codecademy

Codeflix Churn Rates

A “Learn SQL from Scratch” Codecademy Capstone

USER: **NATO2042**

Capstone-tacular Table of Contents

Part 01:

The Company, The Measure, and The Method

1. Codeflix: Who, How, and What?

- **Codeflix** is a video's on demand (VOD) internet streaming service
- The new startup opened its doors in December 2016 and has been operating for at four months
- They are interesting measuring there subscribers churn rates
- The marketing department is particularly interested in the performance of two key acquisition channels of users, identified via segments 30 and 87

2. Churn rates: All you ever wanted to know

- **Churn rate** (sometimes referred to as attrition) is the percentage of subscribers to a service who discontinue their subscriptions to the service within a given time period.
- For a company to expand its clientele, its growth **rate**, as measured by the number of new customers, must exceed its **churn rate**.
- This **rate** is generally expressed as a percentage.
- In this analysis Codeflix requires all donors to remain ‘active’ for a minimum of 31 days so a user can never have a subscription that start and ends on the same month

3a. Method: Data, Tools, and Techniques

- **Codeflix** marketing department has provided us with 2000 records for our analysis in an SQL table called **subscriptions**

- Will we be utilising **SQL** to load, view and manipulate this data to reach our results and conclusions

- In the next slide we will cover some of the common SQL commands used in the project

3b. Method: Data, Tools, and Techniques

- SQL Commands

SELECT	Fetch specified data from database
FROM	Specified table/database
LIMIT	Specify maximum rows to display

Aggregate related	
MIN	Smallest value in column
MAX	Largest value in column
SUM	Addition of values
GROUP BY	Only used with aggregate functions

Data joins	
UNION	Combines two or more select statements
CROSS JOIN	Each row for 1st table joins will all the rows of another table e.g. x * y rows

WITH	Temporary table creator (Common table expression)
AS	Alias creator for column or table
DISTINCT	Selects unique values
AND	Combining operator
OR	Either operator
IS NULL	Is field being calculated empty
CASE / WHEN / THEN / ELSE / END AS	Define set conditions into specified columns

Part 02:

Investigation and Results

4a. Investigation: Step 01 – Get familiar with the data

Calling the first query SELECT / LIMIT we identify four columns of data:

- id
- subscription_start
- subscription_end
- segment

Via the second query SELECT COUNT() we observe there are:

- 2000 records of data

And using the final query SELECT DISTINCT are two distinct segments:

- 30
- 87

```
-- STEP01: Data investigation 1 (Segments = 2 (30, 87)
| Records: 2000)

SELECT *
FROM subscriptions
LIMIT 100;

SELECT COUNT(*)
FROM subscriptions;

SELECT DISTINCT segment
FROM subscriptions;
```

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

4b. Investigation: Step 02 – What are our date ranges

1st query SELECT / MIN / MAX checks the smallest and largest values in the column subscription_start

- Smallest: 2016-12-01
- Largest: 2017-03-30

2nd query runs the same commands against subscription_end:

- Smallest: 2016-12-01
- Largest: 2017-03-30

```
--STEP02: Data investigation 2 (Month start: Dec-2016
| Month End: Mar-2017)

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

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

MIN(subscription_start)	MAX(subscription_start)
2016-12-01	2017-03-30
MIN(subscription_end)	MAX(subscription_start)
2017-01-01	2017-03-30

5a. Step 03 – Defining our ‘months’ table

Query utilises the WITH / AS commands in conjunction with UNION to create a temporary table of first_day/last_day ‘month’ rows.

Thus we are left with a table of the 3 months that we will be using to calculate churn rates between the two segments.

first_day	last_day
2017-01-01	2017-01-31
2017-02-01	2017-02-28
2017-03-01	2017-03-30

```
-- STEP03: Temp table 1 (manual created data ranges)

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,
'2017-03-30' AS last_day)

SELECT *
FROM months;
```

5b. Step 04 – Integrating our initial data with the ‘months’ table

In this step we use the CROSS JOIN command to stitch the temporary ‘months’ table with our initial ‘subscriptions’ data table into a newly created temporary table ‘cross_join’

CROSS JOIN will join each row (x) from table 1 with each row from table 2 (y): so x * y rows

In essence it will enable us to cross reference a user’s (id) subscription_start and subscription_end against the ‘months’ we created before on a row by row basis.

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	2017-03-30

