Analytical SQL Project

Eslam Fayez
ITI

Dataset Background: Customers has purchasing transaction that we shall be monitoring to get intuition behind each customer behavior to target the customers in the most efficient and proactive way, to increase sales/revenue , improve customer retention and decrease churn.

Invoice	StockCode	Quantity	InvoiceDate	Price	Customer_ID	Country
537215	85124C	12	12/5/2010 15:38	2.55	12747	United Kingdom
537215	85124B	6	12/5/2010 15:38	2.55	12747	United Kingdom
537215	84879	16	12/5/2010 15:38	1.69	12747	United Kingdom
537215	85062	24	12/5/2010 15:38	1.65	12747	United Kingdom
537215	85064	6	12/5/2010 15:38	5.45	12747	United Kingdom
537215	82484	36	12/5/2010 15:38	5.55	12747	United Kingdom
537215	21136	8	12/5/2010 15:38	1.69	12747	United Kingdom
538537	22795	16	12/13/2010 10:41	5.95	12747	United Kingdom
538537	48138	2	12/13/2010 10:41	7.95	12747	United Kingdom
538537	82494L	24	12/13/2010 10:41	2.55	12747	United Kingdom
538537	84879	24	12/13/2010 10:41	1.69	12747	United Kingdom
538537	85062	12	12/13/2010 10:41	1.65	12747	United Kingdom
538537	21754	3	12/13/2010 10:41	5.95	12747	United Kingdom
538537	82484	12	12/13/2010 10:41	5.55	12747	United Kingdom

you will be required to answer using SQL Analytical functions you have learnt in the course.

My Analysis here depends on 3 main factors :

1- Time Analysis for sales data

2- Analysis for our product sales to manage our inventory management, pricing, and resource allocation depending on

3- Customer Segmentation

Query1)

```
with cte as(
   SELECT TO_DATE(INVOICEDATE, 'MM/DD/YYYY HH24:MI') AS INVOICEDATE,
         invoice,
         stockcode,
         quantity,
         price,
         customer_id,
         country
   FROM tableRetail)

SELECT
      EXTRACT(MONTH FROM invoicedate) AS SalesMonth,
      SUM(quantity * price) AS TotalSales
FROM cte
GROUP BY EXTRACT(YEAR FROM INVOICEDATE), EXTRACT(MONTH FROM INVOICEDATE)

ORDER BY EXTRACT(YEAR FROM INVOICEDATE), EXTRACT(MONTH FROM INVOICEDATE);
```

SALESMONTH	TOTALSALES
12	13422.96
1	9541.29
2	13336.84
3	17038.01
4	10980.51
5	19496.18
6	13517.01
7	15664.54
8	38374.64
9	27853.82
10	19735.07
11	45633.38
12	11124.13

Business Description: This query provides This query is useful for analyzing sales performance over time.
Businesses can use the results to:
Identify peak sales months.
Monitor trends and seasonality.

Query2)

```
WITH MonthlySales AS (
   SELECT TO_DATE(INVOICEDATE,'MM/DD/YYYY HH24:MI') AS actual_date,
         quantity,
         price
   FROM tableretail
)
SELECT EXTRACT(YEAR FROM actual_date) AS year,
      EXTRACT(MONTH FROM actual_date) AS month,
      SUM(price * quantity) AS total_sales,
      LAG(SUM(price * quantity)) OVER (ORDER BY EXTRACT(YEAR FROM actual_date), EXTRACT(MONTH FROM
actual_date)) AS prev_month_sales,
      (SUM(price * quantity) - LAG(SUM(price * quantity)) OVER (ORDER BY EXTRACT(YEAR FROM actual_date),
EXTRACT(MONTH FROM actual_date))) / LAG(SUM(price * quantity)) OVER (ORDER BY EXTRACT(YEAR FROM
actual_date), EXTRACT(MONTH FROM actual_date)) * 100 AS growth_rate,
      RANK() OVER(ORDER BY sum( QUANTITY * PRICE) DESC) AS MONTH_RANK
FROM MonthlySales
GROUP BY EXTRACT(YEAR FROM actual_date), EXTRACT(MONTH FROM actual_date)
ORDER BY EXTRACT(YEAR FROM actual_date), EXTRACT(MONTH FROM actual_date);
```

