Operation Analytics and Investigating Metric Spike

Project Description:
This project aims to improve operational efficiency by analyzing data to identify improvement areas. As Lead Data Analyst, I will use SQL to explore metric spikes and evaluate performance. The goal is to deliver actionable insights for optimizing operations and supporting data-driven decisions.

Approach:
In this project, I started by understanding the company's objectives and data. I identified key metrics for analysis and used SQL to explore areas like jobs reviewed per hour and user engagement. I refined my queries iteratively, focusing on detecting patterns and anomalies. Finally, I compiled my findings into a report with insights and recommendations to enhance operational efficiency.

Tech-Stack Used:
MySQL WorkBench v8.0.40 was utilized as part of the tech stack, proving to be an excellent tool for database querying due to its easy access, simple configuration, intuitive graphical user interface, and effective troubleshooting support.

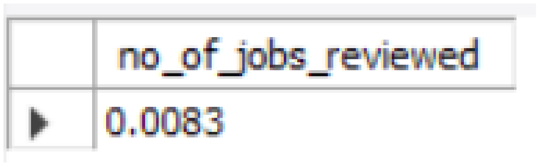
Case Study 1: Job Data Analysis

1. **Jobs Reviewed Over Time:**
    - **Objective:** Calculate the number of jobs reviewed per hour for each day in November 2020.
    - **Task:** Write an SQL query to calculate the number of jobs reviewed per hour for each day in November 2020.

Code (distinct):

```
select count(distinct job_id)/(30*24) as no_of_jobs_reviewed

from job_data;
```

Output:

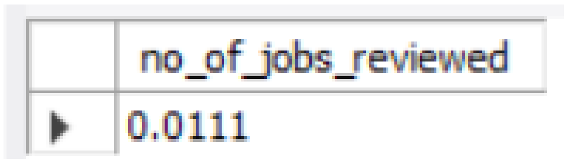


```
no_of_jobs_reviewed
0.0083
```

Code (non-distinct) :

```
select count(job_id)/(30*24) as no_of_jobs_reviewed

from job_data;
```

Output:



```
no_of_jobs_reviewed
0.0111
```

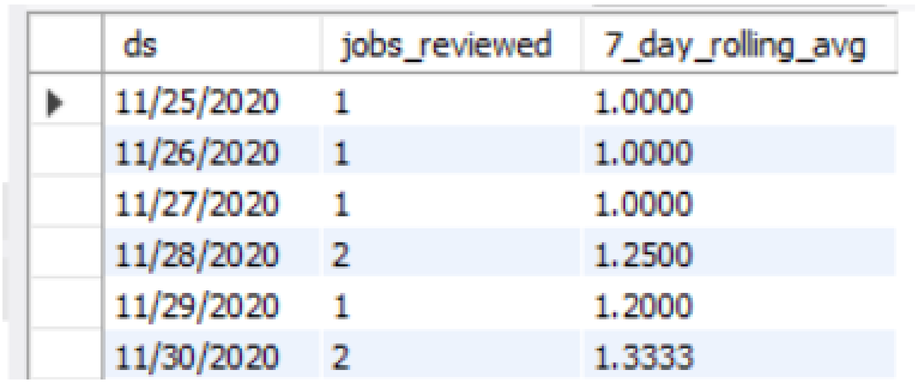
2. Throughput Analysis:

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

Code:

```
select ds,

count(distinct job_id) as jobs_reviewed,

avg(count(distinct job_id)) OVER (ORDER BY ds ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) as 7_day_rolling_avg

from job_data

group by ds

order by ds;
```

Output:



ds	jobs_reviewed	7_day_rolling_avg
11/25/2020	1	1.0000
11/26/2020	1	1.0000
11/27/2020	1	1.0000
11/28/2020	2	1.2500
11/29/2020	1	1.2000
11/30/2020	2	1.3333

