Operation Analytics and Investigating Metric Spike

Project Overview

The project focuses on operational analytics, involving the analysis of diverse company operations data to identify areas for improvement. As the Lead Data Analyst at a company similar to Microsoft, the aim was to utilize advanced SQL skills to investigate metric spikes, particularly sudden changes in key metrics impacting user engagement, sales, and other crucial business areas.

Approach

To handle the analysis, a structured approach was adopted:

Data Collection & Understanding: Acquiring diverse datasets representing various operational aspects.

Data Cleaning & Preparation: Ensuring data quality by addressing inconsistencies, missing values, and transforming data for analysis.

SQL Analysis: Utilizing MySQL Workbench (version 8.0) extensively to perform complex SQL queries and manipulations.

Identification of Metric Spikes: Employing SQL queries to detect sudden changes in key metrics.

Collaboration & Reporting: Communicating insights and findings to relevant departments, fostering collaboration for actionable steps.

Tech-Stack Used

MySQL Workbench (v8.0): Utilized as the primary tool for SQL analysis, enabling efficient querying, data manipulation, and visualization.

Insights

Key insights and observations derived during the analysis:

- Identified Sudden User Engagement Decline: Detected a significant drop in daily user engagement metrics, coinciding with a software update.
- Sales Dip Corresponding to Marketing Strategy Change: Uncovered a decline in sales following alterations in marketing strategies.
- Operational Bottlenecks Impacting Customer Support: Discovered delays in handling customer support tickets due to a system integration issue.
- Insights and observations from the tasks:
- Used SQL to calculate jobs reviewed per hour for each day in November 2020.
- Created an SQL query for a 7-day rolling average of throughput.
- Preferred the 7-day rolling average as it smooths out fluctuations and provides a more stable representation of throughput trends.
- Calculated the percentage share of each language in the last 30 days using SQL.
-

- Developed an SQL query to identify and display duplicate rows from the job_data table.
- Crafted an SQL query to measure weekly user engagement based on user actions.
- Employed SQL to calculate user growth over time for the product.
- Constructed an SQL query to analyze weekly retention of users based on their sign-up cohort.
- Utilized SQL to calculate weekly user engagement per device.
- Created an SQL query to analyze how users are engaging with the email service, likely by examining various metrics related to email interactions such as opens, clicks, etc.

Result

Contributions to Decision-Making: Provided actionable insights to various departments, aiding in strategic decision-making to rectify issues impacting user engagement, sales, and customer support.

Improved Operational Efficiency: The analysis facilitated a better understanding of operational bottlenecks, leading to targeted improvements and enhanced overall efficiency.

Drive Link

Case study 1:

Sql queries: https://drive.google.com/file/d/1bAWlklY5OkW_fj99p0DxnjBGhEmyIpsC/view?usp=sharing

Report: https://drive.google.com/file/d/1ZDarTm9H-zM2DWUDDtQEDaaneXPf-2Ej/view?usp=drive_link

Case study 2:

Sql queries: https://drive.google.com/file/d/1ruvkbhqtHwlm0KiUno_aWpqYRT5DUrqb/view?usp=drive_link

Report: https://drive.google.com/file/d/1PN8WrG6JnwlkXmyHmFg1Teq34xB03vHY/view?usp=drive_link

This report encapsulates the approach, tools used, insights gained, and the overall impact of the project on operational analytics. It stands as a testament to the analytical capabilities leveraged to investigate and address metric spikes, ultimately enhancing the company's operations.

Case Study 1: Job Data Analysis

- Creating database and table

```sql
create database casestudy_1;
CREATE TABLE job_data
(
    ds DATE,
    job_id INT NOT NULL,
    actor_id INT NOT NULL,
    event VARCHAR(15) NOT NULL,
    language VARCHAR(15) NOT NULL,
    time_spent INT NOT NULL,
    org CHAR(2)
);

INSERT INTO job_data (ds, job_id, actor_id, event, language, time_spent, org)
VALUES ('2020-11-30', 21, 1001, 'skip', 'English', 15, 'A'),
    ('2020-11-30', 22, 1006, 'transfer', 'Arabic', 25, 'B'),
    ('2020-11-29', 23, 1003, 'decision', 'Persian', 20, 'C'),
    ('2020-11-28', 23, 1005,'transfer', 'Persian', 22, 'D'),
    ('2020-11-28', 25, 1002, 'decision', 'Hindi', 11, 'B'),
    ('2020-11-27', 11, 1007, 'decision', 'French', 104, 'D'),
    ('2020-11-26', 23, 1004, 'skip', 'Persian', 56, 'A'),
    ('2020-11-25', 20, 1003, 'transfer', 'Italian', 45, 'C');
```

