Analytical SQL Case Study

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.

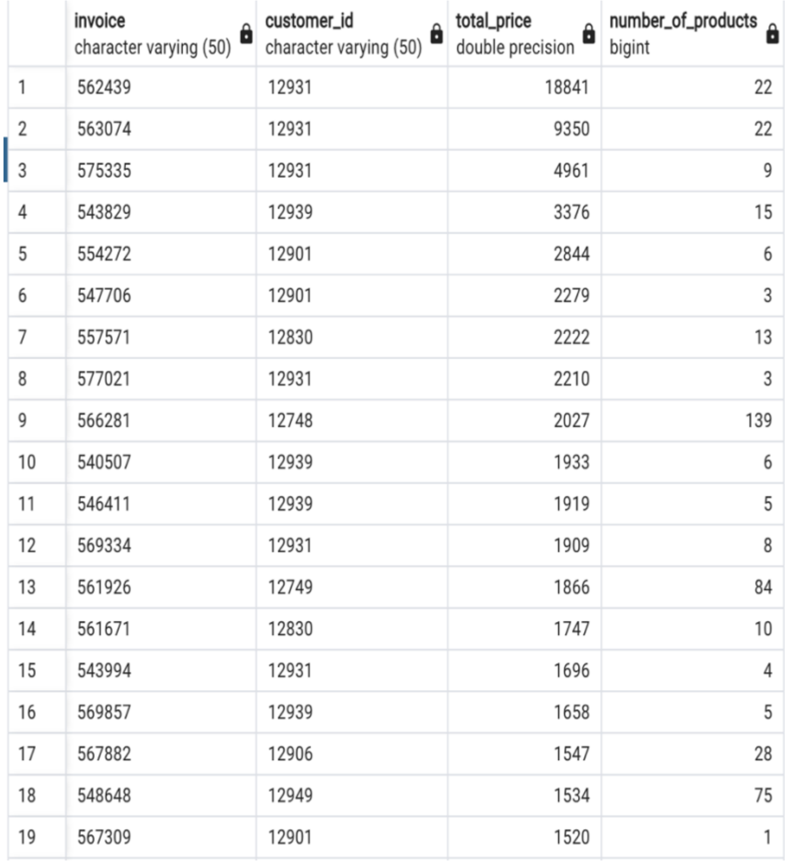
Q1- Using Online Retail dataset

1. ***Total price and quantity per each invoice***

```sql
SELECT Invoice,customer_id, round(SUM(Price * Quantity)) AS Total_Price,
        count(stockcode) as Number_of_Products
FROM tableRetail
GROUP BY Invoice,customer_id
order by Total_Price desc;
```

This query helps to calculate the total price and quantity of each invoice, which can help businesses keep track of their sales revenue and the number of products sold for each customer.

This Query result shows that



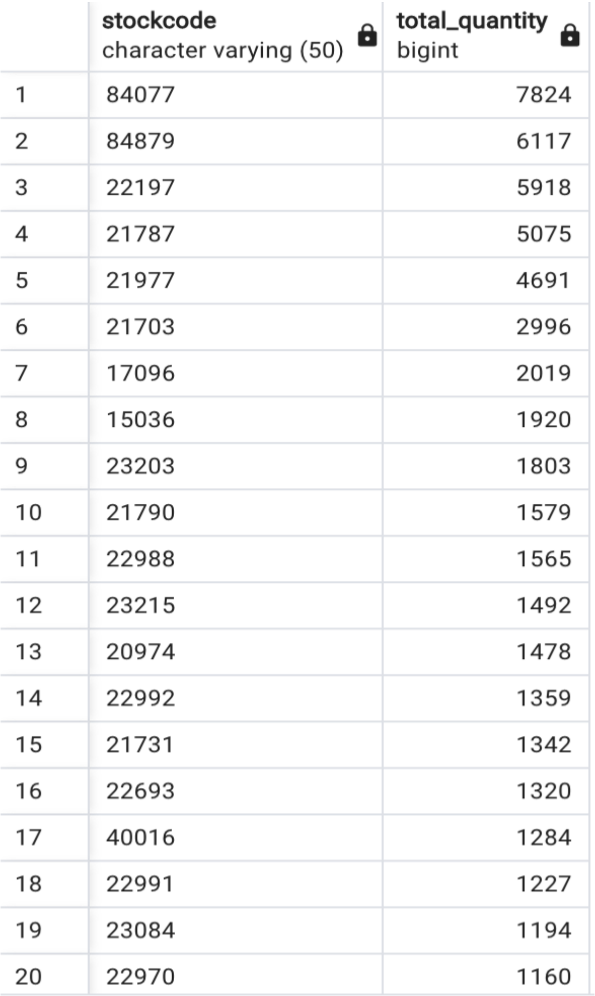
	invoice character varying (50)	customer_id character varying (50)	total_price double precision	number_of_products bigint
1	562439	12931	18841	22
2	563074	12931	9350	22
3	575335	12931	4961	9
4	543829	12939	3376	15
5	554272	12901	2844	6
6	547706	12901	2279	3
7	557571	12830	2222	13
8	577021	12931	2210	3
9	566281	12748	2027	139
10	540507	12939	1933	6
11	546411	12939	1919	5
12	569334	12931	1909	8
13	561926	12749	1866	84
14	561671	12830	1747	10
15	543994	12931	1696	4
16	569857	12939	1658	5
17	567882	12906	1547	28
18	548648	12949	1534	75
19	567309	12901	1520	1

2. *The Most selling product*

```
SELECT StockCode, Total_Quantity
FROM (SELECT StockCode,  SUM(Quantity) AS Total_Quantity,
percent_rank() OVER (ORDER BY SUM(Quantity) DESC) AS percent_rank
FROM tableRetail
GROUP BY StockCode)t
order by Total_Quantity desc;
```

This query helps to identify the products that are most popular among customers based on the total quantity sold. This information can be used to optimize inventory management and marketing strategies.

