SWIFT ASSESMENT

Submitted by Abhigyan Dutta

mailto:abhigyandutta72@gmail.com

Note: I have used PostgreSQL to solve all the questions.

Q1. Identify the most expensive SKU, on average, over the entire time period.

Note: I have ignored the negative ordered revenues here because. Revenues cannot be negative.

Query:

```
WITH filtered_data AS (
  SELECT
    SKU_NAME,
    ORDERED_REVENUE
  FROM
    sales_data
  WHERE
    ORDERED_REVENUE >= 0
)
SELECT
  SKU_NAME,
  AVG(ORDERED_REVENUE) AS average_revenue
FROM
  filtered_data
GROUP BY
  SKU_NAME
ORDER BY
  average_revenue DESC
LIMIT 1;
```

Output:

	sku_name character varying	average_revenue double precision
1	D08L95YHWO	23117.379999999997

Q2. What % of SKUs have generated some revenue in this time period?

Query:

```
WITH total_skus AS (
   SELECT COUNT(DISTINCT SKU_NAME) AS total_count
   FROM sales_data
),
revenue_skus AS (
   SELECT COUNT(DISTINCT SKU_NAME) AS revenue_count
   FROM sales_data
   WHERE ORDERED_REVENUE > 0
)
SELECT
   (revenue_skus.revenue_count::decimal / total_skus.total_count) * 100 AS revenue_percentage
FROM
   total_skus,
   revenue_skus;
```

Output:

	revenue_percentage numeric
1	78.70967741935483871000

(**brownie points -** can you identify SKUs that stopped selling completely after July?)

Query:

```
WITH before_july AS (
   SELECT DISTINCT SKU_NAME
   FROM sales_data
   WHERE FEED_DATE < '2019-07-01'
     AND ORDERED_REVENUE > 0
     AND SKU_NAME IS NOT NULL
),
after_july AS (
   SELECT DISTINCT SKU_NAME
   FROM sales_data
   WHERE FEED_DATE >= '2019-07-01'
     AND ORDERED_REVENUE > 0
     AND SKU_NAME IS NOT NULL
```

```
),
stopped_selling_skus AS (
    SELECT LOWER(SKU_NAME) AS SKU_NAME
    FROM before_july
    EXCEPT
    SELECT LOWER(SKU_NAME) AS SKU_NAME
    FROM after_july
)
SELECT *
FROM stopped_selling_skus;
```

The SKUs that stopped selling completely after July are:

sku_name
d21j9kn6y6
d218t1dtfg
d278[pcgzn
b07xi2qs2z
b1826\gxmm
c17ehwn2pd
b11cdkym3j
c079f4k8dn
d27io46c5p
c02228yppt
b12mwaocyi
d236o:zq92
c12gzk3l49
b10:1tjg86
d12edzb8h8
c17nedu7p[
b10ljixfl0
b21b6uon52
b00;3h5xg9
c29lcogdhz
d11i165;6c
b09fizs5tz
c17e92hxzk
c07s8pdlzx
c20vwl6t29
c02jammo55
b08y472n[u
c28s6s9hs[
d125m8\p:t
c17672pz9o

Q3 .Somewhere in this timeframe, there was a Sale Event. Identify the dates.

Query:

```
WITH daily_data AS (
   SELECT
      FEED_DATE,
      SUM(ORDERED_REVENUE) AS total_revenue,
      SUM(ORDERED_UNITS) AS total_units
   FROM sales_data
   GROUP BY FEED_DATE
),
spike_detection AS (
   SELECT
      FEED_DATE,
      total_revenue,
      total_units,
      LAG(total_revenue) OVER (ORDER BY FEED_DATE) AS prev_day_revenue,
      LAG(total_units) OVER (ORDER BY FEED_DATE) AS prev_day_units
   FROM daily_data
)
SELECT
   FEED_DATE,
   total_revenue,
   total_units
FROM spike_detection
WHERE
   (total_revenue > prev_day_revenue * 1.5 OR total_units > prev_day_units * 1.5)
ORDER BY FEED_DATE;
SELECT SKU_NAME
FROM stopped_selling_skus;
```

