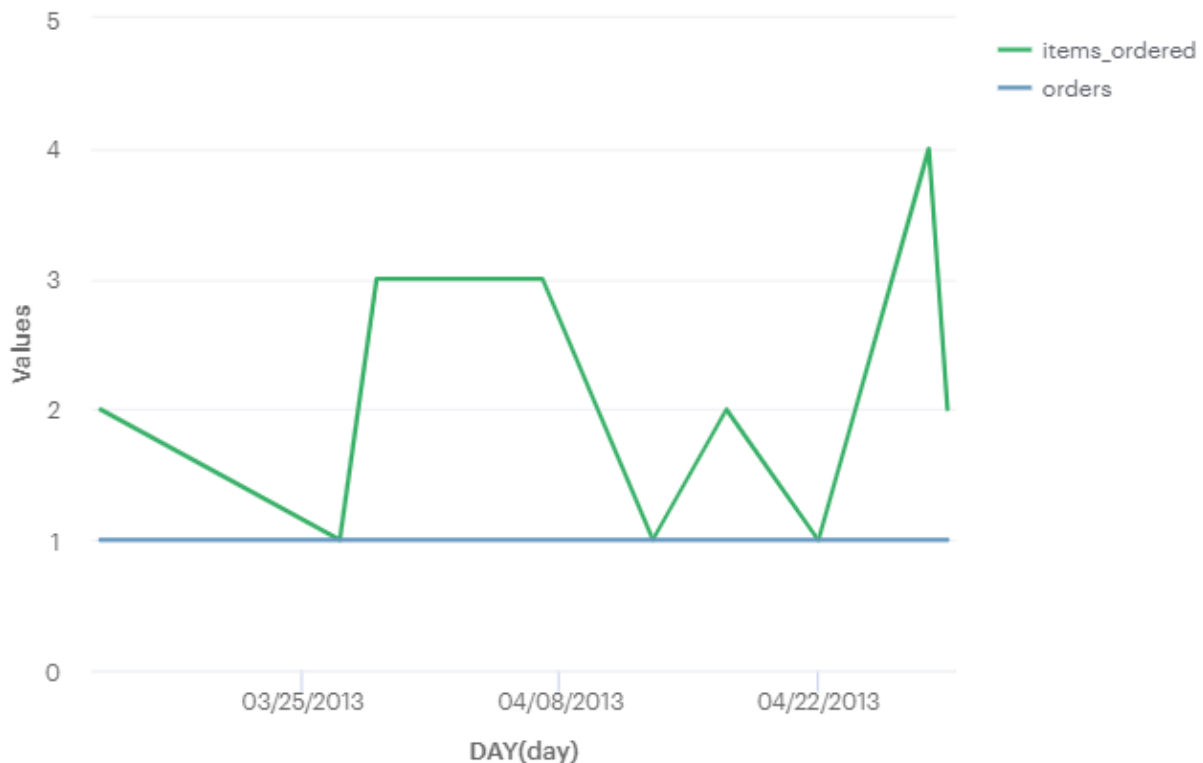


Create a Roll-up Table

Exercise 1: Create a sub-table of orders per day. Make sure you decide whether you are counting invoices or line items.

```
1. select
2.   date(orders.paid_at)      as day,
3.   count(distinct invoice_id) as orders,
4.   count(distinct line_item_id) as items_ordered
5. from
6.   dsv1069.orders
7. group by
8.   date(orders.paid_at)
```

