Operation Analytics and Investigating Metric Spike

Project Description : This project involves performing comprehensive data analysis on two sets of data to derive meaningful insights and metrics. The first task is centred around analyzing job data to understand review patterns, throughput, language distribution, and identifying duplicates. The second task focuses on investigating user engagement metrics, growth, retention, and email interactions from a different dataset.

Project Approach : The project was executed using SQL. Where the database was created and the provided data imported. Specific SQL queries were run to extract the required data which was then analyzed to get the valuable insights.

After implementing the necessary queries still some data needed to be extracted with the help of Python and MS Excel.

Tech Stack Used : The tech stack used included MySQL Workbench Ver 8.0.37 for Win64 on x86_64 (MySQL Community Server - GPL). It provides a robust interface for SQL development and was used for executing SQL queries and managing the database.

Python Version 3.8 utilized for additional data processing, analysis, and visualization tasks that are not easily achievable with SQL alone. Python libraries matplotlib and seaborn were used for data visualization in a few tasks.

PROJECT INSIGHTS

1) Job Data Analysis :

a) Jobs Reviewed Over Time: To Calculate the number of jobs reviewed per hour for each day in November 2020.

review_date	jobs_reviewed	total_time_spent	jobs_reviewed_per_hour
30-11-2020	2	40	180
29-11-2020	1	20	180
28-11-2020	2	33	218.1818
27-11-2020	1	104	34.6154
26-11-2020	1	56	64.2857
25-11-2020	1	45	80

For all the other days there weren't any jobs being reviewed.

Code :

```
#Task 1

SELECT
    review_date,
    jobs_reviewed,
    total_time_spent,
    jobs_reviewed * 3600 / total_time_spent AS jobs_reviewed_per_hour
FROM (
    SELECT
        DATE(ds) AS review_date,
        COUNT(*) AS jobs_reviewed,
        SUM(time_spent) AS total_time_spent
    FROM
        job_data
    WHERE
        ds BETWEEN '2020/11/01' AND '2020/11/30'
    GROUP BY
        review_date
) AS daily_stats
ORDER BY
    review_date desc;
```

b) **Throughput Analysis:** To calculate the 7-day rolling average of throughput (number of events per second).

review_date	daily_event_count	avg_7day_throughput_per_second
25-11-2020	1	0.00001157
26-11-2020	1	0.00001157
27-11-2020	1	0.00001157
28-11-2020	2	0.00001447
29-11-2020	1	0.00001389
30-11-2020	2	0.00001543

A 7-day rolling average provides a clearer picture of trends by smoothing out daily fluctuations, making it easier to identify underlying patterns in throughput.

Code :

```
 #Task 2
WITH daily_events AS (
    SELECT
        DATE(ds) AS review_date,
        COUNT(*) AS daily_event_count
    FROM
        job_data
    GROUP BY
        review_date
),
rolling_avg AS (
    SELECT
        review_date,
        daily_event_count,
        AVG(daily_event_count) OVER (ORDER BY review_date ROWS BETWEEN
6 PRECEDING AND CURRENT ROW) AS avg_7day_throughput
    FROM
        daily_events
)
SELECT
    review_date,
    daily_event_count,
    avg_7day_throughput / 86400 AS avg_7day_throughput_per_second
FROM
    rolling_avg;
```

c) **Language Share Analysis:** To Calculate the percentage share of each language in the last 30 days.

Languages	Percentage
English	12.5
Arabic	12.5
Persian	37.5
Hindi	12.5
French	12.5
Italian	12.5

There were 6 distinct languages, **Persian** being the most frequently used

Code :

```
#Task 3

SELECT
   language AS Languages,
   ROUND(100 * COUNT(*) / (SELECT COUNT(*) FROM job_data), 2) AS
Percentage
FROM
   job_data
GROUP BY
   language;
```

d) **Duplicate Rows Detection:** To identify duplicate rows in the data.

