Step 1: Exploring Dataset:

1-total items and amount for each invoice belong to customer:

Description:

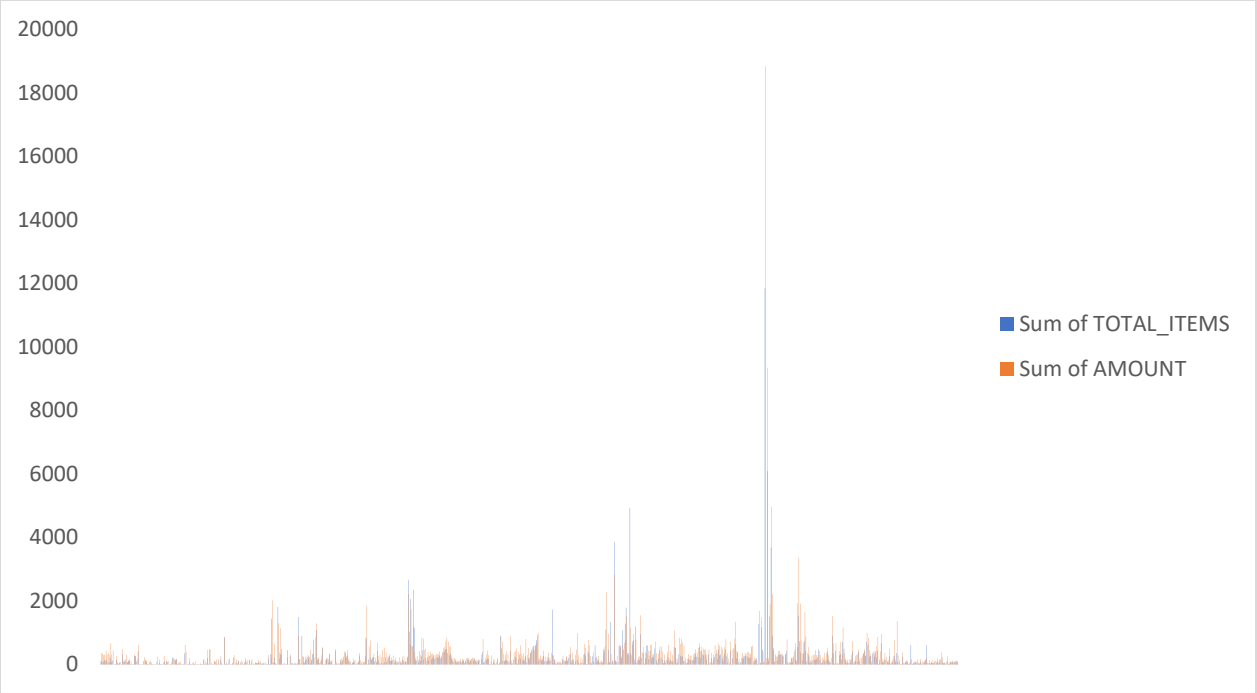
This query helps stores understand what each customer buys. It counts how many items were bought in each purchase (invoice) and calculates how much money was spent. By organizing this data by customer, businesses can see who buys what and how much they spend, which helps them plan their sales and keep customers happy.

Query:

```
/*
this query selects customerid and invoice from tablereatil table
then it calculates total items the customer purchased and the amount for that invoice
*/

select customer_id , invoice , sum(quantity) as total_items , sum(price*quantity) as amount
from tableretail
group by customer_id , invoice
order by customer_id;
```

Output:

CUSTOMER_ID	INVOICE	TOTAL_ITEMS	AMOUNT
12747	537215	108	358.56
12747	538537	105	347.71
12747	541677	88	303.04
12747	545321	146	310.78
12747	551992	116	442.96
12747	554549	145	328.35
12747	558265	92	376.3
12747	563949	62	301.7
12747	569397	208	675.38



Conclusion:

This chart displays total items and amount of each invoice, this query helps businesses analyze customer purchase behavior by providing insights into the total number of items bought and the total amount spent per invoice, aiding in customer relationship management and financial analysis.

2- Top 10 customers by total amount:

Description:

