Notion Data Science Take-Home Assignment

Ernest Salim - es4281@columbia.edu

This notebook serves as a PDF representation for the take-home assessment submission.

Source code can be accessed through this link.

Preemptive Assumptions

1. One customer can have multiple subscriptions (in theory), and thus, cancellation and activation are tied to subscriptions, not customers.
2. Therefore, when the question asks "How many customers are we losing" – I'd rephrase it as "How many subscriptions are we losing". In simpler terms, I used subscription as the level of granularity.
3. I saw it this way to prevent inconsistency during the analysis. For example, since a customer can have two different subscriptions (one still continuing to next month and the other one just issued cancellation), then, if I were tasked to compute the churn rate of the customer instead of a subscription, it'd be difficult and overly complex.
4. This is a fake software product, but if I can tie this to Notion, I'd see one subscription will directly correlate to one Notion workspace (since one customer can have multiple workspaces).
5. All analyses are conducted within the year 2018 only.

Setting Up

```
import sqlite3
import pandas as pd
```

```
conn = sqlite3.connect('./db.sqlite')
```

```
with open('./queries.sql', 'r') as file:
    queries = file.read()
```

```
queries = queries.split(';')
```

Query 1

How much money are we earning — what was our revenue for each month in 2018, for team vs personal plans?

```
WITH monthly_revenue AS (
  SELECT
      strftime('%m', i.period_start) AS month,
      s.plan_id,
      SUM(i.amount_due) as total
  FROM invoices i
  INNER JOIN subscriptions s
```

```
      ON i.subscription_id = s.id
  WHERE i.period_start >= '2018-01-01' AND i.period_start < '2019-01-
01'
  GROUP BY 1, 2
)

SELECT
     month,
    SUM(COALESCE(CASE WHEN plan_id = 'personal' THEN total END, 0)) AS
personal_plan_revenue,
    SUM(COALESCE(CASE WHEN plan_id = 'team' THEN total END, 0)) AS
team_plan_revenue
FROM monthly_revenue
GROUP BY 1
ORDER BY 1 ASC;
```

```
df = pd.read_sql_query(queries[0], conn)
df
```

```
   month  personal_plan_revenue  team_plan_revenue
0     01                   8000             238000
1     02                  20000             559000
2     03                  28000             940000
3     04                  41500            1343000
4     05                  62500            1730000
5     06                  77000            2398000
6     07                  92000            3140000
7     08                 104500            4004000
8     09                 125500            4989000
9     10                 150500            6110000
10    11                 177000            7193000
11    12                 209000            8170000
```

Quick Words

1. The revenue for both personal and team plan increases throughout each month. This may seem odd in a realistic context, but within the scope of this assessment, I have validated that this was the data that we're dealing with.
2. This implementation is under the assumption that customers pay their subscriptions at the beginning of the period. Therefore using `invoice.period_start` as the column for date operations.
3. Dates filtering may not cover all corner cases because I tried to not use date-related functions and tried to incorporate sargable queries as much as possible to make the query execution plan more efficient.

Query 2

How many customers are we losing — what was our customer churn rate for each month in 2018, for team vs. personal plans?