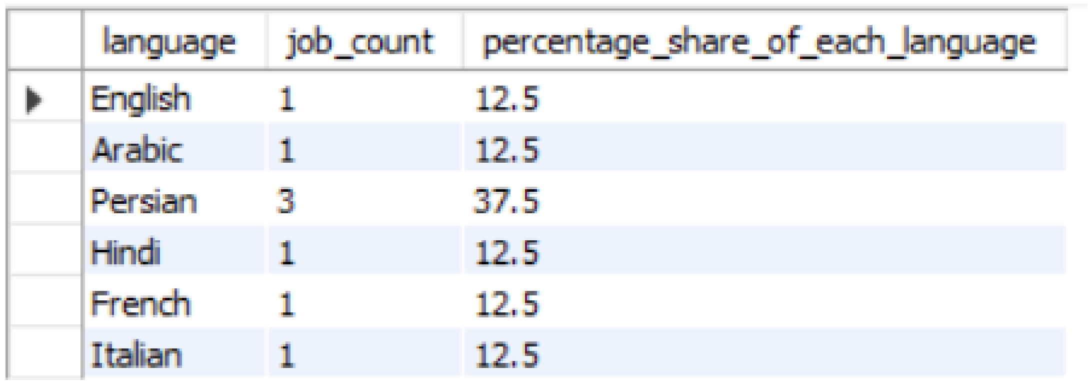
3. Language Share Analysis:

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

Code:

```
select language,
    COUNT(*) AS job_count,
    ROUND((COUNT(*) * 100.0 / SUM(COUNT(*)) over ()), 1) as
percentage_share_of_each_language
from job_data
group by language;
```

Output:



language	job_count	percentage_share_of_each_language
English	1	12.5
Arabic	1	12.5
Persian	3	37.5
Hindi	1	12.5
French	1	12.5
Italian	1	12.5

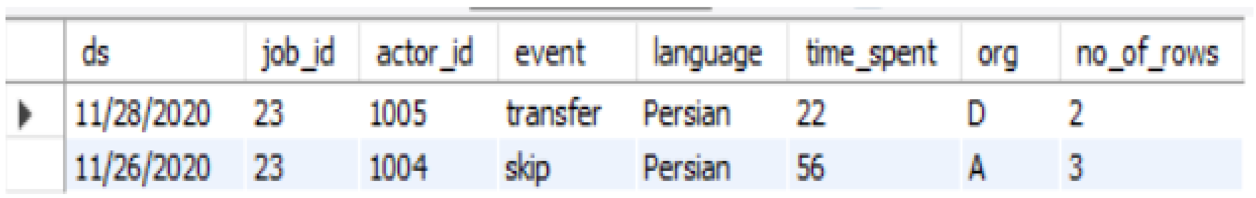
4. Duplicate Rows Detection:

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

Code:

```
select * from (select *, row_number() over (partition by job_id) as
no_of_rows

from job_data) e

where no_of_rows>1;
```

Output:



ds	job_id	actor_id	event	language	time_spent	org	no_of_rows
11/28/2020	23	1005	transfer	Persian	22	D	2
11/26/2020	23	1004	skip	Persian	56	A	3

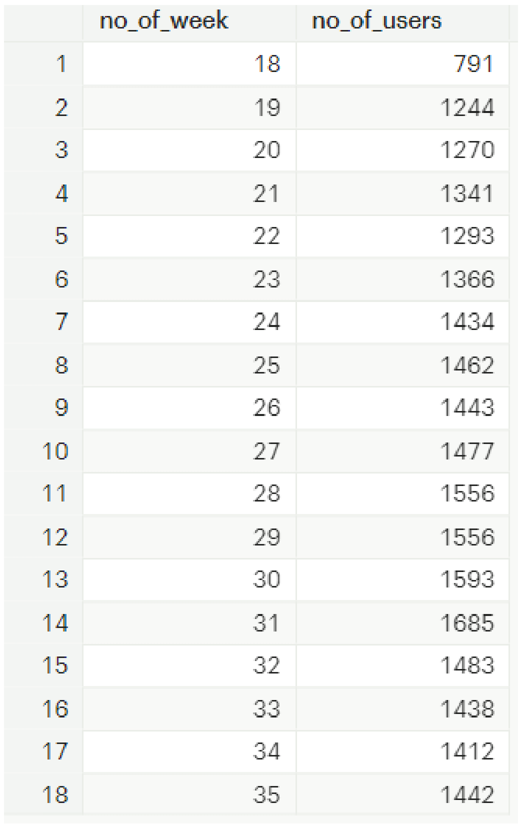
Case Study 2: Investigating Metric Spike

1. **Weekly User Engagement:**
    - **Objective:** Measure the activeness of users on a weekly basis.
    - **Task:** Write an SQL query to calculate the weekly user engagement.

