Operation Analytics and Investigating Metric Spike

Project Report

Project Description

The 'Operation Analytics and Investigating Metric Spike' project focuses on analyzing various company metrics to identify trends, patterns, and anomalies. The goal is to derive valuable insights that can help improve business operations and understand sudden changes in key metrics. The project involved addressing specific case studies requiring advanced SQL techniques to answer business-related questions.

Approach

The approach to this project was methodical and structured in the following steps:

1. Understanding the Problem
2. Data Exploration
3. Data Cleaning and Preprocessing
4. SQL Query Execution
5. Analysis and Interpretation
6. Insights and Reporting

Tech-Stack Used

1. MySQL Workbench: Version 8.0
2. Python: Used for data preprocessing and automation tasks.
3. Microsoft Word: For documenting the project report.
4. Google Drive: For sharing the final project report.

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.

SQL Query:

```
SELECT
   ds AS review_day,
   COUNT(job_id) AS jobs_reviewed,
   (SUM(time_spent) / 3600) AS time_spent_in_hours,
   COUNT(job_id) / (SUM(time_spent) / 3600) AS jobs_reviewed_per_hour
FROM
   job_data
WHERE
   ds BETWEEN '2020-11-01' AND '2020-11-30'
GROUP BY
   ds
```

```
ORDER BY
    review_day;
```

Result:

review_day	jobs_reviewed	time_spent_in_hours	jobs_reviewed_per_hour
2020-11-29	1	0.0056	180.0000
2020-11-28	2	0.0092	218.1818
2020-11-30	2	0.0111	180.0000
2020-11-25	1	0.0125	80.0000
2020-11-26	1	0.0156	64.2857
2020-11-27	1	0.0289	34.6154

2. Throughput Analysis

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

SQL Query:

```
WITH daily_throughput AS (
    SELECT
        ds AS review_day,
        COUNT(event) AS total_events,
        SUM(time_spent) AS total_time_spent_sec,
        (COUNT(event) / NULLIF(SUM(time_spent), 0)) AS daily_throughput
    FROM
        job_data
    WHERE
        ds BETWEEN '2020-11-01' AND '2020-11-30'
    GROUP BY
        ds
),
rolling_avg_throughput AS (
```

```
    SELECT
        review_day,
        daily_throughput,
        AVG(daily_throughput) OVER (
            ORDER BY review_day
            ROWS BETWEEN 6 PRECEDING AND CURRENT ROW
        ) AS rolling_avg_throughput_7d
    FROM
        daily_throughput
)
SELECT
    review_day,
    daily_throughput,
    rolling_avg_throughput_7d
FROM
    rolling_avg_throughput
ORDER BY
    review_day;
```

Result:

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

review_day	daily_throughput	rolling_avg_throughput_7d
2020-11-25	0.0222	0.02220000
2020-11-26	0.0179	0.02005000
2020-11-27	0.0096	0.01656667
2020-11-28	0.0606	0.02757500
2020-11-29	0.0500	0.03206000
2020-11-30	0.0500	0.03505000

Interpretation:

Daily vs. Rolling Average:

The daily throughput varies significantly, as seen with a low of 0.0096 on 2020-11-27 and a peak of 0.0606 on 2020-11-28.

The 7-day rolling average smooths these fluctuations, providing a more stable trend. It starts low at 0.0222 and gradually increases, ending at 0.03505 on 2020-11-30.

Trend Insight:

The rolling average indicates an increasing trend in throughput towards the end of the month, suggesting improved event processing efficiency over time.

Conclusion:

Using a 7-day rolling average is beneficial because it smooths out daily fluctuations and provides a clearer view of throughput trends, helping to identify whether the system is consistently improving or facing bottlenecks.

3. Language Share Analysis

Objective: Calculate the percentage share of each language in the last 30 days.

