802170356	79.33
16	17	2019-08-02 20:10:02	10003	20190802201002	32.01
18	19	2019-08-03 10:08:58	10003	20190803100858	50.40
31	32	2019-08-05 15:05:32	10003	20190805150532	42.29
34	35	2019-08-05 19:17:29	10003	20190805191729	49.75
40	41	2019-08-06 17:01:50	10003	20190806170150	37.49
50	51	2019-08-07 20:14:32	10003	20190807201432	33.16
51	52	2019-08-08 08:16:05	10003	20190808081605	81.74
64	65	2019-08-09 17:05:16	10003	20190809170516	33.72
66	67	2019-08-09 19:01:05	10003	20190809190105	90.97
77	78	2019-08-10 18:03:06	10003	20190810180306	22.59
78	79	2019-08-10 19:10:26	10003	20190810191026	68.86

pandas 操作

```
1 SELECT * FROM t_order
2 where uid = 10003;
```

信息 结果1 概况 状态

id	ts	uid	orderid	amount
3	2019-08-01 10:04:35	10003	20190801100435	63.86
12	2019-08-02 17:03:56	10003	20190802170356	79.33
17	2019-08-02 20:10:02	10003	20190802201002	32.01
19	2019-08-03 10:08:58	10003	20190803100858	50.4
32	2019-08-05 15:05:32	10003	20190805150532	42.29
35	2019-08-05 19:17:29	10003	20190805191729	49.75
41	2019-08-06 17:01:50	10003	20190806170150	37.49
51	2019-08-07 20:14:32	10003	20190807201432	33.16
52	2019-08-08 08:16:05	10003	20190808081605	81.74
65	2019-08-09 17:05:16	10003	20190809170516	33.72
67	2019-08-09 19:01:05	10003	20190809190105	90.97
78	2019-08-10 18:03:06	10003	20190810180306	22.59
79	2019-08-10 19:10:26	10003	20190810191026	68.86

MySQL 使用where关键字

5.查询带有多个条件的数据。

- 多个条件同时满足的情况

  在前一小结基础上，pandas需要使用&符号连接多个条件，每个条件需要加上小括号；SQL需要使用and关键字连接多个条件。例如我们查询uid为10003并且金额大于50的记录。两种方式的实现代码如下：

```
]: order_data[(order_data['uid']==10003) & (order_data['amount'] > 50)]
:
```

	id	ts	uid	orderid	amount
2	3	2019-08-01 10:04:35	10003	20190801100435	63.86
11	12	2019-08-02 17:03:56	10003	20190802170356	79.33
18	19	2019-08-03 10:08:58	10003	20190803100858	50.40
51	52	2019-08-08 08:16:05	10003	20190808081605	81.74
66	67	2019-08-09 19:01:05	10003	20190809190105	90.97
78	79	2019-08-10 19:10:26	10003	20190810191026	68.86

pandas 操作，使用&和()

```
1 SELECT * FROM t_order
2 where uid = 10003
3 and amount > 50;
```

信息 结果1 概况 状态

id	ts	uid	orderid	amount
3	2019-08-01 10:04:35	10003	20190801100435	63.86
12	2019-08-02 17:03:56	10003	20190802170356	79.33
19	2019-08-03 10:08:58	10003	20190803100858	50.4
52	2019-08-08 08:16:05	10003	20190808081605	81.74
67	2019-08-09 19:01:05	10003	20190809190105	90.97
79	2019-08-10 19:10:26	10003	20190810191026	68.86

MySQL 使用where和and关键字

- 多个条件满足其中一个的情况

  与多个条件同时满足使用&相对应的，我们使用|符号表示一个条件满足的情况，而SQL中则用or关键字连接各个条件表示任意满足一个。例如我们查询uid为10003或者金额大于50的记录。

```
: order_data[(order_data['uid']==10003) | (order_data['amount'] > 50)]
```

	id	ts	uid	orderid	amount
1	2	2019-08-01 10:00:06	10001	20190801100006	89.33
2	3	2019-08-01 10:04:35	10003	20190801100435	63.86
4	5	2019-08-01 14:05:15	10001	20190801140515	87.15
5	6	2019-08-01 14:05:29	10004	20190801140529	88.65
7	8	2019-08-02 11:14:24	10009	20190802111424	95.66
8	9	2019-08-02 13:18:01	10005	20190802131801	89.36
9	10	2019-08-02 15:18:34	10001	20190802151834	71.38
10	11	2019-08-02 16:00:14	10005	20190802160014	63.13
11	12	2019-08-02 17:03:56	10003	20190802170356	79.33

pandas 操作，使用|和()

```
1 SELECT * FROM t_order
2 where uid = 10003
3 or amount > 50;
```

