Operation Analytics and Investigating Metric Spike

Project description

Operation analytics involves the systematic analysis of operational data to gain insights into various aspects of a business or system's performance. This can include examining metrics related to efficiency, productivity, quality, and more. Investigating metric spikes is a crucial aspect of operation analytics, as sudden increases or decreases in key metrics can indicate issues, opportunities, or anomalies that require attention.

In this project we have two case studies

- Case Study 1 (Job Data Analysis)
  - In this case study we have to create a database table known as job_data and perform SQL queries on it
- Case Study 2 (Investigating metric spike)
  - In this case study we have to import three tables - users, events and email_events and perform SQL queries on them to gain insights

Approach

For this project, I used the dataset provided by the Trainity team to create the required tables and populate them accordingly in MySQL. The various queries I used to load the data and gain insights are mentioned in the results section

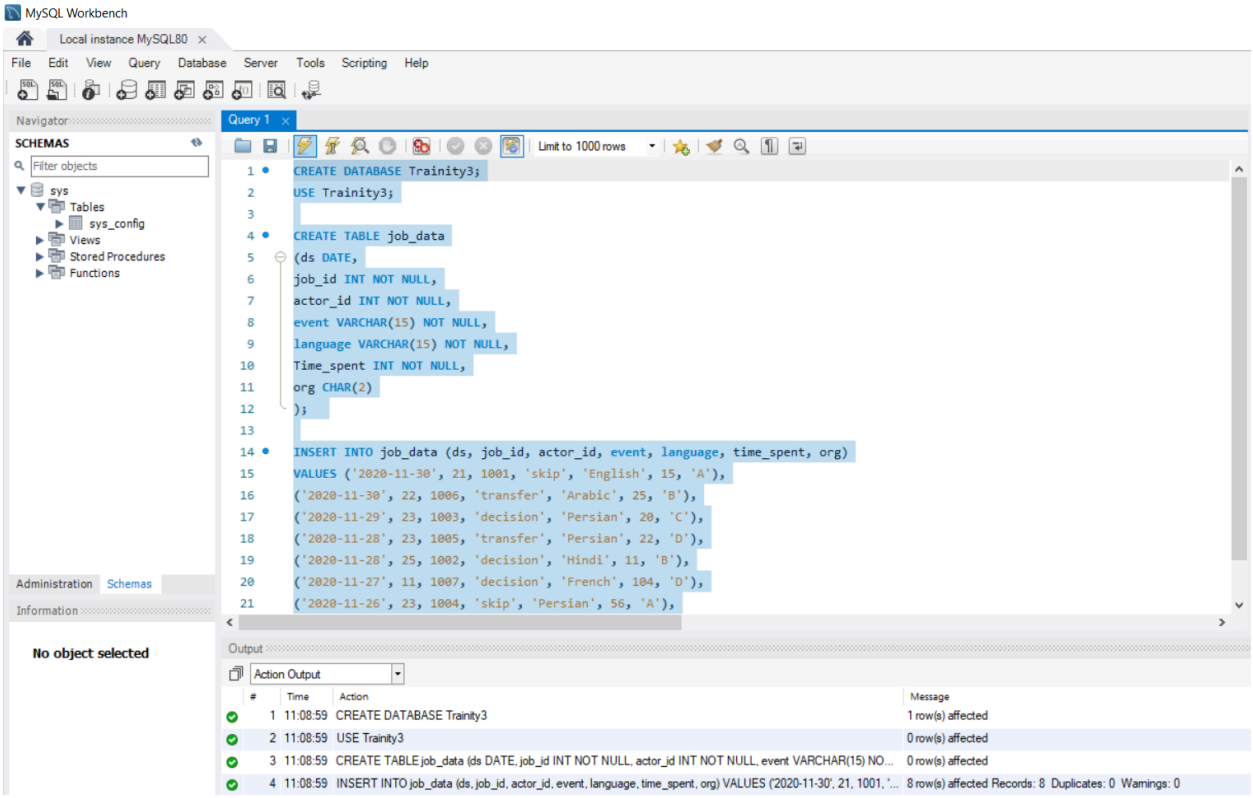
Tech Stack used

For this project, I have chosen MySQL for my database management software as it is the most widely used open-source relational database management system known for its stability and reliability. I have also used MySQL for a number of college projects hence it was my first choice