SQL Query:

```
SELECT

    language,

    SUM(time_spent) AS total_time_spent,

    ROUND(
```

```
        (SUM(time_spent) /
        (SELECT SUM(time_spent)
        FROM job_data
        WHERE STR_TO_DATE(ds, '%Y-%m-%d') >= '2020-11-01' AND
STR_TO_DATE(ds, '%Y-      %m-%d') <= '2020-11-30')) * 100, 2
  ) AS language_share_percentage
FROM
  job_data
WHERE
  STR_TO_DATE(ds, '%Y-%m-%d') BETWEEN '2020-11-01' AND '2020-11-30'
GROUP BY
  language
ORDER BY
  language_share_percentage DESC;
```

Result:

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

language	total_time_spent	language_share_percentage
Arabic	25	8.39
English	15	5.03
French	104	34.90
Hindi	11	3.69
Italian	45	15.10
Persian	98	32.89

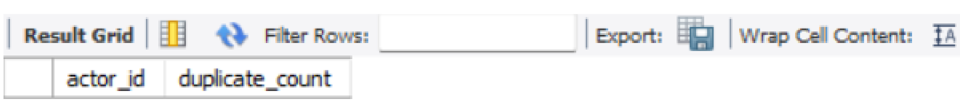
4. Duplicate Rows Detection

Objective: Identify duplicate rows in the data.

SQL Query:

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

Result:



Case Study 2: Investigating Metric Spike

1. Weekly User Engagement

Objective: Measure the activeness of users on a weekly basis.

SQL Query:

```
WITH weekly_data AS (
   SELECT
      YEARWEEK(created_at, 1) AS year_week,
      COUNT(DISTINCT user_id) AS new_users,
      COUNT(DISTINCT CASE WHEN activated_at IS NOT NULL THEN user_id
END) AS activated_users
   FROM
      users
   GROUP BY
      YEARWEEK(created_at, 1)
)

SELECT
   wd.year_week,
   wd.new_users,
```

```
    wd.activated_users,
    (SELECT SUM(new_users)
     FROM weekly_data wd2
     WHERE wd2.year_week <= wd.year_week) AS total_users
FROM
    weekly_data wd
ORDER BY
    wd.year_week;
```

Result:

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

year_week	new_users	activated_users	total_users
201301	26	26	26
201302	29	29	55
201303	47	47	102
201304	36	36	138
201305	30	30	168
201306	48	48	216
201307	41	41	257
201308	39	39	296
201309	33	33	329
201310	43	43	372
201311	33	33	405
201312	32	32	437
201313	33	33	470
201314	40	40	510
201315	35	35	545
201316	42	42	587
201317	48	48	635
201318	48	48	683
201319	45	45	728

. . . . . .

2. User Growth Analysis

Objective: Analyze the growth of users over time for a product.

SQL Query:

```
SELECT
   YEARWEEK(created_at, 1) AS year_week,
   COUNT(user_id) AS new_users,
   SUM(COUNT(user_id)) OVER (ORDER BY YEARWEEK(created_at, 1)) AS
cumulative_users
FROM
   users
GROUP BY
   YEARWEEK(created_at, 1)
ORDER BY
   year_week;
```

Result:

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

year_week	new_users	cumulative_users
201301	26	26
201302	29	55
201303	47	102
201304	36	138
201305	30	168
201306	48	216
201307	41	257
201308	39	296
201309	33	329
201310	43	372
201311	33	405
201312	32	437
201313	33	470
201314	40	510
201315	35	545

. . . . . .

3. Weekly Retention Analysis

Objective: Calculate the weekly retention of users based on their sign-up cohort.

