codecademy

Calculating Codeflix's Churn Rate

Analyze Data with SQL

By Christopher Lim

April 2, 2020

Table of Contents

1. Introduction

1.1 SQL Table (subscriptions)

The subscription table contains the following columns:

- Id - the subscription id.
- subscription_start - the start date of the subscription.
- subscription_end - the end date of the subscription.
- segment - identifies which segment to subscription owner belongs to.

id	subscription_start	subscription_end	segment
1	2016-12-01	2017-02-01	87
2	2016-12-01	2017-01-24	87
3	2016-12-01	2017-03-07	87
4	2016-12-01	2017-02-12	87

1.2 Range of Months to Calculate Churn Rate

By getting the minimum starting date (2016-12-01) and maximum end date (2017-03-31) of the subscriptions, I was able to determine that the range of months to calculate the churn rate is from **January**, **February**, up until **March**.

Note: Normally, we should also calculate the churn rate of December but since the company had a constraint of minimum subscription length of 31 days before they can end their subscription, we start with January.

min_start	max_end
2016-12-01	2017-03-31

```
SELECT MIN(subscription_start) as min_start,
       MAX(subscription_end) as max_end
FROM subscriptions;
```

2. Method

2.1 Months

We first make a temporary table “*months*” which consists of columns:

- first_day - first day of the month
- last_day - last day of the month

first_day	last_day
2017-01-01	2017-01-31
2017-02-01	2017-02-28
2017-03-01	2017-03-31

```
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'
),
```

2.2 Cross Join

Next we make a temporary table named “*cross_join*” where we will **CROSS JOIN** the months table with the subscriptions table.

id	subcription_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
2	2016-12-01	2017-01-24	87	2017-01-01	2017-01-31

Continuation of the code...

```
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
),
```

2.3 Status

Then we make a temporary table *"status"* which has columns:

- Id - user id
- month - first_day of table cross_join given alias as "month"
- is_active_30 - returns 1 if segment 30 and is active else 0
- is_active_87 - returns 1 if segment 87 and is active else 0
- Is_canceled_30 - returns 1 if segment 30 and canceled else 0
- Is_canceled_87 - returns 1 if segment 87 and canceled else 0

Note: This is the crucial point in calculating the churn. The data extracted based on activeness and canceled is crucial in order to get the correct data.

month	is_active_30	is_active_87	is_canceled_30	is_canceled_87
2017-01-01	0	1	0	0
2017-02-01	0	0	0	1
2017-03-01	0	0	0	0
2017-01-01	0	1	0	1

Continuation of the code...

```
status AS
(SELECT first_day as month,
   CASE
     WHEN segment = 30
       AND subscription_start < first_day
       AND (subscription_end > first_day
            OR subscription_end IS NULL)
     THEN 1
     ELSE 0
   END as is_active_30,
   CASE
     WHEN segment = 87
       AND subscription_start < first_day
       AND (subscription_end > first_day
            OR subscription_end IS NULL)
     THEN 1
     ELSE 0
   END as is_active_87,
   CASE
     WHEN segment = 30
       AND (subscription_end
             BETWEEN first_day AND last_day)
     THEN 1
     ELSE 0
   END as is_canceled_30,
   CASE
     WHEN segment = 87
       AND (subscription_end
            BETWEEN first_day AND last_day)
     THEN 1
     ELSE 0
   END as is_canceled_87
 FROM cross_join
),
```

2.4 Status Aggregate

Then we make a temporary table "*status_aggregate*" where we compute the following:

- Total number of active users in segment 30
- Total number of active users in segment 87
- Total number of users who canceled in segment 30
- Total number of users who canceled in segment 87

These results would then give us capability to now compute for the churn rate for each section.
Then finally, we can compute for the churn rate of each month.