信息 结果1 概况 状态

id	ts	uid	orderid	amount
2	2019-08-01 10:00:06	10001	20190801100006	89.33
3	2019-08-01 10:04:35	10003	20190801100435	63.86
5	2019-08-01 14:05:15	10001	20190801140515	87.15
6	2019-08-01 14:05:29	10004	20190801140529	88.65
8	2019-08-02 11:14:24	10009	20190802111424	95.66
9	2019-08-02 13:18:01	10005	20190802131801	89.36
10	2019-08-02 15:18:34	10001	20190802151834	71.38
11	2019-08-02 16:00:14	10005	20190802160014	63.13

MySQL 使用where和or关键字

这里需要特别说明的是有一种情况是需要判断某字段是否为空值。pandas的空值用nan表示，其判断条件需要携程isna()，或者notna()。例如

```
#查找uid不为空的记录
order_data[order_data['uid'].notna()]

#查找uid为空的记录
order_data[order_data['uid'].isna()]
```

MySQL相应的判断语句需要写成 is null 或者is not null。

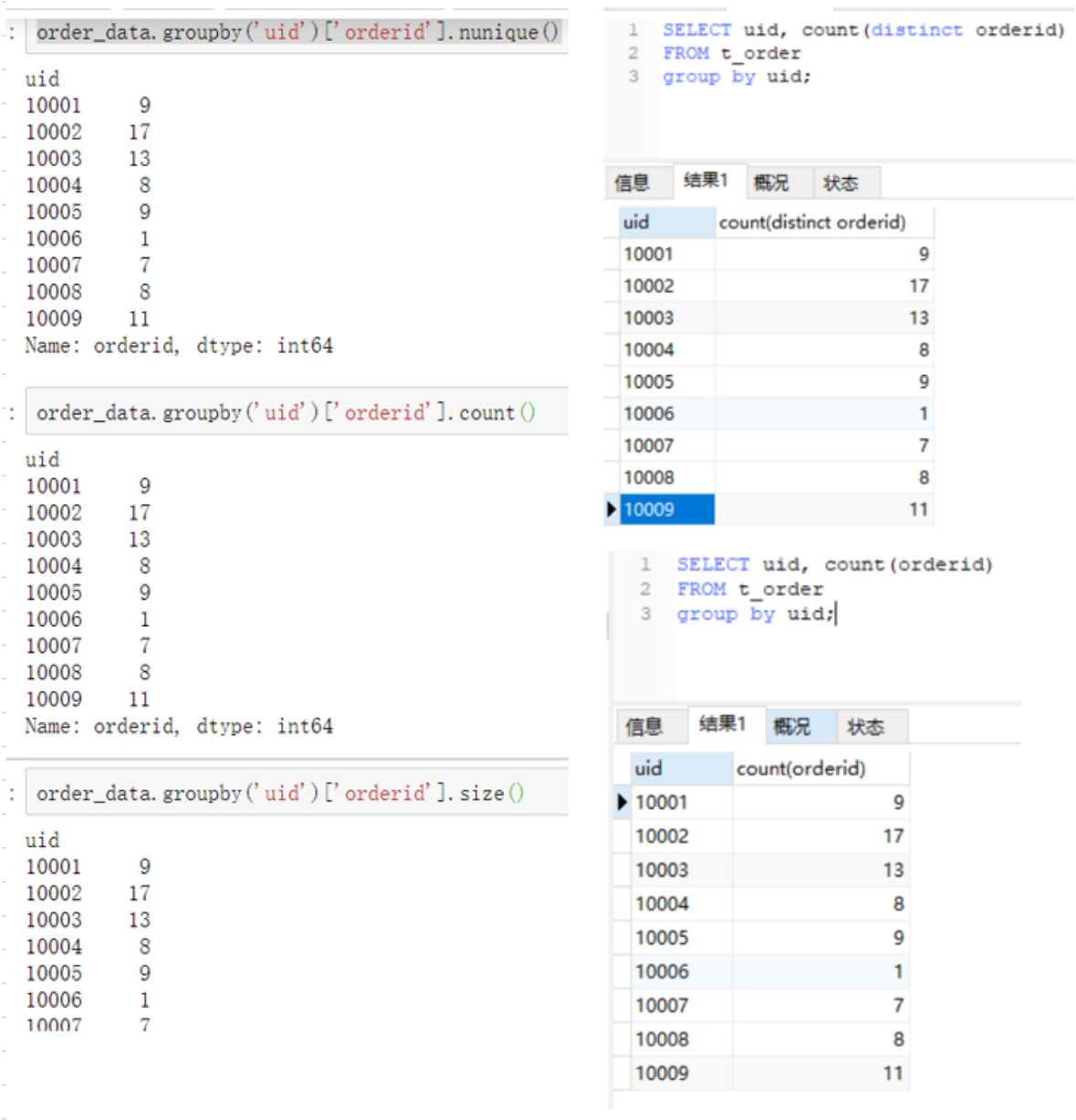
```
select * from t_order where uid is not null;

select * from t_order where uid is null;
```

