BetterHelp Interview Data Challenge

Problem 1: Programming

Using any programming language, write a function that takes in each of the phone numbers found here and returns the area code of the phone number.

a. Which area code is represented the most?

907

b. How many phone numbers have the area code found in part a?

113

The most common area code was 907, which caught my attention since it's assigned to Alaska. That's not something I usually expect to see in datasets with a large US user base. It made me wonder whether the data was synthetic or if it's highlighting a real pattern, pointing out that BetterHelp has stronger reach in areas like Alaska where in-person therapy is harder to access. It'd be worth checking this pattern against a larger dataset, but it stood out as an interesting signal.

```
import pandas as pd
import re
from collections import Counter
df = pd.read_csv('Phone Numbers Challenge.csv')
df.count()
phone_column = df.columns[0]
print(f"Processing column: {phone_column}")
# Cleaning the phone numbers.
cleaned_phones = []
for phone in df[phone_column]:
   if pd.isna(phone):
       cleaned_phones.append(None)
       continue

   # Keeping only the numbers.
   phone_str = str(phone).strip()
   digits_only = re.sub(r'[^\d]', '', phone_str)

   # Removing the country code.
```

```
    if digits_only.startswith('1') and len(digits_only) == 11:
        digits_only = digits_only[1:]

    # Keeping it only if I hve 10 digits.
    if len(digits_only) == 10:
        cleaned_phones.append(digits_only)
    else:
        cleaned_phones.append(None)

df['cleaned_phone'] = cleaned_phones
df.count()
# Extracting the area codes.
area_codes = []
for phone in df['cleaned_phone']:
    if phone:
        area_codes.append(phone[:3])
area_code_counts = Counter(area_codes)
print(f"Total unique area codes: {len(area_code_counts)}")
```

```
Total unique area codes: 10
```

```
# Area codes frequency.
for area_code, count in area_code_counts.most_common():
    print(f"{area_code}: {count} times")
```

```
907: 113 times
426: 107 times
659: 102 times
516: 100 times
382: 99 times
998: 99 times
773: 98 times
877: 98 times
391: 95 times
555: 89 times
```

```
most_common = area_code_counts.most_common(1)[0]
print(f"Most frequent area code: {most_common[0]} it appears {most_common[1]}
times.")

print("Top 3 area codes:")
for i, (area_code, count) in enumerate(area_code_counts.most_common(3), 1):
```

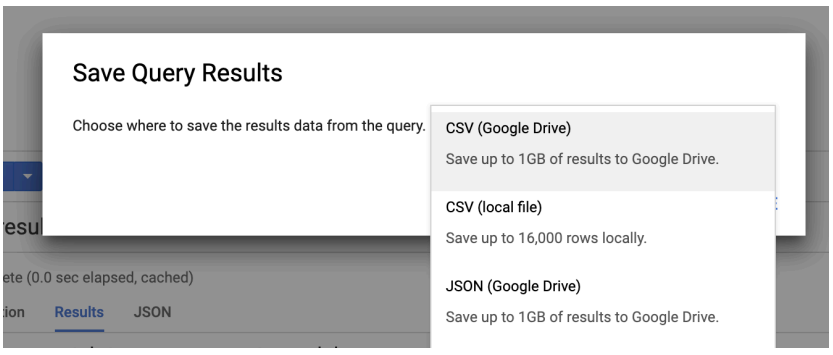
```
    print(f"{i}. {area_code}: {count} times")

print(f"Total unique area codes: {len(area_code_counts)}")
```

```
Most frequent area code: 907 it appears 113 times.
Top 3 area codes:
1. 907: 113 times
2. 426: 107 times
3. 659: 102 times
Total unique area codes: 10
```

Please show your code.

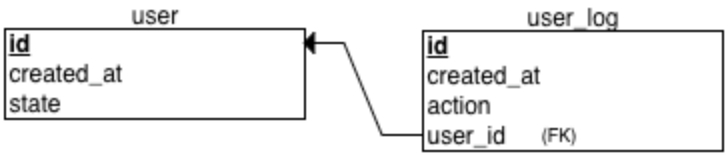
To access the data for Problems 2 and 3:

- A BigQuery project should have been shared with your Gmail account (there won't be an email noting that it's been shared). This project is called "bh-interview-data". You should have SQL access to this project.
    - Go to https://console.cloud.google.com/bigquery?project=bh-interview-data
- For problem 2, the relevant database is called 'User_Data'.
- For problem 3, the relevant database is called 'Trial_Email_Reminder'.
- To query the table, you need to include the project and database name. (for example: SELECT * FROM `bh-interview-data.User_Data.user`)
- To download data as a csv, run a query, click on 'Save Results' and select Google drive. Then go to drive.google.com to download the csv file.