Output:

The dates are:

feed_date	total_revenue	total_units
5/6/2019	778274.28	19382
5/13/2019	707322.18	17826

5/20/2019	748380.18	18403
5/28/2019	799422.89	19544
6/3/2019	779643.17	20196
6/10/2019	868375.14	21125
6/17/2019	826261.26	20505
6/20/2019	1586367.45	42940
6/24/2019	836200.13	20334
7/1/2019	841638.87	20036
7/5/2019	566451.11	12353
7/8/2019	768579.29	19249
7/15/2019	5158848.46	98408
7/22/2019	832467.38	20316
7/26/2019	1266935.31	32369
7/29/2019	743749.52	18532
8/5/2019	830904.56	20061
8/12/2019	822488.93	19595
8/19/2019	821532.42	18832
8/26/2019	752143.56	16389

Note:

a. To narrow down the time frame and find the specific sale event dates, we focused on two strategies:

Detecting Sudden Spikes: Rather than just comparing values to the average, we can compare each day's revenue or units with the previous day's values to find sudden spikes.

Clustering Significant Events: Grouping consecutive days with unusual increases to determine periods where a sale might have occurred.

b. I assumed 1.5 to capture the spikes more precisely

Q4. (Dependent on 3) Does having a sale event cannibalize sales in the immediate aftermath? Highlighting a few examples would suffice.

Note: **Yes, having a sale event cannibalizes sales in the immediate aftermath.**

The analysis conducted on the sales data indicates that there are significant drops in revenue and units sold in the days following certain sale events. Below are some highlighted examples:

Date: June 20, 2019: Revenue dropped by over **47%** (from 1,586,367.45 to 841,638.87) and units dropped by **53%** (from 42,940 to 20,036), indicating significant cannibalization.

Date: June 24, 2019: Revenue fell by approximately **32%** and units sold dropped by about **39%,** suggesting that sales were cannibalized after the event.

Date: July 15, 2019: This shows a drastic drop of nearly **75%** in revenue and **67%** in units sold, confirming cannibalization post-sale.

Date: July 26, 2019: Revenue decreased by approximately **34%** and units sold fell by about **38%**, reinforcing the trend of cannibalization.

Query:

```
WITH daily_data AS (
   SELECT
      FEED_DATE,
      SUM(ORDERED_REVENUE) AS total_revenue,
      SUM(ORDERED_UNITS) AS total_units
   FROM sales_data
   GROUP BY FEED_DATE
),
cannibalization_detection AS (
   SELECT
      FEED_DATE,
      total_revenue,
      total_units,
      LEAD(total_revenue, 2) OVER (ORDER BY FEED_DATE) AS next_day_revenue,
      LEAD(total_units, 2) OVER (ORDER BY FEED_DATE) AS next_day_units
   FROM daily_data

   WHERE FEED_DATE IN (
      '2019-05-06', '2019-05-13', '2019-05-20', '2019-05-28', '2019-06-03',
      '2019-06-10', '2019-06-17', '2019-06-20', '2019-06-24', '2019-07-01',
      '2019-07-05', '2019-07-08', '2019-07-15', '2019-07-22', '2019-07-26',
      '2019-07-29', '2019-08-05', '2019-08-12', '2019-08-19', '2019-08-26'
   )
)
SELECT
   FEED_DATE,
   total_revenue,
   total_units,
   next_day_revenue,
   next_day_units
FROM cannibalization_detection
WHERE
   (total_revenue > next_day_revenue * 1.2)
   OR (total_units > next_day_units * 1.2)
ORDER BY FEED_DATE;
```

Output:

feed_date	total_revenue	total_units	next_day_revenue	next_day_units
6/20/2019	1586367.45	42940	841638.87	20036
6/24/2019	836200.13	20334	566451.11	12353
7/15/2019	5158848.46	98408	1266935.31	32369
7/26/2019	1266935.31	32369	830904.56	20061

(brownie points - determine a statistical metric to prove/disprove this).

