Operation Analytics and Investigating Metric Spike

by Suraj Beloshe

Powered by

trainity

Project Description

1. Data for two cases has been provided in Excel sheets
2. Need to prepare tables and perform analysis
3. With help of this data need to find the following insights which will help the other departments

Case Study 1 (Job Data)

- Number of jobs reviewed
- Throughput
- Percentage share of each language
- Duplicate rows

Case Study 2 (Investigating metric spike)

- User Engagement
- User Growth
- Weekly Retention
- Weekly Engagement
- Email Engagement:

Approach

Case Study 1 - Data is given in an excel file. First, we have to create a database in SQL and then create a table. Once the Table is created we need to observe data in view to answer questions asked by various department

Case Study 2 - There is vast data in three different Excel files.
So understanding the DATA is the first approach. then extracting the same in My Sql. And think that how can answer the questions given. Also need to focus on the connections of tables with each other.

Tech-Stack Used

My SQL workbench version 8.0.32

- MySQL is one of the most popular and widely used SQL databases.
- User-friendly tool for data analysts to work with databases
- It includes a visual SQL query builder, built-in data visualization tools, and collaboration features with other stakeholders.
- The tool also integrates with other commonly used data analysis tools such as Python and R

Microsoft office 365

- Amazing Tool offered by Microsoft.
- Has advance Function which enable power as Data analyst
- Variouse charts helps to visualise data in effective manner

Insights

Case Study 1 (Job Data)

A). Number of jobs reviewed: Amount of jobs reviewed over time.

Task: Calculate the number of jobs reviewed per hour per day for November 2020?

casestudy_1*

Limit to 1000 rows

```sql
-- A) Number of jobs reviewed:  Calculate the number of jo

SELECT ds AS Dates,
       ROUND(COUNT(job_id) / SUM(time_spent) * 3600, 0)
       AS JobsReviewed_Hour_Day
FROM job_data
WHERE ds BETWEEN '2020-11-01' AND '2020-11-30'
GROUP BY ds;
```

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

Dates	JobsReviewed_Hour_Day
2020-11-30	180
2020-11-29	180
2020-11-28	218
2020-11-27	35
2020-11-26	64
2020-11-25	80

- To Count, the number of job_ids for each day selected "ds" column and used Count Function. To calculate the total time spent reviewing jobs for each day used SUM Function. Then Divides the total number of jobs by the total review time and multiply by 3600 to convert it into hours.
- to get the complete number as a day used Round Function result is rounded to 0 decimal places.
- And result given the number of jobs reviewed per day
- On 28 Nov 2020 the total number of job reviews was higher i.e. 218 Jobs per day per hours

Insights

Case Study 1 (Job Data)

B). Throughput: It is the no. of events happening per second.

Task: Let's say the above metric is called throughput. Calculate the 7-day rolling average of throughput? For throughput, do you prefer daily metric or 7-day rolling and why?

casestudy_1*

Limit to 1000 rows

```sql
-- Throughput: It is the no. of events happening per second.
-- Your task: Let's say the above metric is called throughput.
-- Calculate 7 day rolling average of throughput? For throughput,
-- do you prefer daily metric or 7-day rolling and why?

SELECT ds,
       COUNT(event) / SUM(time_spent) AS daily_throughput,
       AVG(COUNT(event) / SUM(time_spent))
       OVER (ORDER BY ds ROWS BETWEEN 6 PRECEDING AND CURRENT ROW)
       AS seven_day_avg_throughput
FROM job_data
GROUP BY ds;
```

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

ds	daily_throughput	seven_day_avg_throughput
2020-11-25	0.0222	0.02222222
2020-11-26	0.0179	0.02003968
2020-11-27	0.0096	0.01656492
2020-11-28	0.0606	0.02757520
2020-11-29	0.0500	0.03206016
2020-11-30	0.0500	0.03505013

- With the given definition of Throughput, we calculate the throughput per day. for that, we divided the total events happened by the total time spent
- we got daily throughput
- we get the average no of events per second further takes the daily throughput values calculated and used avg function with an over clause to calculate the average of the current day and the previous six days
- This gives an idea of the past seven days.
- In general, the 7-day rolling metric is a better representation of the throughput because it takes into account a longer time period and smooths out the fluctuations in the daily metric.