YEAR	MONTH	TOTAL_S...	PREV_MONTH_SALES	GROWTH_RATE	MONTH_RANK
2010	12	13422.96			9
2011	1	9541.29	13422.96	-28.9181372811958	13
2011	2	13336.84	9541.29	39.7802603211935	10
2011	3	17038.01	13336.84	27.7514763617169	6
2011	4	10980.51	17038.01	-35.5528609268336	12
2011	5	19496.18	10980.51	77.5525909088011	5
2011	6	13517.01	19496.18	-30.6684181208832	8
2011	7	15664.54	13517.01	15.8876112394679	7
2011	8	38374.64	15664.54	144.977765066832	2
2011	9	27853.82	38374.64	-27.4160747827211	3
2011	10	19735.07	27853.82	-29.1477075675796	4
2011	11	45633.38	19735.07	131.229886694093	1
2011	12	11124.13	45633.38	-75.6228225917081	11

Business Description: This query provides a monthly overview of sales performance, with calculating growth rate for every month to be more accurate than last query which will be more efficient to calculate sales trend and sales forecasting.

Query3)

```
WITH HOUR_SalesAS (
   SELECT
      TO_CHAR(TO_DATE(INVOICEDATE, 'MM/DD/YYYY HH24:MI'), 'HH AM') AS hourr,
      SUM(QUANTITY * PRICE) AS Total_Income,
      COUNT(*) AS number_of_invoices
   FROM tableRetail
   GROUP BY  TO_CHAR(TO_DATE(INVOICEDATE, 'MM/DD/YYYY HH24:MI'), 'HH AM')
)

SELECT
   hourr,
   Total_Income,
   number_of_invoices,
   rank() OVER (ORDER BY Total_Income DESC) AS row_num
 FROM HOUR_Sales;
```

HOURR	TOTAL_INCOME	NUMBER_OF_INVOICES	ROW_NUM
01 PM	44414.87	2173	1
12 PM	39473.63	2439	2
03 PM	35772.63	1827	3
10 AM	28699.67	944	4
02 PM	24235.95	1710	5
11 AM	21820.18	1713	6
06 PM	21755.01	120	7
09 AM	15988.75	550	8
04 PM	11249.71	698	9
05 PM	8260.93	372	10
08 AM	2065.29	80	11
07 PM	1362.57	199	12
07 AM	535.5	1	13
08 PM	83.69	32	14

Business Description: The query aims to analyze sales data at an hourly level. It calculates the total income and the number of invoices for each hour of the day, By analyzing hourly sales, businesses can identify peak hours and allocate resources accordingly to identify sales pattern

Query4)

```
WITH MonthSales AS (
   SELECT
      TO_CHAR(TO_DATE(INVOICEDATE, 'MM/DD/YYYY HH24:MI'), 'DD') AS actual_date,
      TO_CHAR(TO_DATE(INVOICEDATE, 'MM/DD/YYYY HH24:MI'), 'Mon YYYY') AS monthh,
      quantity,
      price
   FROM tableretail
),
avgg AS (
   SELECT
      monthh,
      SUM(CASE WHEN TO_NUMBER(actual_date) <= 10 THEN Price * Quantity ELSE 0 END) AS
total_sales_of_first_10_days,
      SUM(CASE WHEN TO_NUMBER(actual_date) BETWEEN 11 AND 20 THEN Price * Quantity ELSE 0 END) AS
total_sales_of_second_10_days,
      SUM(CASE WHEN TO_NUMBER(actual_date) BETWEEN 21 AND 31 THEN Price * Quantity ELSE 0 END) AS
total_sales_of_third_10_days
   FROM MonthSales
   GROUP BY monthh
)
SELECT
   sum(total_sales_of_first_10_days) AS avg_10,
   sum(total_sales_of_second_10_days) AS avg_20,
   sum(total_sales_of_third_10_days) AS avg_30
FROM avgg;
```

AVG_10	AVG_20	AVG_30
99662.53	79048.59	77007.26

Business Insight: The query analyzes sales data based on the day of the month.
It calculates the total sales for the first 10 days, second 10 days, and third 10 days of each month. Which will play crucial role to identify the following :
Sales Distribution: Understanding sales patterns across different parts of the month helps businesses allocate resources effectively.
Promotional Timing: Businesses can plan promotions or marketing campaigns based on the sales distribution throughout the month.
Inventory Management: Knowing when sales peak allows for better inventory planning and stock replenishment