Problem 2: SQL queries

There are two data tables used for this problem. Both are found in the ‘User_Data’ database. Below is the relational schema.



The **user** table has all the users (*id*) as well as the current state the user is in (*state*). The **user_log** table tracks every *action* any of the users make (i.e. “Changing state from 3 to 1!”). The *created_at* column in **user_log** table denotes when this *action* occurred.

a. Write a SQL query or use another language (e.g. Python, R) to find what percent of users have ever switched *directly* from state 1 to state 3. Please provide the percentage found as well as the code written.

Users who went 1 to 3 directly: 4,881

Total users: 9,986

Percentage who moved directly 1 to 3 out of total users: 48.9%.

The transition from state 1 to state 3 happened in 49% of all users which shows this path represents a typical user behavior pattern. The high number of users who skipped state 2 creates interesting questions about their behavior and its effect on user participation.

```
users_moved_1_to_3 AS (
  SELECT
    COUNT(DISTINCT user_id) AS users_moved_1_3
  FROM state_changes
  WHERE old_state = 1 AND new_state = 3
),

total_users AS (
  SELECT
    COUNT(DISTINCT id) AS total_user_count
  FROM `bh-interview-data.User_Data.user`
)
```

```sql
SELECT
  u13.users_moved_1_3,
  tu.total_user_count,
  ROUND((u13.users_moved_1_3 / tu.total_user_count) * 100, 1) AS percentage_moved_1_to_3
FROM users_moved_1_to_3 u13
CROSS JOIN total_users tu;
```

 # Verifying with Python

```python
import pandas as pd
import re
df = pd.read_csv('user_log.csv')
df.head()
# Counting the unique users.
unique_ids = df['user_id'].nunique()
print(f"Unique id values: {unique_ids}")
print(f"Loaded {len(df)} rows of data")
print(f"Sample of the data:")
print(df.head())
def extract_states(action_text):
    match = re.search(r'from (\d+) to (\d+)', str(action_text))
    if match:
        return int(match.group(1)), int(match.group(2))
    return None, None
df[['from_state', 'to_state']] = df['action'].apply(
    lambda x: pd.Series(extract_states(x))
)

valid_transitions = df.dropna(subset=['from_state', 'to_state'])

print(f"Found {len(valid_transitions)} valid state transitions")
# Transition counts total
transition_counts = valid_transitions.groupby(['from_state', 'to_state']).size()
print(f"Transition patterns found:")
for (from_s, to_s), count in transition_counts.items():
    print(f"  {from_s} to {to_s}: {count} times")
direct_1_to_3 = valid_transitions[
    (valid_transitions['from_state'] == 1) &
    (valid_transitions['to_state'] == 3)
]
```

```
# Users directly from 1 to 3
users_who_jumped = direct_1_to_3['user_id'].nunique()

user_df = pd.read_csv('user.csv')
total_users = user_df['id'].nunique()

if total_users > 0:
   percentage = (users_who_jumped / total_users) * 100
else:
   percentage = 0

print(f"Users who went 1 to 3 directly: {users_who_jumped}")
print(f"Total users: {total_users}")
print(f"Percentage who moved directly 1 to 3 out of total users:
{percentage:.1f}%")
```

```
Users who went 1 to 3 directly: 4881
Total users: 9986
Percentage who moved directly 1 to 3 out of total users: 48.9%
```

b. Write a SQL query or use another language (e.g. Python, R) that creates the following table that counts the number of users that start in state “x” and have current state “y”. Below is what the table should look like. Please provide the code and the table it produces.