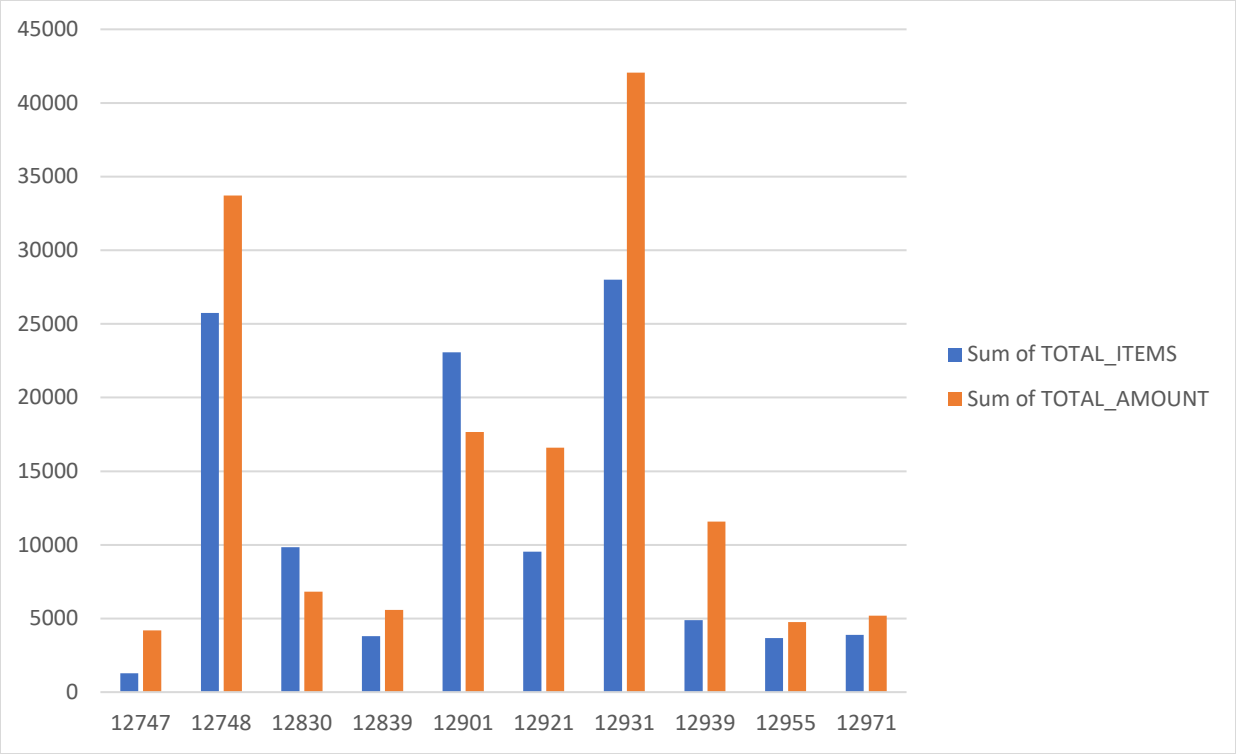
This query aims to rank customers based on the total amount they have spent on purchases; it sums amount spent and total items purchased then it count total invoices and rank based on sum amount for each customer then it gets the top 10 customers only;

Query:

```
/*
 the query calculates the total items purchased and the total amount spent for each invoice by
each customer, organizing this data in a common table expression (CTE) named 'cte'.
Then, it selects customer IDs, counts the number of invoices for each customer,
sums up their total items and total amount spent, and assigns a ranking to each customer based
on their total spending.
The 'dense_rank()' function is used to assign rankings without gaps, ordered by the total
amount spent in descending order and all are in cte named 'ranking'
then it gets the top 10 only
*/
with cte as (
    select customer_id , invoice , sum(quantity) as total_items , sum(price*quantity) as amount
    from tableretail
    group by customer_id , invoice
    order by customer_id
),

ranking as (
select customer_id , count(invoice) as total_invoices , sum(total_items) total_items , sum
(amount) total_amount , dense_rank() over(order by  sum (amount) desc) rank_by_amount
from cte
group by customer_id
)

select customer_id , total_invoices,total_items , total_amount
from ranking
where rank_by_amount <=10;
```

Output:

CUSTOMER_ID	TOTAL_INVOICES	TOTAL_ITEMS	TOTAL_AMOUNT	RANK_BY_AMOUNT
12931	15	28004	42055.96	1
12748	210	25748	33719.73	2
12901	28	23075	17654.54	3
12921	37	9526	16587.09	4
12939	8	4876	11581.8	5
12830	6	9848	6814.64	6
12839	14	3807	5591.42	7
12971	45	3892	5190.74	8
12955	11	3664	4757.16	9



Conclusion:

This chart displays top 10 customers based on total amount it displays total items purchased and total amount for each customer, this query helps businesses figure out which customers spend the most money. By knowing this, they can focus on giving these customers extra attention and perks to keep them happy. This makes sure that the business grows steadily and makes more money over time.