Query5)

```
WITH product_sales AS (
    SELECT Distinct
        stockcode,
        ROUND(SUM(quantity * price) OVER (PARTITION BY stockcode)) AS stock_sales
    FROM tableRetail
)
SELECT
    stockcode,
    stock_sales,
    RANK() OVER (ORDER BY stock_sales DESC) AS product_rnk
FROM product_sales
 ORDER BY product_rnk;
```

Query6)

```
select distinct StockCode , sum(quantity ) over(partition by StockCode) as Quantity_sold

from tableRetail

where quantity > 0

order by Quantity_sold desc;
```

STOCKCODE	STOCK_SALES	PRODUCT_RNK
84879	9115	1
22197	4323	2
21787	4059	3
22191	3461	4
23203	3357	5
21479	2736	6
23215	2697	7
22970	2494	8
22570	2458	9
22992	2308	10

Business Insights: The query analyzes sales and quantity data for different stockcode items. Which will play crucial role to identify the following :

Top-Selling Products: By examining the stock_sales, businesses can identify which products contribute significantly to overall revenue.

Inventory Management: Knowing which products sell well allows businesses to optimize inventory levels.

Pricing Strategies: Analyzing top-selling products helps determine pricing strategies.

Customer Segementation:

On the previous data set:

implement a Monetary model for customers behavior for product purchasing and segment each customer based on the below groups Champions - Loyal Customers - Potential Loyalists – Recent Customers – Promising - Customers Needing Attention - At Risk - Cant Lose Them – Hibernating – Lost The customers will be grouped based on 3 main values

• Recency => how recent the last transaction is (Hint: choose a reference date, which is the most recent purchase in the dataset)

• Frequency => how many times the customer has bought from our store

• Monetary => how much each customer has paid for our products As there are many groups for each of the R, F, and M features, there are also many potential permutations, this number is too much to manage in terms of marketing strategies. For this, we would decrease the permutations by getting the average scores of the frequency and monetary (as both of them are indicative to purchase volume anyway)

Label each customer based on the below values

Group name	Recency score	AVG(Frequency & Monetary) score
Champions	5	5
	5	4
	4	5
Potential Loyalists	5	2
	4	2
	3	3
	4	3
Loyal Customers	5	3
	4	4
	3	5
	3	4
Recent Customers	5	1
Promising	4	1
	3	1
Customers Needing Attention	3	2
	2	3
	2	2
At Risk	2	5
	2	4
	1	3
Cant Lose Them	1	5
	1	4
Hibernating	1	2
Lost	1	1