Insights

Case Study 1 (Job Data)

C) Percentage share of each language: Share of each language for different contents.

Task: Calculate the percentage share of each language in the last 30 days?

casestudy_1*

Limit to 1000 rows

```
-- Percentage share of each language: Share of each language
-- Your task: Calculate the percentage share of each language

SELECT language AS Languages,
CONCAT(ROUND(100 * COUNT(*)/total, 2), '%')
AS percentage
FROM job_data
CROSS JOIN (SELECT COUNT(*) AS total FROM job_data) sub
GROUP BY language, total;
```

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

Languages	percentage
English	12.50%
Arabic	12.50%
Persian	37.50%
Hindi	12.50%
French	12.50%
Italian	12.50%

- We have been told to derive the percentage of each language in the last 30 days but the data is available for 6 days only.
- So considering a sample we proceed with the available data we count of each unique language using the COUNT function.
- Then, we use the CONCAT function to concatenate the percentage value (rounded to 2 decimal places) with the "%" symbol.
- To calculate the percentage, we are using a subquery to find the total count of rows in the "job_data" table, and then dividing the count of each unique language by this total count. This is done using a CROSS JOIN between the "job_data" table and the subquery.
- Finally, we group the results by the "language" column and the total count, which gives us the percentage of each language used in the "job_data" table.

Insights

Case Study 1 (Job Data)

D) Duplicate rows: Rows that have the same value present in them.

Task: Let's say you see some duplicate rows in the data. How will you display duplicates from the table?

casestudy_1*

Limit to 1000 rows

```sql
-- Duplicate rows: Rows that have the same value present in them.
-- Your task: Let's say you see some duplicate rows in the data.
-- How will you display duplicates from the table?

SELECT *
FROM job_data
WHERE (job_id, language) IN (
    SELECT job_id, language
    FROM job_data
    GROUP BY job_id, language
    HAVING COUNT(*) > 1
);
```

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

job_id	actor_id	event	language	time_spent	org	ds
23	1003	decision	Persian	20	C	2020-11-29
23	1005	transfer	Persian	22	D	2020-11-28
23	1004	skip	Persian	56	A	2020-11-26

- we selected all data from a given table where the combination of job_id and language appears more than once in the same table.
- We used the query is using a subquery that groups the data by job_id and language,
- And then filters out any groups that have a count of less than 2 (i.e., only appear once).
- The outer query then uses the resulting list of job_id and language combinations to retrieve all the corresponding data from the table.
- And in result, we found that duplication in job id which is matching with

Insights

Case Study 2 (Investigating metric spike)

A). User Engagement: To measure the activeness of a user. Measuring if the user finds quality in a product/service.

Task: Calculate the weekly user engagement?

casestudy_2*

Limit to 1000 rows

```sql
-- Case Study 2 (Investigating metric spike)
-- A. User Engagement: To measure the activeness
-- Task: Calculate the weekly user engagement?

SELECT
    EXTRACT(WEEK FROM occurred_at) AS Week_no,
    COUNT(DISTINCT user_id) as user_engagement
FROM
    `table-2 events`
GROUP BY Week_no;
```

Result Grid | Filter Rows: | Export: | Wrap Ce

Week_no	user_engagement
17	740
18	1260
19	1287
20	1351
21	1299
22	1381
23	1446
24	1471
25	1459
26	1509
27	1573
28	1577
29	1607

- We selected the week number (extracted from the occurred_at column) and the count of unique users that have engaged with a platform based on an events table named table-2 events.
- The EXTRACT() function is used to extract the week number from the occurred_at column.
- The COUNT() function with DISTINCT keyword is used to count only the unique users who engaged with the platform.
- The GROUP BY clause is used to group the result set by the week number.
- Finally, the result of this query will be a list of week numbers and the corresponding count of unique users who engaged with the platform during each week.

Insights

Case Study 2 (Investigating metric spike)

B). User Growth: Amount of users growing over time for a product.

