Operation Analytics and Investigating Metric Spike

Project Description:

As a Data Analyst, I work closely with various teams, including operations, support, and marketing, to extract important insights from the data they gather and address various queries from different departments across the company.

An essential component of Operational Analytics is examining sudden increases in metrics. For this job, you need to be able to understand and analyse sudden changes in important indicators, like a drop in daily user interaction or sales.

As a Data Analyst, I need to understand the process of investigating metric spikes to address these product-related inquiries on a daily basis.

The project has two case studies. One of the case studies involves the analysis of job data, which includes a table named "job data" with columns for job ID, actor ID, event kind, language of the content, time spent reviewing a job, actor's organization, and date. Another case study examines a metrics spike that includes three tables: "users" which holds detailed information on user accounts, "events" which records user actions (such as login, messaging, and search), and "email events" which focuses on the sending of emails.

Approach:

Firstly, Understand the data given by the different departments and then understand their requirement to drive the valuable insights from the data they have given.

I used MySQL Workbench and SQL queries to create a database and table for both case studies. First, load the data into the tables and do it manually from Table Data Import Wizard and then use advanced SQL queries to analyse the data and provide valuable insights.

Tech-Stack Used:

MySQL Workbench 8.0 – Version 8.0.36 build 3737333 CE (64 bits) is used in this project as it is free and open-source. It is an easy-to-use relational database management system.

Insights:

The SQL queries extract the summary and insights from the database:

Case Study 1: Job Data Analysis

Create a table named "job_data" in the "jobdata_analysis" database, consisting of the following columns: **job_id**, **actor_id, event, language, time_spent, org, ds**.

A. Jobs Reviewed Over Time:

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

```
SELECT ds AS `date`,
       COUNT(job_id)/SUM(time_spent/(60*60)) AS job_reviewed_per_hour
FROM job_data
WHERE ds BETWEEN '2020-11-01' AND '2020-11-30'
GROUP BY ds
ORDER BY ds ASC;
```

date	job_reviewed_per_hour
2020-11-25	80.0000
2020-11-26	64.1026
2020-11-27	34.6021
2020-11-28	217.3913
2020-11-29	178.5714
2020-11-30	180.1802

⇒ 28th November is the day when most of the jobs are reviewed per hour and 27th November is the day when the least job is reviewed per hour.

B. Throughput Analysis:

- Objective: Calculate the 7-day rolling average of throughput (number of events per second).
- Your Task: 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.

```
SELECT date, events_per_second AS daily_avg_tp,
       ROUND(AVG(events_per_second) OVER (ORDER BY date ROWS
       BETWEEN 6 PRECEDING AND CURRENT ROW), 3) AS
       7days_rolling_avg_tp
FROM (
       SELECT ds AS date, ROUND(COUNT(event) / SUM(time_spent), 3)
              AS events_per_second
       FROM job_data
       GROUP BY ds
       ) AS job_data1
ORDER BY date;
```

date	daily_avg_tp	7days_rolling_avg_tp
2020-11-25	0.022	0.022
2020-11-26	0.018	0.020
2020-11-27	0.010	0.017
2020-11-28	0.061	0.028
2020-11-29	0.050	0.032
2020-11-30	0.050	0.035

⇒ Whether to prefer a daily metric or a 7-day rolling average for throughput depends on requirements, and objectives of the analysis.

- Daily Metrics:
  Daily metrics provide granular, real-time insights into daily performance, enabling quick decision-making and operational efficiency monitoring. They are sensitive to short-term changes in throughput, making them ideal for detecting sudden spikes or drops in performance.
- 7-day Rolling Average:
  A 7-day rolling average smooths out fluctuations, providing a stable trend line for identifying long-term trends. It reduces noise and outliers, allowing focus on underlying trends. This method is particularly useful for capturing long-term performance trends like weekly cycles or seasonal variations.