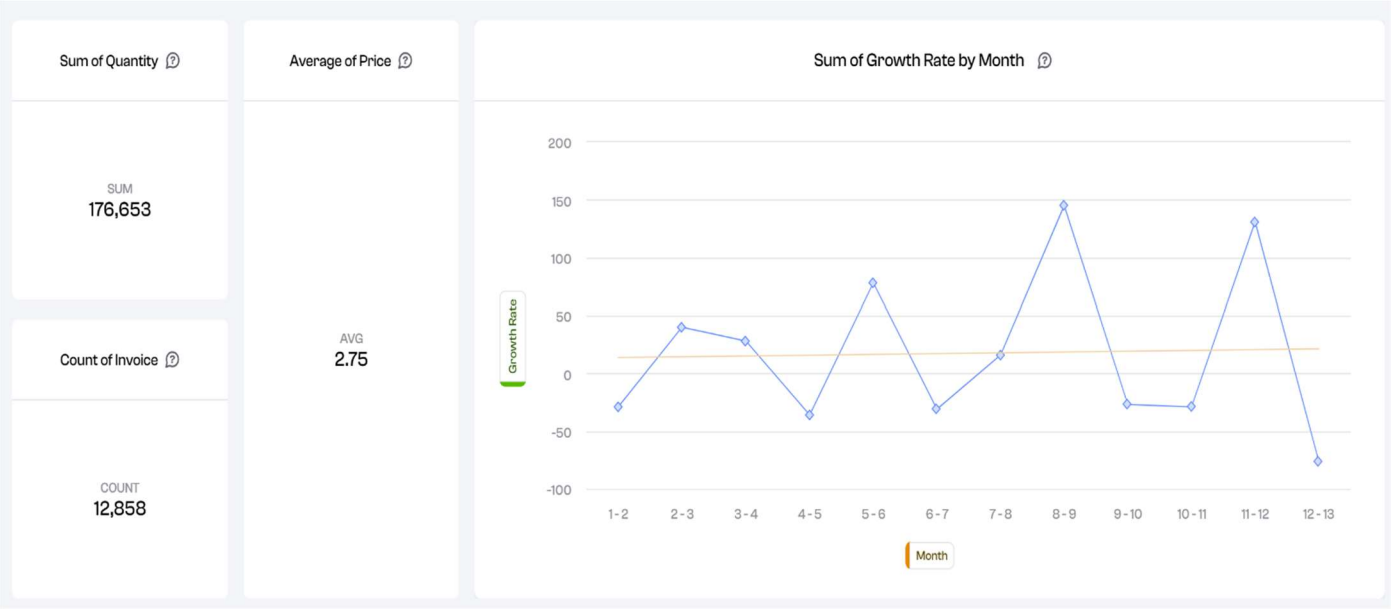
Here's the query:

```
WITH update_retail AS (
    SELECT
        customer_id,
        Quantity,
        Price,
        Invoice,
        ROUND(MAX(TO_DATE(InvoiceDate, 'MM/DD/YYYY HH24:MI')) OVER () - MAX(TO_DATE(InvoiceDate,
'MM/DD/YYYY HH24:MI')) OVER (PARTITION BY customer_id)) AS recency
    FROM tableRetail
),
cte_2 AS (
    SELECT
        customer_id,
        COUNT(DISTINCT invoice) AS frequency,
        SUM(price * quantity) AS monetary,
        recency
    FROM update_retail
    GROUP BY customer_id, recency
),
cte_3 AS (
    SELECT
        customer_id,
        recency,
        frequency,
        NTILE(5) OVER (ORDER BY frequency ) AS F_score,
        monetary,
        ROUND(PERCENT_RANK() OVER (ORDER BY monetary ), 2) AS monetary_percent_rank
    FROM cte_2
),
cte_4 AS (
    SELECT
        customer_id,
        recency,
        frequency,
        monetary,
        monetary_percent_rank,
        NTILE(5) OVER (ORDER BY recency DESC) AS R_score,
        NTILE(5) OVER (ORDER BY (F_score + monetary_percent_rank) / 2) AS FM_score
    FROM cte_3
)
SELECT
    customer_id,
    recency,
    frequency,
    monetary,
    R_score,
    FM_score,
    monetary_percent_rank,
    monetary / SUM(monetary) OVER () AS monetary_percentage,
    CASE
        WHEN R_score = 5 AND FM_score IN (5, 4) THEN 'champions'
        WHEN R_score = 4 AND FM_score = 5 THEN 'champions'
        WHEN R_score = 5 AND FM_score = 2 THEN 'potential loyalist'
        WHEN R_score = 4 AND FM_score IN (2 , 3) THEN 'potential loyalist'
```

```
        WHEN R_score = 3 AND FM_score = 3 THEN 'potential loyalist'
        WHEN R_score = 5 AND FM_score = 3 THEN 'loyal customers'
        WHEN R_score = 4 AND FM_score = 4 THEN 'loyal customers'
        WHEN R_score = 3 AND FM_score IN (5, 4) THEN 'loyal customers'
        WHEN R_score = 5 AND FM_score = 1 THEN 'recent customer'
        WHEN R_score = 4 AND FM_score = 1 THEN 'promising'
        WHEN R_score = 3 AND FM_score = 1 THEN 'promising'
        WHEN R_score = 2 AND FM_score IN (3, 2) THEN 'needs attention'
        WHEN R_score = 3 AND FM_score = 2 THEN 'needs attention'
        WHEN R_score = 2 AND FM_score IN (5, 4) THEN 'At Risk'
        WHEN R_score = 1 AND FM_score = 3 THEN 'At Risk'
        WHEN R_score = 1 AND FM_score IN (5, 4) THEN 'cant lose them'
        WHEN R_score = 1 AND FM_score = 2 THEN 'Hibernating'
        WHEN R_score = 1 AND FM_score = 1 THEN 'lost'
        ELSE 'About to sleep '
    END AS Customer_segmentation
FROM cte_4
ORDER BY customer_id;
```

