codecademy

Calculating Churn Rates

Capstone Project by Klara Mitteregger

Table of Contents

1. Get familiar with Codeflix

1.1 How many different segments do you see?

I use the SELECT and LIMIT function to view the different segments (select * to select all columns from the subscriptions table).

I see 2 different segments, 87 and 30, illustrated below (I’ve selected row 1 and 13 as examples in the table below).

id	subscription_start	subscription_end	segment
1	2016-12-01	2017-02-01	87
13	2016-12-01	Ø	30

```
Query:

select *
from subscriptions
limit 100;
```

1.2 Which months will you be able to calculate churn for?

In order to determine the months we are able to calculate churn for, we need to establish the earliest subscription start date and the latest subscription start date so we know the entire range of subscription dates. Then we know all the months we can calculate churn for.

For this we can use the MAX and MIN functions.
The earliest subscription start date is 2016-12-01, and the latest subscription start date is 2017-03-30.

min (subscription_start	max (subscription_end
2016-12-01	2017-03-30

The minimum subscription length is 31 days - a user can never start and end their subscription in the same month. This means that for start month December, there are no active or end users. So, we cannot calculate December churn rate, but only January, February and March.

```
Query:

select min (subscription_start),
max(subscription_start) from subscriptions;
```

2. What is the overall churn rate by month?

2.1 Creating a temporary table named *months*

Following the instructions, we create a temporary table named *months*. The purpose of this table is to cross join it with our subscriptions table. so that for each subscription we can see for each month whether it was active or not.

We use the WITH function in order to create the temporary table, and UNION in order to stack the 3 months on top of each other – creating 3 rows (January, February and March).

First_day	Last_day
2017-01-01	2017-01-31
2017-02-01	2017-02-28
2017-03-01	2017-03-31

```
Query:

With months as
(select
'2017-01-01' as first_day,
'2017-01-31' as last_day
UNION
select
'2017-02-01' as first_day,
'2017-02-28' as last_day
UNION
select
'2017-03-01' as first_day,
'2017-03-31' as last_day)
select *
from months;
```

2.2 Create a temporary table *cross_join* from subscriptions and months

Now that we have created *months* in our first query, we can select it for our new, revised query without using the WITH function.

Following the instructions, we create a table named *cross_join* by CROSS JOIN *subscriptions* and *months,* so that for each subscription there is a first_day and last_day for each month, creating 3 rows for each subscription id (all possible combinations for each id - January, February and March). See the example table below.

id	Subscription_start	Subscription_end	segment	First_day	Last_day
1	2016-12-01	2017-02-01	87	2017-01-01	2017-01-31
1	2016-12-01	2017-02-01	87	2017-02-01	2017-02-28
1	2016-12-01	2017-02-01	87	2017-03-01	2017-03-31

```
Query:

With months as
(select
'2017-01-01' as first_day,
'2017-01-31' as last_day
UNION
select
'2017-02-01' as first_day,
'2017-02-28' as last_day
UNION
select
'2017-03-01' as first_day,
'2017-03-31' as last_day),
cross_join as
(select * from subscriptions
cross join months)
select * from cross_join;
```

2.3 Create a temporary table *status* from table *cross_join*

Following the instructions, we now add another table to our query named *status*.

We use the CASE WHEN function in order to check which users are active for each month. We need to combine CASE WHEN with the AND and OR functions in order to determine for each id (subscription) over each segment (87 and 30), whether the id was active or not for each month. See the table below (example for id 1)

id	month	Is_active_87	Is_active_30
1	2017-01-01	1	0
1	2017-02-01	0	0
1	2017-03-01	0	0

```
Query:

...
cross join months),
status as
(select
id,
first_day as month,
case when
(subscription_start < first_day) AND (subscription_end >
first_day OR subscription_end is NULL) AND (segment =
87)
 then 1
 else 0
 end as is_active_87,
 case when
(subscription_start < first_day) AND (subscription_end >
first_day OR subscription_end is NULL) AND (segment =
30)
 then 1
 else 0
 end as is_active_30
 from cross_join)
 select * from status;
```

2.4 Add *is_canceled* columns to the *status* temporary table

We want to do the same thing as in 2.3, but now for canceled subscriptions. We need to add two *canceled* columns to our *status* table for each segment, so *is_canceled_87* and *is_canceled_30*. We use the CASE WHEN function again, together with the BETWEEN function, in order to check for each id over each segment, in which month (or not at all) the subscription was canceled.

id	month	Is_active_87	Is_active_30	Is_canceled_87	Is_canceled_30
1	2017-01-01	1	0	0	0
1	2017-02-01	0	0	1	0
1	2017-03-01	0	0	0	0

```
Query:

...
 end as is_active_30,
  case when
 (subscription_end BETWEEN first_day and last_day) and (segment = 87)
 then 1
 else 0
 end as is_canceled_87,
  case when
 (subscription_end BETWEEN first_day and last_day) and (segment = 30)
 then 1
 else 0
 end as is_canceled_30
 from cross_join)
 select * from status;
```

2.5 Create a temporary table named *status_aggregate*

As the name implies, we're now going to add a table to our query named status_aggregate that aggregates the values from our *status_table*. We are summing the 1:s for every column, naming the new columns *sum*... For this we use the SUM function.