So, as per the data provided for analysis of the throughput, I would prefer to use daily metrics to identify the trends and pattern because the data is given for the short term due to which I am unable to identify the trends and patterns in the 7-day rolling average as it smooths out the patterns.

C. Language Share Analysis:

- Objective: Calculate the percentage share of each language in the last 30 days.
- Your Task: Write an SQL query to calculate the percentage share of each language over the last 30 days.

```
SELECT language, COUNT(language) AS language_count,
    ROUND(COUNT(language) * 100.0 / SUM(COUNT(language)) OVER (),
    2) AS percentage_share
FROM job_data
GROUP BY language
ORDER BY language_count ASC;
```

language	language_count	percentage_share
English	1	12.50
Arabic	1	12.50
Hindi	1	12.50
French	1	12.50
Italian	1	12.50
Persian	3	37.50

⇒ As per the data analysis, the Persian language holds more percentage share which means it is widely used for the content.

D. Duplicate Rows Detection:

- Objective: Identify duplicate rows in the data.
- Your Task: Write an SQL query to display duplicate rows from the job_data table.

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

ds	job_id	actor_id	event	language	time_spent	org	duplicate_rows

⇒ There are no duplicate rows in data but there are duplicate job ID which has different actor IDs and different organisations.

```
SELECT *
FROM (
      SELECT *,
      ROW_NUMBER()OVER(PARTITION  BY job_id) AS row_no
      FROM job_data
      ) job_data1
WHERE row_no > 1;
```

ds	job_id	actor_id	event	language	time_spent	org	duplicate_rows
2020-11-28	23	1005	transfer	Persian	00:00:22	D	2
2020-11-26	23	1004	skip	Persian	00:00:56	A	3

Case Study 2: Job Data Analysis

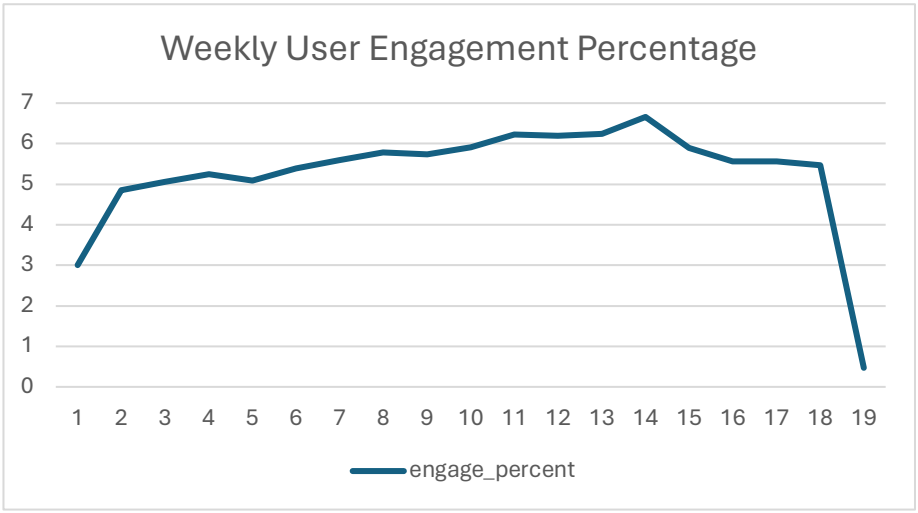
There are 3 tables in the database named metric_spike: **users**, **events**, **email_events**.

A. Weekly User Engagement:

- Objective: Measure the activeness of users on a weekly basis.
- Your Task: Write an SQL query to calculate the weekly user engagement.

```
WITH weekly_user_count AS (
        SELECT YEAR(occurred_at) AS years, WEEK(occurred_at) AS
                weeks, COUNT(DISTINCT user_id) AS weekly_user_count
        FROM events
        WHERE event_type = "engagement"
        GROUP BY years, weeks
        )
SELECT years, weeks, weekly_user_count,
        ROUND((weekly_user_count/SUM(weekly_user_count) OVER ()) *
        100, 2) AS engage_percent
FROM weekly_user_count
ORDER BY years, weeks ASC;
```

