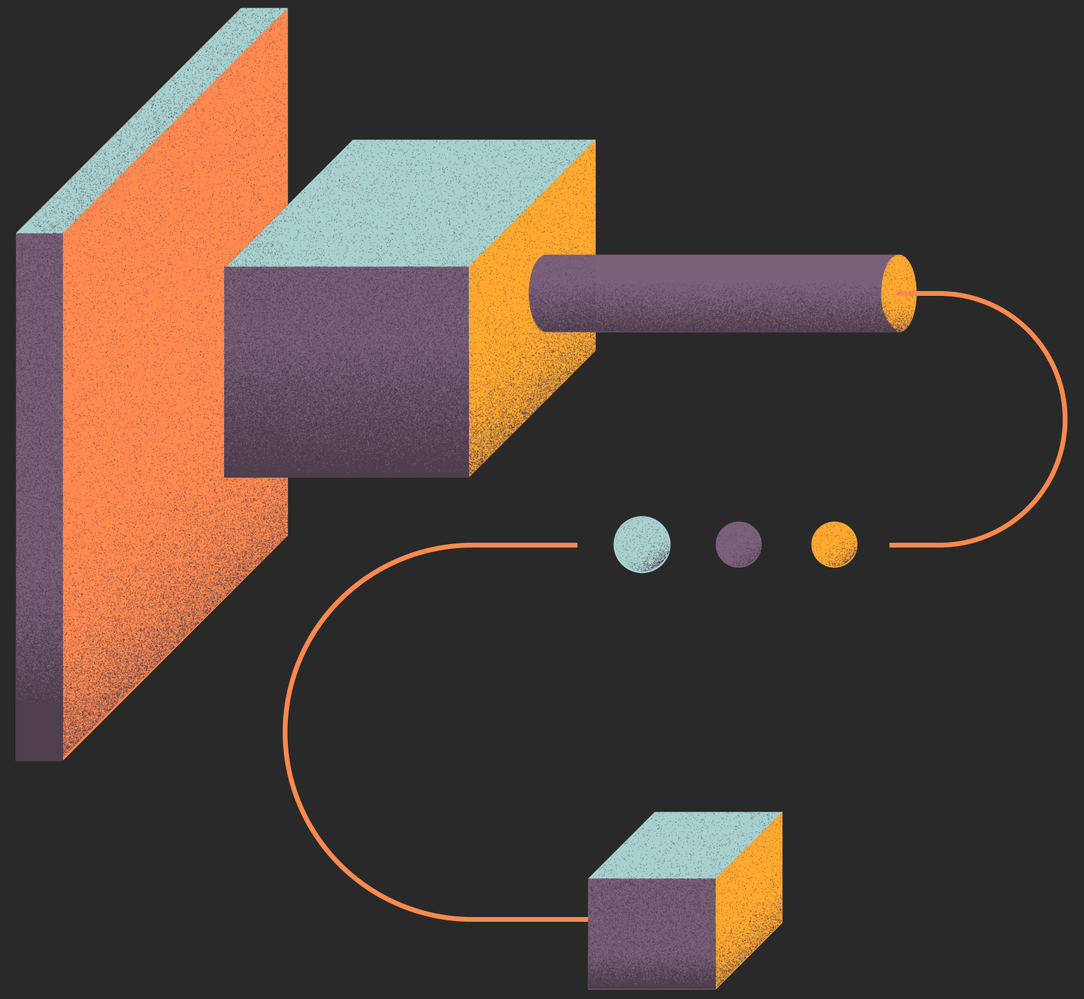
OPERATION ANALYTICS AND INVESTIGATING METRIC SPIKE

A SQL-Based Data Analysis Project

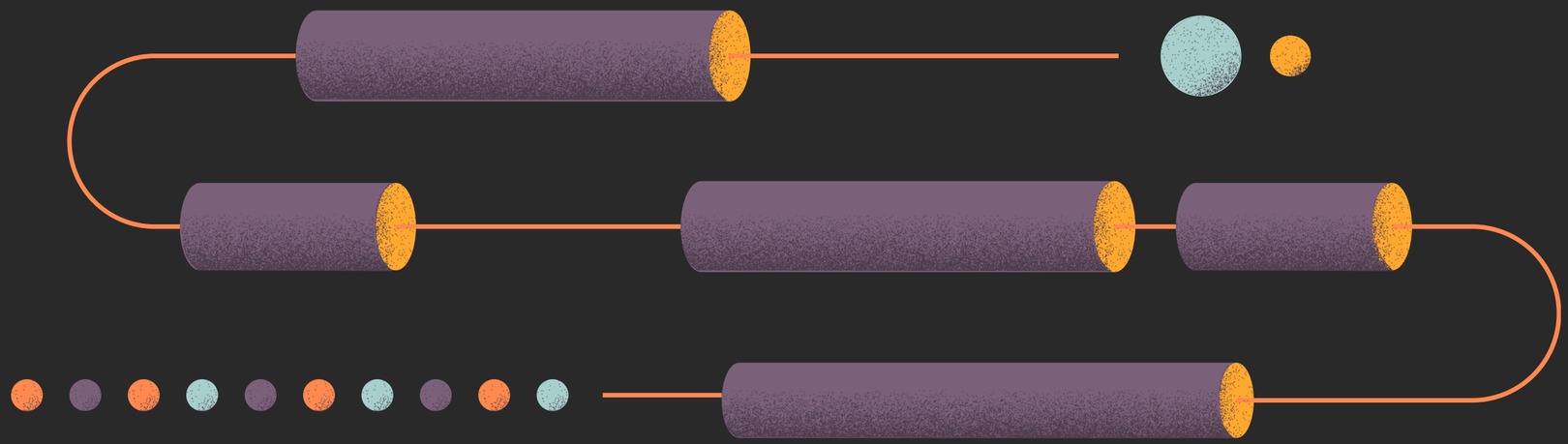


By Rajeev Kumar

INTRODUCTION

Sometimes, numbers in reports suddenly go up or down without any clear reason. This project is all about finding out why! Using SQL, we analyze data to **track unusual changes** in key metrics and understand what caused them.

By writing smart queries, we pull the right data, look for patterns, and figure out if the spike happened due to system issues, business trends, or something else. This helps businesses stay on top of their **operations**, **fix problems** faster, and make **better decisions** based on real data. 🚀



CASE STUDY -1

JOB DATA ANALYSIS

- **job_id:** Unique identifier of jobs
- **actor_id:** Unique identifier of actor
- **event:** The type of event (decision/skip/transfer).
- **language:** The Language of the content
- **time_spent:** Time spent to review the job in seconds.
