CODEFLIX CHURN DATA

LEARN SQL FROM SCRATCH

MONICA LOPEZ

codecademy

TABLE OF CONTENTS

1. GET FAMILIAR WITH CODEFLIX

1.1 UNDERSTAND THE DATA FILE

First thing I needed to do was look at the structure of the data file itself. I can see that there are 4 columns consisting of ID, subscription start and end dates, and a segment ID.

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 UNDERSTAND MORE ABOUT CODEFLIX

Using the code to the left, I was able to discover that Codeflix has data for 4 months: Dec 2017 through March 2017. And it has two segment IDs: 87 and 30.

I added MAX(subscription_end) just in case there were subscriptions that went past the start months. This data shows us we can calculate churn for January 2017 through March 2017.

```
SELECT DISTINCT segment
FROM subscriptions;

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

2. OVERALL MONTHLY CHURN RATE

2.1 CREATING THE CODE FOR MONTHLY CHURN

In creating the code for overall monthly churn, I created the temporary tables of months, cross_join, status, and status_agreggrate and ignored the segment variable.

I followed the methodology taught in the course where a months file is created to cross join with subscriptions, which then allows me to compare the subscription start and end dates. From there I can determine, for any given month, whether or not a subscription was active at the beginning of the month, and whether or not it was canceled that month.

```
WITH months AS
 (SELECT
  '2017-01-01' AS first_day,
  '2017-01-31' 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 cross_join.id,
   cross_join.first_day AS month,
  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, SUM(is_active) AS
 sum_active, SUM(is_canceled) AS
 sum_canceled
 FROM status
 GROUP BY month)
SELECT month,
1.0*sum_canceled/sum_active AS churn_rate
FROM status_aggregate
GROUP BY month;
```

2.2 OVERALL MONTHLY CHURN RATE

We can see that Codeflix's churn rate is increasing over time and is 11% higher from January to March! Something is happening where customers are becoming unsatisfied with their service. We would want to know more about what has changed over the last couple of months to create such a loss in customers.

Month	Churn Rate
2017-01-01	16.1%
2017-02-01	18.9%
2017-03-01	27.2%

**Note: Churn is calculated as cancelations/active subscribers for that month

3. MONTHLY CHURN RATE BY SEGMENTS

3.1 CREATING THE CODE FOR MONTHLY CHURN BY SEGMENT

In creating the code by segment, I followed the structure suggested by the project steps. The beginning temporary tables of months and cross_join remain the same. Then to distinguish between segment 87 and 30, I added the 'AND segment='87' or 'AND segment='30' respectively to the status table, and calculated the aggregates separately as well. I've only shown segment 87 for sake of space, but it is all listed in my code file.

```
...
 status AS
 (SELECT cross_join.id, cross_join.segment,
  cross_join.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_end
   BETWEEN first_day AND last_day
   AND segment = '87'
   THEN 1
   ELSE 0
  END AS is_canceled_87,
  ...
 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 1)
SELECT month,
1.0*sum_canceled_87/sum_active_87 AS '87 Churn
Rate',
1.0*sum_canceled_30/sum_active_30 AS '30 Churn
Rate'
FROM status_aggregate
GROUP BY 1;
```

3.2 MONTHLY CHURN RATE BY SEGMENT

We can see that for each segment, Codeflix's churn rate is still increasing over time. Something happened in March for both segments to more than double their churn rates.

Segment 30 has a much lower churn rate than segment 80. It actually decreased in February, before it more than doubled in March. We would want to look further into what is different about segment 30, and what happened in February for that group to see what is working well; and look at segment 87 to see what is not working for us.

Month	87 Churn Rate	30 Churn Rate
2017-01-01	25.1%	7.6%
2017-02-01	31.7%	7.3%
2017-03-01	47.7%	11.7%

4. BONUS: ADAPTING THE CODE FOR MULTIPLE SEGMENTS

4.1 CREATING THE CODE TO SIMPLIFY MONTHLY CHURN BY SEGMENT

As before, the beginning temporary tables of months and cross_join remain the same. But then to simplify the code, I went back to just calculating active and canceled subscriptions without the additional AND statement. Then I added segment as a selected variable and a grouping variable.

It's creates a more streamlined process and simplifies the code.

```
...
 status AS
 (SELECT cross_join.id, cross_join.first_day
  AS month, cross_join.segment AS 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,
1.0*sum_canceled/sum_active AS churn_rate
FROM status_aggregate
GROUP BY month, segment;
```

4.2 MONTHLY CHURN RATE BY SEGMENT

We can see the results are the same, but the structure is a little different. It creates longer table vs the wide table we had before. The ordering of the group by variables determines which column will be grouped first. In this case, I chose the months to be first and the segments to be second.

Month	Segment	Churn Rate
2017-01-01	30	7.6%
2017-01-01	87	25.1%
2017-02-01	30	7.3%
2017-02-01	87	31.7%
2017-03-01	30	11.7%
2017-03-01	87	47.7%