codecademy

User Churn - Codeflix

Analyze Data with SQL

Athena Mascarenhas

September 14th, 2024

Table of Contents

1. Project Description

1. Project Description

Four months into launching Codeflix, management asks you to look into subscription churn rates. It's early on in the business and people are excited to know how the company is doing

The marketing department is particularly interested in how the churn compares between two segments of users. They provide you with a dataset containing subscription data for users who were acquired through two distinct channels.

The dataset provided to you contains one SQL table, subscriptions. Within the table, there are 4 columns:

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

2. Company Information

2.1 Company Information

Codeflix, a streaming video startup, is interested in measuring their user churn rate. Codeflix requires a minimum subscription length of 31 days, so a user can never start and end their subscription in the same month.

Here is a snippet of the subscriptions table:

Query Results			
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
5	2016-12-01	2017-03-09	87
6	2016-12-01	2017-01-19	87
7	2016-12-01	2017-02-03	87
8	2016-12-01	2017-03-02	87
9	2016-12-01	2017-02-17	87
10	2016-12-01	2017-01-01	87
11	2016-12-01	2017-01-17	87
12	2016-12-01	2017-02-07	87
13	2016-12-01	Ø	30
14	2016-12-01	2017-03-07	30
15	2016-12-01	2017-02-22	30

2.2 Company Information: Date Range and Segments

- The company has been operating for about four months from December 2016 through March 2017.

```
SELECT MIN(subscription_start) as date_min,
MAX(subscription_start) as date_max
FROM subscriptions;
```

date_min	date_max
2016-12-01	2017-03-30

- Since Codeflix requires a minimum subscription length of 31 days, there is no subscription_end dates in December!
- There are two user segments: 87 and 30

```
SELECT DISTINCT segment FROM subscriptions;
```

segment
87
30

3. Churn Rates of Segments

3. Churn Rates of Segments

We will determine the churn rates of segment 87 and segment 30. Here is an overview of the process:

3.1 Create temporary months table

3.2 (Cross) Join subscriptions table with months

3.3 Create temporary status table with active and cancelled subscription flags

3.4 Create temporary table with aggregated active and cancelled subscription flags

3.5 Calculate the churn rates using temporary tables

3.

- 2. What is the overall churn trend since the company started?
- 3. Compare the churn rates between user segments.
- Which segment of users should the company focus on expanding?

3.1 Churn Rates of Segments: Create temporary months table

Since there is no months table available, a temporary months table was created with the first_day and last_day of each of the three months (January through March).

```
--create temporary months table
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;
```

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

3.2 Churn Rates of Segments: (Cross) Join subscriptions table with months

Next, the main subscriptions table was cross joined with months to create a new temporary table called, cross_join.

```
--create temporary cross_join
table
--Limited to 8 records
cross_join as (
  SELECT * from subscriptions
  CROSS JOIN months
)
SELECT * FROM cross_join
LIMIT 8;
```

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
2	2016-12-01	2017-01-24	87	2017-01-01	2017-01-31
2	2016-12-01	2017-01-24	87	2017-02-01	2017-02-28
2	2016-12-01	2017-01-24	87	2017-03-01	2017-03-31
3	2016-12-01	2017-03-07	87	2017-01-01	2017-01-31
3	2016-12-01	2017-03-07	87	2017-02-01	2017-02-28

Note: The code above is a continuation of the WITH clause from 3.1. Due to limited space, we are not including the entire code.

3.3 Churn Rates of Segments: Create temporary status table with active and cancelled subscription flags

```
status as (
  SELECT id, first_day as month,
  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_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_end BETWEEN first_day and last_day
              ) THEN 1
              ELSE 0 END AS is_canceled_87,
  CASE WHEN (
              segment == 30  AND
              subscription_end BETWEEN first_day and last_day
              ) THEN 1
              ELSE 0 END AS is_canceled_30
  FROM cross_join)
  SELECT * FROM status LIMIT 8;
```

Created a temporary status table that will contain the subscription id, month (first day) and the active and cancelled subscription flags for each segment.

The active and canceled flags will be 1 (yes) or 0 (no).

Results of the code will be in the next slide

Note: The code above is a continuation of the WITH clause from 3.1 and 3.2. Due to limited space, we are not including the entire code.