This Query result shows that



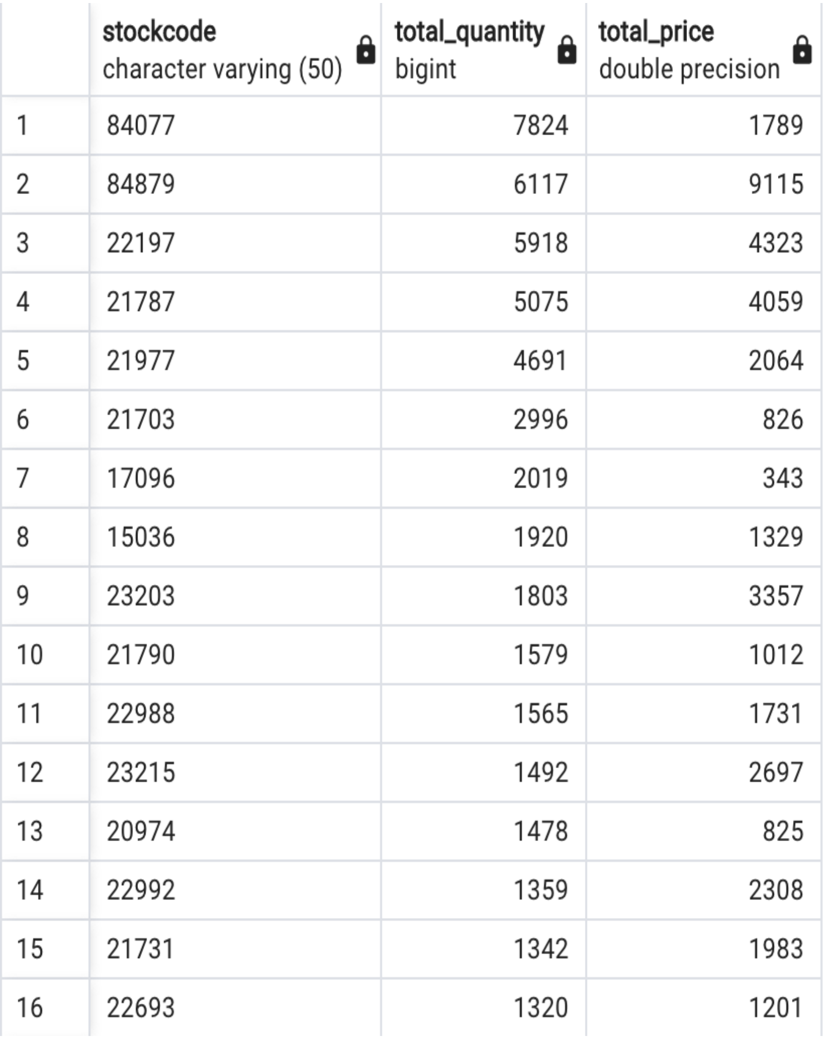
	stockcode character varying (50)	total_quantity bigint
1	84077	7824
2	84879	6117
3	22197	5918
4	21787	5075
5	21977	4691
6	21703	2996
7	17096	2019
8	15036	1920
9	23203	1803
10	21790	1579
11	22988	1565
12	23215	1492
13	20974	1478
14	22992	1359
15	21731	1342
16	22693	1320
17	40016	1284
18	22991	1227
19	23084	1194
20	22970	1160

3. ***Profit analysis by product:***

```
SELECT StockCode,Total_Quantity,Total_Price
FROM (SELECT StockCode, SUM(Quantity) AS Total_Quantity,round(SUM(Price
         * Quantity)) AS Total_Price,
          percent_rank() OVER (ORDER BY SUM(Quantity) DESC) AS percent_rank
FROM tableRetail
GROUP BY StockCode)t
order by Total_Quantity desc;
```

This query helps to analyze the profitability of each product based on its total quantity sold and the total revenue generated. This information can be used to make informed decisions about pricing and product mix.

This Query result shows that



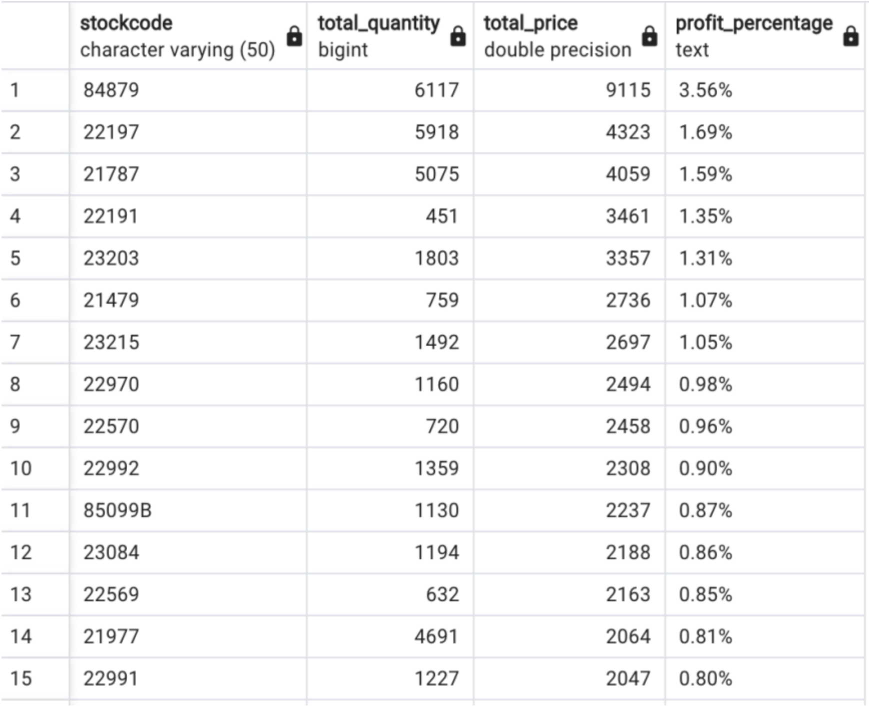
	stockcode character varying (50)	total_quantity bigint	total_price double precision
1	84077	7824	1789
2	84879	6117	9115
3	22197	5918	4323
4	21787	5075	4059
5	21977	4691	2064
6	21703	2996	826
7	17096	2019	343
8	15036	1920	1329
9	23203	1803	3357
10	21790	1579	1012
11	22988	1565	1731
12	23215	1492	2697
13	20974	1478	825
14	22992	1359	2308
15	21731	1342	1983
16	22693	1320	1201

4. *Percentage of profits by product:*

```
SELECT
            StockCode,
            Total_Quantity,
            Total_Price,
            ROUND(CAST(Total_Price / SUM(Total_Price) OVER () * 100 as
            numeric),2) || '%' AS Profit_Percentage
FROM (SELECT    StockCode,
                SUM(Quantity) AS Total_Quantity,
                ROUND(SUM(Price * Quantity)) AS Total_Price,
                percent_rank() OVER (ORDER BY SUM(Price * Quantity)
              DESC) AS Percent_Rank
            FROM
                tableRetail
            GROUP BY
                StockCode) t
ORDER BY Total_Price DESC;
```

This query calculates the percentage of profits generated by each product out of the total profits, which can help businesses identify the most profitable products and optimize their product portfolio.

