REPORT ON OPERATION ANALYTICS AND INVESTIGATING METRIC SPIKE

Introduction:

Operation Analytics is a critical analysis performed to evaluate the end-to-end operations of a company. It involves closely working with various teams such as operations, support, marketing, and others to extract valuable insights from the collected data. By analysing the operations data, companies can identify areas for improvement and make informed decisions to enhance their overall performance.

This type of analysis plays a vital role in predicting the future growth or decline of a company. It enables better automation, fosters improved collaboration between cross-functional teams, and facilitates the development of more efficient workflows.

Investigating metric spikes is an essential component of operation analytics. As a Data Analyst, it is important to understand and address questions such as why there is a decrease in daily engagement or why sales have experienced a decline. By investigating these metric spikes, data analysts can provide explanations and actionable insights to various departments within the company.

1. Operation Analytics

Firstly, creating needed table in MySQL:

Operation Analytics and Inve ● | SQL-Project-1-Table.csv

Python 3 (ipykernel)

```
[1]: import pandas as pd
     import mysql.connector
     from datetime import datetime
     from matplotlib import pyplot as plt
```

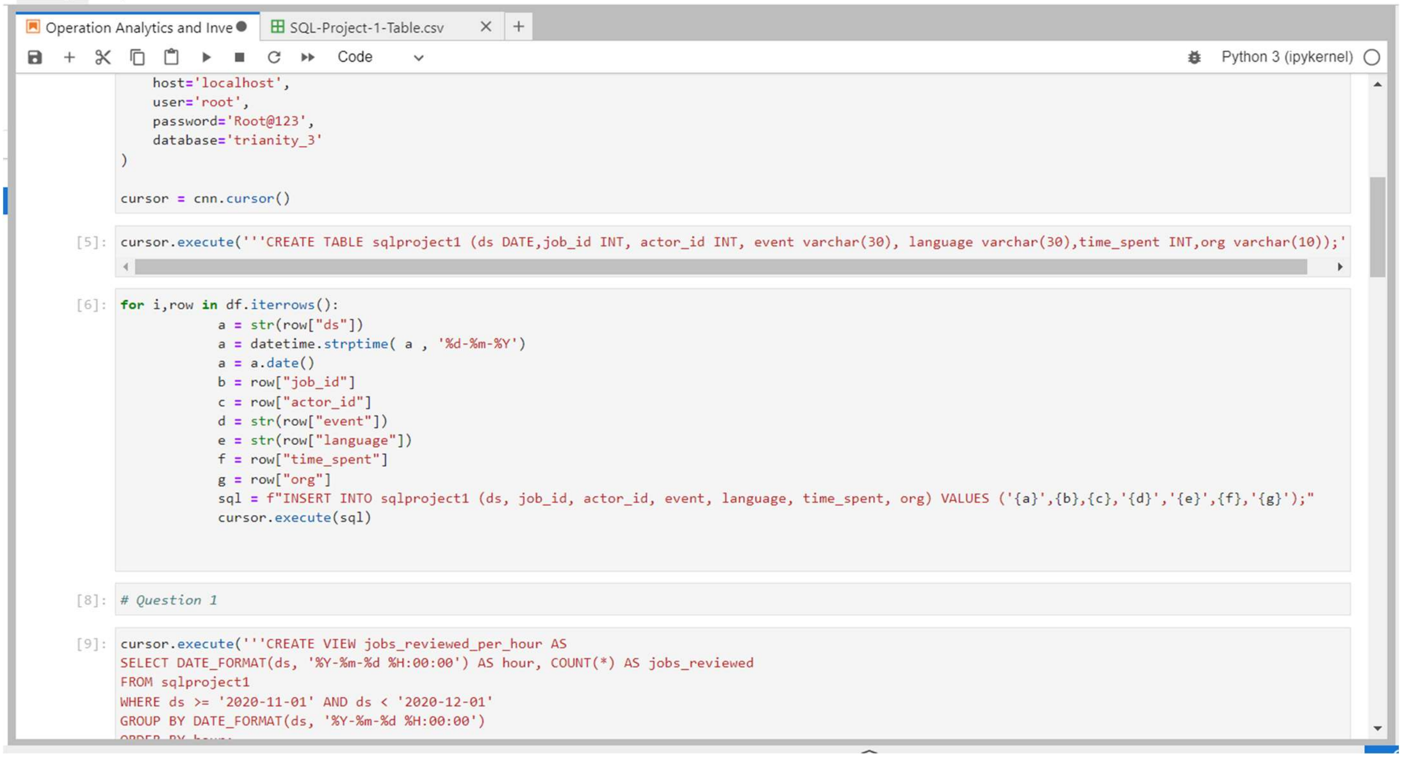
```
[2]: df = pd.read_csv("SQL-Project-1-Table.csv")
```

```
[3]: df.head(10)
```