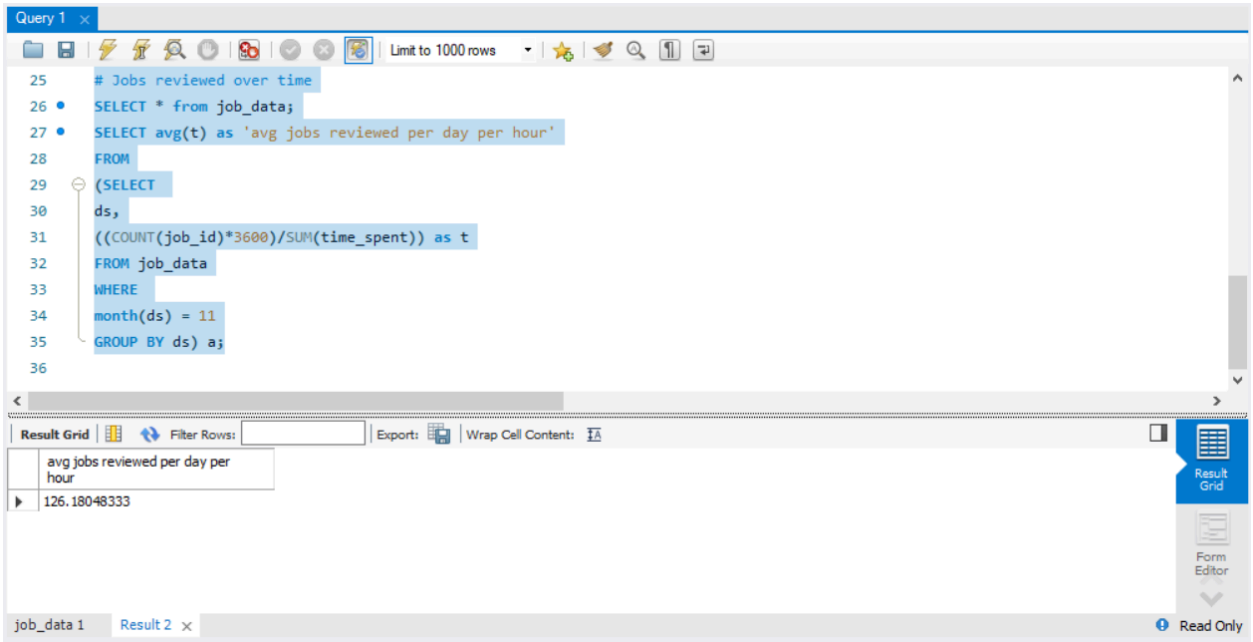
MySQL is optimized for performance, providing fast read and write operations, which is crucial for real-time analytics applications

Insights for Case Study 1

Creating the table



Task 1 : Write an SQL query to calculate the number of jobs reviewed per hour for each day in November 2020.



Therefore the average number of jobs reviewed per hour is approximately 126

Task 2 : 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.

```
38    # Throughput analysis
39 •  SELECT ROUND(COUNT(event)/SUM(time_spent), 2) AS "Weekly Throughput" FROM job_data;
40 •  SELECT ds AS dates, ROUND(COUNT(event)/SUM(time_spent), 2) AS "Daily Throughput" FROM job_data
41    GROUP BY ds ORDER BY ds;
42
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Weekly Throughput
0.03

Result 3 | Read Only

Therefore the weekly or 7-day rolling average for throughput is 0.03

```
38    # Throughput analysis
39 •  SELECT ROUND(COUNT(event)/SUM(time_spent), 2) AS "Weekly Throughput" FROM job_data;
40    SELECT ds AS dates, ROUND(COUNT(event)/SUM(time_spent), 2) AS "Daily Throughput" FROM job_data
41    GROUP BY ds ORDER BY ds;
42
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

dates	Daily Throughput
2020-11-25	0.02
2020-11-26	0.02
2020-11-27	0.01
2020-11-28	0.06
2020-11-29	0.05
2020-11-30	0.05

Result 8 | Read Only

I prefer using the 7-day rolling average for throughput as it smooths out short-term fluctuations and provides a more stable, long-term view of throughput trends. It helps to identify underlying trends and patterns while reducing the impact of random variability.