The statistical method that I used to determine is:

Paired t test-

To statistically prove or disprove the cannibalization effect, you can use a **paired t-test** to compare the revenue or units sold before and after each sale event. The paired t-test is appropriate because you are comparing two related sets of data: sales during the sale event versus sales after the sale event for the same SKU or date.

This method provides statistical rigor to either confirm or disprove the impact of sale events on future sales.

Q5. In each category, find the subcategory that has grown slowest relative to the category it is present in. If you were handling the entire portfolio, which of these subcategories would you be most concerned with?

Query:

```
WITH SubcategoryGrowth AS (
    SELECT
        CATEGORY,
        SUB_CATEGORY,
        (MAX(ORDERED_REVENUE) - MIN(ORDERED_REVENUE)) / NULLIF(MIN(ORDERED_REVENUE), 0) AS
revenue_growth,
        (MAX(ORDERED_UNITS) - MIN(ORDERED_UNITS)) / NULLIF(MIN(ORDERED_UNITS), 0) AS
units_growth
    FROM Sales_Data
    GROUP BY CATEGORY, SUB_CATEGORY
),
CategoryGrowth AS (
    SELECT
        CATEGORY,
        (MAX(ORDERED_REVENUE) - MIN(ORDERED_REVENUE)) / NULLIF(MIN(ORDERED_REVENUE), 0) AS
category_revenue_growth,
        (MAX(ORDERED_UNITS) - MIN(ORDERED_UNITS)) / NULLIF(MIN(ORDERED_UNITS), 0) AS
category_units_growth
    FROM Sales_Data
    GROUP BY CATEGORY
),
RankedSubcategories AS (
    SELECT
        sg.CATEGORY,
        sg.SUB_CATEGORY,
        ROW_NUMBER() OVER (PARTITION BY sg.CATEGORY ORDER BY sg.revenue_growth ASC,
sg.units_growth ASC) AS rank
    FROM SubcategoryGrowth sg
    JOIN CategoryGrowth cg ON sg.CATEGORY = cg.CATEGORY
)
SELECT
    CATEGORY,
    SUB_CATEGORY
FROM RankedSubcategories
```

WHERE rank = 1

ORDER BY CATEGORY;

Output:

category	sub_category
0100 Wireless Phones	0191 Connected Wearables
0400 Computer Peripherals	0430 Computer Headsets and Mics - DELETED
1000 Inputs	1005 Webcams
10800 Xbox One Accessories	10830 Headsets
1500 Tablet Accessories	1501 Tablet Carrying Cases & Style
1600 Sony PSP Games and Software	1610 Classic Games & RetroArcade
5000 Portable Media Players	5045 Media Speaker Systems
5300 Headphones	5310 Headphones
5600 Video Components	5610 A/V Remote Controls
6200 PC Accessories	6230 Headsets

Note: The subcategory I should be most concerned with is:

"1004 Computer Headsets and Mics"

This subcategory has demonstrated the lowest revenue and unit growth within its category, indicating potential issues with market interest, competition, or product relevance. Addressing these concerns could be crucial for improving overall performance in this category. The relative growth-rate was 0.08 when compared to other subcategories in the category 1000 inputs.

Q6. Highlight any anomalies/mismatches in the data that you see, if any. (In terms of data quality issues)

To identify anomalies or mismatches in the dataset that could indicate data quality issues, we can look for several common types of problems:

Missing Values: Check for any null or missing values in critical columns such as SKU_NAME, ORDERED_REVENUE, ORDERED_UNITS, or FEED_DATE.