Code:

```
select extract(week from occurred_at) as no_of_week,

count(distinct user_id) as no_of_users

from events

group by no_of_week;
```

Output:



	no_of_week	no_of_users
1	18	791
2	19	1244
3	20	1270
4	21	1341
5	22	1293
6	23	1366
7	24	1434
8	25	1462
9	26	1443
10	27	1477
11	28	1556
12	29	1556
13	30	1593
14	31	1685
15	32	1483
16	33	1438
17	34	1412
18	35	1442

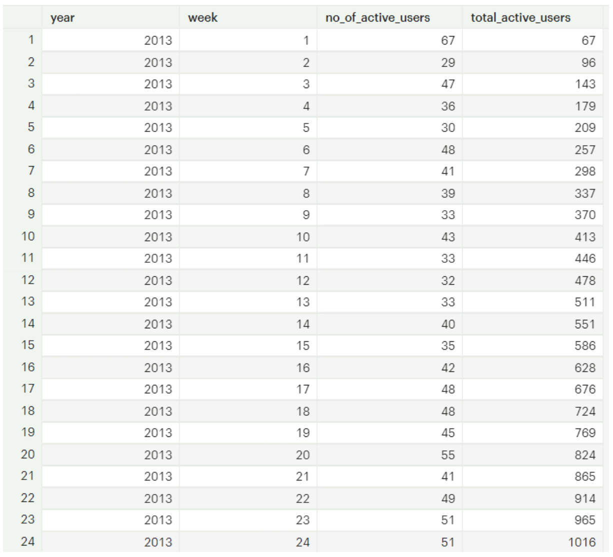
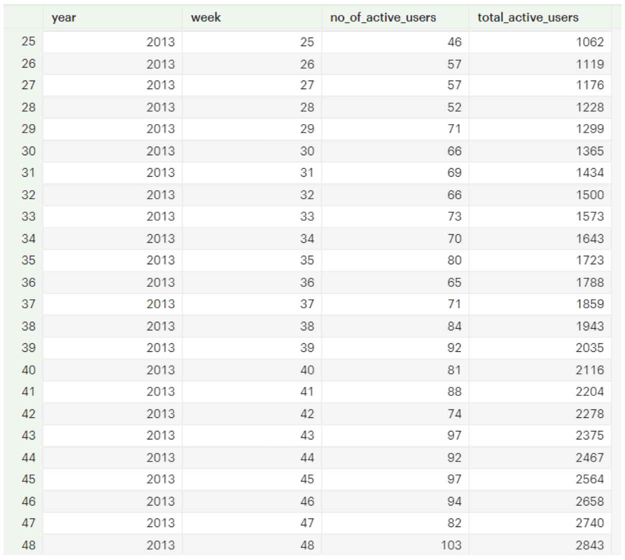
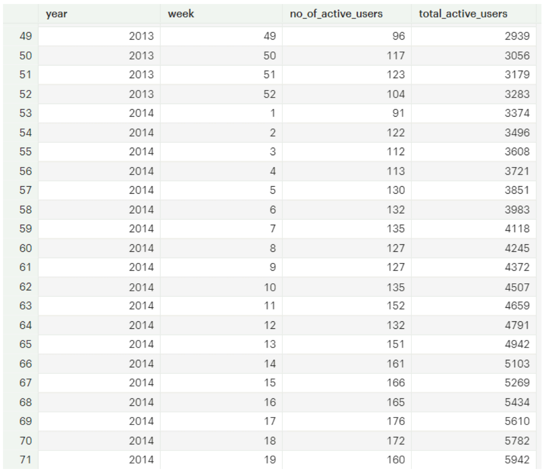
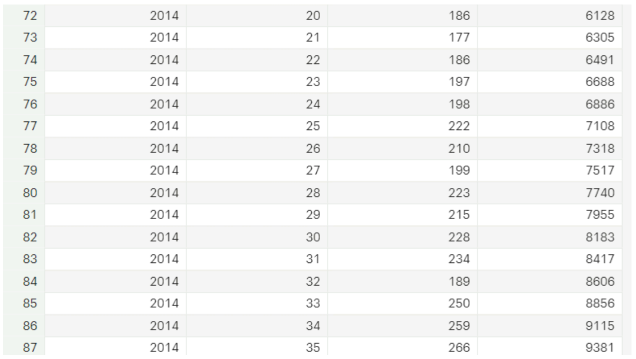
2. User Growth Analysis:

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

Code:

```
select

  year,week,no_of_active_users,

  SUM(no_of_active_users)OVER(ORDER BY year, week ROWS BETWEEN
UNBOUNDED PRECEDING AND CURRENT ROW) AS total_active_users

from (

  select

  extract (year from a.activated_at) as year,

  extract (week from a.activated_at) as week,

  count(distinct user_id) as no_of_active_users

from users a

WHERE state = 'active'

group by year,week

order by year,week

) a;
```

Output:

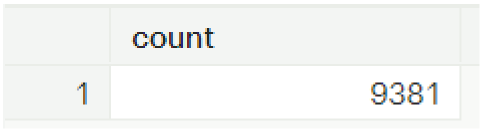





	year	week	no_of_active_users	total_active_users
1	2013	1	67	67
2	2013	2	29	96
3	2013	3	47	143
4	2013	4	36	179
5	2013	5	30	209
6	2013	6	48	257
7	2013	7	41	298
8	2013	8	39	337
9	2013	9	33	370
10	2013	10	43	413
11	2013	11	33	446
12	2013	12	32	478
13	2013	13	33	511
14	2013	14	40	551
15	2013	15	35	586
16	2013	16	42	628
17	2013	17	48	676
18	2013	18	48	724
19	2013	19	45	769
20	2013	20	55	824
21	2013	21	41	865
22	2013	22	49	914
23	2013	23	51	965
24	2013	24	51	1016
25	2013	25	46	1062
26	2013	26	57	1119
27	2013	27	57	1176
28	2013	28	52	1228
29	2013	29	71	1299
30	2013	30	66	1365
31	2013	31	69	1434
32	2013	32	66	1500
33	2013	33	73	1573
34	2013	34	70	1643
35	2013	35	80	1723
36	2013	36	65	1788
37	2013	37	71	1859
38	2013	38	84	1943
39	2013	39	92	2035
40	2013	40	81	2116
41	2013	41	88	2204
42	2013	42	74	2278
43	2013	43	97	2375
44	2013	44	92	2467
45	2013	45	97	2564
46	2013	46	94	2658
47	2013	47	82	2740
48	2013	48	103	2843
49	2013	49	96	2939
50	2013	50	117	3056
51	2013	51	123	3179
52	2013	52	104	3283
53	2014	1	91	3374
54	2014	2	122	3496
55	2014	3	112	3608
56	2014	4	113	3721
57	2014	5	130	3851
58	2014	6	132	3983
59	2014	7	135	4118
60	2014	8	127	4245
61	2014	9	127	4372
62	2014	10	135	4507
63	2014	11	152	4659
64	2014	12	132	4791
65	2014	13	151	4942
66	2014	14	161	5103
67	2014	15	166	5269
68	2014	16	165	5434
69	2014	17	176	5610
70	2014	18	172	5782
71	2014	19	160	5942
72	2014	20	186	6128
73	2014	21	177	6305
74	2014	22	186	6491
75	2014	23	197	6688
76	2014	24	198	6886
77	2014	25	222	7108
78	2014	26	210	7318
79	2014	27	199	7517
80	2014	28	223	7740
81	2014	29	215	7955
82	2014	30	228	8183
83	2014	31	234	8417
84	2014	32	189	8606
85	2014	33	250	8856
86	2014	34	259	9115
87	2014	35	266	9381

Code (counting users from user table having state as active):

```
select count(*) from users

where state = 'active';
```

Output:



	count
1	9381

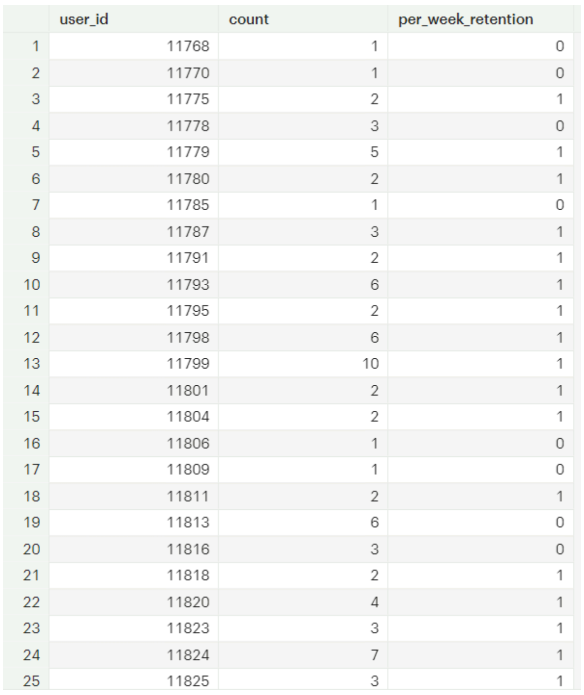
3. Weekly Retention Analysis:

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

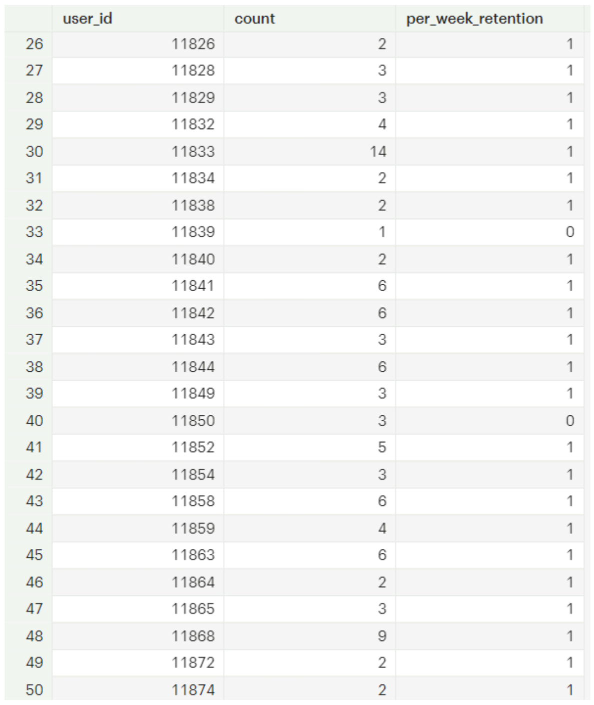
Code:

```
SELECT distinct user_id,

COUNT(user_id), SUM(CASE WHEN retention_week = 1 Then 1 Else 0 END)
as per_week_retention

FROM (

SELECT a.user_id, a.signup_week, b.engagement_week,
b.engagement_week - a.signup_week as retention_week

FROM

(

(SELECT distinct user_id, extract(week from occurred_at) as signup_week
from events

WHERE event_type = 'signup_flow'

and event_name = 'complete_signup') a

LEFT JOIN

(SELECT distinct user_id, extract (week from occurred_at) as
engagement_week FROM events

where event_type = 'engagement') b

on a.user_id = b.user_id)) d

group by user_id

order by user_id;
```

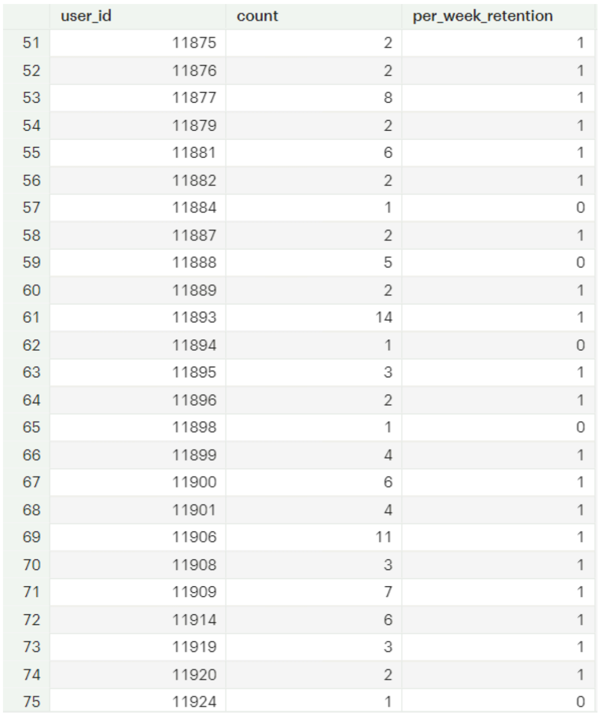
Output (first 100):