还需要注意的是，空字符串或者空格虽然是有值的，但由于“不显示”出来，我们通常认为是空值。这种情况的判断条件和前面一样使用等号即可。感兴趣的朋友可以自己尝试一下。

6.group by聚合操作

使用group by时，通常伴随着聚合操作，这时候需要用到聚合函数。前面提到的count是一种聚合函数，表示计数，除此外还有sum表示求和，max,min表示最大最小值等。pandas和SQL都支持聚合操作。例如我们求每个uid有多少订单量。两种工具的操作如下：



pandas操作，count和size 不去重，nunique去重　SQL操作，加distinct去重

如果想要同时对不同的字段进行不同的聚合操作。例如目标变成：求每个uid的订单数量和订单总金额。写法会稍微不同一些，如下图所示。

```
order_data.groupby('uid').agg({'orderid': np.size, 'amount': np.sum})
```

uid	orderid	amount
10001	9	528.15
10002	17	775.60
10003	13	686.17
10004	8	591.39
10005	9	444.01
10006	1	65.44
10007	7	331.65
10008	8	425.38
10009	11	621.32

pandas操作

```
SELECT uid, count(distinct orderid), sum(amount)
FROM t_order
group by uid;
```

信息 | 结果1 | 概况 | 状态

uid	count(distinc	sum(amount)
10001	9	528.1500039100647
10002	17	775.5999970436096
10003	13	686.1700057983398
10004	8	591.3899993896484
10005	9	444.01000118255615
10006	1	65.44000244140625
10007	7	331.6499996185303
10008	8	425.38000297546387
10009	11	621.3200027942657

SQL操作

更进一步的，我们可以对结果的数据集进行重新命名。pandas可以使用rename方法，MySQL可以使用as 关键字进行结果的重命名。

```
order_df = order_data.groupby('uid').agg({'orderid': np.size, 'amount': np.sum})
order_df.rename(columns={'orderid':'order_cnt', 'amount':'sum_amount'})
```

uid	order_cnt	sum_amount
10001	9	528.15
10002	17	775.60
10003	13	686.17
10004	8	591.39
10005	9	444.01
10006	1	65.44
10007	7	331.65
10008	8	425.38
10009	11	621.32

pandas操作

```
SELECT uid,
count(distinct orderid) as order_cnt,
sum(amount) as sum_amount
FROM t_order
group by uid;
```

信息 | 结果1 | 概况 | 状态

uid	order_cnt	sum_amount
10001	9	528.1500039100647
10002	17	775.5999970436096
10003	13	686.1700057983398
10004	8	591.3899993896484
10005	9	444.01000118255615
10006	1	65.44000244140625
10007	7	331.6499996185303
10008	8	425.38000297546387
10009	11	621.3200027942657

SQL操作

7.join相关操作

join相关的操作有inner join，left join，right join，full join，等。pandas中统一通过pd.merge方法，设置不同的参数即可实现不同的dataframe的连接。而SQL里就可以直接使用相应的关键字进行两个表的连接。为了演示，我们此处引入一个新的数据集，user.csv(对应t_user表)。包含了用户的昵称，年龄信息。数据样例如下所示。