3- Top 10 customers by total invoices:

Description:

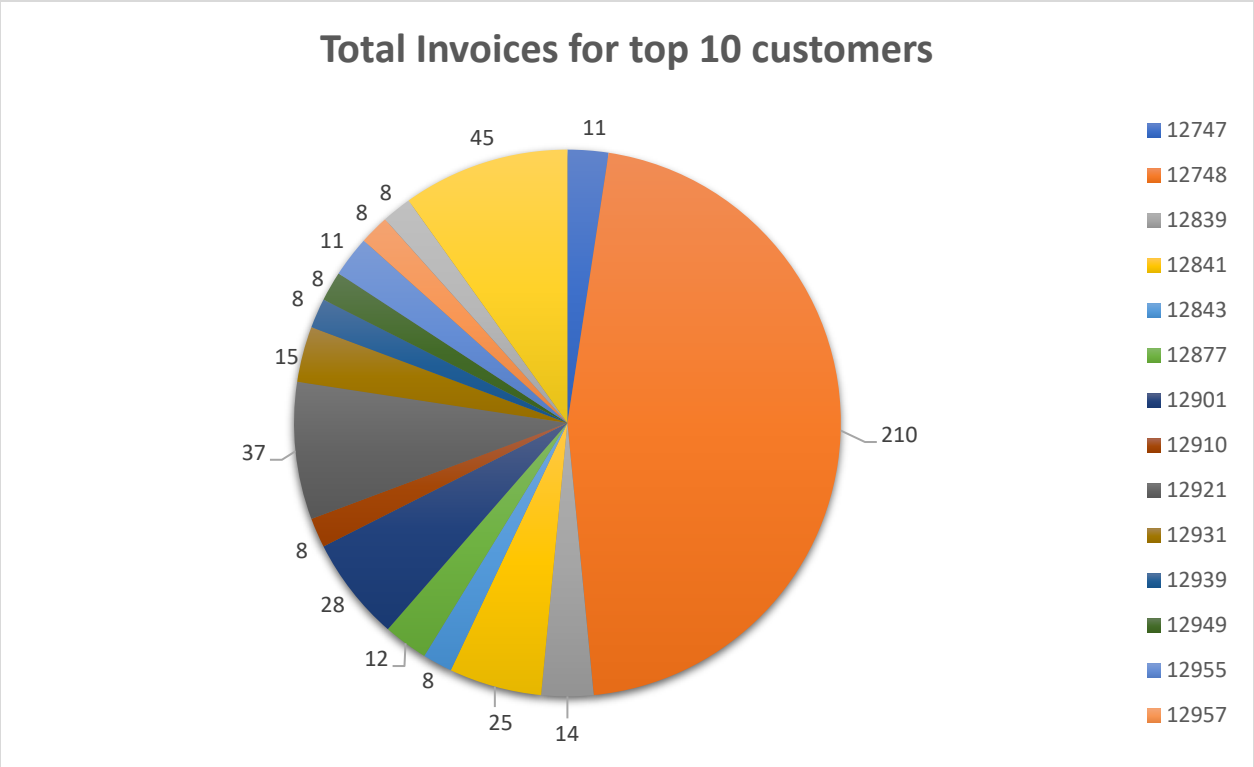
This query aims to rank customers based on the total invoices they purchased; it sums amount spent and total items purchased then it count total invoices and rank based on sum invoices for each customer then it gets the top 10 customers only;

Query:

```
/*
It first organizes the data by grouping transactions according to customer ID and invoice.
Then, it calculates the total number of items purchased and the corresponding total amount
spent for each invoice.
Next, it counts the number of invoices associated with each customer and ranks them
accordingly, using the 'dense_rank()' function.
Finally, it selects the top 10 customers with the highest number of invoices
*/
with cte as (
    select customer_id , invoice , sum(quantity) as total_items , sum(price*quantity) as amount
    from tableretail
    group by customer_id , invoice
    order by customer_id
),

ranking as(
    select customer_id , count(invoice) as total_invoices , sum(total_items) total_items , sum
(amount) total_amount , dense_rank() over(order by count(invoice) desc) rank_by_invoices
    from cte
    group by customer_id
)

select customer_id , total_invoices,total_items , total_amount
from ranking
where rank_by_invoices <=10;
```

Output:

CUSTOMER_ID	TOTAL_INVOICES	TOTAL_ITEMS	TOTAL_AMOUNT
12748	210	25748	33719.73
12971	45	3892	5190.74
12921	37	9526	16587.09
12901	28	23075	17654.54
12841	25	2757	4022.35
12931	15	28004	42055.96
12839	14	3807	5591.42
12877	12	1178	1535.77
12747	11	1275	4196.01



Conclusion:

This chart displays the top 10 customers based on total invoices. It displays total invoices for each customer, this query helps businesses prioritize their efforts towards their most valuable customers, leading to increased customer satisfaction, loyalty, and ultimately, sustainable growth in revenue and profitability.

4- Top selling product per quantity:

Description:

This query selects unique stock codes and calculates the total quantity sold for each code. It orders the results by the total quantity sold in descending order, revealing the top-selling items.