First State	Current State	Number of Users
1	1	996
1	2	603
1	3	596
2	1	1,157
2	2	1,605
2	3	1,154
3	1	1,179
3	2	1,144
3	3	1,552
Total		9,986

I wanted to understand which state users transitioned to after their initial recorded state based on their earliest state change log. The data showed that users who started in state 1 had a 54.6% chance of moving to either state 2 or 3. Users who started in state 3 had the highest retention, with around 45% still in that state. The flow from state 2 was fairly balanced, with users splitting almost evenly between state 1 and 3, while 41% stayed in the state.

```
WITH user_first_states AS (
 SELECT
   user_id,
   CAST(REGEXP_EXTRACT(action, 'to (\\d+)') AS INT64) AS first_state,
   created_at
 FROM (
   SELECT
     user_id,
     action,
     created_at,
     ROW_NUMBER() OVER (PARTITION BY user_id ORDER BY created_at ASC) as firstent
   FROM `bh-interview-data.User_Data.user_log`
   WHERE REGEXP_CONTAINS(action, 'from \\d+ to \\d+')
 )
 WHERE firstent = 1
),

user_now_states AS (
 SELECT
   id as user_id,
   state as current_state
 FROM `bh-interview-data.User_Data.user`
)

SELECT
 COALESCE(f.first_state, n.current_state) AS first_state,
 n.current_state,
 COUNT(*) as num_users
FROM user_now_states n
LEFT JOIN user_first_states f ON f.user_id = n.user_id
GROUP BY COALESCE(f.first_state, n.current_state), n.current_state
ORDER BY COALESCE(f.first_state, n.current_state), n.current_state;
```

 # Verifying with Python

```
log_df = pd.read_csv('user_log.csv')
user_df = pd.read_csv('user.csv')
log_df.head()
log_df.count()
unique_ids = log_df['user_id'].nunique()
print(f"Unique id values: {unique_ids}")
user_df.head()
user_df.count()
unique_ids = user_df['id'].nunique()
print(f"Unique id values: {unique_ids}")
print(f"Loaded {len(log_df)} log entries and {len(user_df)} users")
```

```
Loaded 29937 log entries and 9986 users
```

```
# Finding each user's first state.
def extract_to_state(action_text):
   match = re.search(r'to (\d+)', str(action_text))
   return int(match.group(1)) if match else None

# State changes.
state_changes = log_df[log_df['action'].str.contains('from.*to', na=False)].copy()
state_changes['to_state'] = state_changes['action'].apply(extract_to_state)

state_changes = state_changes.dropna(subset=['to_state'])

# Timestamp.
state_changes['created_at'] = pd.to_datetime(state_changes['created_at'])

# First state for each user.
first_states_from_log = (state_changes
                         .sort_values('created_at')
                         .groupby('user_id')
                         .first()
                         .reset_index()[['user_id', 'to_state']])
first_states_from_log.rename(columns={'to_state': 'first_state_from_log'},
inplace=True)

print(f"Found first states from log for {len(first_states_from_log)} users")
```

```
Found first states from log for 9512 users
```

```
# Current states
current_states = user_df[['id', 'state']].copy()
```

```
current_states.rename(columns={'id': 'user_id', 'state': 'current_state'},
inplace=True)

print(f"Found current states for {len(current_states)} users")
```

```
Found current states for 9986 users
```

```
combined = current_states.merge(first_states_from_log, on='user_id', how='left')

# For users where the first state is null, I use the current state as the first
state.
combined['first_state'] =
combined['first_state_from_log'].fillna(combined['current_state'])

# Integer.
combined['first_state'] = combined['first_state'].astype(int)

combined = combined.drop('first_state_from_log', axis=1)

print(f"Analysis Summary:")
print(f"Total users in analysis: {len(combined)}")
print(f"Users with log entries: {len(first_states_from_log)}")
print(f"Users without log entries, where I'm using the current state as the first
state: {len(combined) - len(first_states_from_log)}")
```

```
Analysis Summary:
Total users in analysis: 9986
Users with log entries: 9512
Users without log entries, where I'm using the current state as the first state: 474
```

```
result_table = (combined
                .groupby(['first_state', 'current_state'])
                .size()
                .reset_index(name='num_users')
                .sort_values(['first_state', 'current_state']))

print(f"First State and Current State Table:")
print(result_table.to_string(index=False))
```

```
First State and Current State Table:
 first_state  current_state  num_users
           1              1        996
           1              2        603
           1              3        596
           2              1       1157
           2              2       1605
           2              3       1154
           3              1       1179
```

```
           3               2       1144
           3               3       1552
```