This Query result shows



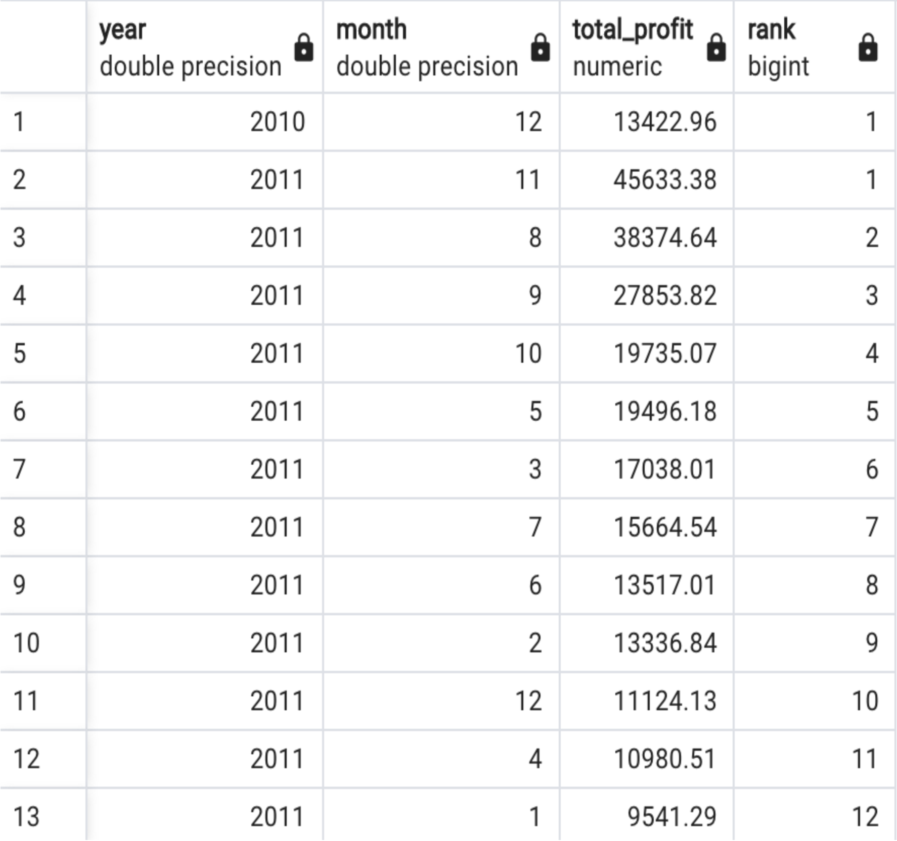
	stockcode character varying (50)	total_quantity bigint	total_price double precision	profit_percentage text
1	84879	6117	9115	3.56%
2	22197	5918	4323	1.69%
3	21787	5075	4059	1.59%
4	22191	451	3461	1.35%
5	23203	1803	3357	1.31%
6	21479	759	2736	1.07%
7	23215	1492	2697	1.05%
8	22970	1160	2494	0.98%
9	22570	720	2458	0.96%
10	22992	1359	2308	0.90%
11	85099B	1130	2237	0.87%
12	23084	1194	2188	0.86%
13	22569	632	2163	0.85%
14	21977	4691	2064	0.81%
15	22991	1227	2047	0.80%

5. ***Most selling month:***

```
SELECT Year,Month,round(Cast(Total_Profit as numeric),2) as Total_Profit,
                RANK() OVER (PARTITION BY Year ORDER BY Total_Profit DESC)
                AS Rank
FROM (SELECT
                           EXTRACT(YEAR FROM TO_DATE(InvoiceDate,
                        'MM/DD/YYYY'))  AS Year,
                           EXTRACT(MONTH FROM TO_DATE(InvoiceDate,
                        'MM/DD/YYYY')) AS Month,
                           SUM(Price * Quantity) AS Total_Profit
                from tableRetail
                group by Year,Month
                )t
ORDER BY Rank,Year, Month;
```

This query helps to identify the month with the highest sales revenue, which can help businesses plan and allocate resources accordingly.

This Query result shows



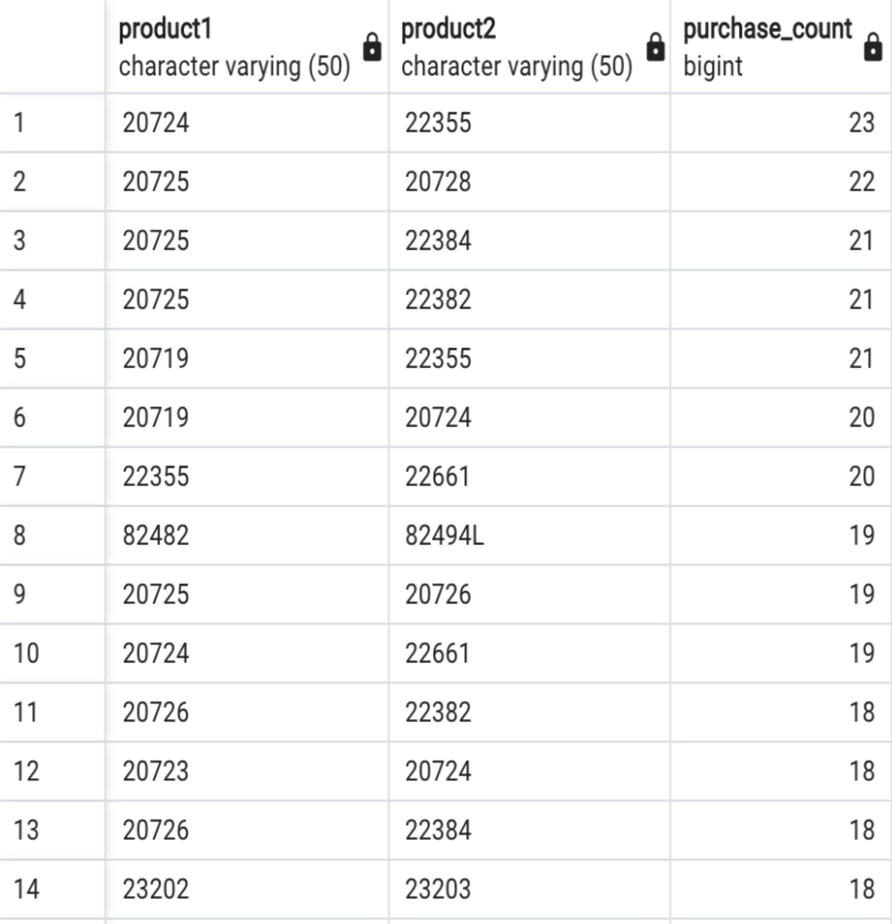
	year double precision	month double precision	total_profit numeric	rank bigint
1	2010	12	13422.96	1
2	2011	11	45633.38	1
3	2011	8	38374.64	2
4	2011	9	27853.82	3
5	2011	10	19735.07	4
6	2011	5	19496.18	5
7	2011	3	17038.01	6
8	2011	7	15664.54	7
9	2011	6	13517.01	8
10	2011	2	13336.84	9
11	2011	12	11124.13	10
12	2011	4	10980.51	11
13	2011	1	9541.29	12

6. ***The Products sold together:***

```
WITH product_pairs AS (SELECT
                              t1.StockCode AS product1,
                              t2.StockCode AS product2,
                              COUNT(DISTINCT t1.Invoice) AS purchase_count
                    FROM tableRetail t1
                              JOIN tableRetail t2 ON t1.Invoice = t2.Invoice AND
                         t1.StockCode < t2.StockCode
                    GROUP BY  t1.StockCode, t2.StockCode)
SELECT product1,product2,purchase_count
FROM product_pairs
ORDER BY purchase_count DESC;
```

This query helps to identify which products are commonly purchased together by customers. This information can be used to optimize product bundling and cross-selling strategies.