3.3 Churn Rates of Segments: Create temporary status table with active and cancelled subscription flags

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
2	2017-01-01	1	0	1	0
2	2017-02-01	0	0	0	0
2	2017-03-01	0	0	0	0
3	2017-01-01	1	0	0	0
3	2017-02-01	1	0	0	0

3.4 Churn Rates of Segments: Create temporary table with aggregated active and cancelled subscription flags

Created a temporary table that stored the month (first day) and the counts of the active and cancelled flags for each segment.

```
--create a temporary table with aggregated flags
status_aggregate AS (
  SELECT month,
                SUM(is_active_87) as active_87,
            SUM(is_active_30) as active_30,
            SUM(is_canceled_87) as canceled_87,
            SUM(is_canceled_30) as canceled_30
    FROM status
    GROUP BY 1
  )
  SELECT * FROM status_aggregate;
```

month	active_87	active_30	canceled_87	canceled_30
2017-01-01	278	291	70	22
2017-02-01	462	518	148	38
2017-03-01	531	716	258	84

Note: The code above is a continuation of the WITH clause from 3.1-3.3. Due to limited space, we are not including the entire code.

3.5 Churn Rates of Segments: Calculate the churn rates using temporary tables

- Finally, calculated the churn rates of each segment for each of the months (January through March).
- Segment 30 has a lower churn rate than segment 87.

```
--Calculate churn rates
SELECT month,
             1.0 * canceled_87/active_87 as
churn_rate_87,
             1.0 * canceled_30/active_30 as
churn_rate_30
  FROM status_aggregate;
```

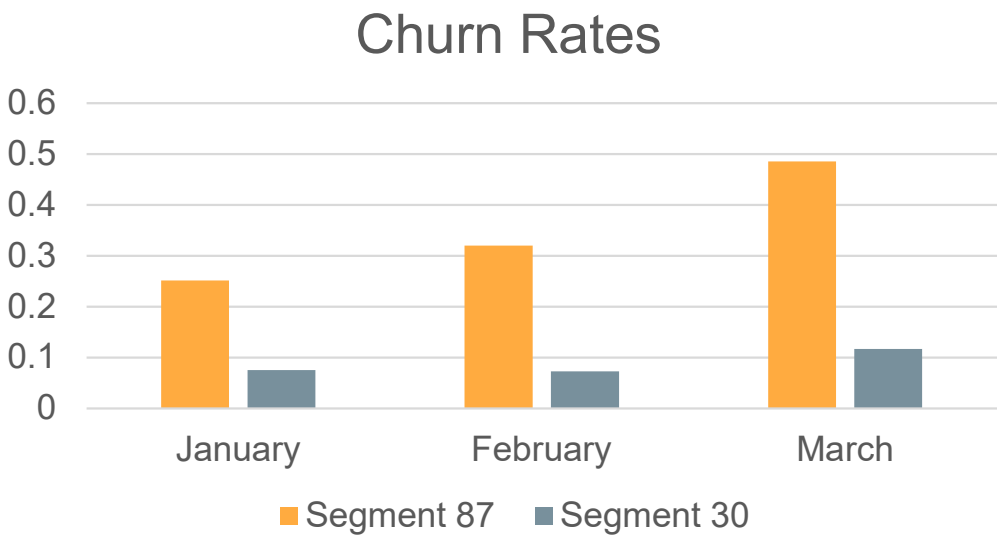
month	churn_rate_87	churn_rate_30
2017-01-01	0.251798561151079	0.0756013745704467
2017-02-01	0.32034632034632	0.0733590733590734
2017-03-01	0.485875706214689	0.11731843575419

Note: The code above is a continuation of the WITH clause from 3.1 - 3.4. Due to limited space, we are not including the entire code.

4. Conclusion

5. Conclusion

- It looks like the churn rate for both segments increase as the months progress
- Segment 87 has a higher churn rate compared to Segment 30. The company should look into what's working for segment 30 and what's not for 87.
- The company should probably consider looking at the new subscriber rate to make sure that the new subscriber rate is higher than the churn rate.



5. Appendix

4.1 Alternate Code (without hardcoding segments)

```
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, 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 active,
                SUM(is_canceled) as canceled
    FROM status
    GROUP BY 1,2
  )
SELECT month, segment,
                1.0 * canceled/active as churn_rate
                FROM status_aggregate;
```

The project tasked us to generate the churn rate for each segment without hardcoding the segments.

This results in a smaller block of code and can accommodate any number of segments.

Results on the next slide. Results should be the same.

4.2 Alternate Code (without hardcoding segments): Results

Results of the Original Code:

month	churn_rate_87	churn_rate_30
2017-01-01	0.251798561151079	0.0756013745704467
2017-02-01	0.32034632034632	0.0733590733590734
2017-03-01	0.485875706214689	0.11731843575419

Results of the Alternate Code:

month	segment	churn_rate
2017-01-01	30	0.0756013745704467
2017-01-01	87	0.251798561151079
2017-02-01	30	0.0733590733590734
2017-02-01	87	0.32034632034632
2017-03-01	30	0.11731843575419
2017-03-01	87	0.485875706214689