Task: Calculate the user growth for product?

```sql
-- B. User Growth: Amount of users growing over time for
-- Your task: Calculate the user growth for product?

SELECT
  year_no,
  week_no,
  no_of_active_users,
  SUM(no_of_active_users) OVER(ORDER BY year_no, week_no
FROM
(
SELECT
  EXTRACT(year from activated_at) as year_no,
  EXTRACT(week from activated_at) as week_no,
  COUNT(distinct user_id) as no_of_active_users
FROM
  project_3db.`table-1 users`
```

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

year_no	week_no	no_of_active_users	CUM_ACTIVE_USERS
2013	0	23	23
2013	1	30	53
2013	2	48	101
2013	3	36	137
2013	4	30	167

- First let's select the year, week, no of active users, and cumulative active users from a subquery.
- The subquery is aggregating data from table-1 users. It is extracting the year and week number from the activated_at column, which presumably represents the time when a user became active on the platform.
- The COUNT() function with DISTINCT keyword is used to count only the unique users who became active each week.
- By using Group by and Order by we set results to come in ascending order.
- used outer query to select the year, week, and a number of active users from the subquery, as well as the cumulative sum of active users over time, which is calculated using the SUM() function with the OVER() clause.
- The OVER() clause is used to specify the window over which the cumulative sum should be calculated.
- Finally gets the desired result.

Insights

Case Study 2 (Investigating metric spike)

C). Weekly Retention: Users getting retained weekly after signing-up for a product.

Task: Calculate the weekly retention of users-sign up cohort?

casestudy_2*

Limit to 1000 rows

```
-- C. Weekly Retention: Users getting retained weekly after signi
-- Your task: Calculate the weekly retention of users-sign up coh

SELECT first_login_week,
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,
```

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

first_login_week	total_users	week_0	week_1	week_2	week_3	week_4	week_5
17	6714	887	975	666	518	397	362
18	5466	1010	688	504	384	312	270
19	3457	676	522	326	283	208	184
20	2631	549	402	315	223	167	140

Result 10

- Weekly retention will be referred to as the login of users since their first log-in. So we need to count the number of logins for each user in each week.
- Retrieved the number of logins of users by week, starting from their first login week. It first selects the user_id and the login_week from the events table where the event_name is 'login' and groups them by user_id and login_week.
- Then, it joins the result with a subquery that retrieves the first_login_week for each user by finding the minimum login_week where the event_name is 'login'.
- Used subquery to calculate the week_number by subtracting the first_login_week from the login_week. In the outer query, the week_number is used in the CASE statement to count the number of logins for each user by week.
- Finally, groups the results by first_login_week and orders the results by the same column.

Insights

Case Study 2 (Investigating metric spike)

D). Weekly Engagement: To measure the activeness of a user. Measuring if the user finds quality in a product/service weekly.

Task: Calculate the weekly engagement per device?

casestudy_2*

Limit to 1000 rows

```sql
-- D. Weekly Engagement: To measure the activeness of a user. Measuring if t
-- Your task: Calculate the weekly engagement per device?

SELECT
  device,
  EXTRACT(year FROM occurred_at) AS year,
  EXTRACT(week FROM occurred_at) AS week_no,
  COUNT(DISTINCT user_id) AS weekly_engagements
FROM `table-2 events`
WHERE event_type = 'engagement'
GROUP BY device, year, week_no
ORDER BY device, year, week_no;
```

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

device	year	week_no	weekly_engagements
acer aspire desktop	2014	17	9
acer aspire desktop	2014	18	26
acer aspire desktop	2014	19	23
acer aspire desktop	2014	20	23
acer aspire desktop	2014	21	29
acer aspire desktop	2014	22	25
acer aspire desktop	2014	23	22
acer aspire desktop	2014	24	24
acer aspire desktop	2014	25	28
acer aspire desktop	2014	26	29

- To get week extracted data from occured_at Coloum with year and week
- then count unique user id as weekly engagement with where condition on event type = engagement
- Further Group by Device, Year and no of week
- and Ordered by Device, Year and no of week

Insights

Case Study 2 (Investigating metric spike)

E). Email Engagement: Users engaging with the email service.