Task 3 : Write an SQL query to calculate the percentage share of each language over the last 30 days.

```
41
42    # Language share analysis
43 •  SELECT language AS languages, ROUND(100 * COUNT(*)/total, 2) AS Percentage, sub.total
44    FROM job_data
45    CROSS JOIN (SELECT COUNT(*) AS total FROM job_data) AS sub
46    GROUP BY language, sub.total
47
48
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

languages	Percentage	total
English	12.50	8
Arabic	12.50	8
Persian	37.50	8
Hindi	12.50	8
French	12.50	8
Italian	12.50	8

Result 10 | Read Only

Task 4 : Write an SQL query to display duplicate rows from the job_data table.

```
48
49    # Duplicate rows detection
50 •  SELECT actor_id,count(*) as Duplicates from job_data
51    Group By actor_id having count(*) >1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

actor_id	Duplicates
1003	2

Result 11 | Read Only

Therefore there are two duplicate rows in the job data table

Insights for Case Study 2

Creating users table

Query 1 | SQL File 2* | Limit to 1000 rows

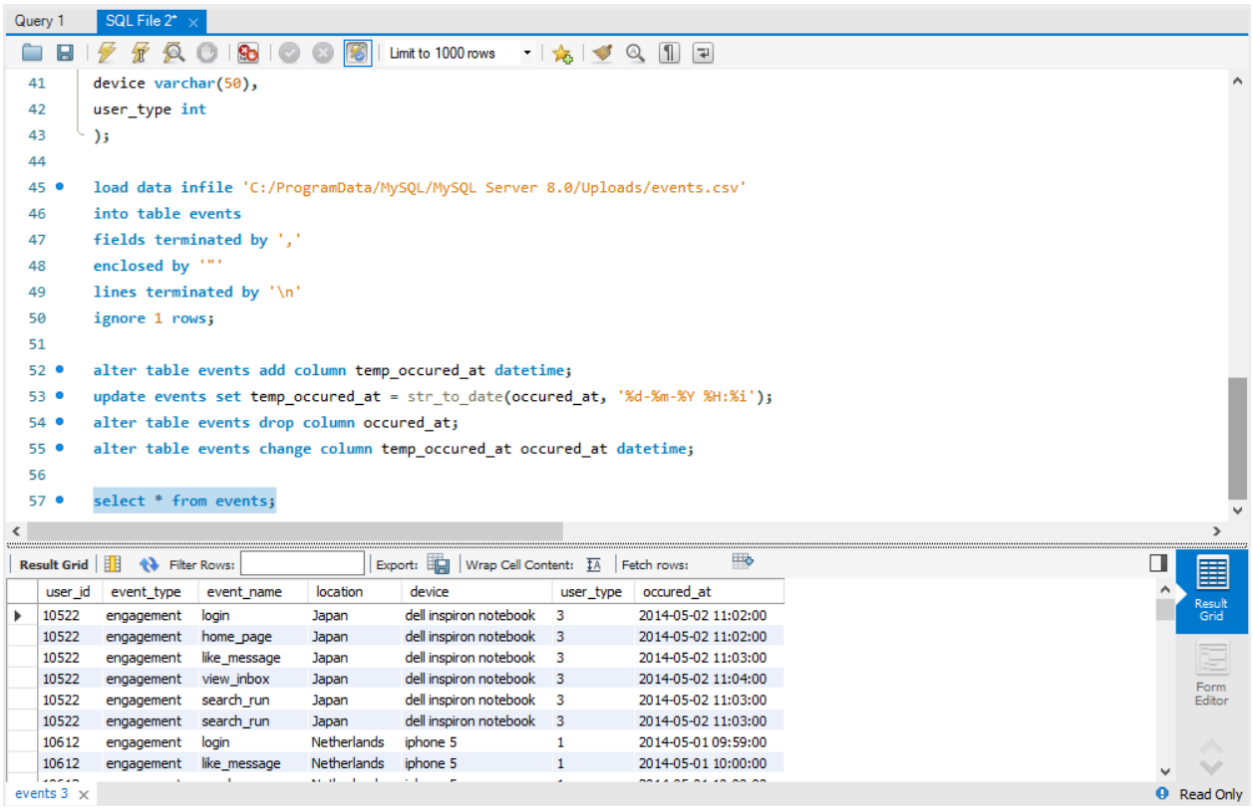
```
17    lines terminated by '\n'
18    ignore 1 rows;
19
20 •  set sql_safe_updates = 0;
21
22 •  alter table users add column temp_created_at datetime;