This Query result shows



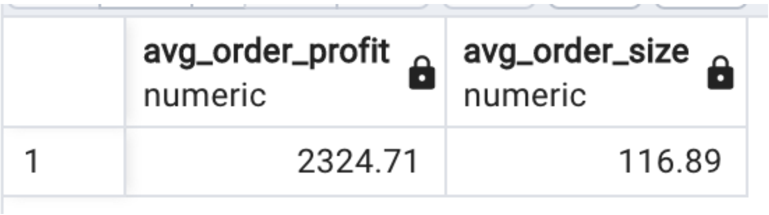
	product1 character varying (50)	product2 character varying (50)	purchase_count bigint
1	20724	22355	23
2	20725	20728	22
3	20725	22384	21
4	20725	22382	21
5	20719	22355	21
6	20719	20724	20
7	22355	22661	20
8	82482	82494L	19
9	20725	20726	19
10	20724	22661	19
11	20726	22382	18
12	20723	20724	18
13	20726	22384	18
14	23202	23203	18

7. ***Average order size by customer:***

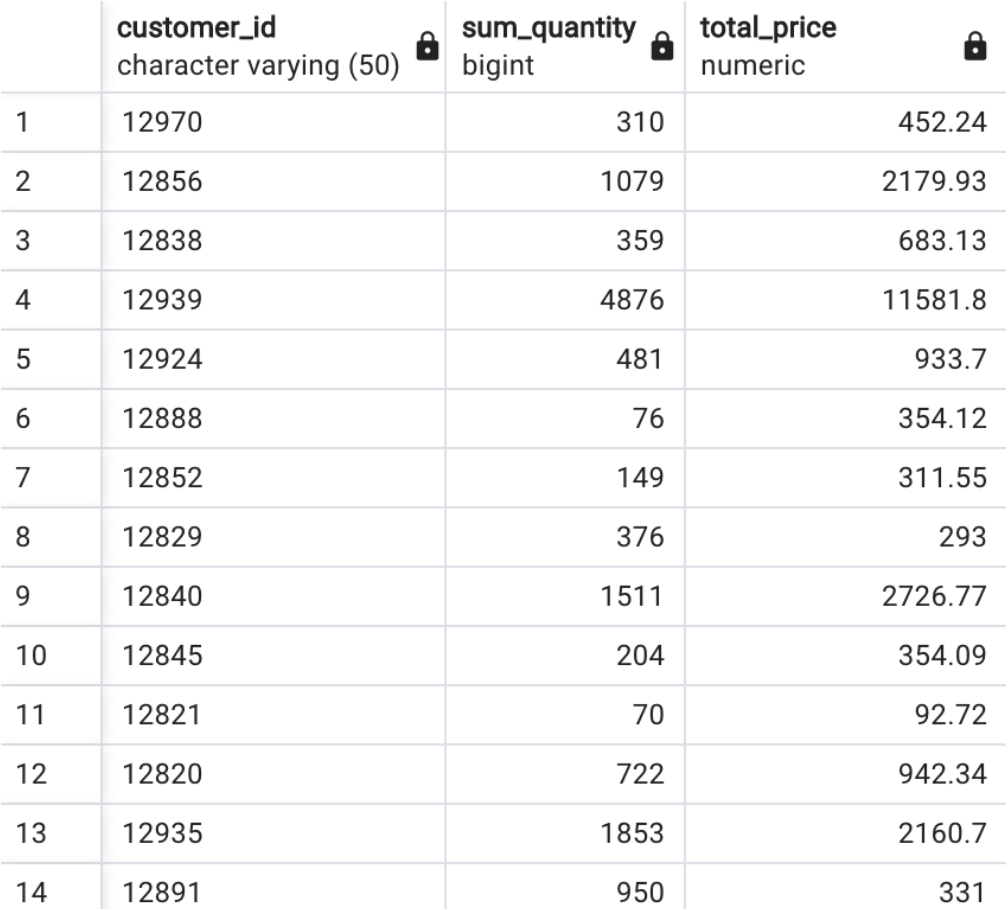
```
SELECT customer_id, Sum_Quantity,
     ROUND(Cast( Total_Price as numeric), 2) AS Total_Price
    FROM (select SUM(Price * Quantity) as Total_Price ,customer_id,
                                   sum(quantity) as Sum_Quantity
                                   from tableRetail
                                   GROUP BY customer_id )tt)
ORDER BY Avg_Order_Size DESC;

&&

SELECT
                    round(Cast(Avg(Total_Price)  as numeric),2) As Avg_Order_profit,
    round(Cast(Avg(Sum_Quantity) as numeric),2) As Avg_Order_size
FROM (SELECT customer_id, Sum_Quantity,
     ROUND(Cast( Total_Price as numeric), 2) AS Total_Price
    FROM (
                                   select
                                   SUM(Price * Quantity) as Total_Price ,
                                   customer_id,
                                   sum(quantity) as Sum_Quantity
                                   from tableRetail
                                   GROUP BY customer_id )tt)t
ORDER BY Avg_Order_Size DESC;
```

This query helps to calculate the average order size (in terms of both revenue and quantity) for each customer, which can help businesses understand customer behavior and tailor marketing and sales strategies accordingly.

