Capstone Project: Calculating Churn Rates

A Student

The Offset:
Understanding the Project

Operation Data: What months can be used?

The company began in December 2016. There is insufficient data for this month to be used, so the start month for data analytics is January 2017.

The months that have data so far are:

- January 2017
- February 2017
- March 2017

Segments: Which Exist?

Two segments exist, Segment 87 and Segment 30.

These will be differentiated between throughout the entire process.

The Initial Goal: Churn Rates per Segment per Month

The initial goal of the analysis process is to discover the churn rate per segment per month.

This can be done with a small amount of SQL coding, which shall be demonstrated in the following slides.

At the end, the final results regarding the Churn Rates will be revealed.

The Coding:
Manipulating and Narrowing the Data

The First Step: The Temporary Month Table

The purpose of this step is to get a table going for the different months in question.

The reason for this is to remove the need for hard-coding month values into the program, instead the code can be abstract, able to work for any sets of months.

The First Step: The Code

```
with months as (
  select
    '2017-01-01' as startDate,
    '2017-01-31' as endDate
  union
  select
    '2017-02-01' as startDate,
    '2017-02-28' as endDate
  union
  select
    '2017-03-01' as startDate,
    '2017-03-31' as endDate
)
select *
from months;
```

To the left is the code used to create the temporary table.

Below is the output, showing the start and end dates to each month.

startDate	endDate
2017-01-01	2017-01-31
2017-02-01	2017-02-28
2017-03-01	2017-03-31

The Second Step: Cross-Joining the Two Tables

The purpose of cross-joining the two tables, subscriptions and months, is to get all of the data into a single table.

This allows for a more wholesome manipulation of the data without needing to join multiple times throughout the rest of the code.

The Second Step: The Code

```
cross_join as (
  select *
  from subscriptions
  cross join months
)
select *
from cross_join
limit 10;
```

To the left is the code creating the cross-joined table.

Below is a selection of the first 10 rows, a snapshot of the result.

id	subscription_start	subscription_end	segment	startDate	endDate
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
3	2016-12-01	2017-03-07	87	2017-03-01	2017-03-31
4	2016-12-01	2017-02-12	87	2017-01-01	2017-01-31

The Third Step: Getting the Status Table

The Status Table is made of a selection of code that narrows the data set down to the following:

- ID (id)
- Month (startDate as month)
- Cancelled in Segment 87 (isCancelled87)
- Active in Segment 87 (isActive87)
- Cancelled in Segment 30 (isCancelled30)
- Active in Segment 30 (isActive30)

The Third Step: The Code

To the right is the code required to narrow the data into its respective groups.

Below is a selection of the output.

id	month	isCancelled87	isActive87	isCancelled30	isActive30
1	2017-01-01	0	1	0	0
1	2017-02-01	1	1	0	0
1	2017-03-01	0	0	0	0
2	2017-01-01	1	1	0	0
2	2017-02-01	0	0	0	0
2	2017-03-01	0	0	0	0
3	2017-01-01	0	1	0	0
3	2017-02-01	0	1	0	0
3	2017-03-01	1	1	0	0
4	2017-01-01	0	1	0	0

```
status as (
  select
    id,
    startDate as month,
    case
      when
            (segment = 87)
        and (subscription_start < startDate)
        and (subscription_end
              between startDate
              and endDate)
      then 1
      else 0
    end as isCancelled87,
    case
      when
            (segment = 87)
        and (subscription_start < startDate)
        and ((subscription_end >= startDate)
            or (subscription_end is null))
      then 1
      else 0
    end as isActive87,
    case
      when
            (segment = 30)
        and (subscription_start < startDate)
        and (subscription_end
              between startDate
              and endDate)
      then 1
      else 0
    end as isCancelled30,
    case
      when
            (segment = 30)
        and (subscription_start < startDate)
        and ((subscription_end >= startDate)
            or (subscription_end is null))
      then 1
      else 0
    end as isActive30
  from cross_join
),
```

The Fourth Step: Aggregating To Month by Month

In this step, the active and cancelled stats are summed up, and they are separated on a month by month basis. The different segments maintain their respective columns.

The Fourth Step: The Code

```
statusAggregate as (
  select
    month,
    sum(isCancelled87) as
sumCancelled87,
    sum(isActive87) as sumActive87,
    sum(isCancelled30) as
sumCancelled30,
    sum(isActive30) as sumActive30
  from status
  group by 1
  order by 1
)
select *
from statusAggregate
limit 10;
```

The code to the right carries out the data aggregation.