Now we have an overview over the total number of active and canceled subscriptions for each segment.

Sum_active_87	Sum_active_30	Sum_canceled_87	Sum_canceled_30
1271	1525	476	144

```
Query:

...
from cross_join),
 status_aggregate as
 (select
 sum(is_active_87) as sum_active_87,
 sum(is_active_30) as sum_active_30,
 sum(is_canceled_87) as sum_canceled_87,
 sum(is_canceled_30) as sum_canceled_30
 from status)
 select * from status_aggregate;
```

2.6 Calculate the churn rates over the three month period (1)

We have the total number of active and canceled subscriptions for each segment, but we are going to calculate the churn rate for each month, so we need the data aggregated over month as well.
We add the GROUP BY function to our query in order to group the total number of active and canceled subscriptions per *month*. The month column will be the *first_day* column that we aliased as *month* in step 2.3. We also need to add the *month* column to our table so we can clearly see the different groups.

month	Sum_active_87	Sum_active_30	Sum_canceled_87	Sum_canceled_30
2017-02-01	278	291	70	22
2017-02-01	462	518	148	38
2017-03-01	531	716	258	84

```
Query:

...
from cross_join),
 status_aggregate as
 (select
  month,
 sum(is_active_87) as sum_active_87,
 sum(is_active_30) as sum_active_30,
 sum(is_canceled_87) as sum_canceled_87,
 sum(is_canceled_30) as sum_canceled_30
 from status
  group by month)
 select * from status_aggregate;
```

2.7 Calculate the churn rates over the three month period (2)

Now we can calculate the churn rates for the two segments over the three month period.
Churn rate = number of canceled subscribers / number of active users
We need to calculate the churn rate for each segment for each month.
We can do this calculation within our query, multiplying it with 1.0 to force a float number, as churn rates will be small and we want to see the decimal points.
We can use the ROUND function to decrease the number of decimals.

month	Churn_rate_87	Churn_rate_30
2017-02-01	0.252	0.076
2017-02-01	0.32	0.073
2017-03-01	0.486	0.117

The churn rates for segment 30 are significantly smaller than for segment 87. Segment 30 has the lower churn rate for every month, and so is performing better than segment 87.

```
Query:

...
 from cross_join),
 status_aggregate as
 (select
  month,
 sum(is_active_87) as sum_active_87,
 sum(is_active_30) as sum_active_30,
 sum(is_canceled_87) as sum_canceled_87,
 sum(is_canceled_30) as sum_canceled_30
 from status
  group by month)
  select
  month,
  round(1.0 * sum_canceled_87/sum_active_87,3) as
churn_rate_87,
    round(1.0 * sum_canceled_30/sum_active_30,3) as
churn_rate_30
    from status_aggregate;
```

3. Comparing the churn rates between segments

3.1 How would you modify this code to support a large number of segments?

If you have a large number of segments, it’s better not to hard code the segments and alias them. Do not use the AND function to define the segments (for example ’AND (segment = 87) … END AS is_active_87,’.

You can remove this criterium and instead add the segment column when you are adding the new table, in order to get active and canceled subscriptions grouped by all segments available in your data. Don’t forget to use GROUP BY to group the segment column in your query, so that you aggregate data over the distinct segment groups.

month	segment	Churn_rate
2017-01-01	30	0.076
2017-02-01	30	0.073
2017-03-01	30	0.117
2017-01-01	87	0.252
2017-02-01	87	0.32
2017-03-01	87	0.486

```
...
cross join months),
status as
(select
id,
first_day as month,
 segment,
case when
(subscription_start < first_day) AND (subscription_end
> first_day OR subscription_end is NULL)
 then 1
 else 0
 end as is_active,
  case when
 (subscription_end BETWEEN first_day and last_day)
 then 1
 else 0
 end as is_canceled
 from cross_join),
 status_aggregate as
 (select
  month,
  segment,
 sum(is_active) as sum_active,
 sum(is_canceled) as sum_canceled
 from status
  group by month,segment)
  select
  month, segment,
  round(1.0 * sum_canceled/sum_active,3) as churn_rate
    from status_aggregate
    order by segment asc;
```