job_id	actor_id	event	language	time_spent	org	ds	row_count

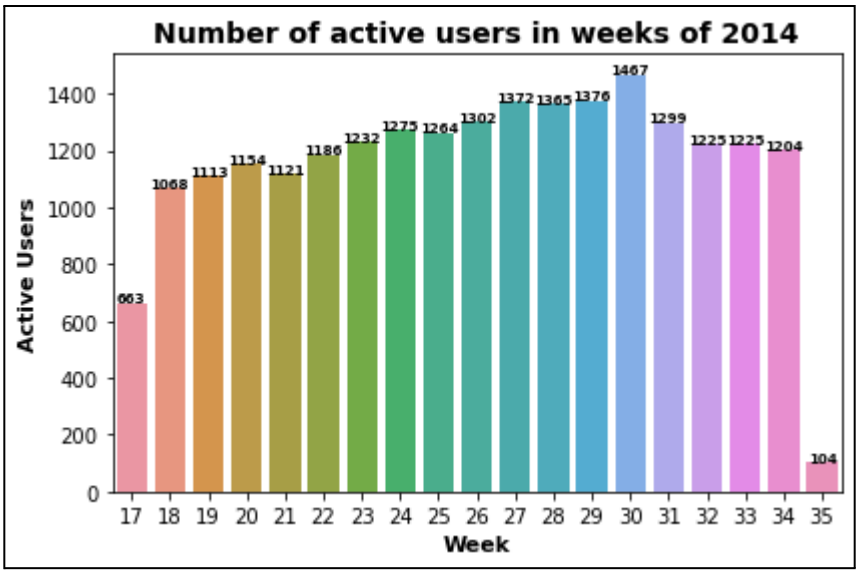
The empty row returned shows that there **wasn't any duplicate** row in the data set.

Code :

```
#Task 4
SELECT
   job_id, actor_id, event, language, time_spent, org, ds,
   COUNT(*) AS row_count
FROM
   job_data
GROUP BY job_id , actor_id , event , language , time_spent , org , ds
HAVING COUNT(*) > 1;
```

2) Investigating Metric Spike :

a) Weekly User Engagement : To measure the activeness of users on a weekly basis.



Highest number of active users were in **30th week(July) of 2014**
Lowest number of active users were in **35th week (September) of 2014**

Code :

```
SELECT
        EXTRACT(YEAR FROM occurred_at) AS year,
        EXTRACT(WEEK FROM occurred_at) AS week,
        COUNT(DISTINCT user_id) AS active_users
FROM
   events
WHERE
   event_type = 'engagement'
GROUP BY year , week
ORDER BY year , week;
```

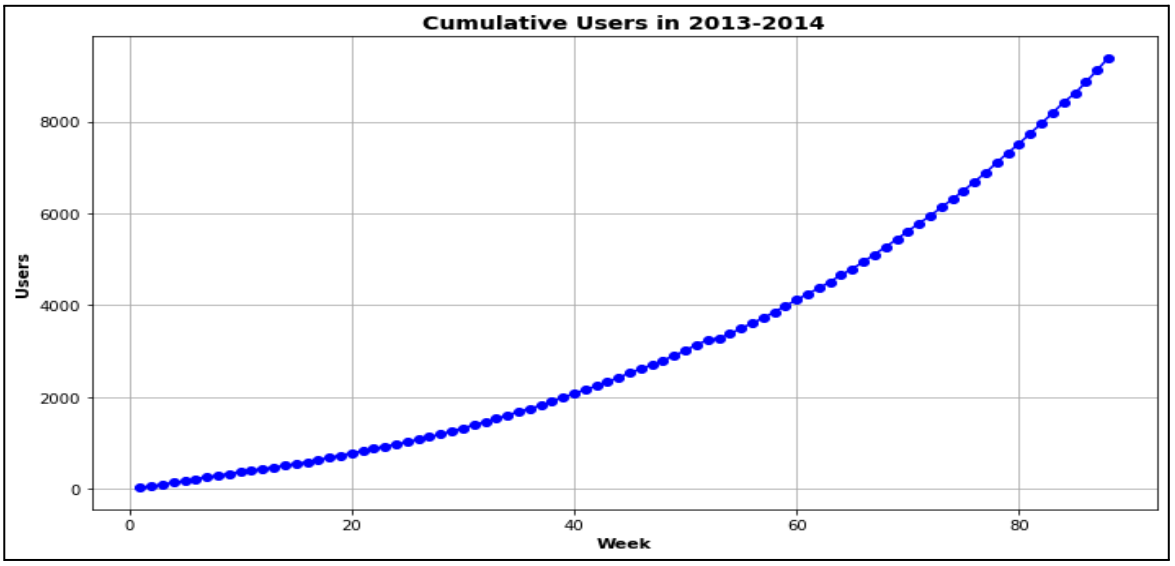
Python code

```
        import matplotlib.pyplot as plt
        import seaborn as sns

        df2=pd.read_csv(r'D:\Trainity\3b1.csv')
        sns.barplot(x=df2['week'], y=df2['active_users'])
        plt.xlabel('Week',fontsize=11,fontweight='bold')
        plt.ylabel('Active Users',fontsize=11,fontweight='bold')
        plt.title('Number of active users in weeks of 2014',fontsize=14,fontweight='bold')
        plt.xticks(rotation=0)
        plt.tight_layout()

        for i, v in enumerate(df2['active_users']):
         plt.text(i, v + 1, str(v), ha='center', fontsize=7,fontweight='bold')
        plt.show()
```