[3]:

	ds	job_id	actor_id	event	language	time_spent	org
0	30-11-2020	21	1001	skip	English	15	A
1	30-11-2020	22	1006	transfer	Arabic	25	B
2	29-11-2020	23	1003	decision	Persian	20	C
3	28-11-2020	23	1005	transfer	Persian	22	D
4	28-11-2020	25	1002	decision	Hindi	11	B
5	27-11-2020	11	1007	decision	French	104	D
6	26-11-2020	23	1004	skip	Persian	56	A
7	25-11-2020	20	1003	transfer	Italian	45	C

```
[4]: cnn = mysql.connector.connect(
         host='localhost',
         user='root',
         password='Root@123',
         database='trianity_3'
     )
```



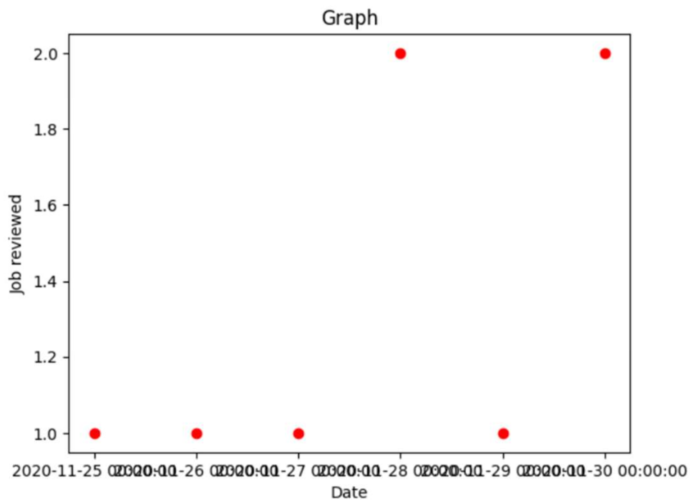
```
Operation Analytics and Inve   SQL-Project-1-Table.csv
Code   Python 3 (ipykernel)

    host='localhost',
    user='root',
    password='Root@123',
    database='trianity_3'
)

cursor = cnn.cursor()

[5]: cursor.execute('''CREATE TABLE sqlproject1 (ds DATE,job_id INT, actor_id INT, event varchar(30), language varchar(30),time_spent INT,org varchar(10));'

[6]: for i,row in df.iterrows():
            a = str(row["ds"])
            a = datetime.strptime( a , '%d-%m-%Y')
            a = a.date()
            b = row["job_id"]
            c = row["actor_id"]
            d = str(row["event"])
            e = str(row["language"])
            f = row["time_spent"]
            g = row["org"]
            sql = f"INSERT INTO sqlproject1 (ds, job_id, actor_id, event, language, time_spent, org) VALUES ('{a}',{b},{c},'{d}','{e}',{f},'{g}');"
            cursor.execute(sql)

[8]: # Question 1

[9]: cursor.execute('''CREATE VIEW jobs_reviewed_per_hour AS
SELECT DATE_FORMAT(ds, '%Y-%m-%d %H:00:00') AS hour, COUNT(*) AS jobs_reviewed
FROM sqlproject1
WHERE ds >= '2020-11-01' AND ds < '2020-12-01'
GROUP BY DATE_FORMAT(ds, '%Y-%m-%d %H:00:00')
```

A) Number of jobs reviewed per hour per day for November 2020:

To calculate the number of jobs reviewed per hour per day for November 2020, you can use the following SQL query:

```
Operation Analytics and Inve   SQL-Project-1-Table.csv
Code   Python 3 (ipykernel)

[8]: # Question 1

[9]: cursor.execute('''CREATE VIEW jobs_reviewed_per_hour AS
SELECT DATE_FORMAT(ds, '%Y-%m-%d %H:00:00') AS hour, COUNT(*) AS jobs_reviewed
FROM sqlproject1
WHERE ds >= '2020-11-01' AND ds < '2020-12-01'
GROUP BY DATE_FORMAT(ds, '%Y-%m-%d %H:00:00')
ORDER BY hour;
''')

[10]: cursor.execute('''SELECT * FROM jobs_reviewed_per_hour''')
Date = []
Job_reviewed = []
for row in cursor:
    Date.append(row[0])
    Job_reviewed.append(row[1])

plt.plot(Date, Job_reviewed, marker='o', linestyle=' ', color='r')

# Set labels and title
plt.xlabel('Date')
plt.ylabel('Job reviewed')
plt.title('Graph')

# Display the graph
plt.show()
```

```python
# Display the graph
plt.show()
```




```python
# Question 2
```

This query extracts the date and hour from the ds column, filters the data for November 2020, and then calculates the count of jobs reviewed for each hour of each day.