Task: Calculate the email engagement metrics?

casestudy_2*

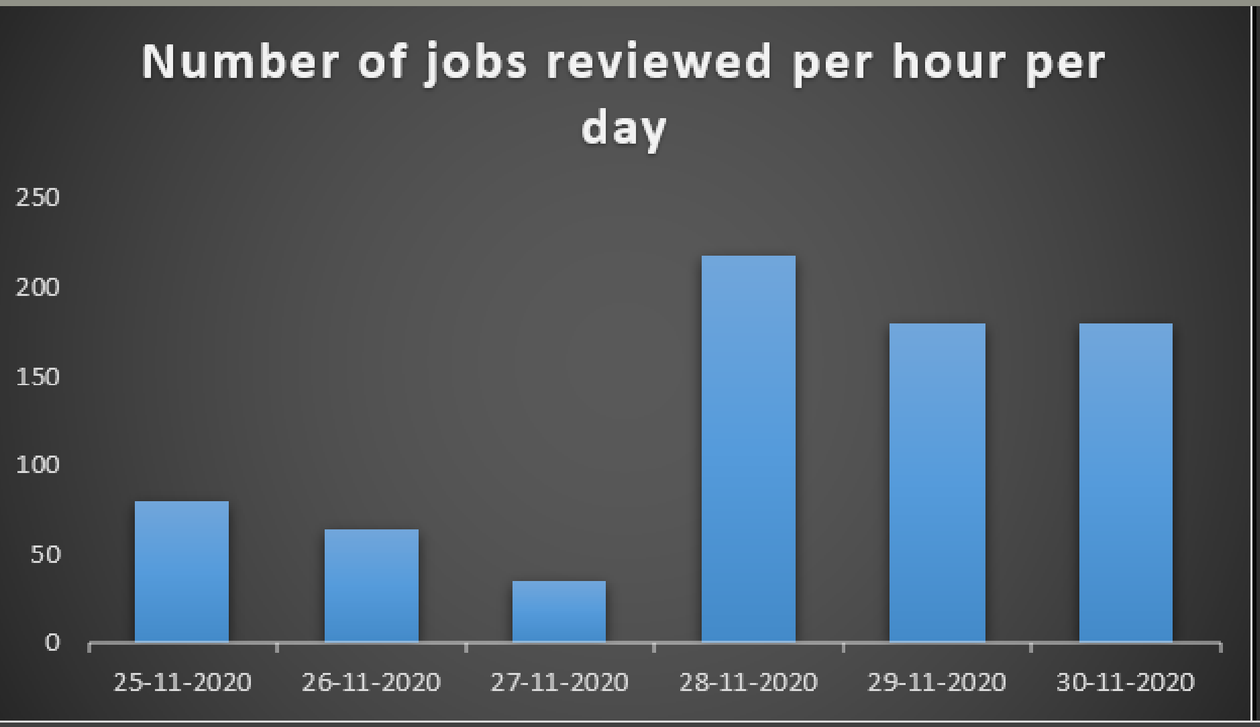
Limit to 1000 rows

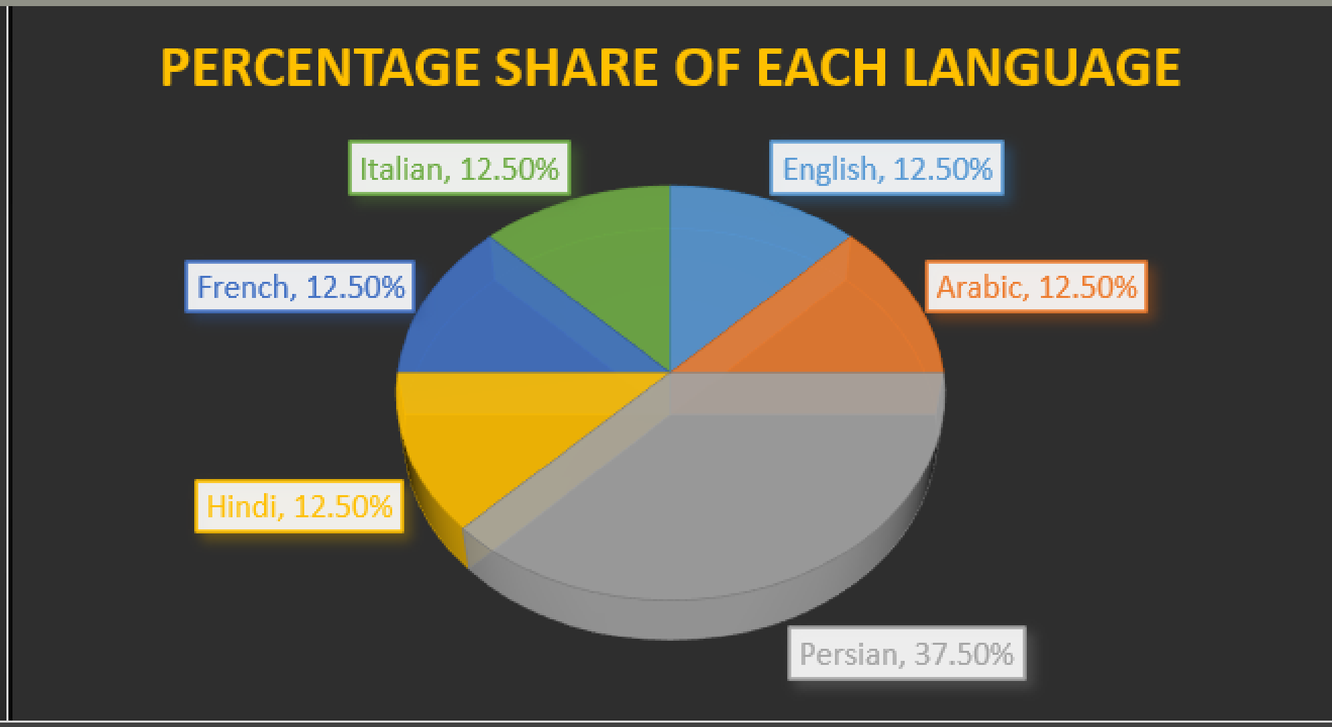
```
-- E. Email Engagement: Users engaging with the email service.
-- Your task: Calculate the email engagement metrics?

WITH email_metrics AS (
    SELECT
        SUM(CASE WHEN action = 'email_open' THEN 1 ELSE 0 END) AS Mail_Open,
        SUM(CASE WHEN action IN ('sent_reengagement_email', 'sent_weekly_digest') THE