Orders per Day

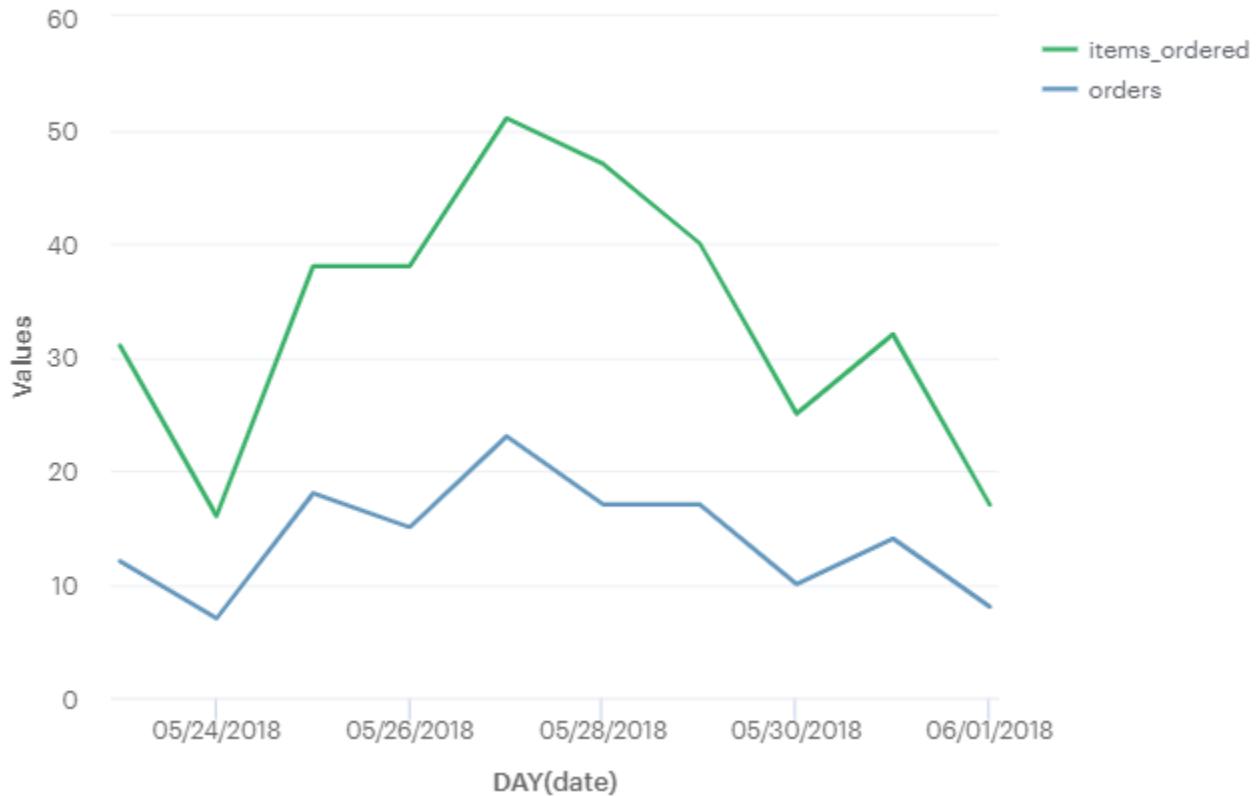


Exercise 2: “Check your joins”. We are still trying to count orders per day. In this step join the sub table from the previous exercise to the dates rollup table so we can get a row for every date. Check that the join works by just running a “select *” query.

```
1. select
2.   *
3. from
4.   dsv1069.dates_rollup
5. left outer join
6.   (
7.     select
8.       date(orders.paid_at)      as day,
9.       count(distinct invoice_id) as orders,
10.      count(distinct line_item_id) as items_ordered
11.    from
12.      dsv1069.orders
13.   group by
14.     date(orders.paid_at)
15.   ) daily_orders
16. on
```

17. `dates_rollup.date = daily_orders.day`

Orders per Day Clean



Exercise 3: “Clean up your Columns” In this step be sure to specify the columns you actually want to return, and if necessary do any aggregation needed to get a count of the orders made per day.

```
1. select
2.     dates_rollup.date,
3.     sum(orders)           as order_count,
4.     sum(items_ordered)   as items_ordered_count,
5.     count(day)           as rows_collapsed
6. from
7.     dsv1069.dates_rollup
8. left outer join
9.     (
10.    select
11.        date(orders.paid_at)           as day,
12.        count(distinct invoice_id)    as orders,
13.        count(distinct line_item_id)  as items_ordered
14.    from
15.        dsv1069.orders
16.    group by
17.        date(orders.paid_at)
18.    ) daily_orders
19. on
20.     dates_rollup.date = daily_orders.day
21. group by
22.     dates_rollup.date
```

	date	order_count	items_ordered_count	rows_collapsed
1	2013-01-01 00:00:00			0
2	2013-01-02 00:00:00			0
3	2013-01-03 00:00:00			0
4	2013-01-04 00:00:00			0
5	2013-01-05 00:00:00			0
6	2013-01-06 00:00:00			0
7	2013-01-07 00:00:00			0
8	2013-01-08 00:00:00			0
9	2013-01-09 00:00:00			0
10	2013-01-10 00:00:00			0