years	weeks	weekly_user_count	engage_percent
2014	17	663	3.01
2014	18	1068	4.85
2014	19	1113	5.06
2014	20	1154	5.24
2014	21	1121	5.09
2014	22	1186	5.39
2014	23	1232	5.60
2014	24	1275	5.79
2014	25	1264	5.74
2014	26	1302	5.91
2014	27	1372	6.23
2014	28	1365	6.20
2014	29	1376	6.25
2014	30	1467	6.66
2014	31	1299	5.90
2014	32	1225	5.56
2014	33	1225	5.56
2014	34	1204	5.47
2014	35	104	0.47



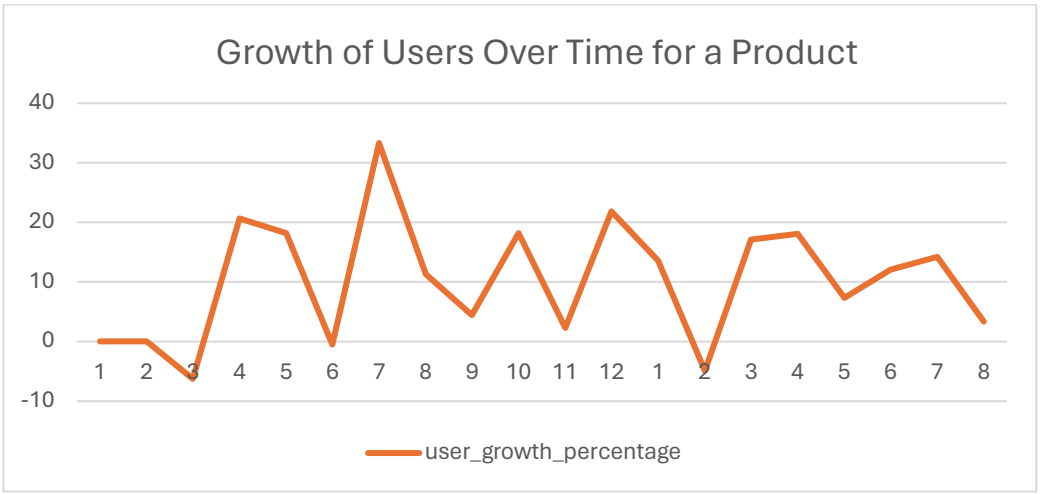
⇒ The engagement of users on a weekly basis started decreasing from week 31 of the year 2014.

B. User Growth Analysis:

- Objective: Analyze the growth of users over time for a product.
- Your Task: Write an SQL query to calculate the user growth for the product.

```
WITH users_1 AS (
        SELECT  YEAR(created_at) AS years, MONTH(created_at) as
                months, COUNT(DISTINCT user_id) AS total_users
        FROM users
        GROUP BY years, months
        )
SELECT years, months, total_users,
      total_users - LAG (total_users) OVER (ORDER BY years, months) AS
      user_growth,
      ROUND((total_users - LAG (total_users) OVER (ORDER BY years,
      months))/LAG(total_users) OVER (ORDER BY years, months)*100, 2) AS
      user_growth_percentage
FROM users_1
ORDER BY years, months;
```

years	months	total_users	user_growth	user_growth_percentage
2013	1	160	NULL	NULL
2013	2	160	0	0.00
2013	3	150	-10	-6.25
2013	4	181	31	20.67
2013	5	214	33	18.23
2013	6	213	-1	-0.47
2013	7	284	71	33.33
2013	8	316	32	11.27
2013	9	330	14	4.43
2013	10	390	60	18.18
2013	11	399	9	2.31
2013	12	486	87	21.80
2014	1	552	66	13.58
2014	2	525	-27	-4.89
2014	3	615	90	17.14
2014	4	726	111	18.05
2014	5	779	53	7.30
2014	6	873	94	12.07
2014	7	997	124	14.20
2014	8	1031	34	3.41



⇒ The growth of the users over time for a product increased from 160 in January 2013 to 1031 in August. Over 20 months, the cumulative growth of the active users increased to 9381.