This Query result shows



	avg_order_profit numeric	avg_order_size numeric
1	2324.71	116.89



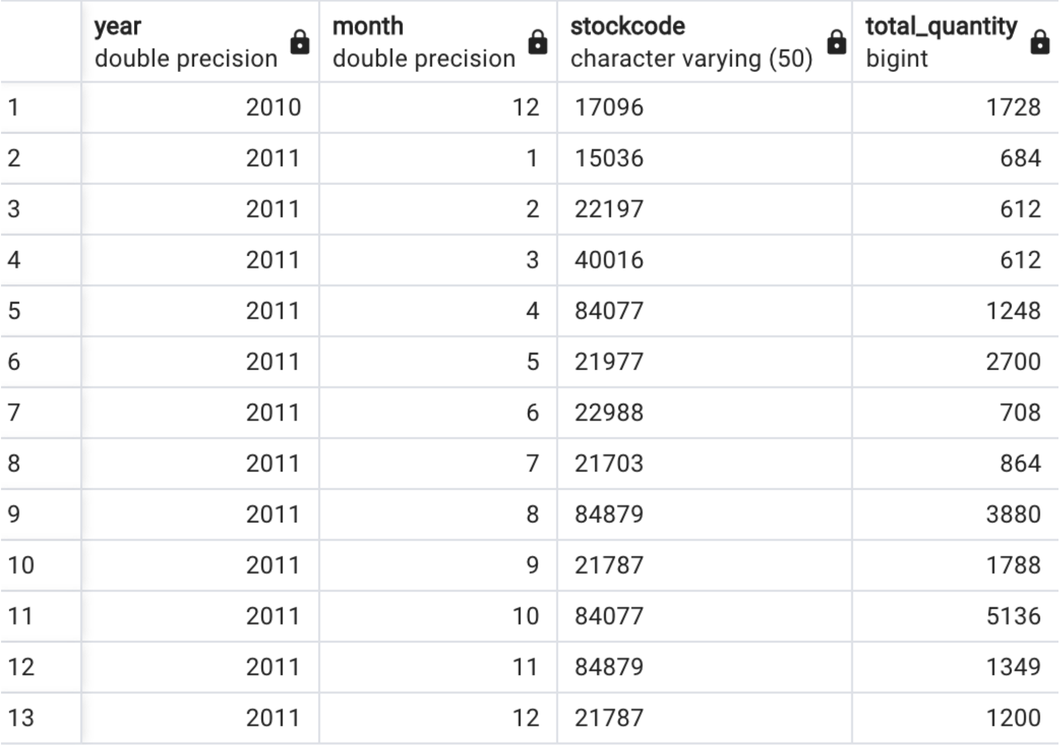
	customer_id character varying (50)	sum_quantity bigint	total_price numeric
1	12970	310	452.24
2	12856	1079	2179.93
3	12838	359	683.13
4	12939	4876	11581.8
5	12924	481	933.7
6	12888	76	354.12
7	12852	149	311.55
8	12829	376	293
9	12840	1511	2726.77
10	12845	204	354.09
11	12821	70	92.72
12	12820	722	942.34
13	12935	1853	2160.7
14	12891	950	331

8. ***Top-selling products by month:***

```
SELECT Year, Month, StockCode, Total_Quantity
FROM (SELECT
EXTRACT(YEAR FROM TO_DATE(InvoiceDate, 'MM/DD/YYYY')) AS Year,
EXTRACT(MONTH FROM TO_DATE(InvoiceDate, 'MM/DD/YYYY')) AS Month,
StockCode,
SUM(Quantity) AS Total_Quantity,
RANK() OVER (PARTITION BY EXTRACT(YEAR FROM
                        TO_DATE(InvoiceDate, 'MM/DD/YYYY')), EXTRACT(MONTH
                        FROM TO_DATE(InvoiceDate, 'MM/DD/YYYY')) ORDER BY
                        SUM(Quantity) DESC) AS rank
                FROM tableRetail
                GROUP BY Year, Month, StockCode) t
WHERE rank = 1
ORDER BY Year, Month;
```

This query helps to identify the top-selling product for each month, which can help businesses understand seasonal trends and adjust their inventory and marketing strategies accordingly.

This Query result shows



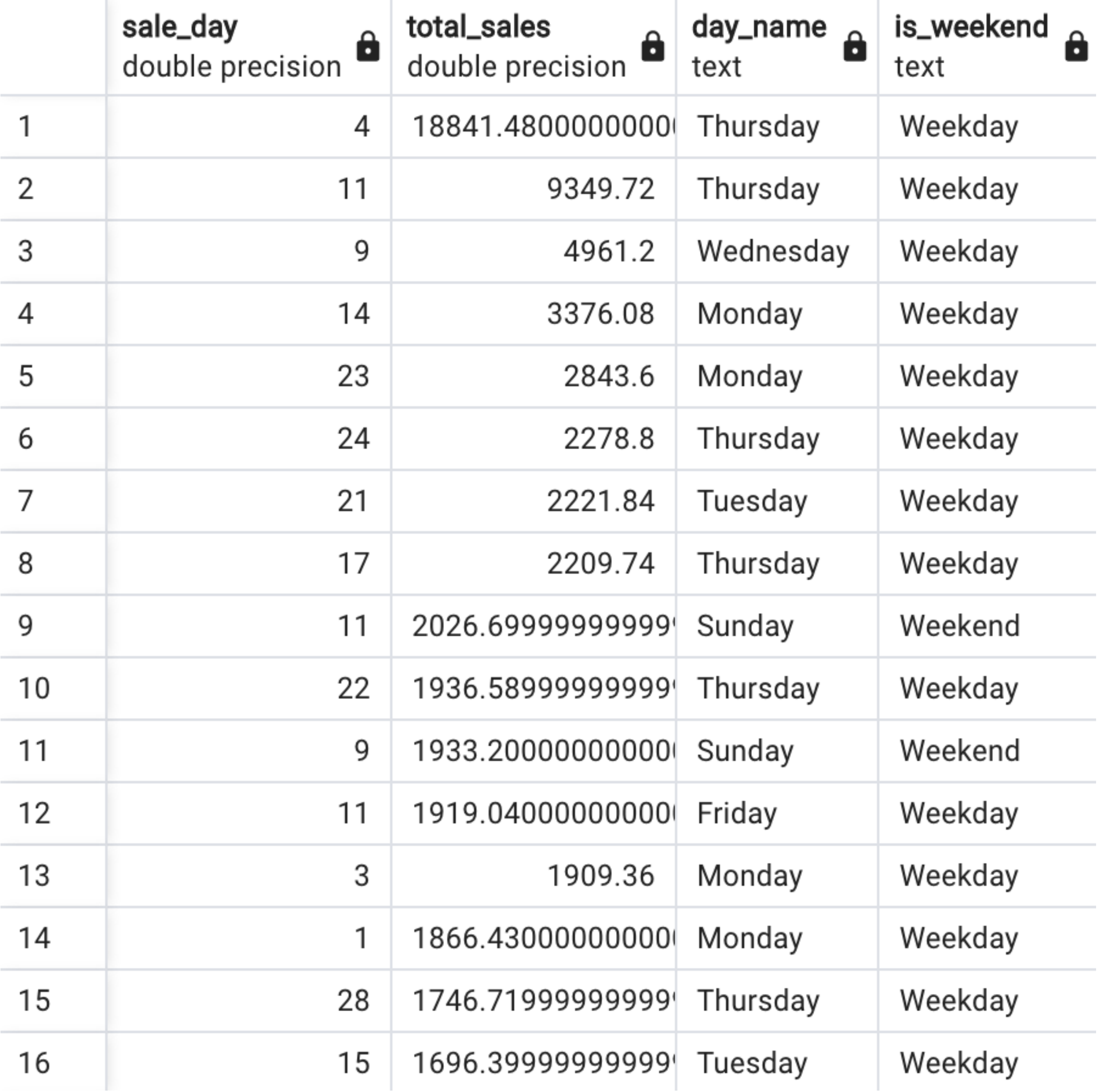
	year double precision	month double precision	stockcode character varying (50)	total_quantity bigint
1	2010	12	17096	1728
2	2011	1	15036	684
3	2011	2	22197	612
4	2011	3	40016	612
5	2011	4	84077	1248
6	2011	5	21977	2700
7	2011	6	22988	708
8	2011	7	21703	864
9	2011	8	84879	3880
10	2011	9	21787	1788
11	2011	10	84077	5136
12	2011	11	84879	1349
13	2011	12	21787	1200