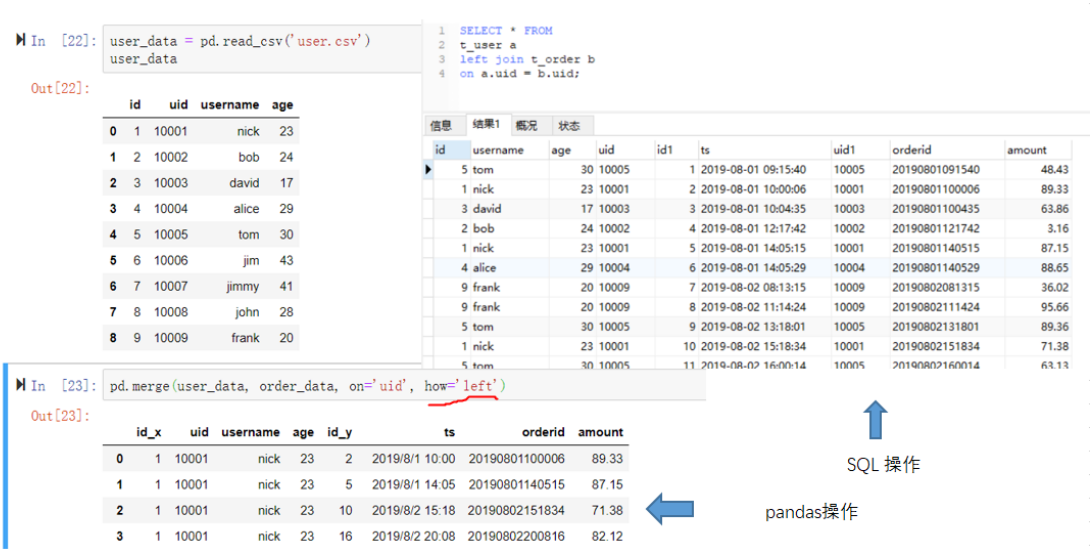
id	uid	username	age
1	10001	nick	23
2	10002	bob	24
3	10003	david	17
4	10004	alice	29
5	10005	tom	30
6	10006	jim	43
7	10007	jimmy	41
8	10008	john	28
9	10009	frank	20

- left join

  首先需要把数据加载进来：

  ```
  user_data = pd.read_csv('user.csv')
  ```

  pandas的merge函数传入4个参数，第一个是连接的主表，第二个是连接从表，第三个连接的key值，第四个是连接的方式，how为left时表示是左连接。SQL操作时基本也是同样的逻辑，要指定主表，从表，连接方式和连接字段。此处我们使用user连接order并查询所有字段和所有记录。具体代码如下所示，由于我们的数据没有空值，所以体现不出左连接的特点，感兴趣的读者可以自己尝试下。



- 其他连接方式

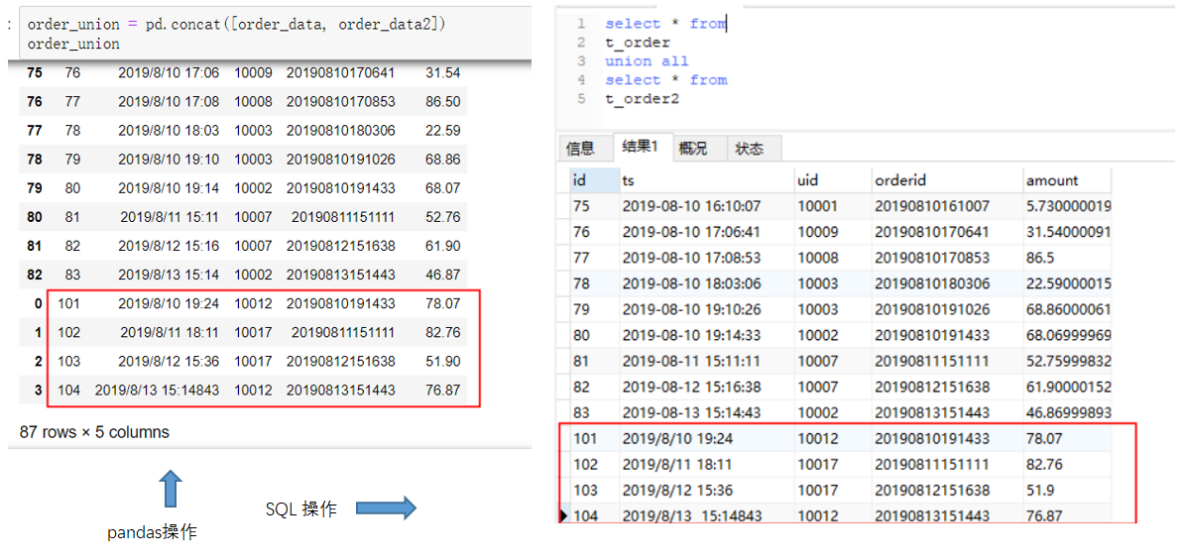
  如果要实现inner join，outer join，right join，pandas中相应的how参数为inner或者不填，outer，right。SQL也是同样直接使用对应的关键字即可。其中inner join 可以缩写为join。本例子中inner join 和left join的结果是一样的，在这里不作结果展示，pandas和SQL代码如下。

  ```
  pd.merge(user_data, order_data, on='uid', how='inner')
  ```

  ```
  SELECT * FROM
  t_user a
  inner join t_order b
  on a.uid = b.uid;
  ```