```
-- STEP04: Temp table 2 (CROSS JOIN merge
subscriptions | months)

cross_join AS
  (
SELECT *
FROM subscriptions
CROSS JOIN months
)

SELECT *
FROM cross_join
LIMIT 10;
```

5c. Step 05 – Defining whether a subscriber is active for the given month

We create another temporary table ‘status’ using data within the ‘cross_join’ table

In conjunction with the CASE / WHEN / (AND/OR) / THEN / ELSE / END AS commands to identify whether a user (id) was active for a given month for a given segment.

id	month	is_active_30	is_active_87
1	2017-01-01	0	1
1	2017-02-01	0	0
1	2017-03-01	0	0

```
--STEP05: Temp table 3A (show active 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
  FROM cross_join)

SELECT *
FROM status
LIMIT 50;
```

5d. Step 06 – Adding whether a subscriber has cancelled in the given month

Expanding on the previous step, we then add two columns to determine whether a user (id) was cancelled for any given month for a given segment.

id	month	is_active_30	is_active_87	is_cancelled_30	is_cancelled_87
1	2017-01-01	0	1	0	0
1	2017-02-01	0	0	0	1
1	2017-03-01	0	0	0	0

```
--STEP06: Temp table 3B (show cancelled months)

   CASE
  WHEN segment = 30
AND (subscription_end
BETWEEN first_day
AND last_day)
THEN 1
ELSE 0
    END AS is_cancelled_30,

   CASE
  WHEN segment = 87
AND (subscription_end
BETWEEN first_day
AND last_day)
THEN 1
ELSE 0
    END AS is_cancelled_87
  FROM cross_join)

SELECT *
FROM status
LIMIT 50;
```

5e. Step 07 – Table summary of active/cancelled by segment

Nearing the end our analysis we tie all the temporary tables into an aggregate based temporary table using the SUM and GROUP BY commands.

This consolidates the data we have calculated and generated thus far into a neat table displaying the sum users (id) active or cancelled by segments for a given month.

month	sum_active_30	sum_active_87	sum_cancelled_30	sum_cancelled_87
2017-01-01	291	278	22	70
2017-02-01	518	462	38	148
2017-03-01	716	531	81	247

```
--STEP07: Temp table 4 (aggregates - summing segements groups
by active and cancelled)

status_aggregate AS
  (SELECT month,
    SUM(is_active_30) AS sum_active_30,
  SUM(is_active_87) AS sum_active_87,
SUM(is_cancelled_30) AS sum_cancelled_30,
SUM(is_cancelled_87) AS sum_cancelled_87
FROM status
GROUP BY month)

SELECT *
FROM status_aggregate;
```

5f. Step 08 – Churn Rates by segment (OUR RESULTS!!!)

With the data created in the previous step we are now ready to calculate the churn rates.

To achieve with we divide the cancellations against the actives and multiple by 1.0 to ensure a float (decimal) return.

As we can clearly see below ‘segment 30’ has a much lower churn rate and is thus the better ‘sticky’ group of subscribers for Codeflix.

month	seg30_churn_rate	seg87_churn_rate
2017-01-01	0.0756013745704467	0.251798561151079
2017-02-01	0.0733590733590734	0.32034632034632
2017-03-01	0.113128491620112	0.465160075329567

```
--STEP08: FINAL CHURN RATE (calculating churn - segment 30
has the lower churn rate @ 11.3% for 3 cumulative months,
whereas segment 87 is at 46.5% over the same period)

SELECT month,
1.0 * sum_cancelled_30 / sum_active_30 AS seg30_churn_rate,
1.0 * sum_cancelled_87 / sum_active_87 AS seg87_churn_rate
FROM status_aggregate;
```

Part 03:

Conclusion

6. Conclusions

- In conclusion segment 30 achieved a much lower (thus favourable) churn rate at only 11.3% over 3 months compared to segment 87 which experienced a very high rate of cumulative churn of over 46% during the same period.

- It would be our recommendation for the Codeflix marketing department to focus on acquisition expansion for the segment 30 group

- It might also be useful to explore if the good practices in segment 30 can be transferred over to the segment 87 acquisition channel.