Chapter 12

Extensions [TBD]

Contents

- When we model the LTV of a customer, we usually only consider factors that we can attribute to them at the start of their first session and not factors that occur sometime after their initial start. Why do we do this? Because the numbers that are generated not at the start of a session are hard to compare against. For example, if you take the "LTV of customers who have spent more than than 20 minutes on our website" it is difficult to understand what to compare this to or even what it means.
- The average revenue per customer number that we came up with before is generally not considered to be the LTV. The LTV is usually a cohort measure – it has a time component to it. In the above queries we are comparing users who have spent a short amount of time in the system to those who have spent a lot of time in the system.
- Because LTV is a cohort number, we need to measure the LTV based on the number of users within a time period, such as week or month. For example, lets consider the case where we want to compute LTV based on month of first purchase.

```
select
        , lhs.cohort
        , sum( amt ) as totaldol
        , count(distinct lhs.userid) as numusers
        , sum( amt )/count(distinct lhs.userid) as numusers
from
        (select
                userid, date_trunc( 'month', min(dt ))::date as cohort
        from
                trans
        group by 1) as lhs
left join
        trans
using(userid)
group by 1
```

- The query above only returns the total revenue by cohort. We may want to return the running total, by month. In other words, we want to make a table where the Y-axis is the cohort and the X-axis is the number of months, starting at zero, and the amount of money that was generated by that cohort that many months afterward. Let's consider the first three months of LTV.

  Before starting this, make a chart of what the data should look like:

cohortdt	numusers	mon_0_amt	mon_1_amt	mon_2_amt
01-01-2011	1,155	25,764	12,885	9,995
02-01-2011	355	7,555	3,456	1,111
03-01-2011	755	2,888	7,925	1,100

```
select
        lhs.cohort
        , count(distinct lhs.user_id) as numusers
        , sum(case when trans.dt::date
                between cohort and (cohort + '1 month'::interval)::date
                then amt else 0 end ) as mon_0_amt
        , sum(case when trans.dt::date
                between (cohort + '1 month'::interval)::date
                        and (cohort + '2 month'::interval)::date
                then amt else 0 end ) as mon_1_amt
        , sum(case when trans.dt::date
                between (cohort + '2 month'::interval)::date
                        and (cohort + '3 month'::interval)::date
                then amt else 0 end ) as mon_2_amt
from
        (select
                userid, date_trunc( 'month', min(dt ))::date as cohort
        from
                trans
        group by 1) as lhs
left join
        trans
using(userid)
group by 1
```

- Calculate the average revenue per user in the cohort for the first three months of LTV.

```
select
        lhs.cohort
        , sum(case when trans.dt::date
                between cohort and (cohort + '1 month'::interval)::date
                then amt else 0 end ) / count(distinct lhs.user_id) as mon_0_PU
        , sum(case when trans.dt::date
                between (cohort + '1 month'::interval)::date
                        and (cohort + '2 month'::interval)::date
                then amt else 0 end ) / count(distinct lhs.user_id) as mon_1_PU
        , sum(case when trans.dt::date
                between (cohort + '2 month'::interval)::date
                        and (cohort + '3 month'::interval)::date
                then amt else 0 end ) / count(distinct lhs.user_id) as mon_2_PU
from
        (select
                userid, date_trunc( 'month', min(dt ))::date as cohort
        from
                trans
        group by 1) as lhs
left join
        trans
using(userid)
group by 1
```

- Calculate the average revenue per *active-user* from that cohort for the first two months of LTV:

```
select
        lhs.cohort
        , sum(case when trans.dt::date
                between cohort and (cohort + '1 month'::interval)::date
                then amt else 0 end )
        / count( distinct case when trans.dt::date
                between cohort and (cohort + '1 month'::interval)::date
                then lhs.userid else null end ) as mon_0_PU
        , sum(case when trans.dt::date
                between (cohort + '1 month'::interval)::date
                        and (cohort + '2 month'::interval)::date
                then amt else 0 end )
        / count( distinct case when trans.dt::date
                between (cohort + '1 month'::interval)::date
                        and (cohort + '2 month'::interval)::date
                then lhs.userid else null end ) as mon_1_PU
        , sum(case when trans.dt::date
                between (cohort + '2 month'::interval)::date
                        and (cohort + '3 month'::interval)::date
                then amt else 0 end )
        / count( distinct case when trans.dt::date
                between (cohort + '2 month'::interval)::date
                        and (cohort + '3 month'::interval)::date
                then lhs.userid else null end ) as mon_2_PU
from
        (select
                userid, date_trunc( 'month', min(dt ))::date as cohort
        from
                trans
        group by 1) as lhs
left join
        trans
using(userid)
group by 1
```