- **org:** The Organization of the actor
- **ds:** The date in the format yyyy/mm/dd (stored as text).

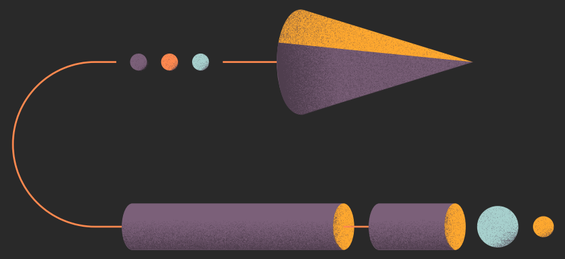
SWIPE TO CHECK THE TASK & SOLUTIONS

PROJECT SETUP

```sql
create database project_3_part_1;
use project_3_part_1;

CREATE TABLE job_data (
    ds DATE,
    job_id INT,
    actor_id INT,
    event VARCHAR(20),
    language VARCHAR(20),
    time_spent INT,
    org CHAR(10)
);
```

```sql
INSERT INTO job_data (ds, job_id, actor_id, event, language, time_spent, org)
VALUES
('2020-11-30', 21, 1001, 'skip', 'English', 15, 'A'),
('2020-11-30', 22, 1006, 'transfer', 'Arabic', 25, 'B'),
('2020-11-29', 23, 1003, 'decision', 'Persian', 20, 'C'),
('2020-11-28', 23, 1005, 'transfer', 'Persian', 22, 'D'),
('2020-11-28', 25, 1002, 'decision', 'Hindi', 11, 'B'),
('2020-11-27', 11, 1007, 'decision', 'French', 104, 'D'),
('2020-11-26', 23, 1004, 'skip', 'Persian', 56, 'A'),
('2020-11-25', 20, 1003, 'transfer', 'Italian', 45, 'C');
```