9. *Top days in 'day_name' by total 'total_sales':*

```
SELECT sale_day, total_sales, day_name, is_weekend
FROM (
        SELECT EXTRACT(DAY FROM TO_DATE(InvoiceDate,
        'MM/DD/YYYY')) AS sale_day,
                SUM(quantity * price) AS total_sales,
                to_char(TO_DATE(InvoiceDate, 'MM/DD/YYYY'), 'Day') AS
        day_name,
                CASE
                        WHEN DATE_PART('dow', TO_DATE(InvoiceDate,
        'MM/DD/YYYY')) IN (0, 6) THEN 'Weekend'
                        ELSE 'Weekday'
                END AS is_weekend,
                RANK() OVER (ORDER BY SUM(quantity * price) DESC) AS
        sales_rank
        FROM tableretail
        GROUP BY EXTRACT(DAY FROM TO_DATE(InvoiceDate,
        'MM/DD/YYYY')), tableretail.InvoiceDate
) AS subquery
order by total_sales desc;
```

The query provides valuable insights to the business about the sales trends based on the day of the week. By analyzing the data, the business can identify the days with the highest sales and plan their operations accordingly. For example, they may want to allocate more staff or resources on those days, offer promotions or discounts to increase sales further, or adjust their inventory levels to ensure that they have enough stock to meet the demand. Knowing which days have the highest sales can also help the business optimize their marketing efforts by targeting their promotions to the right day or time of the week. Additionally, they can use the data to identify the days with lower sales and plan promotions or campaigns to increase sales on those days.

This Query result shows



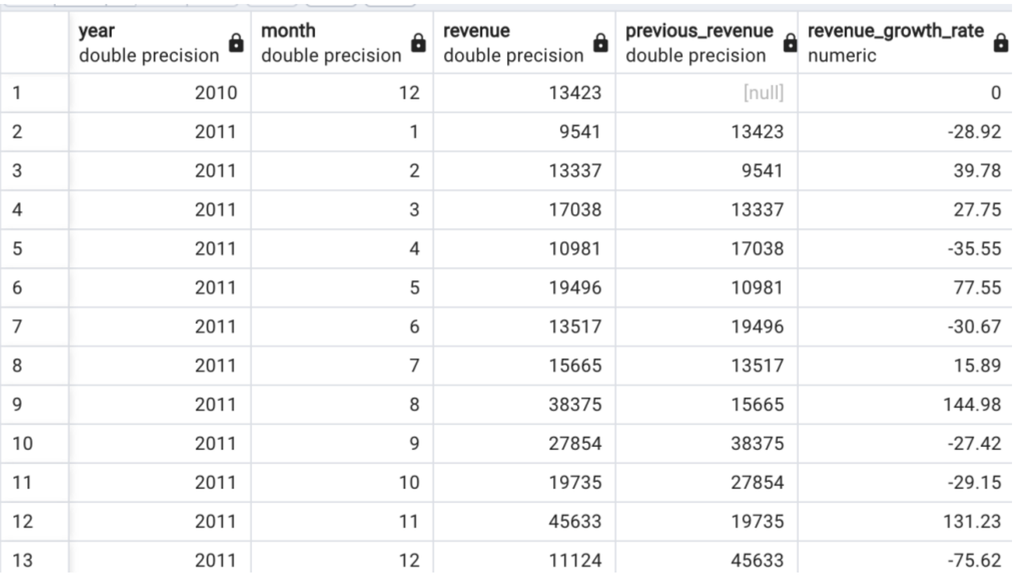
	sale_day double precision	total_sales double precision	day_name text	is_weekend text
1	4	18841.480000000	Thursday	Weekday
2	11	9349.72	Thursday	Weekday
3	9	4961.2	Wednesday	Weekday
4	14	3376.08	Monday	Weekday
5	23	2843.6	Monday	Weekday
6	24	2278.8	Thursday	Weekday
7	21	2221.84	Tuesday	Weekday
8	17	2209.74	Thursday	Weekday
9	11	2026.6999999999	Sunday	Weekend
10	22	1936.5899999999	Thursday	Weekday
11	9	1933.2000000000	Sunday	Weekend
12	11	1919.0400000000	Friday	Weekday
13	3	1909.36	Monday	Weekday
14	1	1866.4300000000	Monday	Weekday
15	28	1746.7199999999	Thursday	Weekday
16	15	1696.3999999999	Tuesday	Weekday

10. Monthly revenue growth rate:

```
WITH revenue_monthly AS (SELECT
                                        EXTRACT(YEAR FROM TO_DATE(InvoiceDate, 'MM/DD/YYYY')) AS
                                   year,
                                        EXTRACT(MONTH FROM TO_DATE(InvoiceDate, 'MM/DD/YYYY')) AS
                                   month,
                                        SUM(Price * Quantity) AS revenue
                        FROMtableRetail
                        GROUP BYyear,month
                        ORDER BYyear,month),
revenue_previous_month AS (
                        SELECT
                                        year,month,
                                        revenue,
                                        LAG(revenue) OVER (ORDER BY year,month) AS previous_revenue
                        FROM
                                        revenue_monthly
)
SELECT year,month,round(revenue) as revenue ,round(previous_revenue) as previous_revenue,
                        CASE
                                WHEN previous_revenue IS NULL THEN 0
                ELSE ROUND(Cast((revenue - previous_revenue) / previous_revenue * 100 as numeric), 2)
                        END AS revenue_growth_rate
FROM revenue_previous_month
ORDER BYyear,month;
```

This query helps to calculate the month-over-month revenue growth rate, which can help businesses track their financial performance and identify areas for improvement.