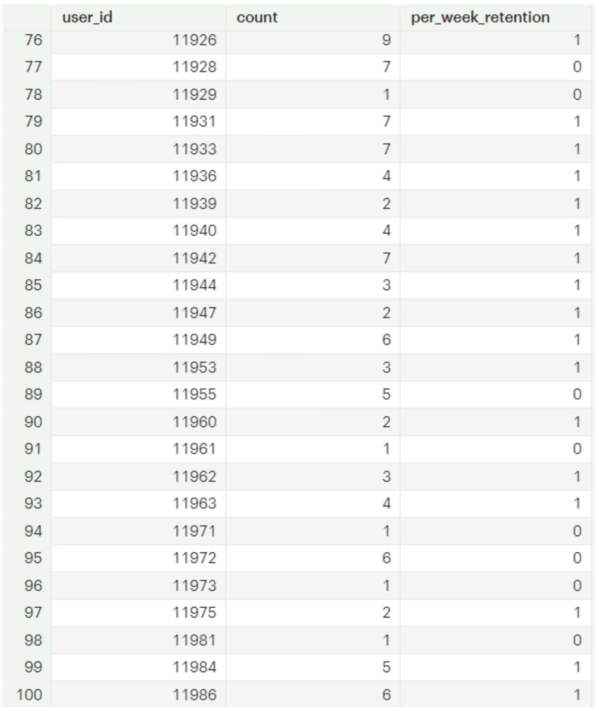
	user_id	count	per_week_retention
1	11768	1	0
2	11770	1	0
3	11775	2	1
4	11778	3	0
5	11779	5	1
6	11780	2	1
7	11785	1	0
8	11787	3	1
9	11791	2	1
10	11793	6	1
11	11795	2	1
12	11798	6	1
13	11799	10	1
14	11801	2	1
15	11804	2	1
16	11806	1	0
17	11809	1	0
18	11811	2	1
19	11813	6	0
20	11816	3	0
21	11818	2	1
22	11820	4	1
23	11823	3	1
24	11824	7	1
25	11825	3	1



	user_id	count	per_week_retention
26	11826	2	1
27	11828	3	1
28	11829	3	1
29	11832	4	1
30	11833	14	1
31	11834	2	1
32	11838	2	1
33	11839	1	0
34	11840	2	1
35	11841	6	1
36	11842	6	1
37	11843	3	1
38	11844	6	1
39	11849	3	1
40	11850	3	0
41	11852	5	1
42	11854	3	1
43	11858	6	1
44	11859	4	1
45	11863	6	1
46	11864	2	1
47	11865	3	1
48	11868	9	1
49	11872	2	1
50	11874	2	1



	user_id	count	per_week_retention
51	11875	2	1
52	11876	2	1
53	11877	8	1
54	11879	2	1
55	11881	6	1
56	11882	2	1
57	11884	1	0
58	11887	2	1
59	11888	5	0
60	11889	2	1
61	11893	14	1
62	11894	1	0
63	11895	3	1
64	11896	2	1
65	11898	1	0
66	11899	4	1
67	11900	6	1
68	11901	4	1
69	11906	11	1
70	11908	3	1
71	11909	7	1
72	11914	6	1
73	11919	3	1
74	11920	2	1
75	11924	1	0



	user_id	count	per_week_retention
76	11926	9	1
77	11928	7	0
78	11929	1	0
79	11931	7	1
80	11933	7	1
81	11936	4	1
82	11939	2	1
83	11940	4	1
84	11942	7	1
85	11944	3	1
86	11947	2	1
87	11949	6	1
88	11953	3	1
89	11955	5	0
90	11960	2	1
91	11961	1	0
92	11962	3	1
93	11963	4	1
94	11971	1	0
95	11972	6	0
96	11973	1	0
97	11975	2	1
98	11981	1	0
99	11984	5	1
100	11986	6	1

Full Output Drive Link:

without week 18 (weekly retention).csv

Code (Specifying week number as 18):

```
SELECT distinct user_id,
COUNT(user_id), SUM(CASE WHEN retention_week = 1 Then 1 Else 0 END)
as per_week_retention
FROM
(
SELECT
a.user_id, a.signup_week, b.engagement_week, b.engagement_week -
a.signup_week as retention_week
FROM
(
(SELECT distinct user_id, extract(week from occurred_at) as signup_week
from events
WHERE event_type = 'signup_flow'
and event_name = 'complete_signup'
and extract(week from occurred_at) = 18) a
LEFT JOIN
(SELECT distinct user_id, extract (week from occurred_at) as
engagement_week FROM events
where event_type = 'engagement') b
on a.user_id = b.user_id) ) d
group by user_id
order by user_id;
```

Full Output Drive Link:
[week 18 (weekly retention).csv]

