Project Description

As an aspiring Data Analyst, I tackled an Operational Analytics project to analyze end-to-end operations. The goal was to investigate metric spikes, understand sudden changes in key metrics, and derive valuable insights to improve various aspects of our operations.

Approach

My approach involved:

1. Analyzing job data to understand review efficiency and language trends

2. Investigating user engagement, growth, and retention patterns

3. Examining device-specific engagement and email interaction metrics

I used SQL queries to extract and analyze data from multiple tables, focusing on creating actionable insights for different departments.

Tech-Stack Used

I primarily used MySQL Workbench for this project. This tool allowed me to write complex SQL queries, manipulate large datasets efficiently, and visualize results. Its user-friendly interface and robust features were crucial for in-depth data analysis and quick query iteration.

Insights

Key insights include:

- Identified peak hours for job reviews, optimizing resource allocation

- Discovered trends in language preferences, informing content strategy

- Uncovered patterns in user engagement across different devices

- Recognized critical periods for user retention after sign-up

- Pinpointed areas for improvement in email engagement

Result

This project significantly enhanced our understanding of operational metrics. By identifying trends and anomalies, we've been able to:

- Improve job review efficiency

- Tailor our language offerings more effectively

- Optimize our user retention strategies

- Enhance our email engagement tactics

These insights have directly contributed to understanding data-driven decision-making as I’m learning SQL and data analytics and helped me understand advanced SQL.

Case Study 1

1. Jobs Reviewed Over Time:

SQL Query: SELECT DISTINCT

ds AS each_days,

COUNT(job_id) / (SUM(time_spent) / 3600) AS jobs_reviewed

FROM

job_data

GROUP BY each_days;

Output:

each_day	jobs_reviewed
2020-11-30	180.0000
2020-11-29	180.0000
2020-11-28	218.1818
2020-11-27	34.6154
2020-11-26	64.2857
2020-11-25	80.0000

2. Throughput Analysis

SQL Query: SELECT a.ds AS day,

a.each_day_throughput,

avg(a.each_day_throughput) OVER (ORDER BY ds rows BETWEEN 6 PRECEDING AND CURRENT row) AS

7_day_avg_throughput

FROM

(SELECT ds, count(job_id) / sum(time_spent) AS each_day_throughput FROM job_data GROUP BY ds) AS a

GROUP BY ds;

Output:

day	each_day_throughput	7_day_avg_throughput
2020-11-25	0.0222	0.02220000
2020-11-26	0.0179	0.02005000
2020-11-27	0.0096	0.01656667
2020-11-28	0.0606	0.02757500
2020-11-29	0.0500	0.03206000
2020-11-30	0.0500	0.03505000

3. Language Share Analysis:

SQL Query: SELECT language,

count(job_id) as no_of_jobs_per_language,

count(job_id)*100 / sum(count(*)) OVER() as percentage_share

FROM job_data

WHERE ds LIKE '2020-11-%'

GROUP by language;

Output:

language	no_of_jobs_per_language	percentage_share
English	1	12.5000
Arabic	1	12.5000
Persian	3	37.5000
Hindi	1	12.5000
French	1	12.5000
Italian	1	12.5000

4. Duplicate Rows Detection:

SQL Query: 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;

Output:

ds	job_id	actor_id	event	language	time_spent	org	duplicate_count

Case Study 2: Investigating Metric Spike

1. **Weekly User Engagement**

SQL Query: SELECT

```
    WEEK(occurred_at) AS week_number,
    COUNT(DISTINCT user_id) AS user_engagement
FROM
    events
GROUP BY WEEK(occurred_at)
ORDER BY WEEK(occurred_at);
```

Output:

week_number	user_engagement
17	663
18	1068
19	1113
20	1154
21	1121
22	1186
23	1232
24	1275
25	1264
26	1302
27	1372
28	1365
29	1376
30	1467
31	1299
32	1225
33	1225
34	1204
35	104

2. User Growth

SQL Query: select a.no_of_users, a.date, sum(no_of_users) over(order by date) as user_growth

from

(SELECT

  COUNT(user_id) AS no_of_users, DATE(created_at) AS date

FROM

  users

WHERE

  state = 'active'

  group by date(created_at)) a;

Output:

no_of_users	date	user_growth
7	2013-01-01	7
7	2013-01-02	14
6	2013-01-03	20
1	2013-01-04	21
2	2013-01-05	23
3	2013-01-06	26
4	2013-01-07	30
2	2013-01-08	32
6	2013-01-09	38
6	2013-01-10	44

4. Weekly Engagement Per Device

SQL Query: SELECT

```
    WEEK(occurred_at) AS week_number,
    device,
    COUNT(DISTINCT user_id) AS user_engagement
FROM
    events
GROUP BY device, WEEK(occurred_at)
ORDER BY WEEK(occurred_at)
LIMIT 10;
```

Output:

week_number	device	user_engagement
17	Acer aspire desktop	9
17	dell inspiron notebook	46
17	htc one	16
17	dell inspiron desktop	18
17	amazon fire phone	4
17	ipad mini	19
17	asus chromebook	21
17	hp pavilion desktop	14
17	ipad air	27
17	acer aspire notebook	20

5. Email Metrics

SQL Query: SELECT week(occurred_at) as Week,

count(DISTINCT (CASE WHEN action = "sent_weekly_digest"

THEN user_id end)) as weekly_digest,

count(distinct (CASE WHEN action = "sent_reengagement_email"

THEN user_id end)) as reengagement_mail,

count(distinct (CASE WHEN action = "email_open"

THEN user_id end)) as opened_email,

count(distinct (CASE WHEN action = "email_clickthrough"

THEN user_id end)) as email_clickthrough

FROM email_events

GROUP BY week(occurred_at)

ORDER BY week(occurred_at);

Output:

week	weekly_digest	reengagement_mail	opened_email	email_clickthrough
17	908	73	310	166
18	2602	157	900	425
19	2665	173	961	476
20	2733	191	989	501
21	2822	164	996	436
22	2911	192	965	478
23	3003	197	1057	529
24	3105	226	1136	549
25	3207	196	1084	524
26	3302	219	1149	550
27	3399	213	1207	613
28	3499	213	1228	594
29	3592	213	1201	583
30	3706	231	1363	625
31	3793	222	1338	444
32	3897	200	1318	416
33	4012	264	1417	490
34	4111	261	1502	481
35	0	48	41	38