8.union操作

union相关操作分为union和union all两种。二者通常用于将两份含有同样字段的数据纵向拼接起来的场景。但前者会进行去重。例如，我现在有一份order2的订单数均，包含的字段和order数据一致，想把两者合并到一个dataframe中。SQL场景下页是期望将order2表和order表合并输出。执行的代码如下：



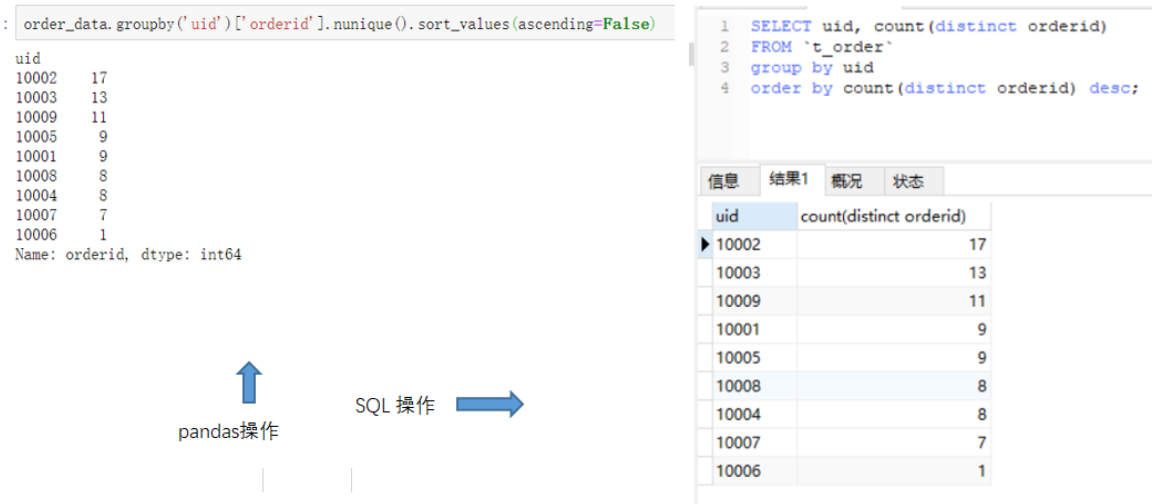
以上是没有去重的情况，如果想要去重，SQL需要用union关键字。而pandas则需要加上去重操作。

```
order_union = pd.concat([order_data, order_data2]).drop_duplicates()
```

```
select * from
t_order
union
select * from
t_order2
```

9.排序操作

我们在实际工作中经常需要按照某一列字段进行排序。pandas中的排序使用sort_values方法，SQl中的排序可以使用order_by关键字。我们用一个实例说明：按照每个uid的订单数从高到低排序。这是在前面聚合操作的基础上的进行的。相应的代码可以参考下方：



