Task 4: using SQL (later on in the dashboard I will reimplement these queries in pandas)

**Conversation volume analysis:**

1. **Messages per Day:**

SELECT

DATE(created\_at) AS day,

COUNT(\*) AS message\_count

FROM

chats

GROUP BY

DATE(created\_at)

ORDER BY

day;

1. **Messages per Week:**

SELECT

YEAR(created\_at) AS year,

WEEK(created\_at) AS week,

COUNT(\*) AS message\_count

FROM

chats

GROUP BY

YEAR(created\_at), WEEK(created\_at)

ORDER BY

year, week;

1. **Message per topic:**

SELECT

CASE

WHEN question LIKE '%flight%' OR question LIKE '%arrival%' OR question LIKE '%departure%' THEN 'Flight Information'

WHEN question LIKE '%menu%' OR question LIKE '%food%' OR question LIKE '%restaurant%' THEN 'Restaurant'

WHEN question LIKE '%hours%' OR question LIKE '%open%' THEN 'Operating Hours'

ELSE 'Other'

END AS topic,

COUNT(\*) AS message\_count

FROM

chats

GROUP BY

topic

ORDER BY

message\_count DESC;

1. **Message per user:**

SELECT

client\_id,

COUNT(\*) AS message\_count

FROM

chats

GROUP BY

client\_id

ORDER BY

message\_count DESC;

**User Engagement Analysis :**

1. **Most Active Users**

SELECT

c.client\_id,

cl.name AS client\_name,

COUNT(\*) AS message\_count,

COUNT(DISTINCT DATE(c.created\_at)) AS active\_days

FROM

chats c

JOIN

clients cl ON c.client\_id = cl.client\_id

GROUP BY

c.client\_id, cl.name

ORDER BY

message\_count DESC;

1. **Average Session Length (assuming a session is a sequence of messages from the same user within 30 minutes)**

WITH sessionized\_chats AS (

SELECT

client\_id,

created\_at,

TIMESTAMPDIFF(MINUTE, LAG(created\_at) OVER (PARTITION BY client\_id ORDER BY created\_at), created\_at) AS time\_diff

FROM chats

),

sessions AS (

SELECT

client\_id,

created\_at, -- ← include this!

SUM(CASE WHEN time\_diff IS NULL OR time\_diff > 30 THEN 1 ELSE 0 END) OVER (PARTITION BY client\_id ORDER BY created\_at) AS session\_id

FROM sessionized\_chats

)

SELECT

AVG(session\_length) AS avg\_session\_length\_minutes

FROM (

SELECT

client\_id,

session\_id,

TIMESTAMPDIFF(MINUTE, MIN(created\_at), MAX(created\_at)) AS session\_length

FROM

sessions

JOIN chats USING (client\_id, created\_at)

GROUP BY

client\_id, session\_id

) AS session\_lengths;

1. **Drop-off Rates (percentage of conversations that end after bot response)**

WITH conversation\_ends AS (

SELECT

client\_id,

created\_at,

LEAD(created\_at) OVER (PARTITION BY client\_id ORDER BY created\_at) AS next\_message\_time,

CASE

WHEN question IS NOT NULL THEN 'user'

WHEN answer IS NOT NULL THEN 'bot'

END AS sender

FROM

chats

),

dropoffs AS (

SELECT

COUNT(\*) AS total\_bot\_messages,

SUM(CASE WHEN next\_message\_time IS NULL AND sender = 'bot' THEN 1 ELSE 0 END) AS drop\_after\_bot

FROM

conversation\_ends

)

SELECT

(drop\_after\_bot / total\_bot\_messages) \* 100 AS drop\_off\_rate\_percentage

FROM

dropoffs;

**Topic performance:**

**1-Most frequently used topics and satisfaction scores:**

SELECT

CASE

WHEN question LIKE '%flight%' OR question LIKE '%arrival%' OR question LIKE '%departure%' THEN 'Flight Information'

WHEN question LIKE '%menu%' OR question LIKE '%food%' OR question LIKE '%restaurant%' THEN 'Restaurant'

WHEN question LIKE '%hours%' OR question LIKE '%open%' THEN 'Operating Hours'

ELSE 'Other'

END AS topic,

COUNT(\*) AS frequency,

ROUND(COUNT(\*) \* 100.0 / (SELECT COUNT(\*) FROM chats), 2) AS percentage

FROM

chats

GROUP BY

topic

ORDER BY

frequency DESC;

Satisfaction Scores (assuming satisfaction can be inferred from conversation length or follow-up questions)

WITH topic\_conversations AS (

SELECT

client\_id,

CASE

WHEN question LIKE '%flight%' OR question LIKE '%arrival%' OR question LIKE '%departure%' THEN 'Flight Information'

WHEN question LIKE '%menu%' OR question LIKE '%food%' OR question LIKE '%restaurant%' THEN 'Restaurant'

WHEN question LIKE '%hours%' OR question LIKE '%open%' THEN 'Operating Hours'

ELSE 'Other'

END AS topic,

COUNT(\*) AS message\_count

FROM

chats

GROUP BY

client\_id, topic

)

SELECT

topic,

AVG(message\_count) AS avg\_messages\_per\_conversation,

COUNT(DISTINCT client\_id) AS unique\_users

FROM

topic\_conversations

GROUP BY

topic

ORDER BY

avg\_messages\_per\_conversation;

**Response Efficiency**

1. **Average Response Time**

WITH response\_times AS (

SELECT

c1.client\_id,

c1.created\_at AS user\_message\_time,

MIN(c2.created\_at) AS bot\_response\_time,

TIMESTAMPDIFF(SECOND, c1.created\_at, MIN(c2.created\_at)) AS response\_time\_seconds

FROM

chats c1

JOIN

chats c2 ON c1.client\_id = c2.client\_id

AND c2.created\_at > c1.created\_at

AND c2.answer IS NOT NULL

WHERE

c1.question IS NOT NULL

GROUP BY

c1.client\_id, c1.created\_at

)

SELECT

AVG(response\_time\_seconds) AS avg\_response\_time\_seconds,

AVG(response\_time\_seconds)/60 AS avg\_response\_time\_minutes

FROM

response\_times;

1. **Number of Turns per Conversation**

WITH conversation\_turns AS (

SELECT

client\_id,

COUNT(\*) AS total\_messages,

SUM(CASE WHEN question IS NOT NULL THEN 1 ELSE 0 END) AS user\_messages,

SUM(CASE WHEN answer IS NOT NULL THEN 1 ELSE 0 END) AS bot\_messages

FROM

chats

GROUP BY

client\_id

)

SELECT

AVG(total\_messages) AS avg\_turns\_per\_conversation,

AVG(user\_messages) AS avg\_user\_messages,

AVG(bot\_messages) AS avg\_bot\_messages

FROM

conversation\_turns;