        SUM(CASE WHEN action = 'email_clickthrough' THEN 1 ELSE 0 END) AS Mail_Click
    FROM `table-3 email_events`
)
SELECT
    Mail_Open,
    Mail_Click,
    Mail_Sent,
    ROUND((Mail_Open * 100.0 / Mail_Sent), 2) AS "% of Email Open",
    ROUND((Mail_Click * 100.0 / Mail_Sent), 2) AS "% of EMail Click"
FROM email_metrics;
```

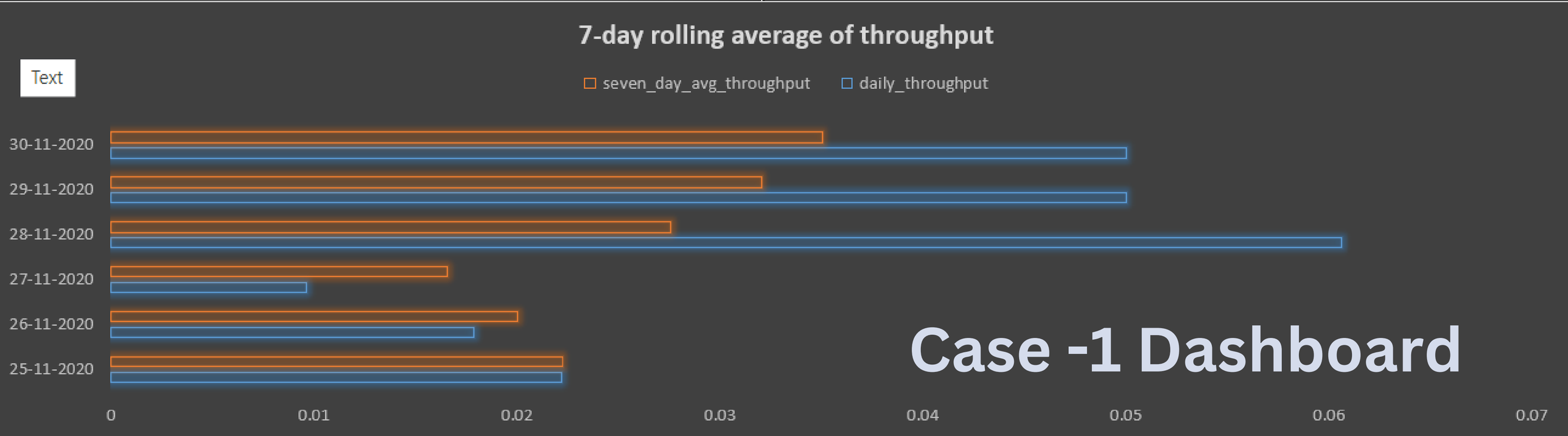
Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Mail_Open	Mail_Click	Mail_Sent	% of Email Open	% of EMail Click
20459	9010	60920	33.58	14.79