- Calculate the LTV for the first three months, per-cohort, for both first-purchase subscribers and first-purchase non-subscribers.

```
select
        date_part('month', firstdt) as cohort
        , subscriber_flag
        , sum(case when trans.dt::date
                between cohort and (cohort + '1 month'::interval)::date
                then amt else 0 end ) / count(distinct lhs.user_id) as mon_0_PU
        , sum(case when trans.dt::date
                between (cohort + '1 month'::interval)::date
                        and (cohort + '2 month'::interval)::date
                then amt else 0 end ) / count(distinct lhs.user_id) as mon_1_PU
        , sum(case when trans.dt::date
                between (cohort + '2 month'::interval)::date
                        and (cohort + '3 month'::interval)::date
                then amt else 0 end ) / count(distinct lhs.user_id) as mon_2_PU
FROM
        (select
                lhs.userid
                , max( case when trans.transtype = 'S' then 1
                        else 0 end ) as subscriber_flag
                , max(firstdt) as firstdt
        from
                (select
                        userid, min(dt) as firstdt
                from
                        trans
                group by 1) as lhs
        left join
                trans
        on
                lhs.userid = trans.userid
                and lhs.firstdt = trans.dt
        group by 1) as lhs

LEFT JOIN
        trans
using( userid)
group by 1,2
order by 2,1;
```

1 More Advanced Joins

In this section we are going to cover a number of common advanced joins.

- In this section we are going to cover some advanced join syntax.

```
        pct          | hr | plaza
---------------------+----+-------
  0.0137942721572615 |  0 |     1
 0.00820199966107439 |  1 |     1
 0.00871038806981867 |  2 |     1
  0.0100321979325538 |  3 |     1
  0.0189798339264531 |  4 |     1
   0.050160989662769 |  5 |     1
  0.0880528723945094 |  6 |     1
  0.0690730384680563 |  7 |     1
  0.0707676664972039 |  8 |     1
  0.0570411794611083 |  9 |     1
  0.0679545839688188 | 10 |     1
  0.0647347907134384 | 11 |     1
  0.0596509066259956 | 12 |     1
  0.0620233858668022 | 13 |     1
  0.0705643111337061 | 14 |     1
  0.0771733604473818 | 15 |     1
  0.0696153194373835 | 16 |     1
  0.0667005592272496 | 17 |     1
  0.0617183528215557 | 18 |     1
  0.0575834604304355 | 19 |     1
  0.0412133536688697 | 20 |     1
  0.0297576681918319 | 21 |     1
  0.0268090154211151 | 22 |     1
   0.023351974241654 | 23 |     1
(24 rows)
```

We join on plaza and then order by hr and this returns the percentage for each hour of the total.

- If we want to do same thing for multiple plazas and multiple days, we can modify the query by removing things from the WHERE clause and adding things to the JOIN:

```
select
        vehiclesez::float / totalez as pct
        , hr
        , lhs.plaza
from
        (select
                vehiclesez
                , mtadt, plaza, hr
        from
                cls.mta
        where
                direction = 'O') as lhs
LEFT JOIN
        (select
                sum( vehiclesez ) as totalez
                , mtadt, plaza
        from
                cls.mta
        where
                direction = 'O'
        group by 2,3) as rhs
using( plaza, mtadt)
order by hr asc;
```

Which should work, but it doesn't because of a division by zero error. How do we handle this? In this case we are going to treat them as NULL:

```
select
        case
                when totalez = 0 then null
                else vehiclesez::float / totalez
                end as pct
        , hr
        , lhs.plaza
from
        (select
                vehiclesez
                , mtadt, plaza, hr
        from
                cls.mta
        where
                direction = 'O') as lhs
LEFT JOIN
        (select
                sum( vehiclesez ) as totalez
                , mtadt, plaza
        from
                cls.mta
        where
                direction = 'O'
        group by 2,3) as rhs
using( plaza, mtadt)
order by plaza, hr;
```