B) Throughput and preference for daily metric or 7-day rolling average:

To calculate the throughput and its 7-day rolling average, you need to define what event constitutes throughput in your context. Assuming it is the number of events happening per second, you can use the following query to calculate the throughput:

```python
# Question 2
```

```python
cursor.execute('''CREATE VIEW rolling_avg_throughput AS
SELECT ds, AVG(count_per_second) OVER (ORDER BY ds ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS rolling_avg_throughput
FROM (
    SELECT ds, COUNT(*) AS count_per_second
    FROM sqlproject1
    GROUP BY ds
) subquery;
''')
```

```python
cursor.execute('''SELECT * FROM rolling_avg_throughput''')

for row in cursor:
    print(row)
```

```
(datetime.date(2020, 11, 25), Decimal('1.0000'))
(datetime.date(2020, 11, 26), Decimal('1.0000'))
(datetime.date(2020, 11, 27), Decimal('1.0000'))
(datetime.date(2020, 11, 28), Decimal('1.2500'))
(datetime.date(2020, 11, 29), Decimal('1.2000'))
(datetime.date(2020, 11, 30), Decimal('1.3333'))
```

In terms of preference for daily metric or 7-day rolling average, it depends on the specific use case and what you want to analyze. The daily metric provides the exact value for each day, which can be useful for tracking daily variations. On the other hand, the 7-day rolling average smooths out daily fluctuations and provides a more stable representation of the overall trend. If you're interested in observing longer-term patterns and minimizing the impact of daily fluctuations, the 7-day rolling average can be a better choice.

C) Percentage share of each language in the last 30 days:

To calculate the percentage share of each language in the last 30 days, you can use the following query:

```
(datetime.date(2020, 11, 30), Decimal('1.3333'))

[14]: #Question 3

[20]: cursor.execute('''
SELECT event, language, COUNT(*) * 100.0 / (SELECT COUNT(*) FROM sqlproject1 WHERE event IS NOT NULL) AS percentage_share
FROM sqlproject1
WHERE event IS NOT NULL
GROUP BY event, language;
''')

for row in cursor:
    print(row)

('skip', 'English', Decimal('12.50000'))
('transfer', 'Arabic', Decimal('12.50000'))
('decision', 'Persian', Decimal('12.50000'))
('transfer', 'Persian', Decimal('12.50000'))
('decision', 'Hindi', Decimal('12.50000'))
('decision', 'French', Decimal('12.50000'))
('skip', 'Persian', Decimal('12.50000'))
('transfer', 'Italian', Decimal('12.50000'))
```

This query counts the number of jobs for each language in the last 30 days and divides it by the total count of jobs in the same period. The result is multiplied by 100 to get the percentage share of each language.

D) Displaying duplicate rows from the table:

To display duplicate rows from the job_data table, you can use the following query:

```
[22]: cursor.execute('''
CREATE VIEW duplicate_rows AS
SELECT *
FROM sqlproject1
WHERE (job_id, actor_id, event, language, time_spent, org, ds) IN (
    SELECT job_id, actor_id, event, language, time_spent, org, ds
    FROM sqlproject1
    GROUP BY job_id, actor_id, event, language, time_spent, org, ds
    HAVING COUNT(*) > 1
);
''')

[23]: cursor.execute('''SELECT * FROM duplicate_rows''')

for row in cursor:
    print(row)

[24]: cnn.commit()

[26]: cnn.close()
cnn.close()
```

This query selects all rows from job_data where the combination of all columns appears more than once in the table, indicating duplicate rows.

2. Investigating Metric Spike

Firstly, creating needed table in MySQL:

Investigating metric spike.ipy | Operation Analytics and Inve | Table-1 users.csv | Table-2 events.csv | Table-3 email_events.csv

Code | Python 3 (ipykernel)

[1]:
```python
import pandas as pd
import mysql.connector
from datetime import datetime
from matplotlib import pyplot as plt
```