- To Calculate Email Engagement Metrics we need to find out the all detail related to email
- Table 3 was Containing actions such as mail open, click, and sent
- So we create a temporary table called **email_metrics** using a common table expression (CTE). The CTE uses three different **SUM** functions to calculate the number of times certain email events occurred
- Further selects the data from the email_metrics table and performs some additional calculations to get our desired metrics
- Finally, we get the effectiveness of the email campaign by showing as per email metrics.

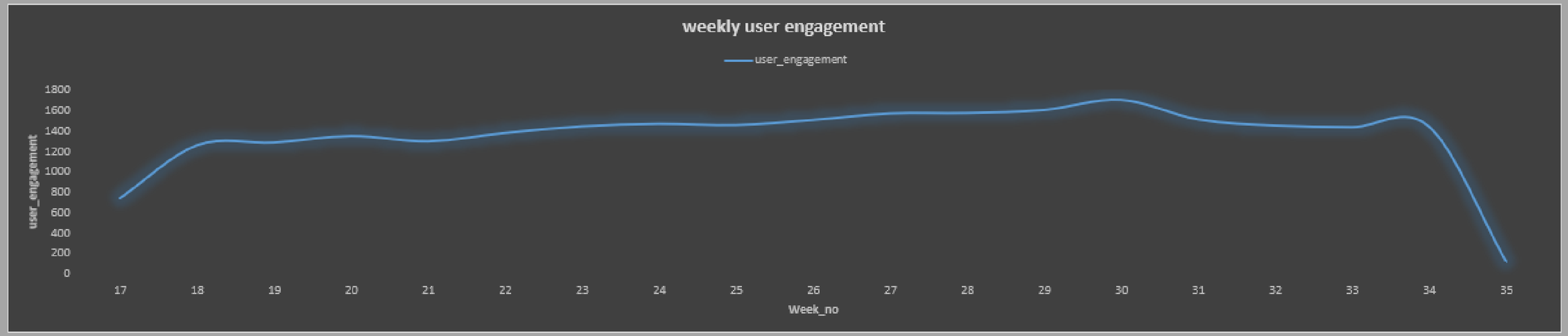


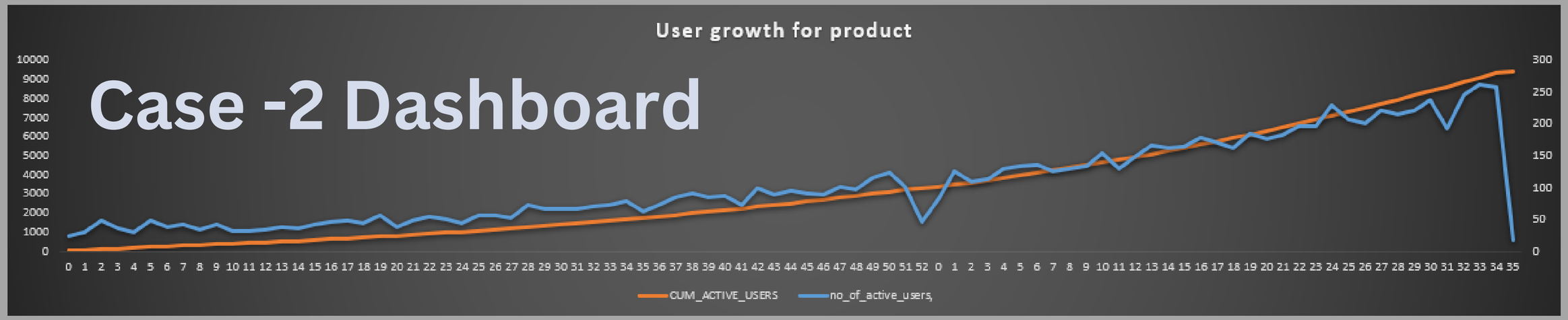


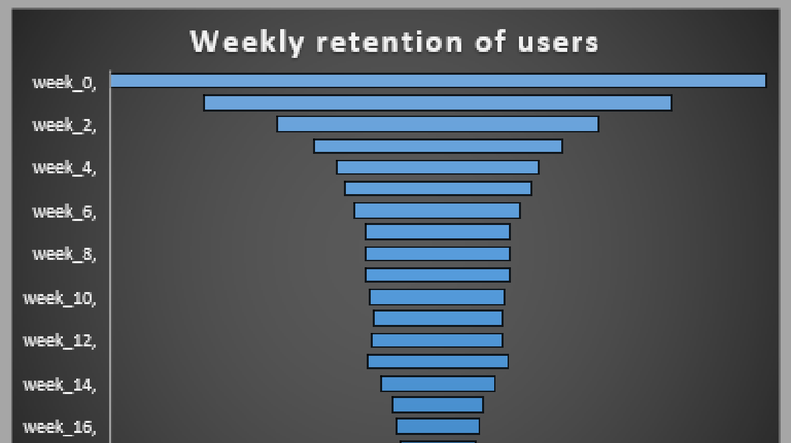


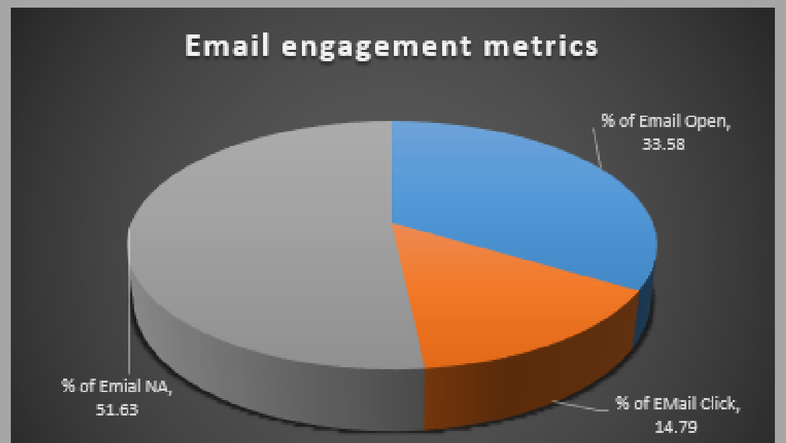
Case -1 Dashboard

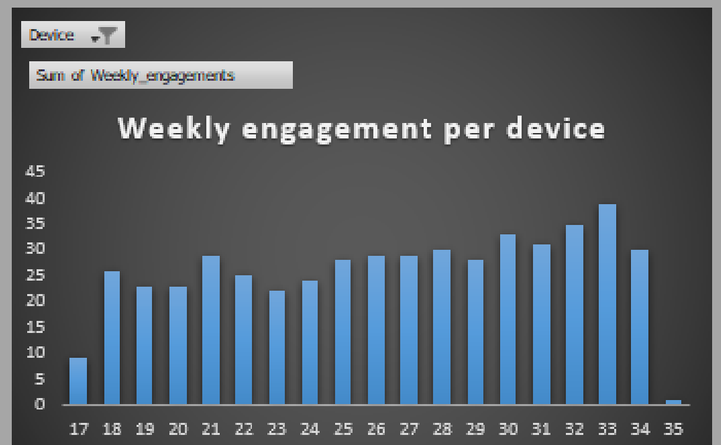
Case -2 Dashboard











Result

- I Learn Basic SQL syntax which I used to perform various operations on tables such as selecting, grouping, and filtering data.
- I understood Data aggregation such as COUNT, and SUM to summarize data. also, Type of Joins, The project uses INNER JOIN to combine data from different tables based on a common field.
- Subqueries: I used subqueries to create derived tables to perform further operations.
- Window functions: used to perform calculations on a specific window of rows within a result set.
- Date and time functions: uses various date and time functions to extract specific information from date and time fields.
- Data type conversion: I understand data type conversion functions to convert data from one data type to another.
- Common Table Expressions (CTEs): this project helps me to understand such advanced concepts I use to create temporary result sets that can be used in subsequent SQL statements.
- CASE statements: used CASE statements to conditionally execute expressions and return values based on a given condition.
- This project also help me to enhance my dashboard building skill in MS Excel 365

Attachment:- Link for project folder

Thank You..