A. WRITE AN SQL QUERY TO CALCULATE THE NUMBER OF JOBS REVIEWED PER HOUR FOR EACH DAY IN NOVEMBER 2020.

SOLUTION

```sql
SELECT
    ds AS review_date,
    COUNT(job_id) AS total_jobs_reviewed,
    SUM(time_spent) / 3600 AS total_time_hours,
    COUNT(job_id) / NULLIF(SUM(time_spent) / 3600, 0) AS Job_reviewd_pr_hr
FROM
    job_data
WHERE
    ds BETWEEN '2020-11-01' AND '2020-11-30'
GROUP BY ds
ORDER BY ds;
```

RESULTS

review_date	total_jobs_reviewed	total_time_hours	Job_reviewd_pr_hr
2020-11-25	1	0.0125	80.0000
2020-11-26	1	0.0156	64.2857
2020-11-27	1	0.0289	34.6154
2020-11-28	2	0.0092	218.1818
2020-11-29	1	0.0056	180.0000
2020-11-30	2	0.0111	180.0000

EXPLAINATION

- First, I select the **date (ds)** and rename it as **review_date** so it's easy to read.
- Then, I count the total number of jobs reviewed each day using **COUNT(job_id)**.
- Next, I calculate the total time spent in hours by dividing **SUM(time_spent)** by 3600.
- To find jobs reviewed per hour, I divide total jobs by total hours and use **NULLIF** to avoid division errors.
- I filter the data to include only **November 2020** and group everything by date.
- Finally, I sort the results by date so the data appears in order.

B. WRITE AN SQL QUERY TO CALCULATE THE 7-DAY ROLLING AVERAGE OF THROUGHPUT. ADDITIONALLY, EXPLAIN WHETHER YOU PREFER USING THE DAILY METRIC OR THE 7-DAY ROLLING AVERAGE FOR THROUGHPUT, AND WHY

SOLUTION

```sql
WITH daily_throughput AS (
    SELECT
        ds AS date,
        COUNT(event) AS total_events,
        SUM(time_spent) AS total_time_spent,
        COUNT(event) / NULLIF(SUM(time_spent), 0) AS daily_throughput
    FROM job_data
    WHERE ds BETWEEN '2020-11-01' AND '2020-11-30'
    GROUP BY ds
)
SELECT
    dt1.date,
    dt1.daily_throughput,
    AVG(dt2.daily_throughput) AS rolling_7_day_avg_throughput
FROM daily_throughput dt1
JOIN daily_throughput dt2
ON dt1.date BETWEEN DATE_SUB(dt2.date, INTERVAL 6 DAY) AND dt2.date
GROUP BY dt1.date
ORDER BY dt1.date;
```

RESULTS

date	daily_throughput	rolling_7_day_avg_throughput
2020-11-25	0.0222	0.03505000
2020-11-26	0.0179	0.03762000
2020-11-27	0.0096	0.04255000
2020-11-28	0.0606	0.05353333
2020-11-29	0.0500	0.05000000
2020-11-30	0.0500	0.05000000

EXPLAINATION

- First, I create a temporary table **(daily_throughput)** to calculate daily metrics.
- It finds total events, total time spent, and daily throughput.
- The main query joins this table to calculate the 7-day rolling average.
- It includes data from the current day and the past 6 days.
- **AVG(dt2.daily_throughput)** gives the rolling average for each date.
- Finally, I group by date and sort the results in order.

Which Metric is Better?