[2]:
```python
df = pd.read_csv("Table-1 users-Copy1.csv")
```

[3]:
```python
df.head(10)
```

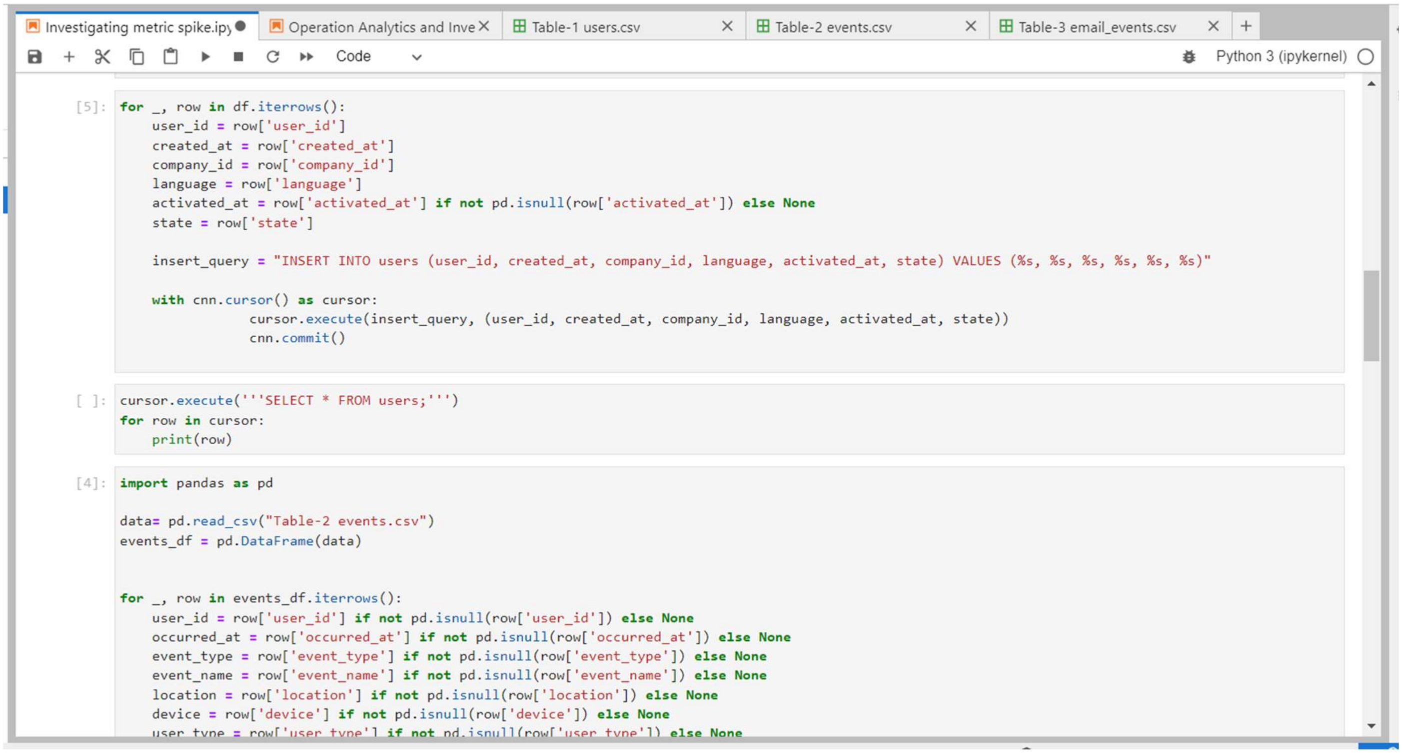
[3]:

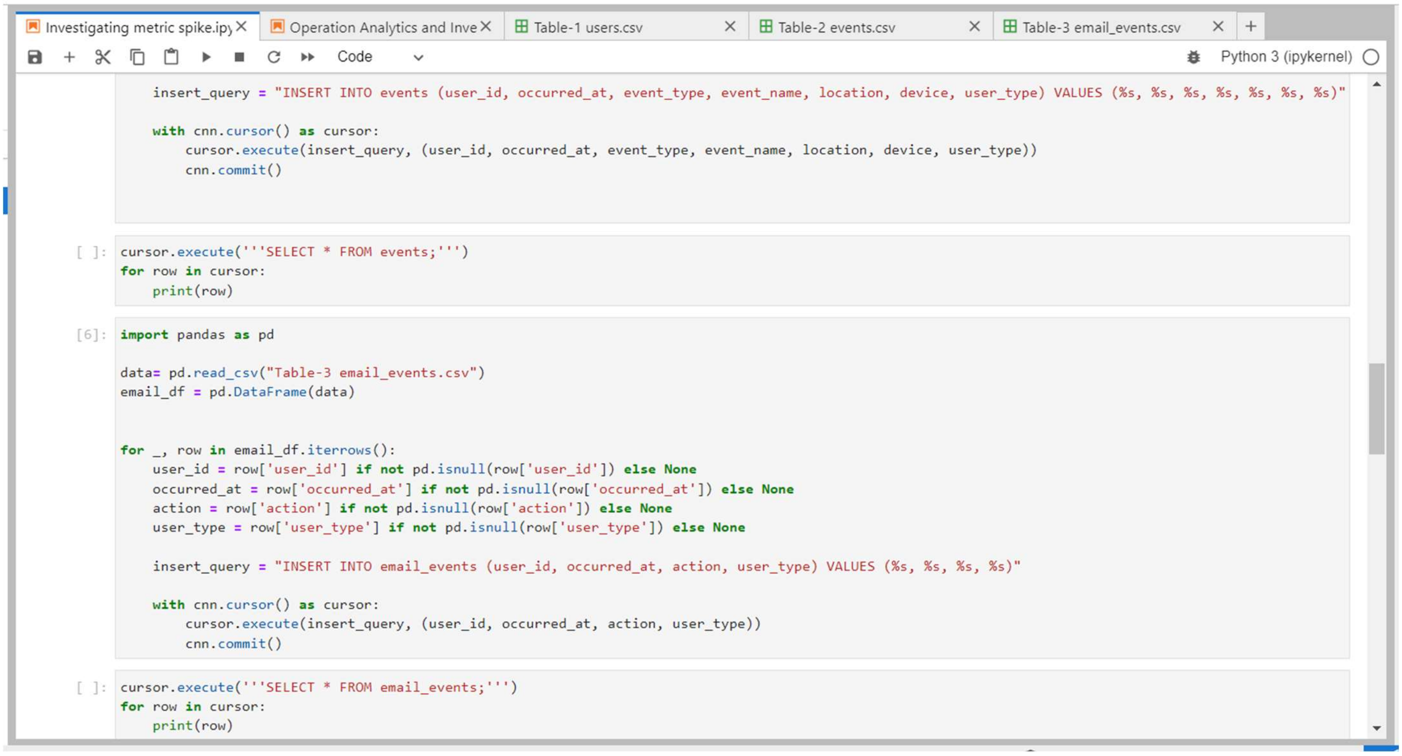
	user_id	created_at	company_id	language	activated_at	state
0	0.0	2013-01-01 20:59:39	5737.0	english	2013-01-01 21:01:07	active
1	1.0	2013-01-01 13:07:46	28.0	english	NaN	pending
2	2.0	2013-01-01 10:59:05	51.0	english	NaN	pending
3	3.0	2013-01-01 18:40:36	2800.0	german	2013-01-01 18:42:02	active
4	4.0	2013-01-01 14:37:51	5110.0	indian	2013-01-01 14:39:05	active
5	5.0	2013-01-01 13:39:51	2463.0	spanish	NaN	pending
6	6.0	2013-01-01 18:37:27	11699.0	english	2013-01-01 18:38:45	active
7	7.0	2013-01-01 16:19:01	4765.0	french	2013-01-01 16:20:28	active
8	8.0	2013-01-01 04:38:30	2698.0	french	2013-01-01 04:40:10	active
9	9.0	2013-01-01 08:04:17	1.0	french	NaN	pending