排序时，asc表示升序，desc表示降序，能看到两种方法都指定了排序方式，原因是默认是会按照升序佩列。在此基础上，可以做到对赌个字段的排序。pandas里，dataframe的多字段排序需要用by指定排序字段，SQL只要将多个字段依次卸载order by之后即可。例如，输出uid，订单数，订单金额三列，并按照订单金额升序，uid降序排列。

```
order_df = order_data.groupby('uid').agg({'orderid': np.size, 'amount': np.sum})
order_df.rename(columns={'orderid':'order_cnt', 'amount':'sum_amount'}, inplace=True)
order_df.sort_values(by=['uid', 'sum_amount'], ascending=[False, True])
```

uid	order_cnt	sum_amount
10009	11	621.32
10008	8	425.38
10007	7	331.65
10006	1	65.44
10005	9	444.01
10004	8	591.39
10003	13	686.17
10002	17	775.60
10001	9	528.15

pandas操作 SQL操作

```
SELECT uid, count(distinct orderid), sum(amount)
FROM `t_order`
group by uid
order by uid desc, sum(amount);
```

信息 结果1 概况 状态

uid	count(distinct orderid)	sum(amount)
10009	11	621.3200027942657
10008	8	425.38000297546387
10007	7	331.6499996185303
10006	1	65.44000244140625
10005	9	444.01000118255615
10004	8	591.3899993896484
10003	13	686.1700057983398
10002	17	775.5999970436096
10001	9	528.1500039100647

在pandas中可能有一些细节需要注意，比如我们将聚合结果先赋值，然后重命名，并指定了inplace=True替换原来的命名，最后才进行排序，这样写虽然有点绕，但整体思路比较清晰。

10.case when 操作

相比于其他操作，case when 操作可能不是那么“通用”。它更常见于SQL场景中，可能会用于分组，可能会用于赋值，也可能用于其他场景。分组，比如按照一定的分数区间分成优良中差。赋值，比如当数值小于0时，按照0计算。我们来举例看一下分组的场景。将每个uid按照总金额分为[0-300)，[300,600),[600,900)，三组。分别用pandas和SQL实现如下，注意这里我们的基础数据是上一步的order_df，SQL中也需要用子查询来实现。

```
def func(x):
    if x < 300:
        return '[0-300)'
    elif x < 600:
        return '[300, 600)'
    else:
        return '[600, 900]'
```

```
order_df['amt_interval'] = order_df['sum_amount'].map(func)
order_df
```

uid	order_cnt	sum_amount	amt_interval
10001	9	528.15	[300, 600)
10002	17	775.60	[600, 900]
10003	13	686.17	[600, 900]
10004	8	591.39	[300, 600)
10005	9	444.01	[300, 600)
10006	1	65.44	[0-300)
10007	7	331.65	[300, 600)
10008	8	425.38	[300, 600)
10009	11	621.32	[600, 900]

pandas操作

```
select uid, order_cnt,
case when sum_amount < 300 then '[0,300)'
     when sum_amount >=300 and sum_amount < 600 then '[300, 600)'
     when sum_amount >=600 and sum_amount < 900 then '[600, 900)'
else 'other' end as amt_interval
from
(
  SELECT uid, count(distinct orderid) order_cnt, sum(amount) as sum_amount
  FROM `t_order`
  group by uid
  order by uid desc, sum(amount)
) a
```

信息 结果1 概况 状态

uid	order_cnt	amt_interval
10009	11	[600, 900)
10008	8	[300, 600)
10007	7	[300, 600)
10006	1	[0,300)
10005	9	[300, 600)
10004	8	[300, 600)
10003	13	[600, 900)
10002	17	[600, 900)
10001	9	[300, 600)

SQL操作

熟悉pandas的朋友应该能想到，pandas的这种分组操作有一种专门的术语叫“分箱”，相应的函数为cut,qcut，能实现同样的效果。为了保持和SQL操作的一致性，此处采用了map函数的方式。您可以自己查阅资料了解另外的实现方式。