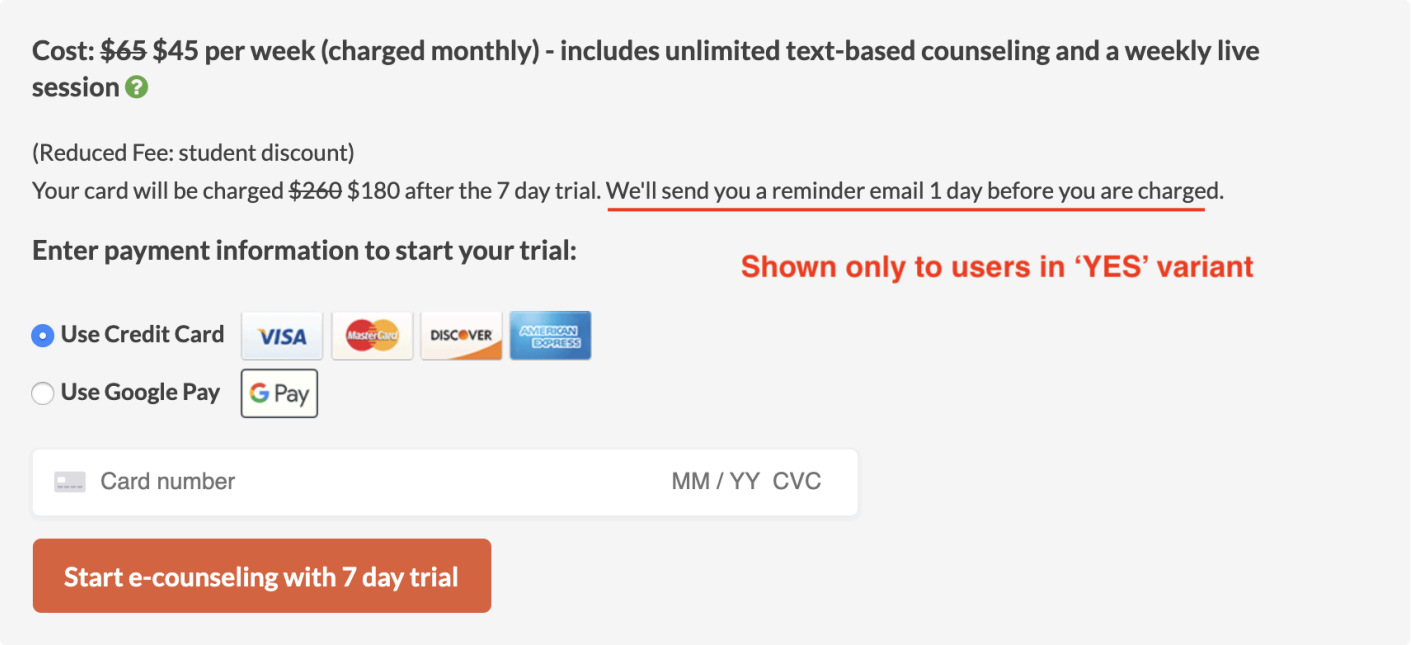
```
total_users_analysis = len(combined)
stayed_same = combined[combined['first_state'] == combined['current_state']]
changed_state = combined[combined['first_state'] != combined['current_state']]

print(f"Summary:")
print(f"Total users: {total_users_analysis}")
print(f"Users who stayed in same state: {len(stayed_same)}
({len(stayed_same)/total_users_analysis*100:.1f}%)")
print(f"Users who changed state: {len(changed_state)}
({len(changed_state)/total_users_analysis*100:.1f}%)")
```

```
Summary:
Total users: 9986
Users who stayed in same state: 4153 (41.6%)
Users who changed state: 5833 (58.4%)
```