This Query result shows



	year double precision	month double precision	revenue double precision	previous_revenue double precision	revenue_growth_rate numeric
1	2010	12	13423	[null]	0
2	2011	1	9541	13423	-28.92
3	2011	2	13337	9541	39.78
4	2011	3	17038	13337	27.75
5	2011	4	10981	17038	-35.55
6	2011	5	19496	10981	77.55
7	2011	6	13517	19496	-30.67
8	2011	7	15665	13517	15.89
9	2011	8	38375	15665	144.98
10	2011	9	27854	38375	-27.42
11	2011	10	19735	27854	-29.15
12	2011	11	45633	19735	131.23
13	2011	12	11124	45633	-75.62

Q2- After exploring the data now we will now implement a Monetary model for customers behavior for product purchasing and segment each customer based on the below groups:

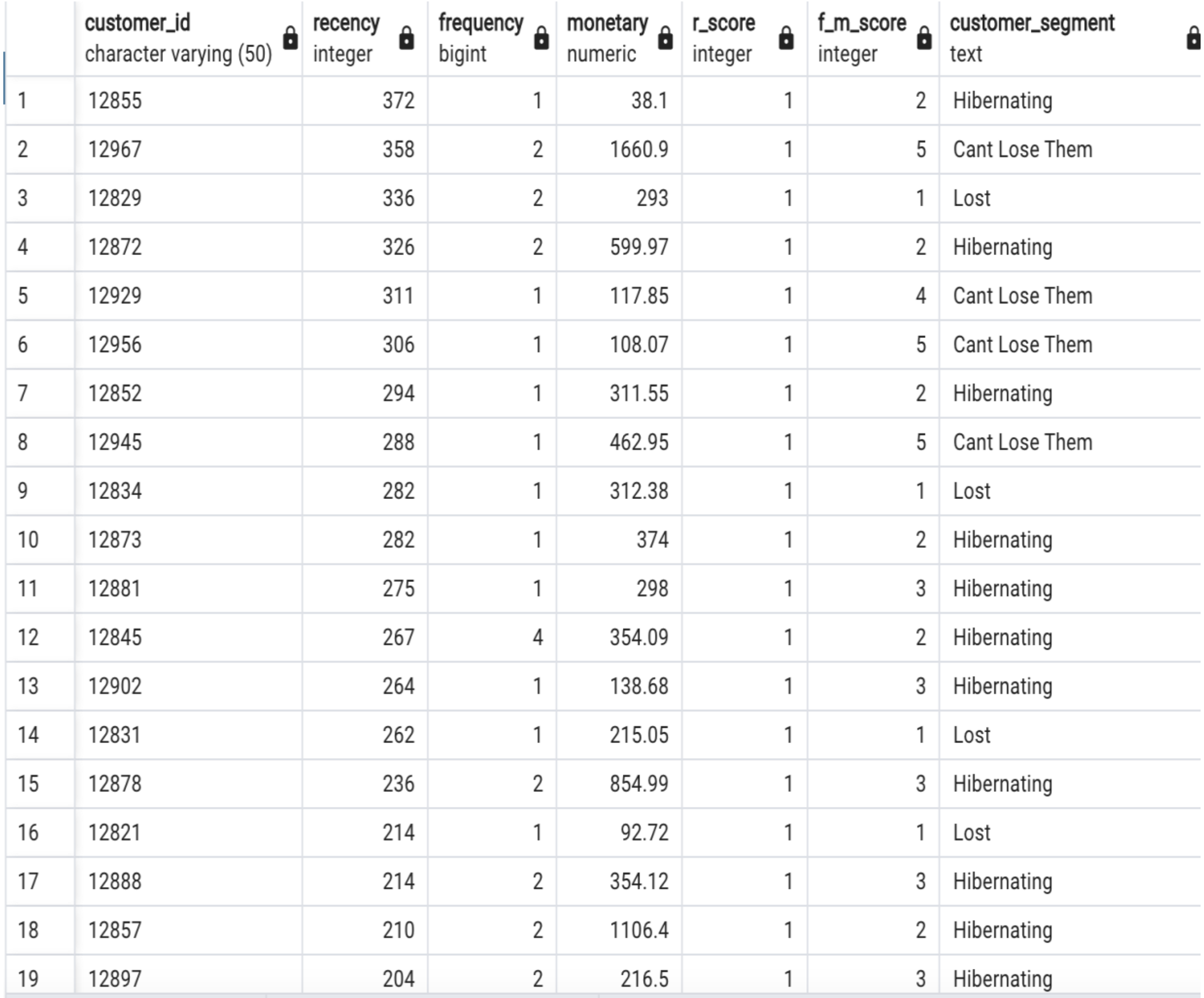
Champions	**Loyal Customers**
Potential Loyalists	**Recent Customers**
Customers Needing Attention	**Promising**
At Risk	**Cant Lose Them**
Hibernating	**Lost**

The customers will be grouped based on 3 main values

- **Recency** => how recent the last transaction is.
- **Frequency** => how many times the customer has bought from our store
- **Monetary** => how much each customer has paid for our products

```
WITH customer_data AS (SELECT
Customer_ID,
(SELECT MAX(TO_DATE(InvoiceDate, 'MM/DD/YYYY HH24:MI')) FROM
                    tableRetail)
MAX(TO_DATE(InvoiceDate, 'MM/DD/YYYY HH24:MI')) AS Recency,
COUNT(Distinct Invoice) AS Frequency,
ROUND(CAST(SUM(Price * Quantity) AS NUMERIC), 2) AS Monetary,
NTILE(5) OVER (ORDER BY (SELECT MAX(TO_DATE(InvoiceDate,
                    'MM/DD/YYYY HH24:MI'))
FROM tableRetail) - MAX(TO_DATE(InvoiceDate, 'MM/DD/YYYY HH24:MI'))
                    DESC) AS R_Score,
NTILE(5) OVER (ORDER BY (SELECT NTILE(5) OVER (ORDER BY
COUNT(Distinct Invoice) DESC)FROM tableRetail
WHERE Customer_ID = t.Customer_ID) + (SELECT NTILE(5) OVER (ORDER
                    BY SUM(Price * Quantity) DESC) FROM tableRetail WHERE
Customer_ID = t.Customer_ID) - 1 DESC) AS F_M_Score
FROM tableRetail t
GROUP BY Customer_ID)
SELECT Customer_ID, Recency, Frequency, Monetary,R_Score,F_M_Score,
CASE
WHEN R_Score = 5 AND F_M_Score IN (5, 4) THEN 'Champions'
WHEN R_Score = 5 AND F_M_Score = 2 THEN 'Potential Loyalists'
WHEN R_Score = 4 AND F_M_Score IN (5, 3) THEN 'Loyal Customers'
WHEN R_Score = 4 AND F_M_Score = 4 THEN 'Loyal Customers'
WHEN R_Score = 3 AND F_M_Score IN (3, 4) THEN 'Potential Loyalists'
WHEN R_Score = 5 AND F_M_Score = 3 THEN 'Loyal Customers'
WHEN R_Score = 4 AND F_M_Score = 3 THEN 'Potential Loyalists'
WHEN R_Score = 4 AND F_M_Score = 2 THEN 'Potential Loyalists'
WHEN R_Score = 3 AND F_M_Score = 5 THEN 'Loyal Customers'
WHEN R_Score = 5 AND F_M_Score = 1 THEN 'Recent Customers'
WHEN R_Score = 4 AND F_M_Score = 1 THEN 'Promising'
WHEN R_Score = 3 AND F_M_Score = 1 THEN 'Promising'
WHEN R_Score = 2 AND F_M_Score IN (2, 3) THEN 'Customers Needing Attention'
WHEN R_Score = 2 AND F_M_Score IN (4, 5) THEN 'At Risk'
WHEN R_Score = 1 AND F_M_Score IN (2, 3) THEN 'Hibernating'
WHEN R_Score = 1 AND F_M_Score IN (4, 5) THEN 'Cant Lose Them'
WHEN R_Score = 1 AND F_M_Score = 1 THEN 'Lost'ELSE 'Undefined'
END AS Customer_Segment
FROM customer_data
ORDER BY Recency DESC;
```