SQL Query:

```
WITH user_cohorts AS (
   -- Get first sign-up event
   SELECT
      user_id,
      MIN(occurred_at) AS first_signup_date
   FROM
      events
   WHERE
      event_type = 'signup_flow'
   GROUP BY
      user_id
),
user_activity AS (
   -- Get activity week and sign-up week
   SELECT
      e.user_id,
      EXTRACT(WEEK FROM e.occurred_at) AS activity_week,
      EXTRACT(YEAR FROM e.occurred_at) AS activity_year,
      uc.first_signup_date,
      EXTRACT(WEEK FROM uc.first_signup_date) AS signup_week,
      EXTRACT(YEAR FROM uc.first_signup_date) AS signup_year
   FROM
      events e
   JOIN
      user_cohorts uc
   ON e.user_id = uc.user_id
   WHERE
```

```
        e.event_type = 'engagement'
),
weekly_retention AS (
    -- Calculate weekly retention
    SELECT
        ua.signup_year,
        ua.signup_week,
        ua.activity_year,
        ua.activity_week,
        COUNT(DISTINCT ua.user_id) AS retained_users
    FROM
        user_activity ua
    WHERE
        (ua.activity_year > ua.signup_year) OR
        (ua.activity_year = ua.signup_year AND ua.activity_week >=
ua.signup_week)
    GROUP BY
        ua.signup_year,
        ua.signup_week,
        ua.activity_year,
        ua.activity_week
)
SELECT
    signup_year,
    signup_week,
    activity_year,
    activity_week,
    retained_users
FROM
    weekly_retention
ORDER BY
    signup_year, signup_week, activity_year, activity_week;
```

Result:

signup_year	signup_week	activity_year	activity_week	retained_users
2014	17	2014	17	72
2014	17	2014	18	59
2014	17	2014	19	24
2014	17	2014	20	16
2014	17	2014	21	11
2014	17	2014	22	16
2014	17	2014	23	11
2014	17	2014	24	9
2014	17	2014	25	6
2014	17	2014	26	8
2014	17	2014	27	8
2014	17	2014	28	8
2014	17	2014	29	7
2014	17	2014	30	9
2014	17	2014	31	6
2014	17	2014	32	5
2014	17	2014	33	1
2014	17	2014	34	2
2014	18	2014	18	163
2014	18	2014	19	114
2014	18	2014	20	73

4. Weekly Engagement Per Device

Objective: Measure the activeness of users on a weekly basis per device.

SQL Query:

```
SELECT
   EXTRACT(YEAR FROM occurred_at) AS year,
   EXTRACT(WEEK FROM occurred_at) AS week,
   device,
   COUNT(DISTINCT user_id) AS engaged_users
FROM
   events
```

```
GROUP BY
   year, week, device
ORDER BY
   year, week, device;
```

Result:

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

year	week	device	engaged_users
2014	17	acer aspire desktop	9
2014	17	acer aspire notebook	20
2014	17	amazon fire phone	4
2014	17	asus chromebook	21
2014	17	dell inspiron desktop	18
2014	17	dell inspiron notebook	46
2014	17	hp pavilion desktop	14
2014	17	htc one	16
2014	17	ipad air	27
2014	17	ipad mini	19
2014	17	iphone 4s	21
2014	17	iphone 5	65
2014	17	iphone 5s	42
2014	17	kindle fire	6
2014	17	lenovo thinkpad	86
2014	17	mac mini	6
2014	17	macbook air	54
2014	17	macbook pro	143
2014	17	nexus 10	16
2014	17	nexus 5	40

. . . . . .

5. Email Engagement Analysis

Objective: Analyze how users are engaging with the email service.

SQL Query:

```
SELECT
    user_id,
    action,
    sum(user_type) as count_engagement
FROM
    email_events
GROUP BY
    user_id, action;
```

Result:

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

user_id	action	count_engagement
0	sent_weekly_digest	17
0	email_open	5
4	sent_weekly_digest	51
4	email_open	15
4	email_clickthrough	12
8	sent_weekly_digest	51
8	email_open	9
8	email_clickthrough	3
11	sent_weekly_digest	17
11	email_open	5
11	email_clickthrough	2
17	sent_weekly_digest	17
17	email_open	4
17	email_clickthrough	1
19	sent_weekly_digest	17

. . . . .

Insights

Key insights from the project include:
1. Throughput analysis revealed smoother trends using rolling averages, indicating performance improvements.
2. Weekly engagement metrics identified spikes in user activity aligned with specific campaigns.
3. Retention analysis highlighted critical periods for user engagement after sign-up.
4. Language analysis showed dominant languages, aiding resource allocation decisions.

Result

The project successfully addressed the outlined objectives. SQL queries extracted key insights from the data, providing actionable information for business decision-making. This enhanced understanding of operational metrics and user behavior contributes to better performance monitoring and strategic planning.