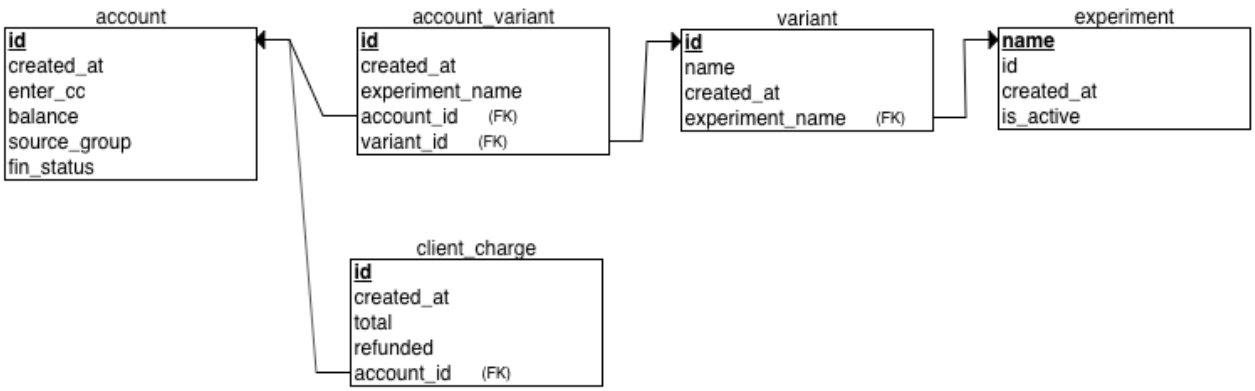
Problem 3: Experiment Challenge

We recently ran an experiment on BetterHelp users who had a 7-day trial. In one variant (the “YES” variant), we would send the users an email 24 hours before their trial expired, reminding them that the trial is about to end. In the other variant (the “NO” variant), the users did not receive this email. We randomly assigned the users to the two variants when they signed up (i.e created an account).

You can sign up at www.betterhelp.com/trial to see the onboarding flow. The screenshot below shows how the experience differs for users in the ‘YES’ variant.



There are five data tables used for this problem. All are found in the 'Trial_Email_Reminder" database. Below is the relational schema.



Analyze the data and answer the following question:

- What are your recommendations for whether we should have a trial ending reminder and why?

I'd recommend using the reminder email. Even though the conversion rate dropped slightly from 83.4% to 78.8%, the users who converted after receiving the reminder were more intentional. This showed in the refund rates since they dropped from 9.7% to 3.2%, and the average revenue per user increased to $304.

What stood out to me is that this wasn't just a conversion rate drop, it highlighted an important shift in user quality. The email seems to create a moment for users to reconsider, which filters out less committed users. The users who continue after the email are likely more engaged, stayed longer and were more valuable overall.

I estimated net revenue by factoring in both the conversion counts and the refund rates.

- No email: 14,185 users * $283.24 average per user * 90.27% success rate = 3.62 million dollars.

- Yes email: 13,590 users * $303.94 average per user * 96.77% success rate = 3.99 million dollars.

The group that received the email had around 595 fewer conversions, but brought in around $367,476 more in net revenue, which is about a 10% increase. On top of that, engagement metrics like average and median tenure and number of charges per user all improved. Users who received the email stayed longer on average, with 18 days compared to 13 days, and had more charges per user. The median user duration increased from 0 to 27 days, indicating a stronger level of commitment and retention. This was true regardless of either acquisition channel or financial status.

BetterHelp should implement the reminder email to every user that signs-up for a trial. I will also suggest a performance dashboard and running surveys to validate that the email reminder is not stopping users who are ready to convert. Tracking lifetime value over time should show whether the email reminder is creating stronger client relationships.

- Briefly describe your analysis method and list any additional insights and assumptions.

The reminder email led to a slight drop in conversions but brought in higher quality users who refunded less, spent more and remained active for longer. There was a higher net revenue in the reminder email group and this effect held across all user segments.

I approached this A/B test by focusing on three key business metrics: conversion rate, refund rate and revenue per user. I started by comparing the three metrics across the trial email groups to understand the impact of the email reminder. I calculated the confidence

intervals of the number of conversions to see if the slight drop was statistically significant by finding the standard error directly in SQL.

Then I took a look at user behavior after the conversion. I looked at average and median revenue per user, customer tenure and charge frequency to get a clearer sense of engagement and long-term value. The email reminder group had a higher average revenue per user of $303.9 and a much lower refund rate of 3.2%, this translated into a higher net revenue.

I also looked at engagement over time. Users who received the reminder had a longer average tenure of 18 days compared to 13, and their median tenure jumped from 0 to 27 days. They were also charged more frequently, with a 14% increase in the average number of charges per user. These patterns suggest the email reminder helped filter in more committed users. These gains could be especially valuable if BetterHelp is focusing on retention strategies, particularly in cost-sensitive segments.