Continuation of the code...

```
status_aggregate AS
(SELECT month,
   SUM(is_active_30) as sum_active_30,
   SUM(is_canceled_30) as sum_canceled_30,
   SUM(is_active_87) as sum_active_87,
   SUM(is_canceled_87) as sum_canceled_87
 FROM status
 GROUP BY month
)
SELECT month,
      ROUND(1.0 * sum_canceled_30 /
            sum_active_30, 4) as churn_30,
      ROUND(1.0 * sum_canceled_87 /
            sum_active_87, 4) as churn_87
FROM status_aggregate;
```

month	sum_active_30	sum_canceled_30	sum_active_87	sum_canceled_87
2017-01-01	291	22	278	70
2017-02-01	518	38	462	148
2017-03-01	716	84	531	258

month	churn_30	churn_87
2017-01-01	0.0756	0.2518
2017-02-01	0.0734	0.3203
2017-03-01	0.1173	0.4859

2.5 Complete Code

```
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
),
status AS
(SELECT id, first_day as month,
   CASE
     WHEN segment = 30
       AND subscription_start < first_day
       AND (subscription_end > first_day OR subscription_end IS NULL)
     THEN 1
     ELSE 0
   END as is_active_30,
   CASE
     WHEN segment = 87
       AND subscription_start < first_day
       AND (subscription_end > first_day OR subscription_end IS NULL)
     THEN 1
     ELSE 0
   END as is_active_87,
   CASE
     WHEN segment = 30
       AND (subscription_end BETWEEN first_day
            AND last_day)
     THEN 1
     ELSE 0
   END as is_canceled_30,
   CASE
     WHEN segment = 87
       AND (subscription_end BETWEEN first_day
             AND last_day)
     THEN 1
     ELSE 0
   END as is_canceled_87
 FROM cross_join
),
status_aggregate AS
(SELECT month,
   SUM(is_active_30) as sum_active_30,
   SUM(is_active_87) as sum_active_87,
   SUM(is_canceled_30) as sum_canceled_30,
   SUM(is_canceled_87) as sum_canceled_87
 FROM status
 GROUP BY month
)
SELECT month, 1.0 * sum_canceled_30 / sum_active_30
as churn_30, 1.0 * sum_canceled_87 / sum_active_87
as churn_87 FROM status_aggregate;
```

3. Results

Results

Based on the results, we can see that overall, Section 87 has a higher churn rate than Section 30.

Month	Churn Rate for Section 87	Churn Rate for Section 30
1st Month	25.18%	7.56%
2nd Month	32.03%	7.34%
3rd Month	48.59%	11.73%

4. Larger Number of Segments

Larger Number of Segments

For the code to support a larger number of segments, by not hard coding and not using brute force, we can get the status and return the **month**, **segment**, **is_active** and **is_canceled** condition for our status temporary variable.

month	segment	is_active	is_canceled
2017-01-01	87	1	0
2017-02-01	87	0	1
2017-03-01	87	0	0
2017-01-01	87	1	0
2017-02-01	87	0	0
2017-03-01	87	0	0

```
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
),
status AS
(SELECT 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
),
```

Larger Number of Segments

Continuation of the code...

```
status_aggregate AS
(SELECT month, segment,
   SUM(is_active) as total_active,
   SUM(is_canceled) as total_canceled
 FROM status
 GROUP BY segment, month
)
SELECT month, segment,
       ROUND(1.0 * total_canceled / total_active, 4) as churn
FROM status_aggregate;
```

Then we take the **sum of all the is_active** and **is_canceled** and then **group them according to group and segment**.

Then we can finally **take the churn of each corresponding month and segment**.

month	segment	total_active	total_canceled
2017-01-01	30	291	22
2017-02-01	30	518	28
2017-03-01	30	716	84
2017-01-01	87	278	70
2017-02-01	87	462	148
2017-03-01	87	531	258

month	segment	churn
2017-01-01	30	0.0756
2017-02-01	30	0.0734
2017-03-01	30	0.1173
2017-01-01	87	0.2518
2017-02-01	87	0.3203
2017-03-01	87	0.4859

2.5 Complete Code

```
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
),
status AS
(SELECT 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 total_active,
   SUM(is_canceled) as total_canceled
 FROM status
 GROUP BY segment, month
)
SELECT month, segment,
      ROUND(1.0 * total_canceled / total_active, 4)
            as churn
FROM status_aggregate;
```

5. Conclusion

Conclusion

In conclusion:

1. Codeflix should **focus on the segment 87 of users to reduce its churn rate** since it has a higher churn rate.
2. A **better solution can be implemented to support a larger number of segments** if the company intends to.

 # The End