Analytical SQL Case Study

Q1:

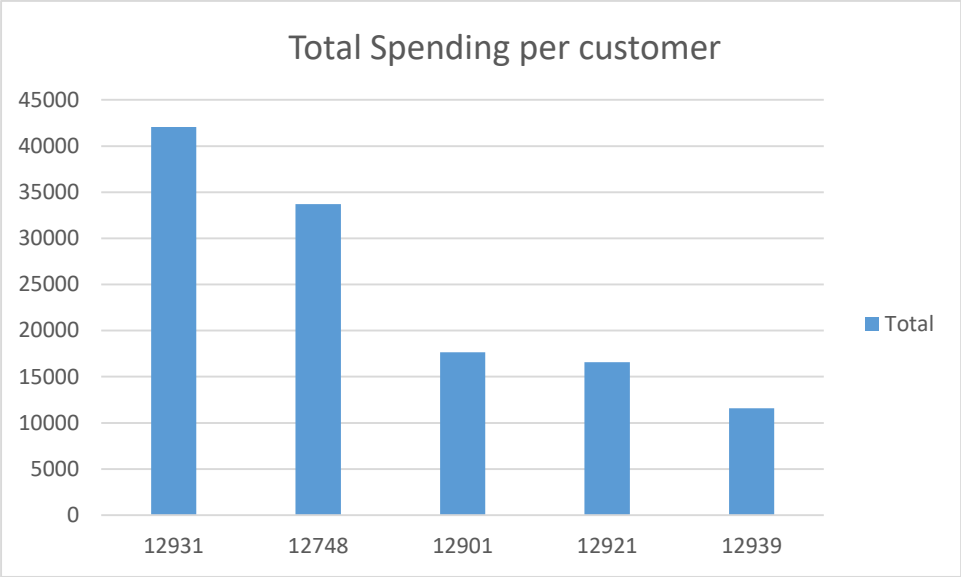
1.1

```sql
-- Calculate total spending for each customer
WITH TO_CUSTOMERS AS (
    SELECT
        Customer_ID,
        SUM(Quantity * Price) AS TotalSpending
    FROM
        tableRetail
    GROUP BY
        Customer_ID
    ORDER BY
        TotalSpending DESC
),

-- Assign rank to each customer based on total spending
ORDER_R AS (
    SELECT
        Customer_ID,
        TotalSpending,
        ROW_NUMBER() OVER(ORDER BY TotalSpending DESC) AS RANK_ORDER
    FROM
        TO_CUSTOMERS
)

-- Select top 5 customers based on rank
SELECT
    *
FROM
    ORDER_R
WHERE
    RANK_ORDER < 6;
```

query output:

CUSTOMER_ID	TOTALSPENDING	RANK_ORDER
12931	42055.96	1
12748	33719.73	2
12901	17654.54	3
12921	16587.09	4
12939	11581.8	5

The query identifies and ranks the top 5 customers based on their total spending. It helps the business focus on high-value customers for targeted marketing and retention strategies, optimizing resource allocation to maximize revenue.