b) User Growth Analysis: To analyze the growth of users over time for a product.



Maximum 266 new users were added in **35th week(September) of 2014**
26 *new users* were the **minimum** added in **1st week(January) of 2013**
On an **average 106** users were added in each week
At the end of 35th week of 2014 there were **9381** total users

Code :

```
WITH weekly_new_users AS (
    SELECT
        YEAR(created_at) AS year,
        WEEK(created_at, 1) AS week,
        COUNT(*) AS new_users
    FROM
        users
    GROUP BY
        YEAR(created_at), WEEK(created_at, 1)
),

cumulative_users AS (
    SELECT
        year,
        week,
        new_users,
        SUM(new_users) OVER (ORDER BY year, week) AS cumulative_users
    FROM
        weekly_new_users
)

SELECT
    year,
    week,
    new_users,
    cumulative_users
FROM
    cumulative_users
```

```
ORDER BY
   year, week;
```

Python Code -

```
import matplotlib.pyplot as plt
import seaborn as sns

df3=pd.read_csv(r'D:\Trainity\3b2.csv')

plt.figure(figsize=(10, 6))
plt.plot(df3['week'], df3['cumulative_users'], marker='o', linestyle='-', color='b')

plt.xlabel('Week', fontsize=11, fontweight='bold')
plt.ylabel('Users', fontsize=11, fontweight='bold')
plt.title('Cumulative Users in 2013-2014', fontsize=14, fontweight='bold')
plt.xticks(rotation=0)
plt.grid(True)
plt.tight_layout()
plt.show()

print(df3['new_users'].describe())
```

c) **Weekly Retention Analysis :** To analyze the retention of users on a weekly basis after signing up for a product.

The user with id **20** and **30** have the highest retention duration of weeks i.e **86 weeks**
The **average** retention duration was **18 weeks**

Code :

```
WITH cohort AS (
   SELECT
         user_id,
         YEARWEEK(created_at, 1) AS signup_week
   FROM
         users
),

weekly_engagement AS (
   SELECT
      user_id,
      YEARWEEK(occurred_at, 1) AS engagement_week
   FROM
      events
   GROUP BY
      user_id, engagement_week
),

retention AS (
   SELECT
      c.user_id,
      c.signup_week,
      w.engagement_week,
```

```
        TIMESTAMPDIFF(WEEK, STR_TO_DATE(CONCAT(c.signup_week, '1'),
'%X%V%w'), STR_TO_DATE(CONCAT(w.engagement_week, '1'), '%X%V%w')) AS
retention_duration_weeks
    FROM
        cohort c
    JOIN
        weekly_engagement w ON c.user_id = w.user_id
    WHERE
        w.engagement_week >= c.signup_week
)

SELECT
    user_id,
    signup_week,
    engagement_week,
    retention_duration_weeks
FROM
    retention
ORDER BY
    signup_week, engagement_week, user_id;
```

d) Weekly Engagement Per Device: To measure the activeness of users on a weekly basis per device.