[22]:
```python
cnn = mysql.connector.connect(
    host='localhost',
    user='root',
    password='Root@123',
    database='trianity_3'
)

cursor = cnn.cursor()
```

[ ]:
```python
cursor.execute('''CREATE TABLE users (
    user_id INT PRIMARY KEY,
    created_at DATETIME,
    company_id INT,
    language VARCHAR(50),
    activated_at DATETIME,
    state VARCHAR(50)
);

CREATE TABLE events (
    user_id INT,
    occurred_at DATETIME,
    event_type VARCHAR(50),
    event_name VARCHAR(50),
    location VARCHAR(50),
    device VARCHAR(50),
    user_type VARCHAR(50)
);

CREATE TABLE email_events (
    user_id INT,
    occurred_at DATETIME,
    action VARCHAR(50),
    user_type VARCHAR(50)
);
```

Insert From CSV File :





To perform the calculations and derive the required metrics for the given case study, you will need to write SQL queries based on the structure of the tables provided. Here's a breakdown of how you can approach each task**:**

A) User Engagement:

User engagement refers to the level of activity and interaction exhibited by users towards a product or service. It is an important metric that measures how actively users are using and finding value in the product or service. By calculating the weekly user engagement, we can track the ongoing level of user activity and determine if users are consistently engaging with the product/service over time. This information helps to assess the effectiveness of the product/service and identify areas for improvement or further engagement strategies.

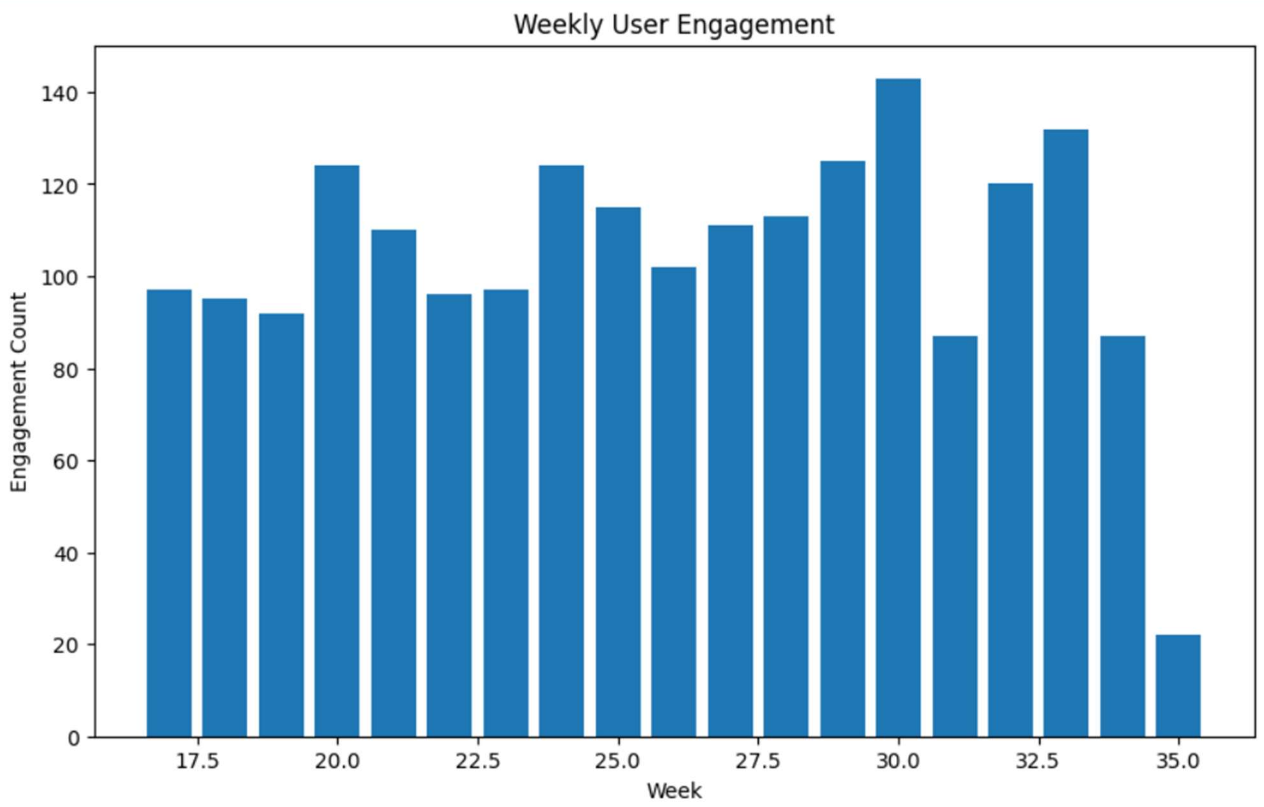
To calculate the weekly user engagement, you can use the events table to count the number of relevant events (such as login events, messaging events, etc.) per user per week. Here's an example query:

```
[10]: # Question 1
```

```
[16]: cursor.execute('''CREATE VIEW weekly_user_engagement AS
SELECT e.user_id, WEEK(e.occurred_at) AS week, COUNT(*) AS engagement_count
FROM events e
WHERE e.event_type IN ('engagement')
GROUP BY e.user_id, week;
''')
```

```
[ ]: cursor.execute('''SELECT *
FROM weekly_user_engagement
ORDER BY week;
''')
for row in cursor:
    print(row)
```

```
[23]: query = "SELECT * FROM weekly_user_engagement"
df = pd.read_sql(query, cnn)

# Plot the data
plt.figure(figsize=(10, 6))
plt.bar(df['week'], df['engagement_count'])
plt.xlabel('Week')
plt.ylabel('Engagement Count')
plt.title('Weekly User Engagement')
plt.show()
```