- **Daily throughput** is useful for tracking short-term changes, but it can be inconsistent due to daily fluctuations.
- **7-day rolling average** smooths out spikes and drops, making it better for spotting long-term trends.
- **I prefer** the 7-day rolling average because it provides a more stable and meaningful view of throughput performance. 🚀

C. WRITE AN SQL QUERY TO CALCULATE THE PERCENTAGE SHARE OF EACH LANGUAGE OVER THE LAST 30 DAYS.

SOLUTION

```sql
WITH total_events AS (
    SELECT COUNT(*) AS total FROM job_data
    WHERE ds >= '2020-11-01'
)
SELECT
    jd.language,
    ROUND(100 * COUNT(*) / te.total, 2) AS percentage_share,
    te.total AS total_events
FROM job_data jd
JOIN total_events te
ON 1=1  -- This ensures every row gets access to total count
WHERE ds >= '2020-11-01'
GROUP BY jd.language,te.total
ORDER BY percentage_share DESC;
```

RESULTS

language	percentage_share	total_events
Persian	37.50	8
English	12.50	8
Arabic	12.50	8
Hindi	12.50	8
French	12.50	8
Italian	12.50	8

EXPLAINATION

- First, I create a temporary **table (total_events)** to count the total events for the last 30 days **(WHERE ds >= '2020-11-01')**.
- Then, I select the **language** from job_data.
- I calculate the percentage share of each language by dividing its count by the total events and multiplying by 100 **(ROUND(100 * COUNT(*) / te.total, 2)**).
- The **JOIN** total_events te **ON 1=1** ensures each row has access to the total event count.
- I group by **jd.language** and te.total to calculate **percentages** for each language.
- Finally, I order the results by percentage share in descending order.

D. WRITE AN SQL QUERY TO DISPLAY DUPLICATE ROWS FROM THE JOB_DATA TABLE.

SOLUTION

```sql
SELECT
    ds,
    job_id,
    actor_id,
    event,
    language,
    time_spent,
    org,
    COUNT(*) AS duplicate_count
FROM
    job_data
GROUP BY ds , job_id , actor_id , event , language , time_spent , org
HAVING COUNT(*) > 1
ORDER BY duplicate_count DESC;
```

RESULTS

ds	job_id	actor_id	event	language	time_spent	org	duplicate_count

EXPLAINATION

- Retrieves all relevant columns from the job_data table.
- **COUNT(*)** AS duplicate_count counts occurrences of each unique row.
- Groups data based on all selected columns.
- Ensures that only identical rows (all column values matching) are considered duplicates.
- Filters out records that appear only once.
- Keeps only rows where **COUNT(*)** is greater than 1.
- Sorts duplicate records in descending order based on their frequency.
- Helps identify the most frequently occurring duplicates first.

CASE STUDY -2

INVESTIGATING METRIC SPIKE

- **users:** Contains one row per user, with descriptive information about that user's account.
- **events:** Contains one row per event, where an event is an action that a user has taken (e.g., login, messaging, search).
- **email_events:** Contains events specific to the sending of emails.

SWIPE TO CHECK THE TASK & SOLUTIONS →

PROJECT SETUP

```
create database project_3_part_2;
use project_3_part_2;
```

USERS TABLE

```
CREATE TABLE users (
    user_id INT,
    created_at VARCHAR(100),
    company_id INT,
    language VARCHAR(50),
    activated_at VARCHAR(50),
    state VARCHAR(100)
);
SHOW VARIABLES LIKE 'secure_file_priv';

LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server 9.2/Uploads/users.csv"
INTO TABLE users
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
```

```
alter table users add column temp_created_at datetime;

UPDATE users
SET
    temp_created_at = STR_TO_DATE(created_at, '%d-%m-%Y %H:%i');
alter table users drop column created_at;
alter table users change column temp_created_at created_at datetime after user_id;

alter table users add column temp_activated_at datetime;
SET SQL_SAFE_UPDATES = 0;
UPDATE users
SET
    temp_activated_at = STR_TO_DATE(activated_at, '%d-%m-%Y %H:%i');
alter table users drop column activated_at;
alter table users change column temp_activated_at activated_at datetime after language;
```

PROJECT SETUP

EVENTS TABLE

```sql
CREATE TABLE events (
    user_id INT,
    occurred_at VARCHAR(100),
    event_type VARCHAR(50),
    event_name VARCHAR(1000),
    location VARCHAR(100),
    device VARCHAR(100),
    user_type INT
);

LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server 9.2/Uploads/events.csv"
INTO TABLE events
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
```

```sql
alter table events
add column temp_occurred_at datetime;
SET SQL_SAFE_UPDATES = 0;
UPDATE events
SET
    temp_occurred_at = STR_TO_DATE(occurred_at, '%d-%m-%Y %H:%i');
alter table events
drop column occurred_at;
alter table events
change column temp_occurred_at occurred_at datetime
after user_id;
```