C. Weekly Retention Analysis:

- Objective: Analyze the retention of users on a weekly basis after signing up for a product.
- Your Task: Write an SQL query to calculate the weekly retention of users based on their sign-up cohort.

```
SELECT first_login AS weeks,
        SUM(CASE WHEN week_number = 0 THEN 1 ELSE 0 END) AS Week_0,
        SUM(CASE WHEN week_number = 1 THEN 1 ELSE 0 END) AS Week_1,
        SUM(CASE WHEN week_number = 2 THEN 1 ELSE 0 END) AS Week_2,
        SUM(CASE WHEN week_number = 3 THEN 1 ELSE 0 END) AS Week_3,
        SUM(CASE WHEN week_number = 4 THEN 1 ELSE 0 END) AS Week_4,
        SUM(CASE WHEN week_number = 5 THEN 1 ELSE 0 END) AS Week_5,
        SUM(CASE WHEN week_number = 6 THEN 1 ELSE 0 END) AS Week_6,
        SUM(CASE WHEN week_number = 7 THEN 1 ELSE 0 END) AS Week_7,
        SUM(CASE WHEN week_number = 8 THEN 1 ELSE 0 END) AS Week_8,
        SUM(CASE WHEN week_number = 9 THEN 1 ELSE 0 END) AS Week_9,
        SUM(CASE WHEN week_number = 10 THEN 1 ELSE 0 END) AS
        Week_10,
        SUM(CASE WHEN week_number = 11 THEN 1 ELSE 0 END) AS
        Week_11,
        SUM(CASE WHEN week_number = 12 THEN 1 ELSE 0 END) AS
        Week_12,
        SUM(CASE WHEN week_number = 13 THEN 1 ELSE 0 END) AS
        Week_13,
        SUM(CASE WHEN week_number = 14 THEN 1 ELSE 0 END) AS
        Week_14,
        SUM(CASE WHEN week_number = 15 THEN 1 ELSE 0 END) AS
        Week_15,
        SUM(CASE WHEN week_number = 16 THEN 1 ELSE 0 END) AS
        Week_16,
        SUM(CASE WHEN week_number = 17 THEN 1 ELSE 0 END) AS
        Week_17,
        SUM(CASE WHEN week_number = 18 THEN 1 ELSE 0 END) AS
        Week_18
    FROM
        (
        SELECT event_login.user_id, event_login.login_week,
        first_event.first_login,
        event_login.login_week - first_event.first_login AS week_number
        FROM
            (
            SELECT user_id, WEEK(occurred_at) AS login_week
            FROM events
            GROUP BY 1, 2
            ) event_login,
            (
            SELECT user_id, MIN(WEEK(occurred_at)) AS first_login
```

```
        FROM events
        GROUP BY 1
        ) first_event
    WHERE event_login.user_id = first_event.user_id
    ) event_1
GROUP BY first_login
ORDER BY first_login ASC;
```