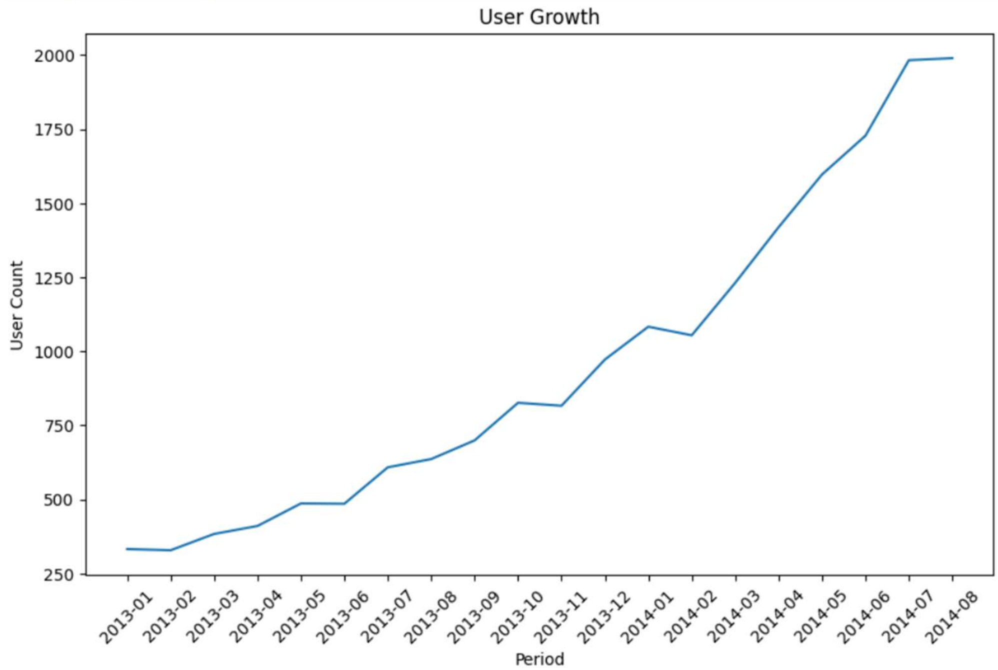
This query groups the events by user and week and calculates the count of events for each combination.

B) User Growth:

User growth refers to the increase in the number of users over a specific period of time. It is a key metric for understanding the expansion and adoption of a product or service. By calculating the user growth, we can track the rate at which new users are joining the product/service. This information provides insights into the popularity and reach of the product/service and helps in strategic planning, resource allocation, and assessing the overall success of the user acquisition efforts.

To calculate the user growth over time, you can use the users table to count the number of new users added per week. Here's an example query:

```
[ ]: # Question 2

[24]: cursor.execute('''CREATE VIEW user_growth AS
SELECT DATE_FORMAT(u.created_at, '%Y-%m') AS period, COUNT(DISTINCT u.user_id) AS user_count
FROM users u
GROUP BY period;
''')

[25]: query = "SELECT * FROM user_growth"
df = pd.read_sql(query, cnn)

# Plot the data
plt.figure(figsize=(10, 6))
plt.plot(df['period'], df['user_count'])
plt.xlabel('Period')
plt.ylabel('User Count')
plt.title('User Growth')
plt.xticks(rotation=45)
plt.show()
```



This query groups the users by the week they were created and calculates the count of users for each week.