And the Query shows:

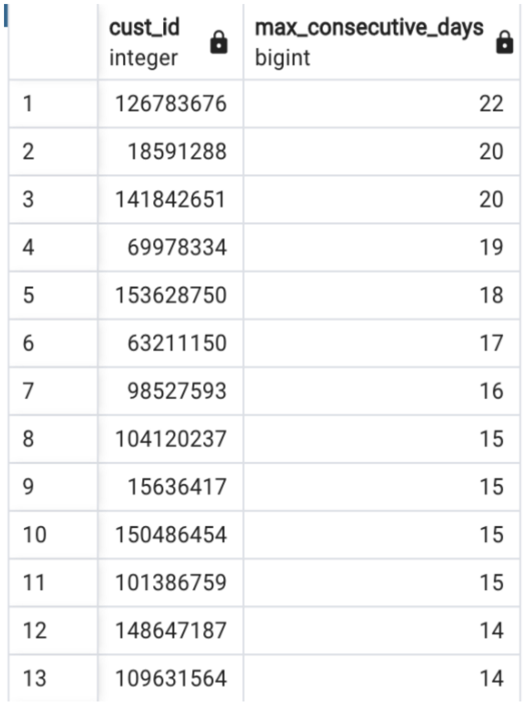


	customer_id	recency	frequency	monetary	r_score	f_m_score	customer_segment
	character varying (50)	integer	bigint	numeric	integer	integer	text
1	12855	372	1	38.1	1	2	Hibernating
2	12967	358	2	1660.9	1	5	Cant Lose Them
3	12829	336	2	293	1	1	Lost
4	12872	326	2	599.97	1	2	Hibernating
5	12929	311	1	117.85	1	4	Cant Lose Them
6	12956	306	1	108.07	1	5	Cant Lose Them
7	12852	294	1	311.55	1	2	Hibernating
8	12945	288	1	462.95	1	5	Cant Lose Them
9	12834	282	1	312.38	1	1	Lost
10	12873	282	1	374	1	2	Hibernating
11	12881	275	1	298	1	3	Hibernating
12	12845	267	4	354.09	1	2	Hibernating
13	12902	264	1	138.68	1	3	Hibernating
14	12831	262	1	215.05	1	1	Lost
15	12878	236	2	854.99	1	3	Hibernating
16	12821	214	1	92.72	1	1	Lost
17	12888	214	2	354.12	1	3	Hibernating
18	12857	210	2	1106.4	1	2	Hibernating
19	12897	204	2	216.5	1	3	Hibernating

Q3- Using new dataset we will answer some Questions:

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

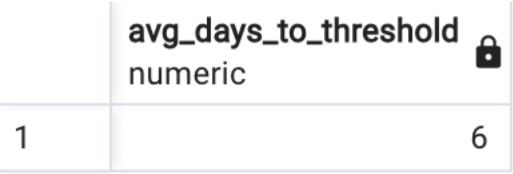
```
WITH customer_purchases AS (SELECT
                                        Cust_Id,
                                        Calendar_Dt,
                                        Amt_LE,
                                        LAG(Calendar_Dt) OVER (PARTITION
                            BY Cust_Id ORDER BY Calendar_Dt) AS
                            prev_date
                        FROM
                                        transactions
              ),
              consecutive_days AS (
                        SELECT
                                        Cust_Id,
                                        Calendar_Dt,
                                        Amt_LE,
                                        prev_date,
                                        COALESCE((Calendar_Dt -
                            prev_date)::integer - 1, 0) AS day_diff,
                                        SUM(CASE WHEN
                            COALESCE((Calendar_Dt - prev_date)::integer - 1,
                            0) = 1 THEN 1 ELSE 0 END)
                                                        OVER (PARTITION BY
                                   Cust_Id ORDER BY Calendar_Dt) AS
                                   consecutive_days
                        FROM
                                        customer_purchases
              )
              SELECT
                        Cust_Id,
                        MAX(consecutive_days) AS max_consecutive_days
              FROM
                        consecutive_days
              GROUP BY
                        Cust_Id
                          order by max_consecutive_days desc ;
```



	cust_id integer	max_consecutive_days bigint
1	126783676	22
2	18591288	20
3	141842651	20
4	69978334	19
5	153628750	18
6	63211150	17
7	98527593	16
8	104120237	15
9	15636417	15
10	150486454	15
11	101386759	15
12	148647187	14
13	109631564	14

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

```
WITH customer_totals AS (
            SELECT
                    Cust_Id,
                    Calendar_Dt,
                    Amt_LE,
                    SUM(Amt_LE) OVER (PARTITION BY Cust_Id
                ORDER BY Calendar_Dt) AS running_total_spent
            FROM
                    transactions
),
customer_thresholds AS (
            SELECT
                    Cust_Id,
                    Calendar_Dt,
                    running_total_spent
            FROM
                    customer_totals
            WHERE
                    running_total_spent < 250
),
customer_thresholds_reached AS (
        SELECT
                Cust_Id,
                Calendar_Dt,
                running_total_spent
        FROM
                customer_totals
        WHERE
                running_total_spent >= 250
),
customer_threshold_counts AS (
        SELECT
                Cust_Id,
                COUNT(Calendar_Dt) AS days_to_threshold
        FROM
                customer_thresholds
        WHERE
                Cust_Id IN (SELECT Cust_Id FROM
              customer_thresholds_reached)
        GROUP BY
                Cust_Id
        ORDER BY
                Cust_Id
)
SELECT ROUND(AVG(days_to_threshold)) AS avg_days_to_threshold
FROM customer_threshold_counts;
```



	avg_days_to_threshold numeric
1	6