year	week	device	active_users
2014	17	macbook pro	143
2014	18	macbook pro	252
2014	19	macbook pro	266
2014	20	macbook pro	256
2014	21	macbook pro	247
2014	22	macbook pro	251
2014	23	macbook pro	266
2014	24	macbook pro	255
2014	25	macbook pro	275
2014	26	macbook pro	269
2014	27	macbook pro	302
2014	28	macbook pro	295
2014	29	macbook pro	295
2014	30	macbook pro	322
2014	31	macbook pro	321
2014	32	macbook pro	307
2014	33	macbook pro	312
2014	34	macbook pro	292
2014	35	macbook pro	17

year	week	device	active_users
2014	17	amazon fire phone	4
2014	18	amazon fire phone	9
2014	19	samsumg galaxy tablet	6
2014	20	samsumg galaxy tablet	9
2014	21	amazon fire phone	5
2014	22	amazon fire phone	5
2014	25	samsumg galaxy tablet	12
2014	26	samsung galaxy note	9
2014	27	amazon fire phone	10
2014	28	amazon fire phone	6
2014	29	amazon fire phone	12
2014	30	samsumg galaxy tablet	9
2014	31	samsumg galaxy tablet	8
2014	32	samsumg galaxy tablet	6
2014	33	samsumg galaxy tablet	12
2014	34	amazon fire phone	11

26 devices were used all over different weeks . Where **macbook pro** was the most used device in all the weeks, with highest usage in the 30th week of 2014. **Amazon fire phone** was the least used device in the 17th week of 2014.

Code :

```
(query to extract weekly activeness of each device )
        SELECT
                extract(year from occurred_at) AS year,
                extract(week from occurred_at) AS week,
                device,
        COUNT(DISTINCT user_id) AS active_users
        FROM
                Events
        GROUP BY
                year,week, device
        ORDER BY
                year,week, device;

(query to extract the max and min used device per week )
    WITH weekly_device_engagement AS (
        SELECT
                extract(year from occurred_at) AS year,
                extract(week from occurred_at) AS week,
                device,
        COUNT(DISTINCT user_id) AS active_users
        FROM
                 events
        GROUP BY
                 year, week, device
      ),

        ranked_devices AS (
         SELECT
                year,
                week,
                device,
                active_users,
               RANK() OVER (PARTITION BY week ORDER BY active_users DESC) AS
               ranking
        FROM
         weekly_device_engagement
        )

        SELECT
                year,
                week,
                device,
                active_users
                FROM
                  ranked_devices
        WHERE
                ranking = 1 or ranking=26
        ORDER BY
                year,week;
```

e) Email Engagement Analysis : To analyze how users are engaging with the email service

action	action_count	action_rate
sent_weekly_digest	57267	1
email_open	20459	35.726
email_clickthrough	9010	15.733
sent_reengagement_email	3653	6.379

4 email-events occurred in which **57267** emails were sent and only **35.726%** emails were opened and only **6.379%** emails were sent back a re engagement email.

Code :

```
(query to get the email services response per week )
SELECT
        extract(year from occurred_at) AS year,
        extract(week from occurred_at) AS week,
        action,
COUNT(DISTINCT user_id) AS engaged_users
FROM
        Email_events
GROUP BY
        year, week, action
ORDER BY
        year, week, action;
(query to get the email services action rate )
 SELECT
        action,
        COUNT(*) AS action_count,
        COUNT(*) * 1.0 / (SELECT COUNT(*) FROM email_events where action
            ='sent_weekly_digest') AS action_rate
FROM
        email_events
GROUP BY
        action
ORDER BY
         action_count DESC;
```

CONCLUSION

The project helped me to have hands-on experience of SQL and helped me to understand its functioning and execution better.

A report by **Sagar Chandan**