Note: To make the best use of your time, we recommend focusing on the top metric(s) for this decision, accurate analysis and clear/easy to follow recommendations.

```
 # A/B test metrics.

 # Conversion rate with confidence intervals.

WITH conversion_stats AS (

   SELECT

       v.name as variant,

       COUNT(*) as total_users,

       SUM(CASE WHEN a.balance > 0 THEN 1 ELSE 0 END) as converted_users,

       AVG(CASE WHEN a.balance > 0 THEN 1.0 ELSE 0.0 END) as conversion_rate,

       SQRT(AVG(CASE WHEN a.balance > 0 THEN 1.0 ELSE 0.0 END) *

            (1 - AVG(CASE WHEN a.balance > 0 THEN 1.0 ELSE 0.0 END)) / COUNT(*)) as
conversion_se

   FROM `bh-interview-data.Trial_Email_Reminder.account` a

   JOIN `bh-interview-data.Trial_Email_Reminder.account_variant` av ON a.id = av.account_id

   JOIN `bh-interview-data.Trial_Email_Reminder.variant` v ON av.variant_id = v.id
```

```
    WHERE a.enter_cc = 1
    GROUP BY v.name
)
SELECT
    variant,
    total_users,
    converted_users,
    ROUND(conversion_rate * 100, 2) as conversion_rate_pct,
    ROUND((conversion_rate - 1.96 * conversion_se) * 100, 2) as conversion_rate_ci_lower,
    ROUND((conversion_rate + 1.96 * conversion_se) * 100, 2) as conversion_rate_ci_upper
FROM conversion_stats
ORDER BY variant;
```

```
 # Conversion rate
 SELECT
    v.name as variant,
    COUNT(DISTINCT a.id) as total_users,
    COUNT(DISTINCT CASE WHEN a.enter_cc = 1 AND a.balance > 0 THEN a.id END) as converted_users,
    ROUND(COUNT(DISTINCT CASE WHEN a.enter_cc = 1 AND a.balance > 0 THEN a.id END) * 100.0 /
          COUNT(DISTINCT a.id), 2) as conversion_rate_pct
 FROM `bh-interview-data.Trial_Email_Reminder.account` a
 JOIN `bh-interview-data.Trial_Email_Reminder.account_variant` av ON a.id = av.account_id
 JOIN `bh-interview-data.Trial_Email_Reminder.variant` v ON av.variant_id = v.id
 WHERE a.enter_cc = 1
 GROUP BY v.name;
```

```
# Revenue per user by trial.

WITH revenue_data AS (

   SELECT

       v.name as variant,

       a.id as account_id,

       COALESCE(SUM(CASE WHEN cc.refunded = 0 THEN cc.total ELSE 0 END), 0) as
net_revenue_cents

   FROM `bh-interview-data.Trial_Email_Reminder.account` a

   JOIN `bh-interview-data.Trial_Email_Reminder.account_variant` av ON a.id = av.account_id

   JOIN `bh-interview-data.Trial_Email_Reminder.variant` v ON av.variant_id = v.id

   LEFT JOIN `bh-interview-data.Trial_Email_Reminder.client_charge` cc ON a.id = cc.account_id

   WHERE a.enter_cc = 1

   GROUP BY v.name, a.id

)

SELECT

   variant,

   ROUND(AVG(net_revenue_cents) / 100.0, 2) as avg_revenue_per_user_dollars,

   ROUND(APPROX_QUANTILES(net_revenue_cents, 100)[OFFSET(50)] / 100.0, 2) as
median_revenue_per_user_dollars

FROM revenue_data

GROUP BY variant;

# Refund rate by trial.

SELECT

   v.name as variant,

   COUNT(DISTINCT cc.id) as total_charges,
```

```
    COUNT(DISTINCT CASE WHEN cc.refunded = 1 THEN cc.id END) as refunded_charges,
    ROUND(COUNT(DISTINCT CASE WHEN cc.refunded = 1 THEN cc.id END) * 100.0 /
          COUNT(DISTINCT cc.id), 2) as refund_rate_pct
FROM `bh-interview-data.Trial_Email_Reminder.account` a
JOIN `bh-interview-data.Trial_Email_Reminder.account_variant` av ON a.id = av.account_id
JOIN `bh-interview-data.Trial_Email_Reminder.variant` v ON av.variant_id = v.id
JOIN `bh-interview-data.Trial_Email_Reminder.client_charge` cc ON a.id = cc.account_id
WHERE a.enter_cc = 1
GROUP BY v.name;
# User behavior metrics.
# User tenure and charges per converted user.
WITH user_charge_summary AS (
  SELECT
      a.id,
      v.name as variant,
      COUNT(cc.id) as total_charges,
      SUM(CASE WHEN cc.refunded = 0 THEN cc.total ELSE 0 END) as net_revenue_cents,
      MAX(cc.created_at) as last_charge_date,
      MIN(cc.created_at) as first_charge_date,
      DATE_DIFF(MAX(cc.created_at), MIN(cc.created_at), DAY) as customer_tenure_days
  FROM `bh-interview-data.Trial_Email_Reminder.account` a
  JOIN `bh-interview-data.Trial_Email_Reminder.account_variant` av ON a.id = av.account_id
  JOIN `bh-interview-data.Trial_Email_Reminder.variant` v ON av.variant_id = v.id
  LEFT JOIN `bh-interview-data.Trial_Email_Reminder.client_charge` cc ON a.id = cc.account_id
```

```
    WHERE a.enter_cc = 1 AND a.balance > 0
    GROUP BY a.id, v.name
)
SELECT
    variant,
    ROUND(AVG(total_charges), 2) as avg_charges_per_user,
    ROUND(AVG(net_revenue_cents) / 100.0, 2) as avg_net_revenue_dollars,
    ROUND(AVG(customer_tenure_days), 1) as avg_tenure_days,
    ROUND(APPROX_QUANTILES(customer_tenure_days, 100)[OFFSET(50)], 1) as median_tenure_days
FROM user_charge_summary
GROUP BY variant;
```