$$\text{Churn Rate} = \frac{\text{Accounts Cancelled in Month}}{\text{Accounts at Start of Month}}$$

```
WITH monthly_churns AS (
  SELECT
    strftime('%m', canceled_at) AS churned_month,
    plan_id,
  COUNT(id) AS churned_customers
  FROM subscriptions
  WHERE canceled_at != 'None' AND canceled_at >= '2018-01-01' AND
canceled_at < '2019-01-01'
  GROUP BY 1, 2
),
monthly_starting_active_accounts AS (
  SELECT
    strftime('%m', i.period_end) AS start_month,
    s.plan_id,
     COUNT(i.id) AS active_customers
  FROM invoices i
  INNER JOIN subscriptions s
     ON i.subscription_id = s.id
  WHERE i.period_end >= '2018-01-01' AND i.period_end < '2019-01-01'
  GROUP BY 1, 2
),
monthly_combined_churns AS (
  SELECT
    mc.churned_month AS month,
    mc.plan_id,
    mc.churned_customers,
    msaa.active_customers,
    COALESCE(ROUND(CAST(mc.churned_customers AS FLOAT) /
msaa.active_customers, 4), 0) AS churn_rate
  FROM monthly_churns mc
  LEFT JOIN monthly_starting_active_accounts msaa
      ON mc.churned_month = msaa.start_month AND
         mc.plan_id = msaa.plan_id
  ORDER BY 1, 2
)

-- PIVOTED FOR BETTER CLARITY
SELECT
     month,
    SUM(CASE WHEN plan_id = 'personal' THEN churn_rate END) AS
personal_plan_churn_rate,
    SUM(CASE WHEN plan_id = 'team' THEN churn_rate END) AS
```

```
team_plan_churn_rate
FROM monthly_combined_churns
GROUP BY 1;
```

```
df = pd.read_sql_query(queries[1], conn)
df
```

	month	personal_plan_churn_rate	team_plan_churn_rate
0	01	0.0000	0.0000
1	02	0.3750	0.1875
2	03	0.1750	0.2742
3	04	0.1607	0.1856
4	05	0.1084	0.1679
5	06	0.1040	0.1465
6	07	0.1429	0.1373
7	08	0.1685	0.0962
8	09	0.1292	0.1433
9	10	0.1036	0.1301
10	11	0.1462	0.1450
11	12	0.1130	0.1645

Quick Words

1. Based on initial assumption is that the denominator value for churn rate calculation is the number of active subscriptions on that month.
2. Because one subscription can only have one corresponding invoice data each month (unless there are some technical errors), then I can use `COUNT(invoice.id)` as the main metric for computing active accounts at the start at each month.
3. The churn rate data for January is 0 because we don't have previous month's data on which active accounts are still continuing through January. By using `invoice.period_end` to detect all active accounts through their invoice activities, the query should be able to generalize if we have 2017 data.

Query 3

Let's say we want to run ads for our product — what is our lifetime value (LTV) for team vs. personal plans?

$$\text{Lifetime Value} = \frac{\text{Average Monthly Revenue per Account}}{\text{Monthly Churn Rate}}$$

```
WITH monthly_arpa AS (
  SELECT
    strftime('%m', i.period_start) AS month,
    s.plan_id,
    AVG(i.amount_due) as avg_revenue
  FROM invoices i
  INNER JOIN subscriptions s
    ON i.subscription_id = s.id
```

```
  WHERE i.period_start >= '2018-01-01' AND i.period_start < '2019-01-
01'
  GROUP BY 1, 2
),
monthly_churns AS (
  SELECT
  strftime('%m', canceled_at) AS churned_month,
  plan_id,
  COUNT(id) AS churned_customers
  FROM subscriptions
  WHERE canceled_at != 'None' AND canceled_at >= '2018-01-01' AND
canceled_at < '2019-01-01'
  GROUP BY 1, 2
),
monthly_starting_active_accounts AS (
  SELECT
    strftime('%m', i.period_end) AS start_month,
    s.plan_id,
     COUNT(i.id) AS active_customers
  FROM invoices i
  INNER JOIN subscriptions s
     ON i.subscription_id = s.id
  WHERE i.period_end >= '2018-01-01' AND i.period_end < '2019-01-01'
  GROUP BY 1, 2
),
monthly_churn_rate AS (
  sELECT
    mc.churned_month AS month,
    mc.plan_id,
    COALESCE(ROUND(CAST(mc.churned_customers AS FLOAT) /
msaa.active_customers, 4), 0) AS churn_rate
  FROM monthly_churns mc
  LEFT JOIN monthly_starting_active_accounts msaa
    ON mc.churned_month = msaa.start_month AND
      mc.plan_id = msaa.plan_id
  ORDER BY 1, 2
),
monthly_ltv AS (
  SELECT
    ma.month,
    ma.plan_id,
    COALESCE(ROUND(ma.avg_revenue / mcr.churn_rate, 4), 0) AS ltv
  FROM monthly_arpa ma
  JOIN monthly_churn_rate mcr
    ON ma.month = mcr.month AND
      ma.plan_id = mcr.plan_id
  ORDER BY 1 ASC
)

-- PIVOTED FOR BETTER CLARITY
```

```
SELECT
      month,
    SUM(CASE WHEN plan_id = 'personal' THEN ltv END) AS
personal_plan_ltv,
    SUM(CASE WHEN plan_id = 'team' THEN ltv END) AS team_plan_ltv
FROM monthly_ltv
GROUP BY 1;
```

```
df = pd.read_sql_query(queries[2], conn)
df
```