23 •  update users set temp_created_at = str_to_date(created_at, '%d-%m-%Y %H:%i');
24 •  alter table users drop column created_at;
25 •  alter table users change column temp_created_at created_at datetime;
26
27 •  alter table users add column temp_activated_at datetime;
28 •  update users set temp_activated_at = str_to_date(activated_at, '%d-%m-%Y %H:%i');
29 •  alter table users drop column activated_at;
30 •  alter table users change column temp_activated_at activated_at datetime;
31
32 •  select * from users;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

user_id	company_id	language	state	created_at	activated_at
0	5737	english	active	2013-01-01 20:59:00	2013-01-01 21:01:00
3	2800	german	active	2013-01-01 18:40:00	2013-01-01 18:42:00
4	5110	indian	active	2013-01-01 14:37:00	2013-01-01 14:39:00
6	11699	english	active	2013-01-01 18:37:00	2013-01-01 18:38:00
7	4765	french	active	2013-01-01 16:19:00	2013-01-01 16:20:00
8	2698	french	active	2013-01-01 04:38:00	2013-01-01 04:40:00
11	3745	english	active	2013-01-01 08:07:00	2013-01-01 08:09:00
13	4025	english	active	2013-01-02 12:27:00	2013-01-02 12:29:00
15	4259	english	active	2013-01-02 15:39:00	2013-01-02 15:41:00

users 2 | Read Only

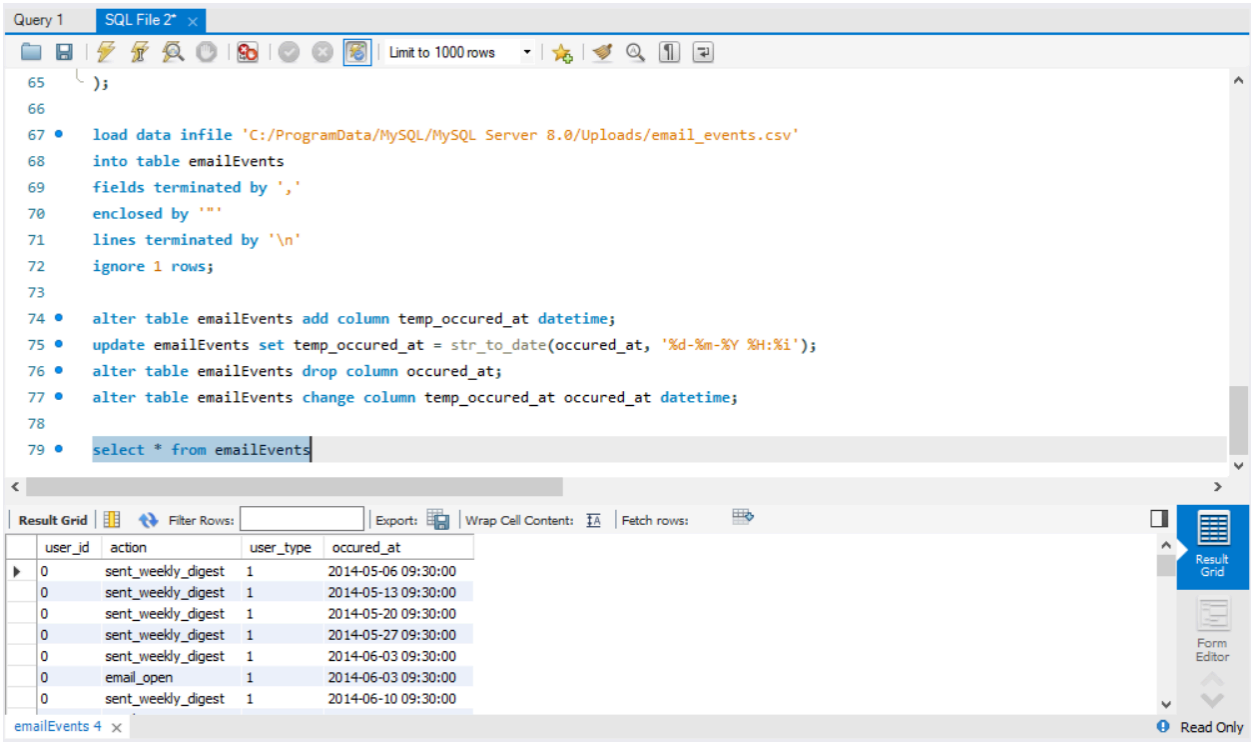
Creating events table