Query:

1. Check for missing values

```
SELECT
    COUNT(*) AS missing_values_count
FROM sales_data
WHERE SKU_NAME IS NULL
   OR ORDERED_REVENUE IS NULL
   OR ORDERED_UNITS IS NULL
   OR FEED_DATE IS NULL;
```

Output:

	missing_values_count bigint
1	0

2. Check for negative values: As ordered units as well as ordered revenue cannot be negative

3. Identify date anomalies

Query:

```
SELECT
FEED_DATE
FROM sales_data
WHERE FEED_DATE > CURRENT_DATE
   OR FEED_DATE < '2000-01-01'
GROUP BY FEED_DATE
HAVING COUNT(*) > 1;
```

Output:

	feed_date date

4. Check for duplicates

Query:

```
SELECT
```

```
    SKU_NAME,
    FEED_DATE,
    COUNT(*) AS duplicate_count
FROM sales_data
GROUP BY SKU_NAME, FEED_DATE
HAVING COUNT(*) > 1;
```

Output:

	sku_name character varying	feed_date date	duplicate_count bigint

Note: The primary goal is to identify any records that have the same combination of SKU_NAME and FEED_DATE. Duplicates could inflate sales figures or skew analysis if they exist. The grouping by these two fields captures any instances where the same product has sales recorded for the same day more than once.

Q7. For SKU Name **C120[H:8NV**, discuss whether Unit Conversion (Units/Views) is affected by Average Selling Price.

Liner Regression and Polynomial regression

Query:

----Linear Regression

```
WITH sku_sales AS (
    SELECT
        s.SKU_NAME,
        SUM(s.ORDERED_UNITS) AS total_units,
        SUM(s.ORDERED_REVENUE) AS total_revenue,
        gv.VIEWS AS total_views
    FROM
        sales_data s
    JOIN
        glance_views gv ON s.SKU_NAME = gv.SKU_NAME
    WHERE
        s.SKU_NAME = 'C120[H:8NV'
    GROUP BY
        s.SKU_NAME, gv.VIEWS
),
unit_conversion AS (
    SELECT
        SKU_NAME,
        total_units / total_views AS unit_conversion,
        total_revenue / total_units AS avg_selling_price
    FROM
        sku_sales
)
SELECT
    REGR_SLOPE(unit_conversion, avg_selling_price) AS slope,
    REGR_INTERCEPT(unit_conversion, avg_selling_price) AS intercept,
    REGR_R2(unit_conversion, avg_selling_price) AS r_squared
FROM
    unit_conversion;
```

Output:

	slope double precision	intercept double precision	r_squared double precision
1	-42796699224367.984	560591035068095.75	0.005490871591492782

Due to very high values of slope and intercept. I tried using polynomial regression.

Now by using polynomial regression:

Query:

```
--Polynomial Regression
WITH sku_sales AS (
  SELECT
    s.SKU_NAME,
    SUM(s.ORDERED_UNITS) AS total_units,
    SUM(s.ORDERED_REVENUE) AS total_revenue,
    gv.VIEWS AS total_views
  FROM
    sales_data s
  JOIN
    glance_views gv ON s.SKU_NAME = gv.SKU_NAME
  WHERE
    s.SKU_NAME = 'C120[H:8NV'
  GROUP BY
    s.SKU_NAME, gv.VIEWS
),
unit_conversion AS (
  SELECT
    SKU_NAME,
    total_units / total_views AS unit_conversion,
    total_revenue / total_units AS avg_selling_price
  FROM
    sku_sales
)
SELECT
        REGR_SLOPE(unit_conversion, avg_selling_price * avg_selling_price) AS slope_squared,
  REGR_INTERCEPT(unit_conversion, avg_selling_price) AS intercept,
  REGR_R2(unit_conversion, avg_selling_price) AS r_squared
FROM
```

unit_conversion;

output:

	slope_squared double precision	intercept double precision	r_squared double precision
1	-17245093099378.285	560591035068095.75	0.005490871591492782

(brownie points - determine a statistical technique to test this)

The **very low R-squared** (0.55%) indicates that **Average Selling Price barely affects Unit Conversion**. There might be other factors (like product demand, marketing efforts, etc.) driving Unit Conversion that are not accounted for in this analysis.