 # Days to conversion and average charge amount.

```
SELECT
    v.name as variant,
    DATE_DIFF(cc.created_at, a.created_at, DAY) as days_to_conversion,
    COUNT(*) as users,
    AVG(cc.total) / 100.0 as avg_charge_amount_dollars
FROM `bh-interview-data.Trial_Email_Reminder.account` a
JOIN `bh-interview-data.Trial_Email_Reminder.account_variant` av ON a.id = av.account_id
JOIN `bh-interview-data.Trial_Email_Reminder.variant` v ON av.variant_id = v.id
JOIN `bh-interview-data.Trial_Email_Reminder.client_charge` cc ON a.id = cc.account_id
WHERE a.enter_cc = 1 AND cc.refunded = 0
GROUP BY v.name, days_to_conversion
ORDER BY days_to_conversion;
```

 # Segmentation.

```sql
# Performance by acquisition chanel.

WITH source_performance AS (
   SELECT
       v.name as variant,
       a.source_group,
       COUNT(*) as users,
       AVG(CASE WHEN a.balance > 0 THEN 1.0 ELSE 0.0 END) as conversion_rate,
       AVG(CASE WHEN cc.refunded = 1 THEN 1.0 ELSE 0.0 END) as refund_rate,
       AVG(COALESCE(cc.total, 0)) / 100.0 as avg_revenue_dollars
   FROM `bh-interview-data.Trial_Email_Reminder.account` a
   JOIN `bh-interview-data.Trial_Email_Reminder.account_variant` av ON a.id = av.account_id
   JOIN `bh-interview-data.Trial_Email_Reminder.variant` v ON av.variant_id = v.id
   LEFT JOIN `bh-interview-data.Trial_Email_Reminder.client_charge` cc ON a.id = cc.account_id
   WHERE a.enter_cc = 1
   GROUP BY v.name, a.source_group
)
SELECT
   variant,
   source_group,
   users,
   ROUND(conversion_rate * 100, 2) as conversion_rate_pct,
   ROUND(refund_rate * 100, 2) as refund_rate_pct,
   ROUND(avg_revenue_dollars, 2) as avg_revenue_dollars
FROM source_performance
```

```sql
WHERE users >= 100

ORDER BY source_group, variant;

 # Performance by financial status.

SELECT

   v.name as variant,

   a.fin_status,

   COUNT(*) as total_users,

   ROUND(AVG(CASE WHEN a.balance > 0 THEN 1.0 ELSE 0.0 END) * 100, 2) as conversion_rate_pct,

   ROUND(AVG(COALESCE(cc.total, 0)) / 100.0, 2) as avg_revenue_dollars,

   ROUND(AVG(CASE WHEN cc.refunded = 1 THEN 1.0 ELSE 0.0 END) * 100, 2) as refund_rate_pct

FROM `bh-interview-data.Trial_Email_Reminder.account` a

JOIN `bh-interview-data.Trial_Email_Reminder.account_variant` av ON a.id = av.account_id

JOIN `bh-interview-data.Trial_Email_Reminder.variant` v ON av.variant_id = v.id

LEFT JOIN `bh-interview-data.Trial_Email_Reminder.client_charge` cc ON a.id = cc.account_id

WHERE a.enter_cc = 1

GROUP BY v.name, a.fin_status

ORDER BY a.fin_status, v.name;

# Cross-segment performance by source and financial status.

SELECT

   v.name as variant,

   a.source_group,

   a.fin_status,

   COUNT(*) as users,

   ROUND(AVG(CASE WHEN a.balance > 0 THEN 1.0 ELSE 0.0 END) * 100, 2) as conversion_rate_pct,
```

```
    ROUND(AVG(CASE WHEN cc.refunded = 1 THEN 1.0 ELSE 0.0 END) * 100, 2) as refund_rate_pct,
    ROUND(AVG(COALESCE(cc.total, 0)) / 100.0, 2) as avg_charge_amount_dollars,
    ROUND(STDDEV(COALESCE(cc.total, 0)) / 100.0, 2) as charge_amount_stddev_dollars
FROM `bh-interview-data.Trial_Email_Reminder.account` a
JOIN `bh-interview-data.Trial_Email_Reminder.account_variant` av ON a.id = av.account_id