```
device varchar(50),
user_type int
);

load data infile 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/events.csv'
into table events
fields terminated by ','
enclosed by '"'
lines terminated by '\n'
ignore 1 rows;

alter table events add column temp_occured_at datetime;
update events set temp_occured_at = str_to_date(occured_at, '%d-%m-%Y %H:%i');
alter table events drop column occured_at;
alter table events change column temp_occured_at occured_at datetime;

select * from events;
```

user_id	event_type	event_name	location	device	user_type	occured_at
10522	engagement	login	Japan	dell inspiron notebook	3	2014-05-02 11:02:00
10522	engagement	home_page	Japan	dell inspiron notebook	3	2014-05-02 11:02:00
10522	engagement	like_message	Japan	dell inspiron notebook	3	2014-05-02 11:03:00
10522	engagement	view_inbox	Japan	dell inspiron notebook	3	2014-05-02 11:04:00
10522	engagement	search_run	Japan	dell inspiron notebook	3	2014-05-02 11:03:00
10522	engagement	search_run	Japan	dell inspiron notebook	3	2014-05-02 11:03:00
10612	engagement	login	Netherlands	iphone 5	1	2014-05-01 09:59:00
10612	engagement	like_message	Netherlands	iphone 5	1	2014-05-01 10:00:00

Creating emailEvents table



```
);

load data infile 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/email_events.csv'
into table emailEvents
fields terminated by ','
enclosed by '"'
lines terminated by '\n'
ignore 1 rows;

alter table emailEvents add column temp_occured_at datetime;
update emailEvents set temp_occured_at = str_to_date(occured_at, '%d-%m-%Y %H:%i');
alter table emailEvents drop column occured_at;
alter table emailEvents change column temp_occured_at occured_at datetime;

select * from emailEvents
```

user_id	action	user_type	occured_at
0	sent_weekly_digest	1	2014-05-06 09:30:00
0	sent_weekly_digest	1	2014-05-13 09:30:00
0	sent_weekly_digest	1	2014-05-20 09:30:00
0	sent_weekly_digest	1	2014-05-27 09:30:00
0	sent_weekly_digest	1	2014-06-03 09:30:00
0	email_open	1	2014-06-03 09:30:00
0	sent_weekly_digest	1	2014-06-10 09:30:00

Task 1 : Write an SQL query to calculate the weekly user engagement.

```sql
# Weekly user engagement
select extract(week from occured_at) as week_number,
count(distinct user_id) as active_user
from events
where event_type = 'engagement'
group by week_number
order by week_number
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

week_number	active_user
17	663
18	1068
19	1113
20	1154
21	1121
22	1186
23	1232

Result 5 | Read Only

Task 2 : Write an SQL query to calculate the user growth for the product.

```sql
# User growth analysis
select year, week_num, num_users, sum(num_users)
over (order by year, week_num) as cum_users
from( select extract(year from created_at) as year, extract(week from created_at) as week_num, count(distinct user_id) as num_user
from users
group by year, week_num
order by year, week_num)sub
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

year	week_num	num_users	cum_users
2013	0	23	23
2013	1	30	53
2013	2	48	101
2013	3	36	137
2013	4	30	167
2013	5	48	215
2013	6	38	253

Result 9 | Read Only

Task 3 : Write an SQL query to calculate the weekly retention of users based on their sign-up cohort.