weeks	Week_0	Week_1	Week_2	Week_3	Week_4	Week_5	Week_6	Week_7	Week_8	Week_9	Week_10	Week_11	Week_12	Week_13	Week_14	Week_15	Week_16	Week_17	Week_18
17	663	472	324	251	205	187	167	146	145	145	136	131	132	143	116	91	82	77	5
18	596	362	261	203	168	147	144	127	113	122	106	118	127	110	97	85	67	4	0
19	427	284	173	153	114	95	91	81	95	82	68	65	63	42	51	49	2	0	0
20	358	223	165	121	91	72	63	67	63	65	67	41	40	33	40	0	0	0	0
21	317	187	131	91	74	63	75	72	58	48	45	39	35	28	2	0	0	0	0
22	326	224	150	107	87	73	63	60	55	48	41	39	31	1	0	0	0	0	0
23	328	219	138	101	90	79	69	61	54	47	35	30	0	0	0	0	0	0	0
24	339	205	143	102	81	63	65	61	38	39	29	0	0	0	0	0	0	0	0
25	305	218	139	101	75	63	50	46	38	35	2	0	0	0	0	0	0	0	0
26	288	181	114	83	73	55	47	43	29	0	0	0	0	0	0	0	0	0	0
27	292	199	121	106	68	53	40	36	1	0	0	0	0	0	0	0	0	0	0
28	274	194	114	69	46	30	28	3	0	0	0	0	0	0	0	0	0	0	0
29	270	186	102	65	47	40	1	0	0	0	0	0	0	0	0	0	0	0	0
30	294	202	121	78	53	3	0	0	0	0	0	0	0	0	0	0	0	0	0
31	215	145	76	57	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
32	267	188	94	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
33	286	202	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
34	279	44	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
35	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

D. Weekly Engagement Per Device:

- Objective: Measure the activeness of users on a weekly basis per device.
- Your Task: Write an SQL query to calculate the weekly engagement per device.

```
WITH weekly_user_count AS
        (
        SELECT WEEK(occurred_at) AS weeks, device, COUNT(DISTINCT
        user_id) AS weekly_user_count
        FROM events
        WHERE event_type = "engagement"
        GROUP BY weeks, device
        )
SELECT weeks, device, weekly_user_count,
        ROUND((weekly_user_count/SUM(weekly_user_count) OVER ()) * 100,
        2) AS engage_percent
FROM weekly_user_count
ORDER BY engage_percent DESC;
```

weeks	device	weekly_user_count	engage_percent
30	macbook pro	322	1.10
31	macbook pro	321	1.09
33	macbook pro	312	1.06
32	macbook pro	307	1.05
27	macbook pro	302	1.03
28	macbook pro	295	1.00
29	macbook pro	295	1.00
34	macbook pro	292	0.99
25	macbook pro	275	0.94
26	macbook pro	269	0.92
19	macbook pro	266	0.91
23	macbook pro	266	0.91
20	macbook pro	256	0.87
24	macbook pro	255	0.87
18	macbook pro	252	0.86
22	macbook pro	251	0.85
21	macbook pro	247	0.84
28	lenovo thinkpad	220	0.75
29	lenovo thinkpad	209	0.71
31	lenovo thinkpad	207	0.71
30	lenovo thinkpad	206	0.70
27	lenovo thinkpad	202	0.69
25	lenovo thinkpad	197	0.67
34	lenovo thinkpad	193	0.66
26	lenovo thinkpad	192	0.65
33	lenovo thinkpad	191	0.65
19	lenovo thinkpad	178	0.61
32	lenovo thinkpad	179	0.61
22	lenovo thinkpad	176	0.60
23	lenovo thinkpad	176	0.60
20	lenovo thinkpad	173	0.59
21	lenovo thinkpad	167	0.57
24	lenovo thinkpad	165	0.56
27	iphone 5	163	0.56
30	macbook air	159	0.54
18	lenovo thinkpad	153	0.52
23	iphone 5	152	0.52
24	macbook air	152	0.52
26	iphone 5	152	0.52

⇒ Most of the weekly engagement per device was observed for Macbook Pro, Lenovo Thinkpad and Iphone 5 user.