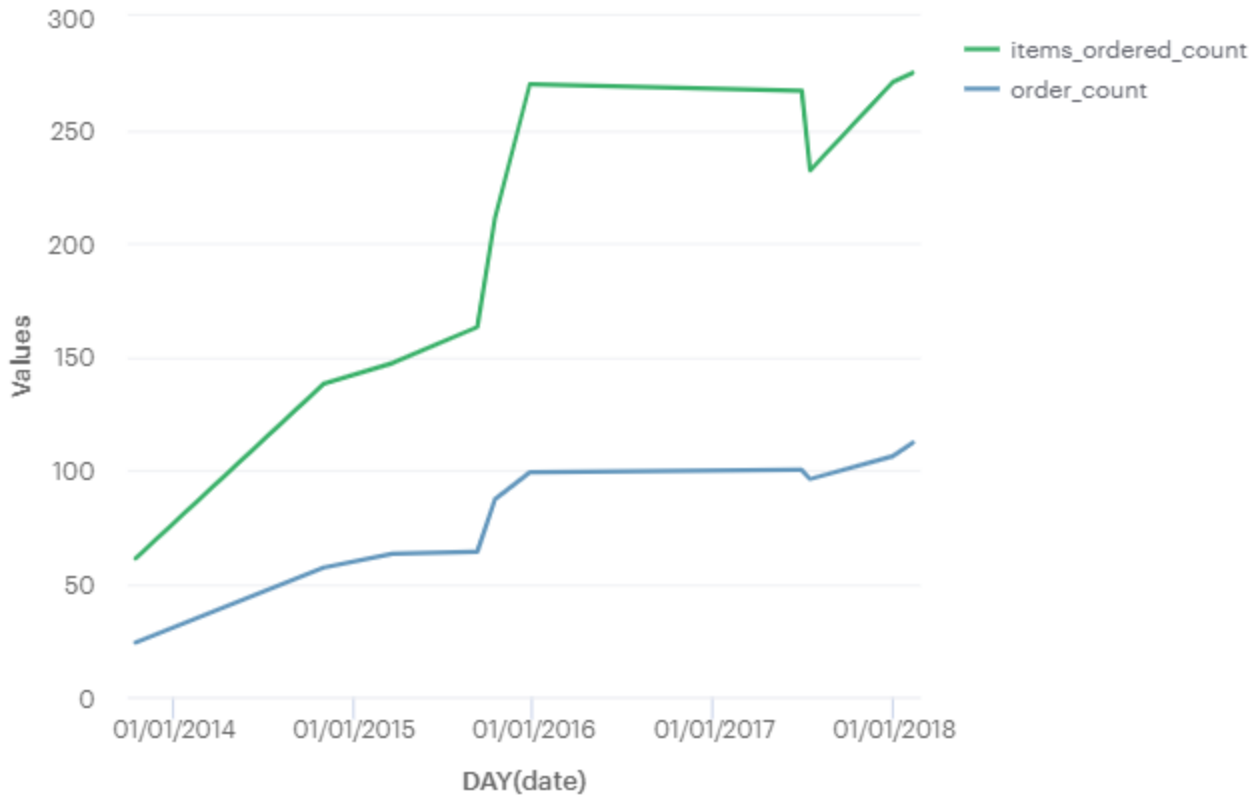
Exercise 4: Weekly Rollup. Figure out which parts of the JOIN condition need to be edited create 7 day rolling orders table.

```

1. select *
2. from
3.   dsv1069.dates_rollup
4. left outer join
5.   (
6.     select
7.       date(orders.paid_at)      as day,
8.       count(distinct invoice_id) as orders,
9.       count(distinct line_item_id) as items_ordered
10.    from
11.      dsv1069.orders
12.    group by
13.      date(orders.paid_at)
14.    ) daily_orders
15. on
16.   dates_rollup.date >= daily_orders.day
17. and
18.   dates_rollup.d7_ago < daily_orders.day