EMAIL_EVENTS TABLE

```sql
create table email_events(
user_id int,
occurred_at varchar(100),
action varchar(100),
user_type int
);

LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server 9.2/Uploads/email_events.csv"
INTO TABLE email_events
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
```

```sql
select * from email_events;
alter table email_events
add column temp_occurred_at datetime;
SET SQL_SAFE_UPDATES = 0;
update email_events
set temp_occurred_at = str_to_date(occurred_at, '%d-%m-%Y %H:%i');
alter table email_events
drop column occurred_at;
alter table email_events
change column temp_occurred_at occurred_at datetime
after user_id;
```

A. WRITE AN SQL QUERY TO CALCULATE THE WEEKLY USER ENGAGEMENT.

SOLUTION

```sql
SELECT
    YEAR(occurred_at) AS year,
    WEEK(occurred_at, 1) AS week_number,
    COUNT(DISTINCT user_id) AS weekly_active_users
FROM
    events
WHERE
    event_type = 'engagement'
        AND event_name = 'login'
GROUP BY year , week_number
ORDER BY year , week_number;
```

RESULTS

year	week_number	weekly_active_users
2014	18	701
2014	19	1054
2014	20	1094
2014	21	1147
2014	22	1113
2014	23	1173
2014	24	1219
2014	25	1262
2014	26	1249

1262

EXPLAINATION

- **Extract Year and Week** – I used YEAR(occurred_at) to get the year and WEEK(occurred_at, 1) to get the week number (starting from Monday).
- **Count Unique Users** – COUNT(DISTINCT user_id) ensures each user is counted only once per week.
- **Filter Login Events** – The WHERE condition selects only engagement events where users logged in.
- **Group and Sort Data** – GROUP BY year, week_number organizes the data by week, and ORDER BY arranges it in order.

B. WRITE AN SQL QUERY TO CALCULATE THE USER GROWTH FOR THE PRODUCT.

SOLUTION

```
SELECT
    YEAR(created_at) AS year,
    MONTH(created_at) AS month,
    COUNT(user_id) AS new_users,
    SUM(COUNT(user_id))
    OVER (ORDER BY YEAR(created_at),
    MONTH(created_at)) AS total_users
FROM users
GROUP BY year, month
ORDER BY year, month;
```

RESULTS

year	month	new_users	total_users
2013	1	160	160
2013	2	160	320
2013	3	150	470
2013	4	181	651
2013	5	214	865
2013	6	213	1078
2013	7	284	1362
2013	8	316	1678
2013	9	330	2008

EXPLAINATION

1. **Extract Year and Month** – I used YEAR(created_at) to get the year and MONTH(created_at) to get the month from the created_at column.
2. **Count New Users** – COUNT(user_id) counts how many new users were created in each month.
3. **Calculate Total Users** – The SUM(COUNT(user_id)) OVER (ORDER BY YEAR(created_at), MONTH(created_at)) calculates the running total of users, adding up new users month by month.
4. **Group and Sort Data** – GROUP BY year, month groups the data by year and month, and ORDER BY year, month arranges the data in order.

C. WRITE AN SQL QUERY TO CALCULATE THE WEEKLY RETENTION OF USERS BASED ON THEIR SIGN-UP COHORT.

SOLUTION

```sql
SELECT
    YEAR(u.created_at) AS signup_year,
    WEEK(u.created_at) AS signup_week,
    YEAR(e.occurred_at) AS engagement_year,
    WEEK(e.occurred_at) AS engagement_week,
    COUNT(DISTINCT e.user_id) AS retained_users
FROM
    users u
JOIN
    events e
    ON u.user_id = e.user_id
WHERE
    e.event_type = 'engagement'
GROUP BY
    signup_year, signup_week, engagement_year, engagement_week
ORDER BY
    signup_year, signup_week, engagement_year, engagement_week;
```