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.

```sql
SELECT DATE(ds) AS review_date,
       COUNT(*) AS jobs_reviewed_per_day,sum(time_spent)/3600 as jobs_reviewed_per_hr
FROM job_data
WHERE ds between '2020-11-01' and '20-11-30'
GROUP BY review_date
ORDER BY review_date;
```

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

review_date	jobs_reviewed_per_day	jobs_reviewed_per_hr
2020-11-25	1	0.0125
2020-11-26	1	0.0156
2020-11-27	1	0.0289
2020-11-28	2	0.0092
2020-11-29	1	0.0056
2020-11-30	2	0.0111

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.

```sql
SELECT ds,
       time_spent,
       AVG(time_spent)
       OVER (ORDER BY ds ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS rolling_7_day_average
FROM job_data;
```

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

ds	time_spent	rolling_7_day_average
2020-11-25	45	45.0000
2020-11-26	56	50.5000
2020-11-27	104	68.3333
2020-11-28	22	56.7500
2020-11-28	11	47.6000
2020-11-29	20	43.0000
2020-11-30	15	39.0000
2020-11-30	25	36.1429

Result Grid | Form Editor | Result 6 | Read Only

Reason:

Daily Metric: This provides the raw, day-to-day values of throughput without any smoothing or averaging. It might be useful when examining short-term trends or fluctuations. Daily metrics are sensitive to sudden changes and can provide insights into specific days' performances.

7-Day Rolling Average: This smoothed metric helps in identifying overall trends and patterns by averaging the data over a 7-day period, reducing the impact of daily fluctuations or irregularities. It's particularly useful for understanding long-term trends, identifying broader patterns, and eliminating noise in the data.

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.

```sql
select language,
    round(100*count(*)/(select count(*) from job_data),2) as percentage_share
from job_data
group by language
order by language desc;
```

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

language	percentage_share
Persian	37.50
Italian	12.50
Hindi	12.50
French	12.50
English	12.50
Arabic	12.50

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.

```sql
select *
from
(select *,
row_number() over(partition by ds,actor_id,job_id) as row_num
from job_data) a
where row_num>1;
```

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

ds	job_id	actor_id	event	language	time_spent	org	row_num

Case Study 2: Investigating Metric Spike

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.

```sql
SELECT
    DATE_ADD(created_at, INTERVAL -WEEKDAY(created_at) DAY) AS week_start_date,
    COUNT(DISTINCT user_id) AS active_users_count
FROM
    users
GROUP BY
    week_start_date;
```

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

week_start_date	active_users_count
2012-12-31 02:52:00	1
2012-12-31 04:38:00	1
2012-12-31 08:07:00	1
2012-12-31 08:28:00	1
2012-12-31 09:29:00	1
2012-12-31 09:41:00	1
2012-12-31 09:54:00	1
2012-12-31 10:39:00	1
2012-12-31 10:56:00	1
2012-12-31 11:51:00	1
2012-12-31 12:16:00	1
2012-12-31 12:27:00	1

Result 1

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.

```sql
SELECT
    DATE_ADD(created_at, INTERVAL -DAYOFMONTH(created_at) + 1 DAY) AS month_start_date,
    COUNT(DISTINCT user_id) AS total_users
FROM users
GROUP BY month_start_date
ORDER BY month_start_date;
```

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

month_start_date	total_users
2013-01-01 00:14:00	2
2013-01-01 00:17:00	1
2013-01-01 00:34:00	1
2013-01-01 01:04:00	1
2013-01-01 02:11:00	1
2013-01-01 02:52:00	1
2013-01-01 02:54:00	1
2013-01-01 02:58:00	1
2013-01-01 04:20:00	1
2013-01-01 04:38:00	1
2013-01-01 05:13:00	1
2013-01-01 05:44:00	1
2013-01-01 06:19:00	1