	month	personal_plan_ltv	team_plan_ltv
0	01	0.0000	0.0000
1	02	1333.3333	48086.0215
2	03	2857.1429	35341.8003
3	04	3111.3877	55236.5754
4	05	4612.5461	65628.9951
5	06	4807.6923	80238.2386
6	07	3498.9503	87960.1098
7	08	2967.3591	126895.1879
8	09	3869.9690	88813.9624
9	10	4826.2548	100136.1918
10	11	3419.9726	92723.1711
11	12	4424.7788	82227.9031

Quick Words

1. Again, the January LTV was zero because of the missing January churn rate data.
2. Can be recomputed depending on the time period of interest. For e.g. quarterly or yearly.
3. Inefficient (but necessary) query as it contains multiple CTEs, in the real life setting, this could've been easily prevented by using materialized views or SQL models (dbt).

Future Action Items

1. Improve query performance.
    a. Incorporate indexing (if not yet implemented).
    b. Could have used materialized views, precomputed SQL models, or any other forms of higher level abstractions (depending on technologies and tools) to reduce redundant CTEs and queries, especially when computing LTV as we could just easily use the result from the second query.
    c. Pay attention to query execution plan to see if there's still anything that could've been optimized.
2. Tables are seemed to be normalized, leading me to believe these tables are scraped directly from production. I would try denormalizing each table first, separating them into respective facts and dimension tables.

    a. This'll not only reduce the amount joins needed to be done, but it'll also create a more intuitive and interpretable view of the data, especially for business and non-technical teams.
    b. We can create a more appropriate date dimension table to cover corner cases relating to missing dates or months that could potentially muddle the analysis.
3. Visualize analysis using BI tools or plotting libraries.
4. For LTVs, can be interpreted in a more intuitive way, such as
    a. Computing MoM change for better clarity on their trends.
    b. Tie the result to the business context. Maybe computing CAC (customer acquisition cost) or other useful metrics to give a better meaning into the LTV result.

Extra Insights (Optional)

These are some findings that I found from the database that may or may not be useful in the context of this assessment.

If anything, these insights helped in creating guardrails and assumptions during the implementation.

Customers

1. All customers are unique (1658 customers).
2. No delinquent nor deleted users.
3. There's only one unique currency (USD) and account_balance (0).

Invoices

1. One-to-many relationship with customers table.
2. All invoices are unique (5603 invoices).
3. All invoices are completed / paid by corresponding customers.
4. Each of the invoice amounts is computed correctly with respect to its respective subscription plan.
5. Days difference between period_start and period_end are perfectly computed (28 days for February, 30 or 31 days for the rest).

Subscriptions

1. All subscriptions are unique (1658 entries). Every subscription has a perfect matching with their corresponding customer.
2. Based on context and heuristics, there should be multiple subscriptions tied to a single customer, creating a one-to-many relationship.
3. There's an exclusive relationship between plan_id and plan_amount, with one plan_id value maps exclusively to a unique numeric plan_amount value.
4. Mismatched values between invoice quantity column and subscriptions quantity column.
5. Personal plans can only have one quantity in their seats.

Closing

The end of the notebook. There are still so many insights and details can be uncovered from this database alone.

Any feedback is greatly appreciated :)