This query segments customers into different categories based on their purchase history. Here's a breakdown of the steps involved:

1. **Calculate customer scores:**
    - **Monetary score:** This score (between 0 and 1) reflects a customer's ranking based on their total spending compared to others.
    - **Recency score:** This score (between 1 and 5) indicates how recently a customer made a purchase, with 5 being the most recent.
    - **Frequency score:** This score (between 1 and 5) represents a customer's purchase frequency compared to others, with 5 being the most frequent.
    - **FM Score:** This combines the monetary and frequency scores (between 1 and 10) to get a holistic view of customer value.
2. **Segment customers:**
    - Based on the recency and FM scores, the query assigns each customer a segment label like "Champions," "Loyal Customer," "At Risk," etc. These labels represent different customer categories based on their purchase behavior.



Sum of Growth Rate by Month (Chart):

There were 7 inflection points, where the direction of the trend changed.

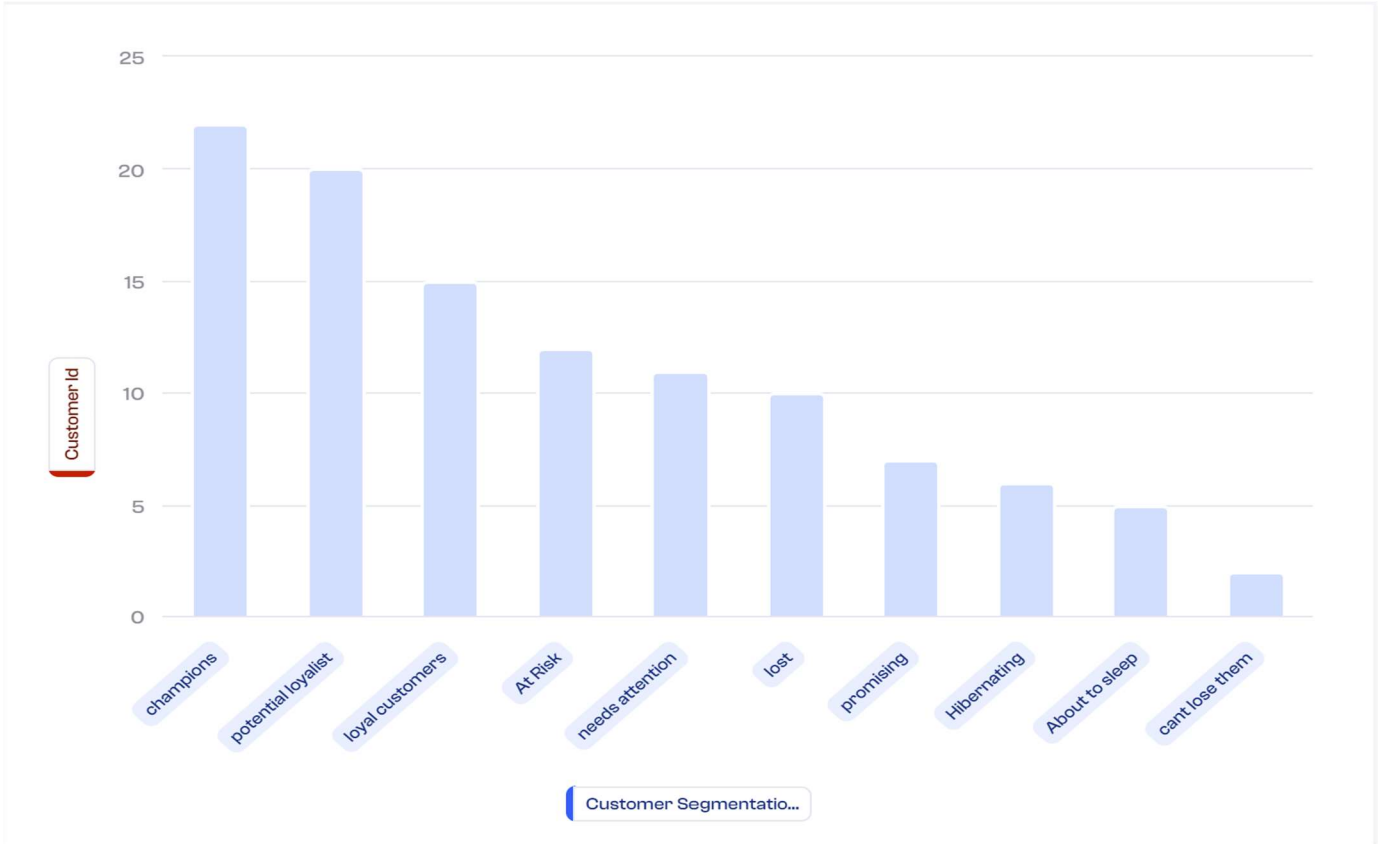
The average growth rate was 17.50.