The table below is the output of said groupings, with the summations of each group of individuals per segment.

month	sumCancelled87	sumActive87	sumCancelled30	sumActive30
2017-01-01	70	279	22	291
2017-02-01	148	467	38	518
2017-03-01	258	541	84	718

The Final Step: The Churn Rates

In this step two things happen, the first is that the Churn Rates are calculated in a raw amount, with there being no limit to the raw number of decimals.

The second is that the churn rates are limited to four decimal places then multiplied into percentages.

The Final Step: The Code

```
rawChurn as (
  select
    month,
    (1.0 * sumCancelled87 / sumActive87) as rawChurn87,
    (1.0 * sumCancelled30 / sumActive30) as rawChurn30
  from statusAggregate
  group by 1
  order by 1
)
select
  month as 'Month',
  (round(rawChurn87, 4) * 100) as 'Percent Churn (87)',
  (round(rawChurn30, 4) * 100) as 'Percent Churn (30)'
from rawChurn
group by 1
order by 1;
```

The code on the left executes the proper steps to calculate the percent churn per segment.

The image below shows the final table representing the data.

Month	Percent Churn (87)	Percent Churn (30)
2017-01-01	25.09	7.56
2017-02-01	31.69	7.34
2017-03-01	47.69	11.7

The Results:
The Resulting Churn

The Results: The Data

As the table below shows, the churn for each month per segment is as follows:

- January 2017: Segment 87 had a churn of 25.09%, Segment 30 had a churn of 7.56%
- February 2017: Segment 87 had a churn of 31.69%, Segment 30 had a churn of 7.34%
- March 2017: Segment 87 had a churn of 47.69%, Segment 30 had a churn of 11.7%

Month	Percent Churn (87)	Percent Churn (30)
2017-01-01	25.09	7.56
2017-02-01	31.69	7.34
2017-03-01	47.69	11.7

The Results: Analysis

The segment of users that has the most decline is Segment 87. The churn rate is abysmal, starting at about 25% and finishing at almost 50%.

Segment 30 has a much lower churn rate, hovering at about 7.5% for January and February. There was a peak of about 12% churn in March, however this was only half of the initial churn Segment 87 experienced and only a quarter of the churn experienced by Segment 87 in March.

The company should work on expanding Segment 30, as it has the higher retention rate between the two segments, by a significant margin.

Bonus:
An Unknown Number of Segments

The Changes: Status

The Status temporary table gets a change where the ID column is removed and replaced with the Segment column.

The Cancelled and Active column is now a single pair instead of a pair of columns per segment.

The Changes: Status Code

```
status as (
  select
    segment,
    startDate as month,
    case
      when
            (subscription_start < startDate)
        and (subscription_end
              between startDate
              and endDate)
      then 1
      else 0
    end as isCancelled,
    case
      when
            (subscription_start < startDate)
        and ((subscription_end >= startDate)
            or (subscription_end is null))
      then 1
      else 0
    end as isActive
  from cross_join
),
```

The code to the left shows the changes made. The logic stays the same for both isActive and isCancelled, while the id column has been replaced with segment.

The Changes: StatusAggregate

Three changes were made, they are as follows:

- Adding the segment column in front of the Month column
- The segment specific active and cancelled columns have been removed and aggregated into a single pair of columns
- Instead of only being grouped based on the month, the table is grouped based on segment and month

The Changes: StatusAggregate

```
statusAggregate as (
  select
    segment,
    month,
    sum(isCancelled) as sumCancelled,
    sum(isActive) as sumActive
  from status
  group by 1,2
  order by 1
),
```

The code to the left shows the three changes, the segment section being added, only two columns are sums, and the table is now grouped by 1,2 instead of 1.

The Changes: Churn Calculation

The only changes in this is the use of grouping by both segment and month and the actually addition of the segment column to the table. The image to the right shows both of these being implemented. Below is an actual image of the final table.

Segment	Month	Percent Churn
30	2017-01-01	7.56
30	2017-02-01	7.34
30	2017-03-01	11.7
87	2017-01-01	25.09
87	2017-02-01	31.69
87	2017-03-01	47.69

```
rawChurn as (
  select
    segment,
    month,
    (1.0 * sumCancelled / sumActive) as decChurn
  from statusAggregate
  group by 1,2
  order by 1
)
select
  segment as 'Segment',
  month as 'Month',
  (round(decChurn, 4) * 100) as 'Percent Churn'
from rawChurn
group by 1,2
order by 1;
```