RESULTS

signup_year	signup_week	engagement_year	engagement_week	retained_users
2013	0	2014	17	2
2013	0	2014	18	3
2013	0	2014	19	3
2013	0	2014	20	3
2013	0	2014	21	2
2013	0	2014	22	4
2013	0	2014	23	3
2013	0	2014	24	6
2013	0	2014	25	4

ult 10

EXPLAINATION

- **Get Signup Year and Week** – I used YEAR(u.created_at) and WEEK(u.created_at) to find the year and week users signed up.
- **Get Engagement Year and Week** – I used YEAR(e.occurred_at) and WEEK(e.occurred_at) to find when users engaged after signing up.
- **Count Retained Users** – COUNT(DISTINCT e.user_id) counts unique users who engaged in the system each week after signup.
- **Join Users and Events** – I joined the users and events tables using user_id to link signups with engagement activities.
- **Filter Engagement Events** – The WHERE condition selects only engagement events.
- **Group and Sort Data** – I grouped the data by signup and engagement weeks and sorted it by year and week.

D. WRITE AN SQL QUERY TO CALCULATE THE WEEKLY ENGAGEMENT PER DEVICE.

SOLUTION

```sql
SELECT
    YEAR(occurred_at) AS year,
    WEEK(occurred_at) AS week,
    device,
    COUNT(DISTINCT user_id) AS weekly_active_users
FROM
    events
WHERE
    event_type = 'engagement'
GROUP BY
    YEAR(occurred_at),
    WEEK(occurred_at), device
ORDER BY
    year, week, device;
```

RESULTS

year	week	device	weekly_active_users
2014	17	acer aspire desktop	9
2014	17	acer aspire notebook	20
2014	17	amazon fire phone	4
2014	17	asus chromebook	21
2014	17	dell inspiron desktop	18
2014	17	dell inspiron notebook	46
2014	17	hp pavilion desktop	14
2014	17	htc one	16
2014	17	ipad air	27

ult 3

EXPLAINATION

- **Get Year and Week** – I used YEAR(occurred_at) and WEEK(occurred_at) to get the year and week when each engagement event occurred.
- **Device Type** – The device column tells us which device the user used for the engagement (e.g., mobile, desktop).
- **Count Active Users** – COUNT(DISTINCT user_id) counts how many unique users engaged each week per device.
- **Filter Engagement Events** – The WHERE condition filters only engagement events (e.g., logins or activities).
- **Group and Sort Data** – I grouped the data by year, week, and device, and used ORDER BY to sort the results by year, week, and device type.

E. WRITE AN SQL QUERY TO CALCULATE THE EMAIL ENGAGEMENT METRICS.

SOLUTION

```sql
SELECT
    YEAR(occurred_at) AS year,
    WEEK(occurred_at) AS week,
    action,
    COUNT(DISTINCT user_id) AS users_engaged
FROM
    email_events
GROUP BY
    YEAR(occurred_at), WEEK(occurred_at), action
ORDER BY
    year, week, action;
```

RESULTS

year	week	action	users_engaged
2014	17	email_clickthrough	166
2014	17	email_open	310
2014	17	sent_reengagement_email	73
2014	17	sent_weekly_digest	908
2014	18	email_clickthrough	425
2014	18	email_open	900
2014	18	sent_reengagement_email	157
2014	18	sent_weekly_digest	2602
2014	19	email_clickthrough	476

sult 2 ×

EXPLAINATION

- **Get Year and Week** – I used YEAR(occurred_at) and WEEK(occurred_at) to extract the year and week when each email event happened.
- **Action Type** – The action column tells us the type of email engagement (e.g., opened, clicked).
- **Count Engaged Users** – COUNT(DISTINCT user_id) counts how many unique users engaged with emails each week.
- **Group Data by Year, Week, and Action** – I grouped the data by year, week, and action so we can see the engagement for each type of action.
- **Sort Results** – I used ORDER BY to sort the results by year, week, and action.

What I Have Learned and Achieved ?

- I improved my SQL skills by writing **complex queries**, using joins, and working with **different types of data.**
- I analyzed job data, user activity, and email events to understand important trends.
- I learned how to track **data over tim**e, like calculating rolling averages and measuring user retention.
- I found and fixed problems in the data, such as duplicate rows.
- I understood key metrics like user **growth**, **engagement**, and how users interact with emails.
- I got better at working with multiple tables to get a complete view of the data.

RAJEEV KUMAR