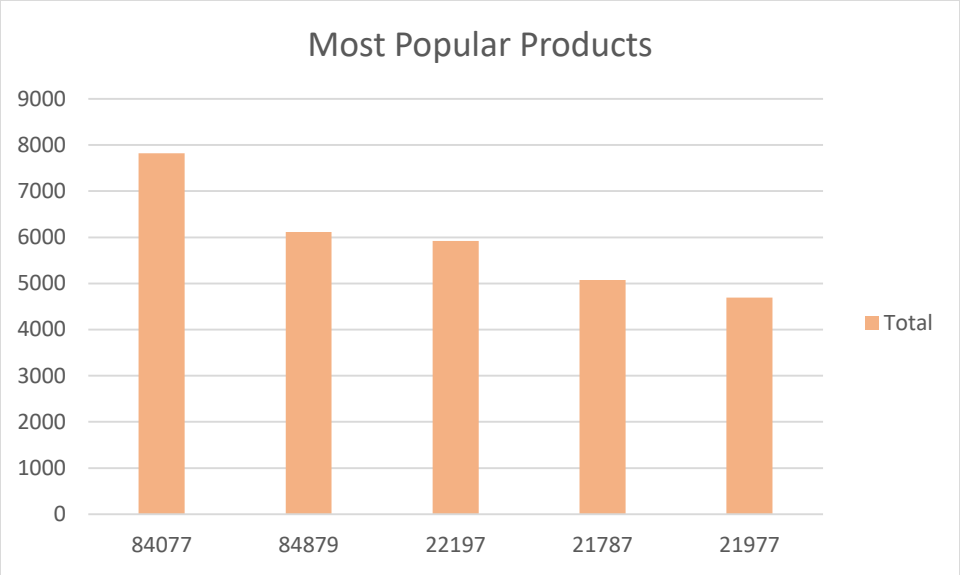


1.2

```
-- Calculate total quantity sold for each product
WITH TO_PRODUCT AS (
    SELECT
        StockCode,
        SUM(Quantity) AS TotalSold
    FROM
        tableRetail
    GROUP BY
        StockCode
    ORDER BY
        TotalSold DESC
),

-- Assign rank to each product based on total quantity sold
ORDER_R AS (
    SELECT
        StockCode,
        TotalSold,
        ROW_NUMBER() OVER(ORDER BY TotalSold DESC) AS "RANK"
    FROM
        TO_PRODUCT
)

-- Select top 5 products based on rank
SELECT
    *
FROM
    ORDER_R
WHERE
    "RANK" < 6;
```

Query Output:

STOCKCODE	TOTALSOLD	RANK
84077	7824	1
84879	6117	2
22197	5918	3
21787	5075	4
21977	4691	5



Business Meaning: Identifying the top-selling products allows businesses to focus on high-demand items. This information aids in inventory management, marketing strategies, and ensuring the availability of popular products to meet customer demand.

1.3

```
-- Extract month from InvoiceDate and calculate total sales for each month
WITH DETAILS AS (
    SELECT
        EXTRACT(MONTH FROM TO_DATE(InvoiceDate, 'MM/DD/YYYY HH24:MI')) AS SalesMonth,
        SUM(Quantity * Price) AS TotalSales
    FROM
        tableRetail
    GROUP BY
        EXTRACT(MONTH FROM TO_DATE(InvoiceDate, 'MM/DD/YYYY HH24:MI'))
)

-- Select the month with the highest total sales
SELECT
    SalesMonth,
    TotalSales
FROM (
    -- Assign rank to each month based on total sales
    SELECT
        SalesMonth,
        TotalSales,
        ROW_NUMBER() OVER (ORDER BY TotalSales DESC) AS Rank
    FROM
        DETAILS
)
-- Filter to only include the month with the highest total sales (Rank = 1)
WHERE
    Rank = 1;
```

SALESMONTH	TOTALSALES
11	45633.38

The SQL query identifies the month with the highest total sales from the tableRetail dataset. This information helps businesses focus on strategic planning, marketing, and resource allocation during peak sales periods, optimizing revenue.

1.4

```
WITH DETAILS AS (
   SELECT
      EXTRACT(MONTH FROM TO_DATE(InvoiceDate, 'MM/DD/YYYY HH24:MI')) AS SalesMonth,
      SUM(Quantity * Price) AS TotalSales
   FROM
      tableRetail
   GROUP BY
      EXTRACT(MONTH FROM TO_DATE(InvoiceDate, 'MM/DD/YYYY HH24:MI'))
)
SELECT
   SalesMonth,
   TotalSales
FROM (
   SELECT
      SalesMonth,
      TotalSales,
      ROW_NUMBER() OVER (ORDER BY TotalSales ASC) AS Rank
   FROM
      DETAILS
)
WHERE
   Rank = 1;
```