C) Weekly Retention:

Weekly retention measures the percentage of users who continue to engage with a product or service after signing up, specifically on a weekly basis. It is an essential metric for evaluating the effectiveness of user onboarding and retention strategies. By calculating the weekly retention of users in the sign-up cohort, we can determine the percentage of users who remain active in subsequent weeks. This information helps in identifying the retention rate, assessing the impact of user engagement initiatives, and making data-driven decisions to improve user retention over time.

To calculate the weekly retention of the user sign-up cohort, you will need to define a cohort (e.g., users who signed up in a particular week) and track their activity in subsequent weeks. Here's an example query:

```
[ ]: # Question 3

•[26]: cursor.execute('''CREATE VIEW weekly_retention AS
SELECT u.created_at, WEEK(e.occurred_at) AS week, COUNT(DISTINCT u.user_id) AS retention_count
FROM users u
LEFT JOIN events e ON u.user_id = e.user_id
WHERE e.occurred_at >= u.created_at
GROUP BY u.created_at, week;
''')

[29]: cursor.execute('''SELECT *
FROM weekly_retention
''')
for row in cursor:
    print(row)

(datetime.datetime(2013, 1, 1, 4, 38, 30), 17, 1)
(datetime.datetime(2013, 1, 1, 4, 38, 30), 18, 1)
(datetime.datetime(2013, 1, 1, 4, 38, 30), 19, 1)
(datetime.datetime(2013, 1, 1, 4, 38, 30), 20, 1)
(datetime.datetime(2013, 1, 1, 4, 38, 30), 30, 1)
(datetime.datetime(2013, 1, 1, 8, 7, 45), 24, 1)
(datetime.datetime(2013, 1, 1, 8, 7, 45), 25, 1)
(datetime.datetime(2013, 1, 1, 8, 7, 45), 30, 1)
(datetime.datetime(2013, 1, 1, 8, 7, 45), 31, 1)
(datetime.datetime(2013, 1, 1, 14, 37, 51), 19, 1)
```

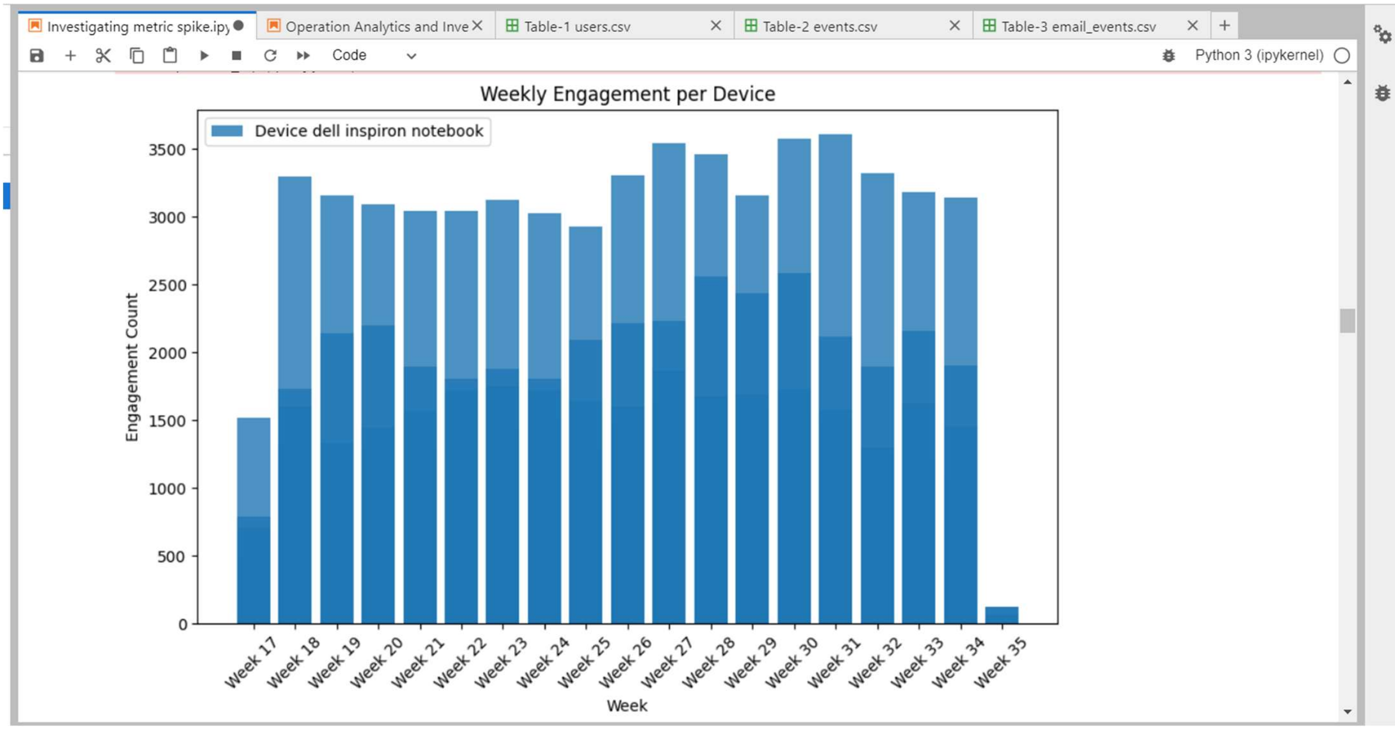
This query joins the users and events tables based on the user ID and calculates the count of distinct users retained for each combination of signup week and retention week.