```
),
cte2 AS (
SELECT DISTINCT user_id,
EXTRACT(week FROM occured_at) AS engagement_week
FROM events
WHERE event_type = 'engagement'
)
SELECT COUNT(user_id) AS total_engaged_users,
SUM(CASE WHEN retention_week > 8 THEN 1 ELSE 0 END) AS retained_users
FROM (
SELECT a.user_id, a.signup_week, b.engagement_week, b.engagement_week - a.signup_week AS retention_week
FROM cte1 a
LEFT JOIN cte2 b ON a.user_id = b.user_id
ORDER BY a.user_id
) sub;
```

total_engaged_users	retained_users
615	96

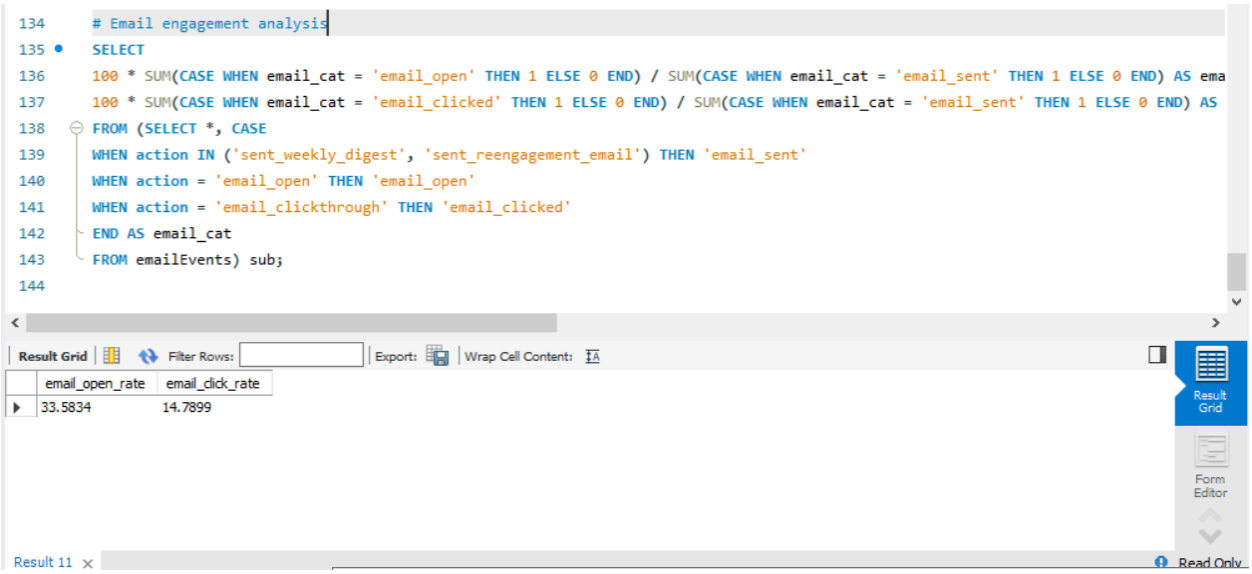
Therefore the number of retained users weekly is 96 from a total of 615

Task 4 : Write an SQL query to calculate the weekly engagement per device.

```
# Weekly engagement per device
WITH cte AS ( SELECT EXTRACT(year FROM occured_at)||'-'||EXTRACT(week FROM occured_at)
AS weeknum, device, COUNT(DISTINCT user_id) AS usercnt
FROM events
WHERE event_type = 'engagement'
GROUP BY weeknum, device
ORDER BY weeknum
)
SELECT weeknum, device, usercnt
FROM cte;
```

weeknum	device	usercnt
1	acer aspire desktop	198
1	acer aspire notebook	338
1	amazon fire phone	89
1	asus chromebook	355
1	dell inspiron desktop	360
1	dell inspiron notebook	677
1	hp pavilion desktop	339

Task 5 : Write an SQL query to calculate the email engagement metrics.



Therefore the email open rate is approximately 34 while the email click rate is approximately 15

Results

Hence we were able to the MySQL database management software to fire multiple queries that gave us valuable insights into Operational Analytics and Investigating Metric Spike

The results of each tasks are highlighted above