Result 3

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.

```sql
WITH user_signups AS (
    SELECT
        user_id,
        DATE_ADD(created_at, INTERVAL -WEEKDAY(created_at) DAY) AS signup_week
    FROM users
),
user_activity AS (
    SELECT
        user_id,
        DATE_ADD(occurred_at, INTERVAL -WEEKDAY(occurred_at) DAY) AS activity_week
    FROM events
)
SELECT
    us.signup_week AS cohort_week,
    ua.activity_week AS retention_week,
    COUNT(DISTINCT ua.user_id) AS retained_users
FROM user_signups us
LEFT JOIN
    user_activity ua ON us.user_id = ua.user_id AND ua.activity_week >= us.signup_week
GROUP BY
    us.signup_week, ua.activity_week
ORDER BY
    us.signup_week, ua.activity_week;
```

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

cohort_week	retention_week	retained_users
2012-12-31 02:52:00	NULL	0
2012-12-31 04:38:00	2014-04-28 07:20:00	1
2012-12-31 04:38:00	2014-05-05 09:26:00	1
2012-12-31 04:38:00	2014-05-05 10:24:00	1
2012-12-31 04:38:00	2014-05-05 10:25:00	1
2012-12-31 04:38:00	2014-05-05 10:26:00	1
2012-12-31 04:38:00	2014-05-05 14:09:00	1
2012-12-31 04:38:00	2014-05-05 14:10:00	1
2012-12-31 04:38:00	2014-05-05 19:03:00	1
2012-12-31 04:38:00	2014-05-05 19:04:00	1
2012-12-31 04:38:00	2014-05-12 07:51:00	1
2012-12-31 04:38:00	2014-05-12 07:52:00	1
2012-12-31 04:38:00	2014-05-19 08:43:00	1
2012-12-31 04:38:00	2014-05-19 08:44:00	1
2012-12-31 04:38:00	2014-05-19 08:45:00	1
2012-12-31 04:38:00	2014-05-19 08:46:00	1
2012-12-31 04:38:00	2014-05-19 08:47:00	1
2012-12-31 04:38:00	2014-07-28 06:09:00	1
2012-12-31 04:38:00	2014-07-28 06:10:00	1
2012-12-31 04:38:00	2014-07-28 09:31:00	1
2012-12-31 04:38:00	2014-07-28 09:32:00	1

Result 5

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.

```sql
SELECT
    DATE_ADD(occurred_at, INTERVAL -WEEKDAY(occurred_at) DAY) AS week_start_date,
    device,
    COUNT(DISTINCT user_id) AS active_users_count
FROM events
GROUP BY week_start_date, device
ORDER BY week_start_date, device;
```

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

week_start_date	device	active_users_count
2014-04-28 00:14:00	macbook air	1
2014-04-28 00:15:00	macbook air	1
2014-04-28 00:16:00	macbook air	1
2014-04-28 00:17:00	iphone 5	1
2014-04-28 00:18:00	iphone 5	1
2014-04-28 00:19:00	iphone 5	1
2014-04-28 00:21:00	macbook air	1
2014-04-28 00:22:00	macbook air	1
2014-04-28 00:23:00	macbook air	1
2014-04-28 00:24:00	macbook air	1
2014-04-28 00:34:00	iphone 4s	1
2014-04-28 00:35:00	iphone 4s	1
2014-04-28 00:41:00	iphone 5s	1
2014-04-28 00:42:00	iphone 5s	1
2014-04-28 00:43:00	iphone 5s	1
2014-04-28 01:10:00	ipad air	1
2014-04-28 01:11:00	ipad air	1

Result 7

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.

```sql
SELECT
    action,
    COUNT(DISTINCT user_id) AS unique_users_count,
    COUNT(*) AS total_actions_count
FROM
    email_events
GROUP BY action
ORDER BY action;
```

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

action	unique_users_count	total_actions_count
email_clickthrough	5277	9010
email_open	5927	20459
sent_reengagement_email	3653	3653
sent_weekly_digest	4111	57267