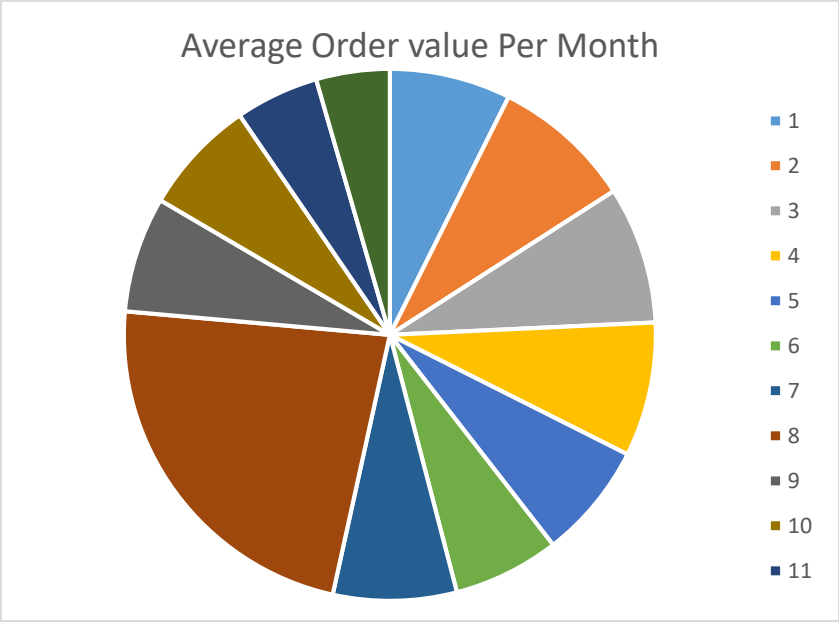
SALESMONTH	TOTALSALES
1	9541.29

The SQL query identifies the month with the lowest total sales from tableRetail. This insight enables businesses to target sales and marketing efforts during this period for potential improvements and strategic adjustments

1.5

```
-- Extract month from InvoiceDate and calculate average order value for each month
SELECT
    EXTRACT(MONTH FROM TO_DATE(InvoiceDate, 'MM/DD/YYYY HH24:MI')) AS SalesMonth,
    AVG(Quantity * Price) AS AvgOrderValue
FROM
    tableRetail
GROUP BY
    EXTRACT(MONTH FROM TO_DATE(InvoiceDate, 'MM/DD/YYYY HH24:MI'))
ORDER BY
    SalesMonth;
```

SALESMONTH	AVGORDERVALUE
1	20.6969414316703
2	23.9871223021583
3	23.4360522696011
4	22.9238204592902
5	19.8333468972533
6	18.1436375838926
7	21.0828263795424
8	64.4951932773109
9	19.7404819277108
10	19.6760418743769
11	14.26043125
12	12.5560562659846



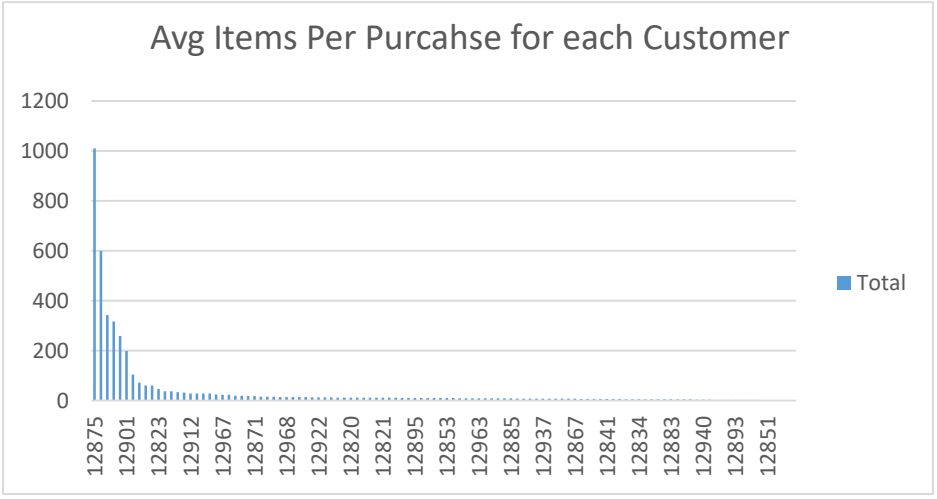
This query calculates the average order value for each month, assisting businesses in understanding customer spending patterns and tailoring marketing strategies accordingly.

1.6

```
-- Analyze customer purchasing behavior
SELECT
    Customer_ID,
    COUNT(Invoice) AS TotalPurchases,
    ROUND(AVG(Quantity)) AS AvgItemsPerPurchase
FROM
    tableRetail
GROUP BY
    Customer_ID
ORDER BY
    TotalPurchases DESC;
```

CUSTOMER_ID	TOTALPURCHASES	AVGITEMSPERPURCHASE
12748	4596	6
12921	720	13
12867	538	8
12841	420	7
12856	314	3
12839	314	12
12957	238	11
12949	215	11
12749	199	7
12955	180	20
12836	175	6
12877	154	8
12971	153	25

This query summarizes customer purchasing behavior by counting total transactions and calculating the average items per purchase. It helps identify loyal customers, understand typical purchase sizes, and informs targeted marketing strategies. The results contribute to customer engagement, inventory planning, and operational optimization for enhanced business performance.