JOIN `bh-interview-data.Trial_Email_Reminder.variant` v ON av.variant_id = v.id
LEFT JOIN `bh-interview-data.Trial_Email_Reminder.client_charge` cc ON a.id = cc.account_id
WHERE a.enter_cc = 1
GROUP BY v.name, a.source_group, a.fin_status
HAVING COUNT(*) >= 50
ORDER BY a.source_group, a.fin_status, v.name;
```

Data Description

Table	Column	Description
account	*id*	This is the user's account id. Each user only has one account.
account	*created_at*	This is the time when the user signed up for BetterHelp ("sign up" = entered their email)
account	*enter_cc*	1 = "entered credit card" ; 0 = "did not enter credit card" **Important:** If a user entered their credit card, then they started the 7-day trial.
account	*balance*	How much money the user has paid in **cents**. **Important:** If the user has any balance, this means that they converted to being a paying user. In other words, if the balance is 0, the user either canceled their trial before the trial period or was refunded after.
account	*source_group*	Which source we credit for bringing the user to BetterHelp (Facebook, SEO, podcast, etc)

account	*fin_status*	Financial status of the user (Good, Fair, Poor)
variant	*id*	This is the id for which variant a user is in
variant	*name*	“YES” or “NO” for if they received an email reminder that their trial is ending
account_variant	*variant_id/ account_id*	This table shows which variant (variant_id) the user (account_id) is in
experiment	*id/name*	The experiment id and name of this experiment.
client_charge	*refunded*	**Important:** 1 = “client was refunded”, 0 = “client was charged”
client_charge	*total*	The amount the client was charged for. If refunded=1, it means this charge was refunded.

Problem 4: Feedback

Please let us know how long the challenge took and if you have feedback or suggestions to make it better. Thanks!

The full challenge took me around 6 hours from start to finish, including time spent verifying my answers in Python. I particularly liked how practical the questions were. This was exactly the kind of analysis that I would be expected to do on the job and having those types of questions definitely made the challenge engaging as well as challenging.

Some of the questions were slightly vague in section 2. In the first question, I was not completely sure if the percentage of users who transitioned from state 1 to state 3 should be calculated from all users or only users who had any state transitions. In the second question, I doubted if users with incomplete logs should be discarded. I'm aware that this may be the intention of the exercise to see how candidates deal with the uncertainty of the problem statement. I think it would be beneficial to add some clarifications regarding the handling of the edge cases to allow the candidate to focus more on the analysis itself.

I enjoyed the challenge and I like that it pushed me to think, dig deeper into the data, and look at it from different perspectives, which I believe are the type of skills that are desired for this type of role. Overall, it was a great balance of exploratory and applied questions. Thank you very much for the opportunity to solve it.