- Let's create a running average for the last two hours (3 data points) of inbound traffic using the EZ-pass for each plaza.

```
select
        lhs.hr, lhs.mtadt, lhs.plaza, lhs.vehiclesez
        , sum(rhs.vehiclesez)::float/count(rhs.vehiclesez) as running
from
        (select
                vehiclesez, hr, mtadt, plaza
        from cls.mta
                where direction = 'I' ) as lhs
left join
        (select
                vehiclesez, hr, mtadt, plaza
        from cls.mta
                where direction = 'I' ) as rhs
on
        lhs.plaza = rhs.plaza
        and (
                (lhs.hr >= 2
                        and lhs.mtadt = rhs.mtadt
                        and rhs.hr >= lhs.hr - 2
                        and rhs.hr <= lhs.hr)
                OR
                (lhs.hr = 1
                        and (
                                (lhs.mtadt = rhs.mtadt and rhs.hr <= 1)
                                or (lhs.mtadt -1 = rhs.mtadt and rhs.hr = 23) )
                )
                OR
                (lhs.hr = 0
                        and (
                                ( lhs.mtadt -1 = rhs.mtadt and rhs.hr in (22, 23))
                                or (lhs.mtadt = rhs.mtadt and rhs.hr=0)
                                )
                )
                )
group by 1,2,3,4;
```

- Keep in mind that the hour goes from 0-23, there is no hour 24 in the dataset. The above query works by matching all rows that fulfill one of three criteria.

  1. If the hour ≥ 2 then just match based on the same date and the right hand side hour is in the previous two hours.
  2. If the hour is equal to 1 then match either the zero or 1 hour with the same date or the 23 hour from yesterday.
  3. If the hour is equal to 0 then match either the hour zero on the same day or hour 22 or 23 from yesterday.

  Once the match is completed the dataset will be three times as large as expected since each row in the left hand table matches *three* in the right hand table. In order to create the average we then use the GROUP BY to collapse those three rows into one, as defined by the left hand table.

- We can also rewrite the query to leverage the contents – and in particular the time aspect of the information. But to do so we will need to convert our hour and time information into a timestamp. There are a number of ways to construct a timestamp, what we will do is combine the date and time using string concatenation and then convert it via the double-colon operator. The following query successfully demonstrates how this would occur:

```
select ('2015-10-04' || ' ' || 2::varchar || ':00')::timestamp ;
      timestamp
---------------------
 2015-10-04 02:00:00
(1 row)
```

- To do the time comparison we will use an INTERVAL operator to assist in the time comparison, as can be seen in the final query below.

```
select
        lhs.mtatime, lhs.plaza, lhs.vehiclesez
        , sum(rhs.vehiclesez)::float/count(rhs.vehiclesez) as running
from
    ( select
            (mtadt || ' ' || hr::varchar || ':00')::timestamp as mtatime
            , plaza, vehiclesez
    from
            cls.mta where direction = 'I' and plaza = 1 ) as lhs
left join
    ( select
            (mtadt || ' ' || hr::varchar || ':00')::timestamp as mtatime
        , plaza, vehiclesez
    from cls.mta where direction = 'I' and plaza = 1 ) as rhs
on
        lhs.plaza = rhs.plaza
        and rhs.mtatime >= lhs.mtatime - interval '2 hour'
        and rhs.mtatime <= lhs.mtatime
group by 1,2,3
order by 1;
```

As in the previous version of this query, we first expand the dataset threefold using the time matching. The data is then collapsed back down into its original form.

- One number we want to calculate is the average revenue per user (ARPU) for a -AAS company. In these situations, customer transaction data is provided in a "long" format with each row representing a single sale.

- Calculating the ARPU is often the first step in calculating a user's lifetime value (or long-term value) ("LTV"). For companies providing a service over time, the calculation of LTV is critical for ensuring profitability. Studying the per-unit economics of a product are often the domain of the data science team and the exercise undertaken below is a common exercise.

- Our major goal for this module is to create a cohort based estimate of how much revenue is generated by an average customer over time. In particular, companies often focus on understanding things like the 90-day ARPU or 120-day ARPU, which represents how much money an average customer generates over the first 90 and 120 days of a customer's lifetime.

- We call this a "cohort" based estimate because we group users based we create this estimate based on when a user first enters the system or alternatively, first becomes a customer. This is usually done on a monthly or weekly basis. So we often consider all new customers who

2 OLAP: Cube and Rollup

TBD

3 Schemas

TBD

Star and Snowflakes

Facts and Dimensions

4 Keys

TBD

5 Data Exploration Strategies

TBD

Goal is to look at data and try to piece how it goes together. Start with just a list of tables and go through "how do we explore this data?"

1. Look at the data (top-10 for each)
2. Think about how it might go together (draw a schema diagram)
3. What do we expect to be unique?
4. How do we test to see if it is unique?
5. What are some one-off things that we can do?

6 Query Strategies

Query strategies: Add a section "Query Strategy" to each lecture which covers some of the more descriptive aspects covered in class.

- Write out what you want.
- Write out conditions for getting it and where data is.
- Build from the inside out.
- For aggregation queries, explain what they doing and dig hard into the data shape.