E. Email Engagement Analysis:

- Objective: Analyze how users are engaging with the email service.
- Your Task: Write an SQL query to calculate the email engagement metrics.

```
WITH email_metrics AS
        (
        SELECT user_id,
        COUNT(*) AS total_emails_sent,
        SUM(CASE WHEN action = 'sent_weekly_digest' THEN 1 ELSE 0 END)
                AS sent_weekly_digest,
        SUM(CASE WHEN action = 'email_open' THEN 1 ELSE 0 END) AS
                email_open,
        SUM(CASE WHEN action = 'email_clickthrough' THEN 1 ELSE 0 END)
                AS email_clickthrough,
        SUM(CASE WHEN action = 'sent_reengagement_email' THEN 1 ELSE 0
                END) AS sent_reengagement_emai
        FROM email_events
        GROUP BY user_id
        )
SELECT user_id, total_emails_sent, sent_weekly_diges, email_open,
   email_clickthrough, sent_reengagement_email
CASE WHEN total_emails_sent > 0 THEN ROUND((sent_weekly_digest/
        total_emails_sent) * 100, 2) ELSE 0 END AS sent_weekly_rate,
CASE WHEN total_emails_sent > 0 THEN ROUND((email_open/
        total_emails_sent) * 100, 2) ELSE 0 END AS email_open_rate,
CASE WHEN total_emails_sent > 0 THEN ROUND((email_clickthrough/
        total_emails_sent) * 100, 2) ELSE 0 END AS clickthrough_rate,
CASE WHEN total_emails_sent > 0 THEN ROUND((sent_reengagement_email
/ total_emails_sent) * 100, 2) ELSE 0 END AS reengagement_rate
FROM email_metrics;
```

user_id	sent_weekly_rate	email_open_rate	clickthrough_rate	reengagement_rate
0	77.27	22.73	0.00	0.00
4	65.38	19.23	15.38	0.00
8	80.95	14.29	4.76	0.00
11	70.83	20.83	8.33	0.00
17	77.27	18.18	4.55	0.00
19	73.91	21.74	4.35	0.00
20	60.71	28.57	10.71	0.00
22	62.96	25.93	11.11	0.00
30	72.00	24.00	4.00	0.00
49	73.91	21.74	4.35	0.00
59	68.00	20.00	12.00	0.00
64	70.83	20.83	8.33	0.00
66	77.27	22.73	0.00	0.00
67	77.27	22.73	0.00	0.00
78	60.71	25.00	14.29	0.00
80	65.38	26.92	7.69	0.00
83	77.27	18.18	4.55	0.00
86	70.83	16.67	12.50	0.00
98	66.67	25.93	7.41	0.00
101	54.55	27.27	18.18	0.00
108	69.23	30.77	0.00	0.00
117	80.95	19.05	0.00	0.00
120	80.95	19.05	0.00	0.00
124	68.00	24.00	8.00	0.00
128	62.96	22.22	14.81	0.00
134	56.67	30.00	13.33	0.00
136	68.00	24.00	8.00	0.00
138	77.27	22.73	0.00	0.00
140	80.95	14.29	4.76	0.00
145	85.00	15.00	0.00	0.00
150	62.96	22.22	14.81	0.00
155	80.95	19.05	0.00	0.00
163	56.67	33.33	10.00	0.00
170	72.00	24.00	4.00	0.00
171	69.23	23.08	7.69	0.00
172	69.23	23.08	7.69	0.00
173	72.00	28.00	0.00	0.00
175	64.29	21.43	14.29	0.00
179	78.26	21.74	0.00	0.00

⇒ Most of the users are engaged with sent weekly digests which means users as delivered a digest email showing relevant conversations from the previous day.

Results:

While doing this project, I have learned about advanced SQL queries to extract insights from the relational database which helps to identify patterns and trends in user behaviours.

From the Job data Analysis case study, some of the insights are derived are as follows:

- Most of the jobs reviewed per hour were on 28th November.
- Daily metrics would be preferred to identify the trends and patterns for the short-term data.
- The Persian language is widely used for the content.

From the Operational Analytics for investigating metrics spike, some of the insights are derived are as follows:

- The weekly user engagement started decreasing from week 31 of the year 2014.
- The growth of the active users over time for a product increased from 160 to 9381.
- Macbook Pro, Lenovo Thinkpad and Iphone 5 are the devices where most of the users are active weekly.
- Most of the users are engaged with sent weekly digests.

Overall, this project helped us to derive some valuable insights to make the product better and enhance user interactions and engagement with the product.