D) Weekly Engagement:

Measuring weekly engagement per device helps to understand how users interact with a product or service across different devices. By tracking the level of activity on a weekly basis and categorizing it by device type, we can gain insights into user preferences and behaviors. This information enables us to optimize the user experience for different devices, identify any disparities in engagement across devices, and tailor strategies to improve user engagement on specific platforms or devices.

To calculate the weekly engagement per device, you can use the events table and group the events by the device and week. Here's an example query:

```
[ ]: #Question 4

[35]: query = "SELECT * FROM weekly_engagement_per_device"
df = pd.read_sql(query, cnn)

# Plot the data
plt.figure(figsize=(10, 6))
plt.bar(df['week'], df['engagement_count'], align='center', alpha=0.8)
plt.xlabel('Week')
plt.ylabel('Engagement Count')
plt.title('Weekly Engagement per Device')

# Set x-axis tick labels to be week numbers
week_labels = ['Week {}'.format(week) for week in df['week'].unique()]
plt.xticks(df['week'].unique(), week_labels, rotation=45)

# Show device names as a legend
devices = df['device'].unique()
device_labels = ['Device {}'.format(device) for device in devices]
plt.legend(device_labels)

plt.show()
```



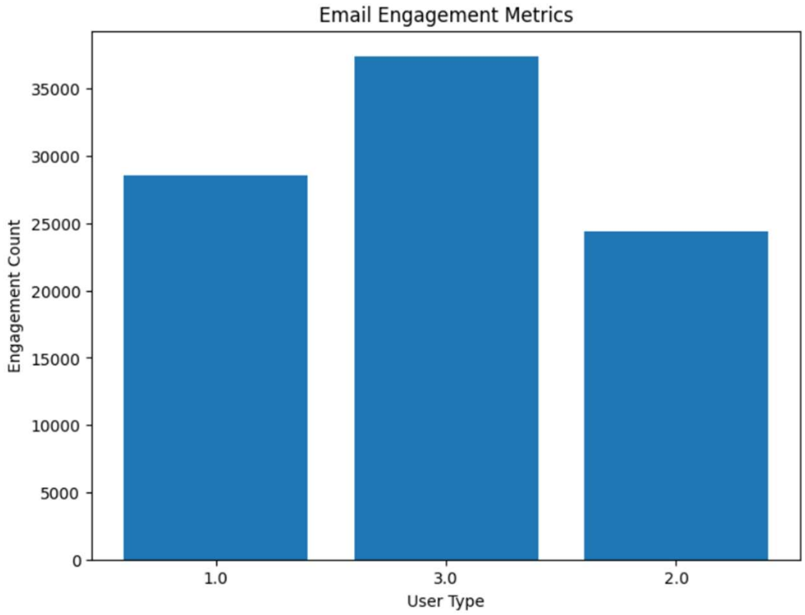
This query groups the events by the device and week and calculates the count of events for each combination.

E) Email Engagement:

Email engagement metrics focus on measuring user interactions and activities related to email services. These metrics provide insights into how users engage with emails, such as opening, clicking links, or performing specific actions within the email. By analyzing email engagement, we can evaluate the effectiveness of email campaigns, assess user interest and response rates, and make data-driven decisions to optimize email content, design, and delivery. Email engagement metrics help in improving user communication, increasing conversions, and enhancing overall email marketing strategies.

To calculate the email engagement metrics, you will need to specify the specific metrics you want to calculate based on the email_events table. For example, you could calculate the number of sent emails, opened emails, clicked links, etc. Here's an example query to calculate the number of sent emails:

```
[ ]: # Question 5

[36]: cursor.execute('''CREATE VIEW email_engagement_metrics AS
      SELECT e.user_type, COUNT(*) AS engagement_count
      FROM email_events e
      GROUP BY e.user_type;
      ''')

[37]: query = "SELECT * FROM email_engagement_metrics"
      df = pd.read_sql(query, cnn)

      # Plot the data as a bar chart
      plt.figure(figsize=(8, 6))
      plt.bar(df['user_type'], df['engagement_count'])
      plt.xlabel('User Type')
      plt.ylabel('Engagement Count')
      plt.title('Email Engagement Metrics')

      plt.show()
```



Email Engagement Metrics
35000
30000
25000
20000
15000
10000
5000
0
Engagement Count
1.0
3.0
2.0
User Type