There were 7 dips and 5 peaks observed.

- With a high confidence level, we can predict that the negative trend in growth rate won't continue in the future.

Actions:

- Based on the instability at sales we are facing almost every quarter , appropriate actions should be taken to mitigate the decline. This could include analyzing the causes of the decline, implementing strategies to boost growth, and closely monitoring the monthly data to identify any potential improvements.



Count of Customer Id by Customer segmentation:

High Performing 'Champions' in Customer Segmentation

The 'champions' category in the 'CUSTOMER_SEGMENTATION' column has shown the highest performance. These customers are likely to be loyal and valuable to the business.

Identify and Target Potential Loyal Customers:

Identifying and targeting customers who have the potential to become loyal can also help improve customer segmentation. By understanding their preferences and needs, personalized marketing campaigns can be created to attract and retain these customers.

Monitor and Address Customer Churn:

Monitor customer churn closely and take proactive measures to prevent it. By analyzing customer behavior and identifying potential reasons for churn, appropriate actions, such as improving product or service offerings, can be taken to reduce customer attrition.

Monitor the trend and adjust accordingly

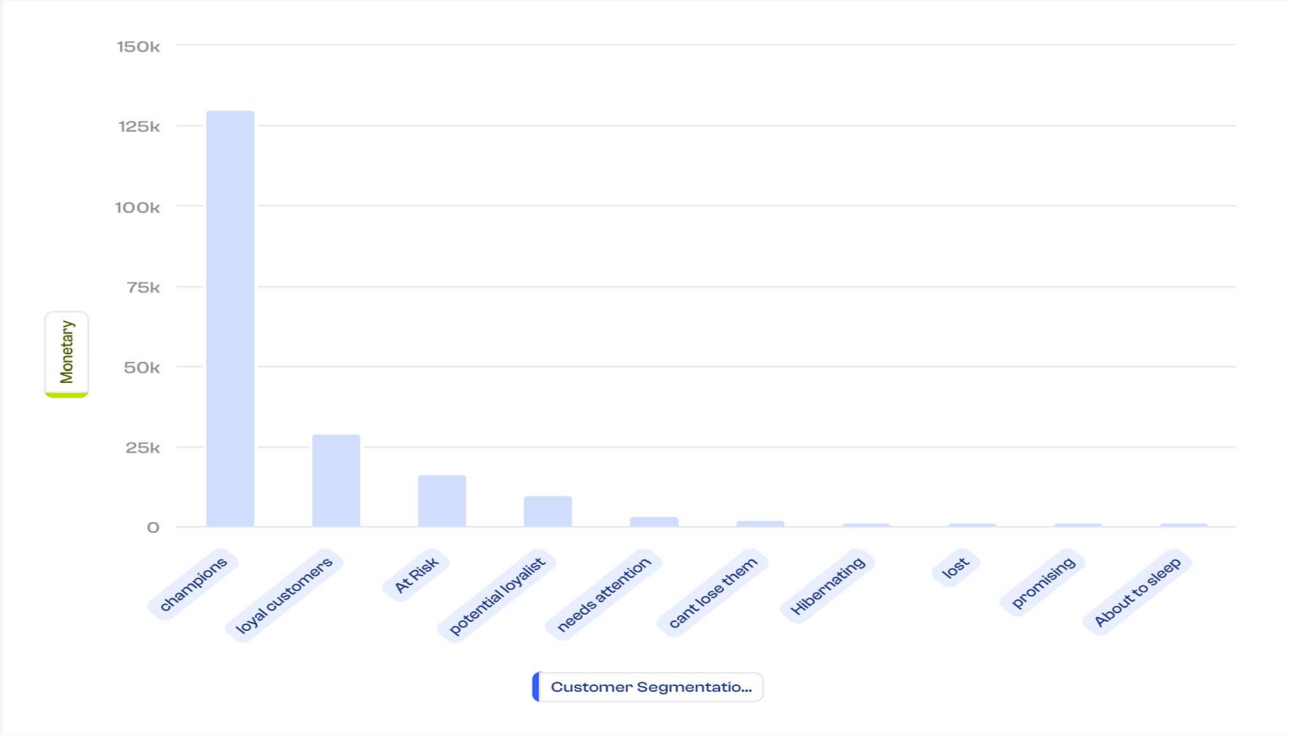
As the ascending trend in 'NUMBER_OF_INVOICES' with respect to 'HOURR' continues, it is important to monitor the data regularly and make necessary adjustments in business operations or resource allocation to accommodate the increasing demand.