1.7

```
-- Analyze customer transaction history
SELECT
    Customer_ID,
    COUNT(DISTINCT Invoice) AS TotalTransactions,
    MAX(TO_DATE(InvoiceDate, 'MM/DD/YYYY HH24:MI')) AS LastPurchaseDate
FROM
    tableRetail
GROUP BY
    Customer_ID
ORDER BY
    LastPurchaseDate;
```

CUSTOMER_ID	TOTALTRANSACTIONS	LASTPURCHASEDATE
12855	1	12/2/2010 9:37:00 AM
12967	2	12/16/2010 7:10:00 PM
12829	2	1/7/2011 11:13:00 AM
12872	2	1/17/2011 10:52:00 AM
12929	1	2/1/2011 12:08:00 PM
12956	1	2/6/2011 4:08:00 PM
12852	1	2/18/2011 8:47:00 AM
12945	1	2/24/2011 2:15:00 PM
12834	1	3/2/2011 9:49:00 AM
12873	1	3/2/2011 2:58:00 PM
12881	1	3/9/2011 11:44:00 AM
12845	4	3/17/2011 1:34:00 PM
12902	1	3/20/2011 12:06:00 PM

Identify customers who haven't made a purchase recently to assess churn and implement retention strategies.

1.8

```
-- Analyze customer purchase history
SELECT
    DISTINCT(Customer_ID),
    FIRST_VALUE(InvoiceDate) OVER (PARTITION BY Customer_ID ORDER BY TO_DATE(InvoiceDate,
'MM/DD/YYYY HH24:MI')) AS FirstPurchaseDate,
    FIRST_VALUE(InvoiceDate) OVER (PARTITION BY Customer_ID  ORDER BY TO_DATE(InvoiceDate,
'MM/DD/YYYY HH24:MI') DESC) AS LastPurchaseDate
FROM
    tableRetail
ORDER BY
    Customer_ID;
```

CUSTOMER_ID	FIRSTPURCHASEDATE	LASTPURCHASEDATE
12747	12/5/2010 15:38	12/7/2011 14:34
12748	12/1/2010 12:48	12/9/2011 12:20
12749	5/10/2011 15:25	12/6/2011 9:56
12820	1/17/2011 12:34	12/6/2011 15:12
12821	5/9/2011 15:51	5/9/2011 15:51
12822	9/13/2011 13:46	9/30/2011 10:04
12823	2/16/2011 12:15	9/26/2011 7:35
12824	10/11/2011 12:49	10/11/2011 12:49
12826	12/9/2010 15:21	12/7/2011 10:25
12827	10/26/2011 15:44	12/4/2011 12:17
12828	8/1/2011 16:16	12/7/2011 8:45
12829	12/14/2010 14:54	1/7/2011 11:13
12830	6/21/2011 10:53	11/2/2011 11:54

This query identifies the first and last purchase dates for each unique customer in the tableRetail dataset. The results offer insights into customer lifecycle duration, aiding in the development of targeted retention strategies, personalized marketing, and improved overall customer experience.

1.9

```sql
-- Analyze customer shopping behavior
SELECT
    Customer_ID,
    COUNT(DISTINCT StockCode) AS UniqueProductsPurchased
FROM
    tableRetail
GROUP BY
    Customer_ID
ORDER BY
    UniqueProductsPurchased DESC;
```

CUSTOMER_ID	UNIQUEPRODUCTSPURCHASED
12748	1768
12867	240
12921	233
12841	229
12957	208
12949	166
12749	160
12856	155
12836	148
12970	116
12838	115
12965	108
12933	102