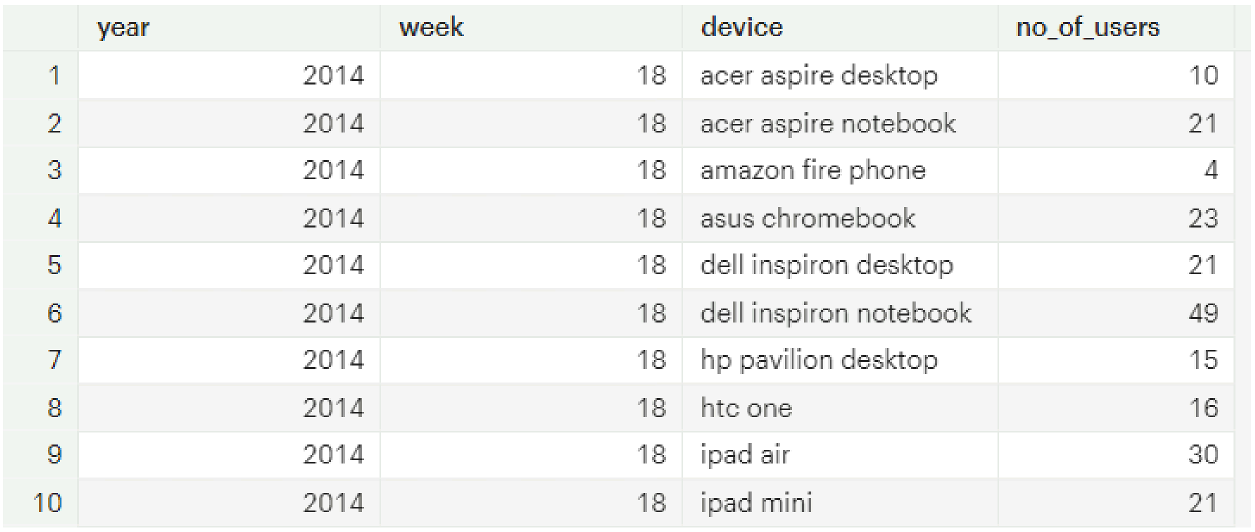
4. Weekly Engagement Per Device:

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

Code:

```
SELECT
  extract(year from occurred_at) as year,
  extract(week from occurred_at) as week,
  device,
  COUNT(distinct user_id) as no_of_users
FROM
  events
where event_type = 'engagement'
GROUP by 1,2,3
order by 1,2,3;
```

Output (first 10):



	year	week	device	no_of_users
1	2014	18	acer aspire desktop	10
2	2014	18	acer aspire notebook	21
3	2014	18	amazon fire phone	4
4	2014	18	asus chromebook	23
5	2014	18	dell inspiron desktop	21
6	2014	18	dell inspiron notebook	49
7	2014	18	hp pavilion desktop	15
8	2014	18	htc one	16
9	2014	18	ipad air	30
10	2014	18	ipad mini	21

Full Output Drive Link:
weekly engagement.csv

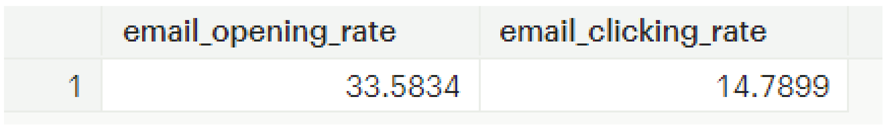
5. Email Engagement Analysis:

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

Code:

```
SELECT

  100.0*SUM(CASE when email_cat = 'email_opened' then 1 else 0
end)/SUM(CASE when email_cat = 'email_sent' then 1 else 0 end)

  as email_opening_rate,

  100.0*SUM(CASE when email_cat = 'email_clicked' then 1 else 0
end)/SUM(CASE when email_cat = 'email_sent' then 1 else 0 end)

   as email_clicking_rate

FROM (SELECT   *, CASE

    WHEN action in ('sent_weekly_digest','sent_reengagement_email')

      then 'email_sent'

    WHEN action in ('email_open')

      then 'email_opened'

    WHEN action in ('email_clickthrough')

      then 'email_clicked'

  end as email_cat from email_events ) a;
```

Output:



```
     email_opening_rate   email_clicking_rate
1               33.5834               14.7899
```

Result:

Through this project, I was able to identify key operational improvements and gain a deeper understanding of the company’s performance metrics. By using advanced SQL, I uncovered patterns and anomalies, which enhanced my decision-making abilities. This experience not only sharpened my analytical skills but also reinforced the importance of data-driven strategies in optimizing operations.