Investigate the causes of steep positive and negative slopes

To understand the factors contributing to the steepest positive and negative slopes observed in the data, further investigation should be conducted. This can help in identifying potential drivers behind high and low invoice counts during specific hours.

Identify reasons for dips and peaks

Examining the 9 dips and 6 peaks observed in the data can help in understanding the reasons behind fluctuations in invoice counts. Identifying the causes can assist in implementing strategies to mitigate or leverage these variations for better business outcomes.



The graph represents different customer segments and their respective monetary contributions.

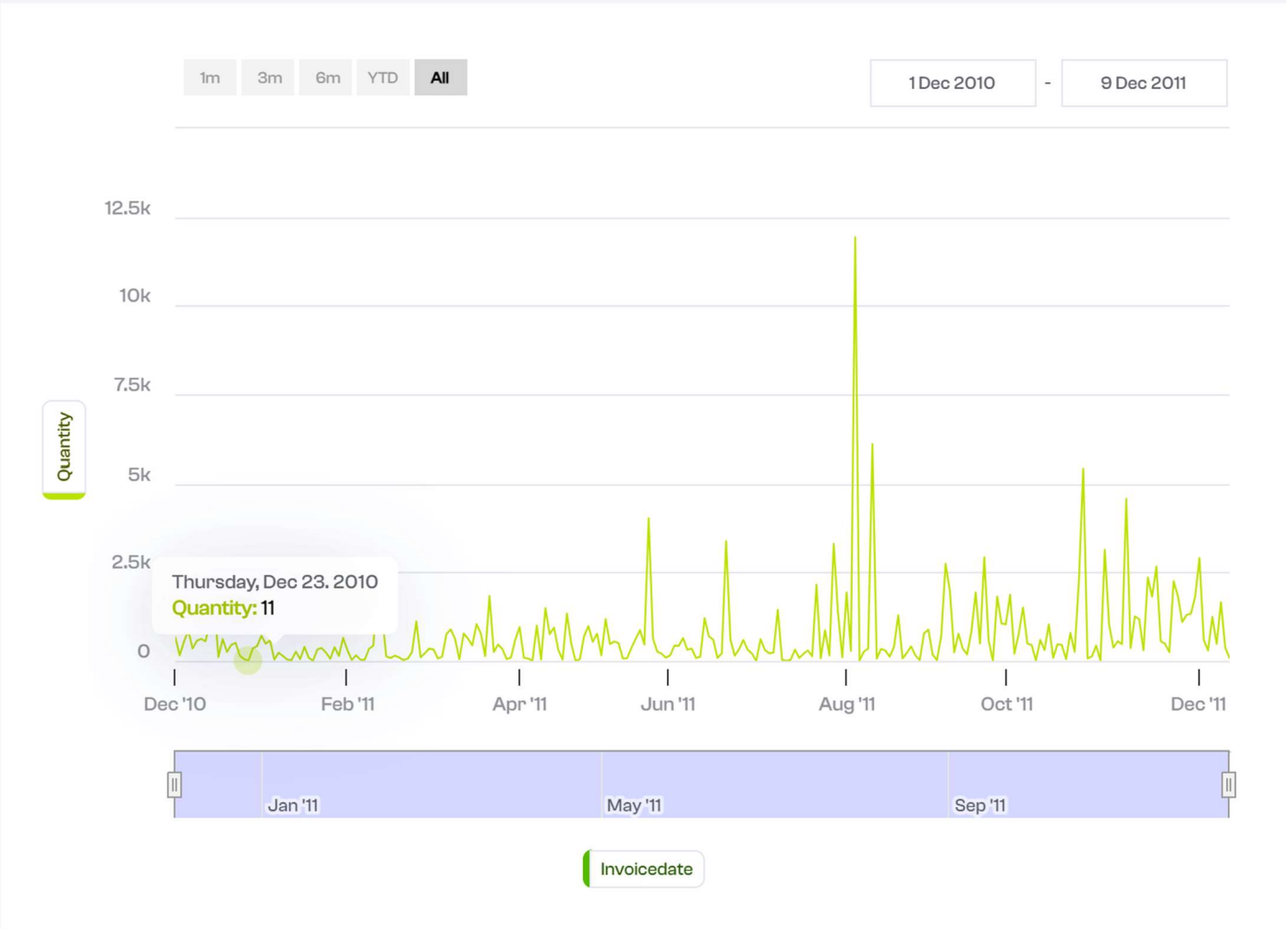
The Champions segment stands out prominently, contributing significantly to the revenue.