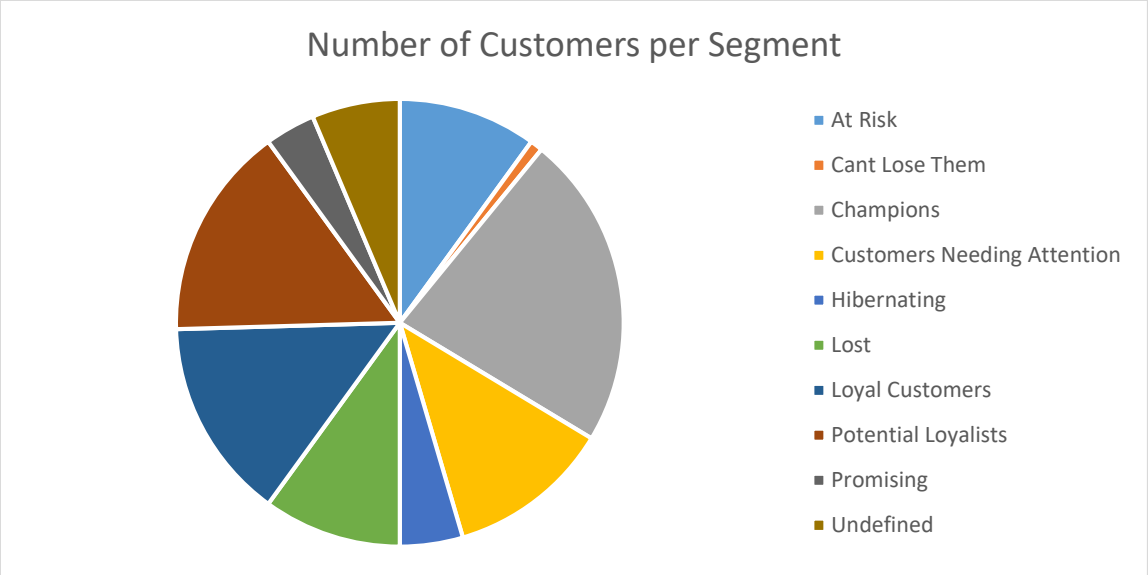
Understand the diversity of products purchased by each customer, aiding in personalized recommendations

2

```
-- Calculate Recency, Frequency, and Monetary values for each customer
WITH CUSTOMER AS (
  SELECT
    Customer_ID,
    ROUND(TO_DATE('12/9/2011 12:20:00 PM', 'MM/DD/YYYY HH:MI:SS PM') -
MAX(TO_DATE(InvoiceDate, 'MM/DD/YYYY HH24:MI'))) AS Recency,
    COUNT(DISTINCT Invoice) AS Frequency,
    SUM(Quantity * Price) AS Monetary
  FROM tableRetail
  GROUP BY Customer_ID
),

-- Calculate R_score, F_score, M_score using NTILE function
R_F_M AS (
  SELECT
    Customer_ID,
    Recency,
    Frequency,
    Monetary,
    NTILE(5) OVER (ORDER BY Recency DESC) AS R_score,
    NTILE(5) OVER (ORDER BY Frequency) AS F_score,
    NTILE(5) OVER (ORDER BY Monetary) AS M_score
  FROM CUSTOMER
),

-- Calculate FM_Score as the average of F_score and M_score
FM AS (
  SELECT
    Customer_ID,
    Recency,
    Frequency,
    Monetary,
    R_score,
    F_score,
    M_score,
    NTILE(5) OVER (ORDER BY (F_score + M_SCORE) / 2) AS FM_Score
  FROM R_F_M
)

-- Select relevant columns and define customer segments
SELECT
    Customer_ID,
    Recency,
    Frequency,
    Monetary,
    R_score,
    FM_Score,
    CASE
        WHEN (R_score = 5 AND FM_Score = 5) OR (R_score = 5 AND FM_Score = 4) OR (R_score = 4
AND FM_Score = 5) THEN 'Champions'
```

```
        WHEN (R_score = 5 AND FM_Score = 2) OR (R_score = 4 AND FM_Score = 2) OR (R_score = 3
AND FM_Score = 3) OR (R_score = 4 AND FM_Score = 3) THEN 'Potential Loyalists'
        WHEN (R_score = 5 AND FM_Score = 3) OR (R_score = 4 AND FM_Score = 4) OR (R_score = 3
AND FM_Score = 5) OR (R_score = 3 AND FM_Score = 4) THEN 'Loyal Customers'
        WHEN R_score = 5 AND FM_Score = 1 THEN 'Recent Customers'
        WHEN (R_score = 4 AND FM_Score = 1) OR (R_score = 3 AND FM_Score = 1) THEN 'Promising'
        WHEN (R_score = 3 AND FM_Score = 2) OR (R_score = 2 AND FM_Score = 3) OR (R_score = 2
AND FM_Score = 2) THEN 'Customers Needing Attention'
        WHEN (R_score = 2 AND FM_Score = 5) OR (R_score = 2 AND FM_Score = 4) OR (R_score = 1
AND FM_Score = 3) THEN 'At Risk'
        WHEN (R_score = 1 AND FM_Score = 5) OR (R_score = 1 AND FM_Score = 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 'Undefined'
    END AS Cust_Segment
FROM FM
ORDER BY CUSTOMER_ID DESC;
```