11.更新和删除操作

更新和删除都是要改变原有数据的操作。对于更新操作，操作的逻辑是：先选出需要更新的目标行，再进行更新。pandas中，可以使用前文提到的方式进行选择操作，之后可以直接对目标列进行赋值，SQL中需要使用update关键字进行表的更新。示例如下：将年龄小于20的用户年龄改为20。

```
user_data
```

	id	uid	username	age
0	1	10001	nick	23
1	2	10002	bob	24
2	3	10003	david	17
3	4	10004	alice	29
4	5	10005	tom	30
5	6	10006	jim	43
6	7	10007	jimmy	41
7	8	10008	john	28
8	9	10009	frank	20

```
user_data.loc[user_data['age']<20, 'age']=20
user_data
```

	id	uid	username	age
0	1	10001	nick	23
1	2	10002	bob	24
2	3	10003	david	20
3	4	10004	alice	29
4	5	10005	tom	30
5	6	10006	jim	43
6	7	10007	jimmy	41
7	8	10008	john	28
8	9	10009	frank	20

pandas操作

```
select * from t_user;
update t_user set age = 20 where age < 20;
select * from t_user;
```

信息 结果1 结果2 概况 状态

id	username	age	uid
1	nick	23	10001
2	bob	24	10002
3	david	17	10003
4	alice	29	10004
5	tom	30	10005
6	jim	43	10006
7	jimmy	41	10007
8	john	28	10008
9	frank	20	10009

```
select * from t_user;
update t_user set age = 20 where age < 20;
select * from t_user;
```

信息 结果1 结果2 概况 状态

id	username	age	uid
1	nick	23	10001
2	bob	24	10002
3	david	20	10003
4	alice	29	10004
5	tom	30	10005
6	jim	43	10006
7	jimmy	41	10007
8	john	28	10008
9	frank	20	10009

SQL操作

删除操作可以细分为删除行的操作和删除列的操作。对于删除行操作，pandas的删除行可以转换为选择不符合条件进行操作。SQL需要使用delete关键字。例如删除年龄为30岁的用户：

```
user_data[user_data['age']!=30]
```

	id	uid	username	age
0	1	10001	nick	23
1	2	10002	bob	24
2	3	10003	david	20
3	4	10004	alice	29
5	6	10006	jim	43
6	7	10007	jimmy	41
7	8	10008	john	28
8	9	10009	frank	20

pandas操作

```
delete from t_user where age=30;
select * from t_user;
```

信息 结果1 概况 状态

id	username	age	uid
1	nick	23	10001
2	bob	24	10002
3	david	20	10003
4	alice	29	10004
6	jim	43	10006
7	jimmy	41	10007
8	john	28	10008
9	frank	20	10009

SQL操作

对于删除列的操作。pandas需要使用drop方法。SQL也需要使用drop关键字。

```
user_data.drop(['uid'], inplace=True, axis=1)
user_data
```

	id	username	age
0	1	nick	23
1	2	bob	24
2	3	david	20
3	4	alice	29
4	5	tom	30
5	6	jim	43
6	7	jimmy	41
7	8	john	28
8	9	frank	20

pandas操作

```
13  alter table t_user drop column uid;
14  select * from t_user;
```

信息 结果1 概况 状态

id	username	age
1	nick	23
2	bob	24
3	david	20
4	alice	29
6	jim	43
7	jimmy	41
8	john	28
9	frank	20

SQL操作

总结:

简单粗暴，如下图所示:

序号	操作	pandas	SQL
1	查看全部数据	order_data	select *
2	查看前n行数据	head(n)	limit n
3	查询特定列	[],loc,iloc	select 列名
4	去重	nunique	distinct
5	条件	指定条件	where
6	多条件同时满足	&	and
7	多条件满足其一	|	or
8	聚合操作	groupby	group by
9	重命名	rename	as
10	连接	merge	join
11	左连接	merge,how=left	left join
12	全连接	merge,how=full	full join
13	合并	concat	union all
14	合并去重	concat drop_dupicate	union
15	排序	sort_values	order by
16	条件操作分组	map函数或cut	case when
17	更新	选择-赋值	update
18	删除行	选择相反条件	delete
19	删除列	drop,axis=1	drop column

需要说明的是，pandas和SQL是两种不同的工具，本文进行比较并不像说明孰优孰劣，只是为了对于二者的类似操作加深理解，从而方便实际工作中更高效的使用二者。实际工作中的操作可能比本文涉及到的复杂很多，甚至会有多种组合的方式出现，也可能会有本文没有提及的情况。但我们掌握了本文的方法，就可以以不变应万变，遇到复杂情况也可从容应对了，希望对你有所帮助！后台回复“对比”可以获得本次联系的数据样例，您可自行进行练习。