Champions Segment:

Despite being only 20 percent of the customer base, the Champions segment contributes 80 percent of the sales.

This suggests that the Champions are high-value customers who make larger or more frequent purchases compared to other segments.

Their spending habits result in a high customer lifetime value.



Title: Overall Trend of Global Metric is Decreasing

Explanations:

1. Major Outliers: There are 8 major outliers in the dataset, indicating significant deviations from the general trend.

2. Minor Outliers: There are 13 minor outliers in the dataset, indicating smaller deviations from the general trend.

<u>ACTIONS:</u>

- Investigate Major Outliers: Further analysis should be done to understand the reasons behind the major outliers and their impact on the overall trend.

- Monitor Minor Outliers: Although the minor outliers may not have a significant impact on the overall trend, they should still be monitored and analyzed to identify any underlying factors.

PROBLEM-3)

Given the following data set as csv file

Cust_Id	Date	Amount
145272	11/5/2019	1.59
145272	11/6/2019	2.98
145272	11/7/2019	2.19
145272	11/8/2019	8.74
1026223	11/3/2019	2
1026223	11/7/2019	33
1026223	11/8/2019	25.5
1767267	11/1/2019	132.69
1767267	11/2/2019	18.64
1767267	11/3/2019	0.4
1767267	11/4/2019	126.33
1767267	11/6/2019	1.92
1767267	11/7/2019	10.07

1)

What is the maximum number of consecutive days a customer made purchases?

Here's the query:

```
WITH cte AS (
    SELECT
        Cust_Id,
        amount,
        "Date",
        "Date" - ROW_NUMBER() OVER (PARTITION BY Cust_Id ORDER BY "Date") AS diff
    FROM
        transactions
), max_cons_days AS (
    SELECT
        cust_id,
        amount,
        COUNT(diff) OVER (PARTITION BY cust_id) AS count_per_diff
    FROM
        cte
    WHERE amount > 0
)
SELECT MAX(count_per_diff), cust_id
FROM max_cons_days
GROUP BY cust_id;
```

2)

On average, How many days/transactions does it take a customer to reach a spent threshold of 250 L.E?

From what I understood, I needed to count the number of transactions or days on which the customer made transactions until they reached a total spending of 250 L.E. in the store.

Here's the query:

```
WITH cte_1 AS (
    SELECT
        cust_id,
        "Date",
        amount,
        SUM(amount) OVER (PARTITION BY cust_id ORDER BY "Date") AS total_amount
    FROM transactions
),
cte_2 AS (
    SELECT
        cust_id,
        "Date",
        amount,
        total_amount,
        CASE WHEN total_amount >= 250 THEN 1 ELSE 0 END AS reached_threshold
    FROM cte_1
    WHERE total_amount > 250
),
cte_3 AS (
    SELECT
        cust_id,
        "Date",
        amount,
        total_amount,
        CASE WHEN total_amount < 250 THEN 1 ELSE 0 END AS before_reached_threshold
    FROM cte_1
    WHERE total_amount <= 250
),
final AS (
    SELECT
        c.cust_id,
        COUNT(c.before_reached_threshold) AS before_reaching_threshold
    FROM cte_3 c
    WHERE c.cust_id IN (SELECT cust_id FROM cte_2)
    GROUP BY c.cust_id
)
SELECT
    AVG(before_reaching_threshold) AS avg_before_reaching_threshold
  FROM final;
```