CUSTOMER_ID	RECENCY	FREQUENCY	MONETARY	R_SCORE	FM_SCORE	CUST_SEGMENT
12971	168	45	5190.74	2	5	At Risk
12970	7	4	452.24	5	3	Loyal Customers
12968	112	1	135.95	2	1	Undefined
12967	358	2	1660.9	1	3	At Risk
12966	9	1	160.18	4	1	Promising
12965	89	1	771.91	2	2	Customers Needing Attentio
12963	8	8	1856.63	5	4	Champions
12962	7	2	266.39	5	2	Potential Loyalists
12957	9	8	4017.54	4	5	Champions
12956	306	1	108.07	1	1	Lost
12955	1	11	4757.16	5	5	Champions
12953	9	1	329.85	4	2	Potential Loyalists
12952	5	4	1387.79	5	4	Champions



This SQL script analyzes retail customer data to segment customers based on Recency (R), Frequency (F), and Monetary (M) values. The resulting segments include categories like "Champions," "Loyal Customers," "At Risk," and others. These segments guide targeted marketing and customer relationship strategies, helping businesses optimize engagement and revenue.

3.1

```sql
-- Step 1: Assign a row number to each purchase for every customer, ordered by purchase date
WITH row_number AS (
  SELECT
    cust_id,
    calendar_dt,
    ROW_NUMBER() OVER (PARTITION BY cust_id ORDER BY calendar_dt) AS orderr
  FROM
    customers
),

-- Step 2: Calculate the difference between the purchase date and its row number to identify consecutive purchases (consecutive purchases will have the same group id)
consecutiveDays AS (
  SELECT
    cust_id,
    calendar_dt,
    orderr,
    calendar_dt - orderr AS group_id
  FROM
    row_number
),

-- Step 3: Group consecutive purchases by customer and group id
max_consecutive_days AS (
  SELECT
    cust_id,
    group_id,
    COUNT(*) AS consecutive_days
  FROM
    consecutiveDays
  GROUP BY
    cust_id, group_id
)

-- Final Step: Select the maximum consecutive days for each customer
SELECT
  cust_id,
  MAX(consecutive_days) AS max_consecutive_days
FROM
  max_consecutive_days
GROUP BY
  cust_id
ORDER BY
  cust_id;
```

Query Output:

CUST_ID	MAX_CONSECUTIVE_DAYS
26592	35
45234	9
54815	3
60045	15
66688	5
113502	6
145392	6
150488	9
151293	3
175749	2
196249	3
211629	5
217534	25

3.2

```
-- Calculate total amount and days to reach threshold for each customer and date
WITH cust_amt AS (
    SELECT
        cust_id,
        calendar_dt,
        SUM(Amt_LE) OVER (PARTITION BY Cust_Id ORDER BY Calendar_Dt) AS total,
        calendar_dt - FIRST_VALUE(calendar_dt) OVER (PARTITION BY cust_id ORDER BY calendar_dt) AS
days_to_reach_threshold
    FROM
        customers
),
-- Identify the minimum days to reach the threshold for each customer
threshold_days AS (
    SELECT
        cust_id,
        MIN(days_to_reach_threshold) AS days_to_reach_threshold
    FROM
        cust_amt
    WHERE
        total >= 250
    GROUP BY
        cust_id
)

-- Calculate the rounded average of days to reach the threshold across all customers
SELECT
    ROUND(AVG(days_to_reach_threshold)) AS average_days_threshold
FROM
    threshold_days;
```

SALESMONTH	TOTALSALES
11	45633.38