```

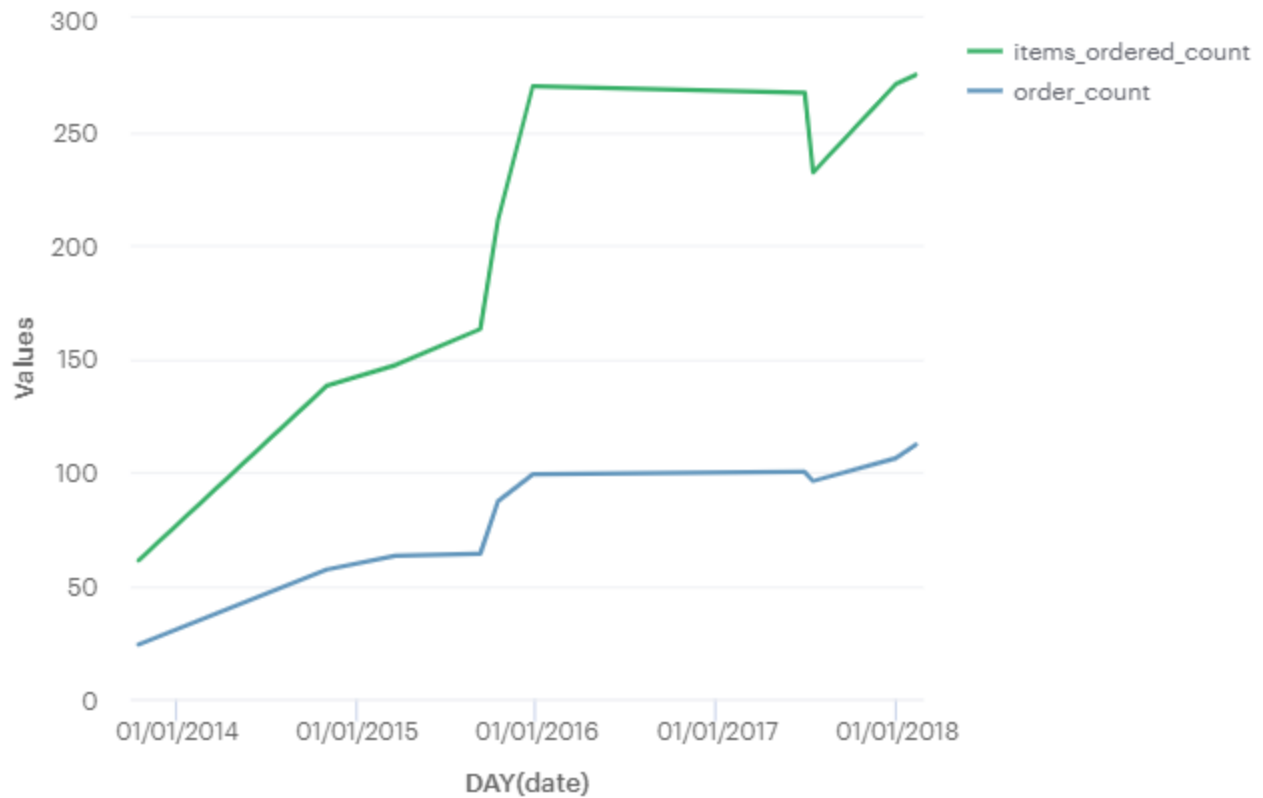
Weekly Roll-up



Exercise 5: Column Cleanup. Finish creating the weekly rolling orders table, by performing any aggregation steps and naming your columns appropriately.

```
1. select
2.   dates_rollup.date,
3.   sum(orders)      as order_count,
4.   sum(items_ordered) as items_ordered_count,
5.   count(day)      as rows_collapsed
6. from
7.   dsv1069.dates_rollup
8. left outer join
9.   (
10.    select
11.      date(orders.paid_at)      as day,
12.      count(distinct invoice_id) as orders,
13.      count(distinct line_item_id) as items_ordered
14.    from
15.      dsv1069.orders
16.    group by
17.      date(orders.paid_at)
18.    ) daily_orders
19. on
20.   dates_rollup.date >= daily_orders.day
21. and
22.   dates_rollup.d7_ago < daily_orders.day
23. group by
24.   dates_rollup.date
```

Weekly Roll-up Clean



Check for anomalies:

```
1. select *
2. from dsv1069.events
3. where event_time > '2018-06-02'
```



Looks like your query didn't return any results

Try broadening your query.

Mode Report Link: <https://app.mode.com/sum14/reports/2ba17c913105>