Query:

```
/*

This query selects all  stock codes and the total quantity sold for each stock code.

The SUM() function calculates the total quantity sold for each stock code,
and OVER() function partitions the data based on stock code.

Use DISTINCT to avoid redundancy, and ORDER BY total_quantity DESC to get the top sells

*/
select distinct stockcode , sum(quantity) over(partition by stockcode) as total_quantity
from tableretail
order by total_quantity desc;
```

Output:

STOCKCODE	TOTAL_QUANTITY
84077	7824
84879	6117
22197	5918
21787	5075
21977	4691
21703	2996
17096	2019
15036	1920
23203	1803

5- Top 10 products sold for each month:

Description:

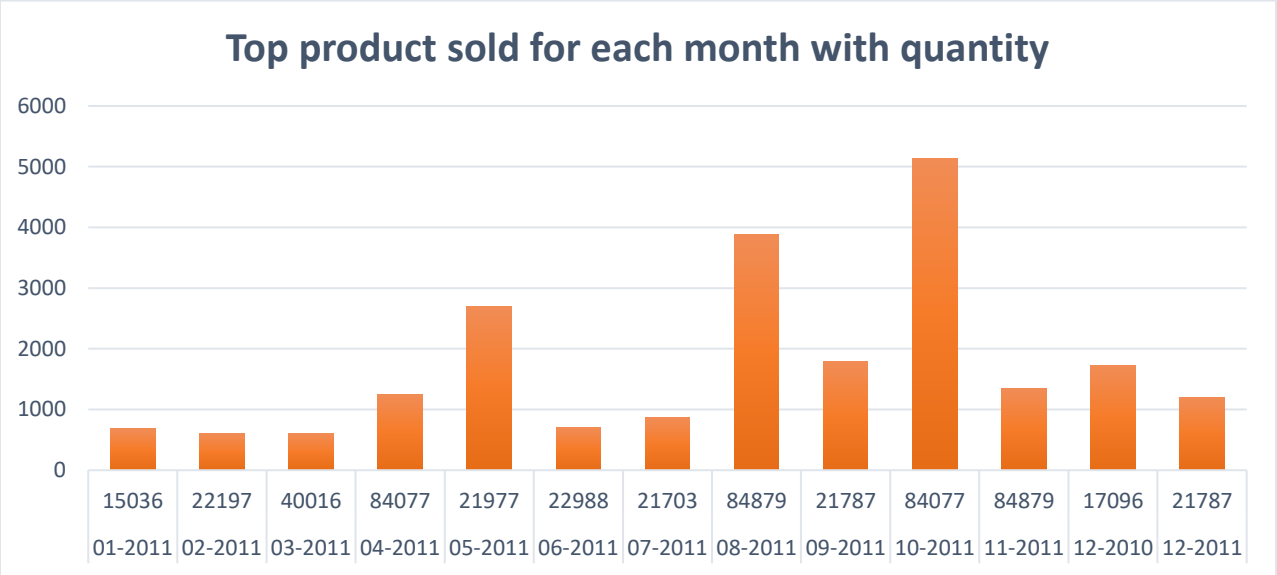
This query finds the most popular product for each month. It looks at how many of each product were sold every month and then picks the one that sold the most.

Query:

```
/*
for CTE 'cte':
-Calculates the total quantity sold for each stock code, partitioned by month.
-Formats the invoice date to extract the month and year.
-Removes duplicate stock codes to ensure each product is counted only once.

Ranking (CTE 'ranking'):
-Assigns a rank to each product based on the total quantity sold within each month, using the
'dense_rank()' function.
-This step ensures that products are ranked within each month separately.

Main Query:
-Selects the month, top-selling stock code, and total quantity sold for each month from the
'ranking' CTE.
-Filters the results to include only the top-ranked product (rank = 1) for each month.
*/
with cte as(
    select distinct stockcode , to_char(to_date(invoicedate,'mm/dd/yyyy hh24:mi'),'mm-yyyy') as
"month", sum(quantity) over(partition by stockcode , to_char(to_date(invoicedate,'mm/dd/yyyy
hh24:mi'),'mm-yyyy') ) total_quantity
    from tableretail
    order by stockcode
),

ranking as(
    select stockcode , "month" , total_quantity , dense_rank() over(partition by "month" order by
total_quantity desc) "rank"
    from cte
)

select  "month" , stockcode , total_quantity
from ranking
where "rank" =1;
```

Output:

month	STOCKCODE	TOTAL_QUANTITY
01-2011	15036	684
02-2011	22197	612
03-2011	40016	612
04-2011	84077	1248
05-2011	21977	2700
06-2011	22988	708
07-2011	21703	864
08-2011	84879	3880
09-2011	21787	1788



Conclusion:

This chart displays each month the most product sold along with the quantity sold. This query finds the most popular product for each month. This helps businesses see which products are in demand each month, so they can focus on selling those and make more profit.

6- Monthly amount for top 10 customers:

Description:

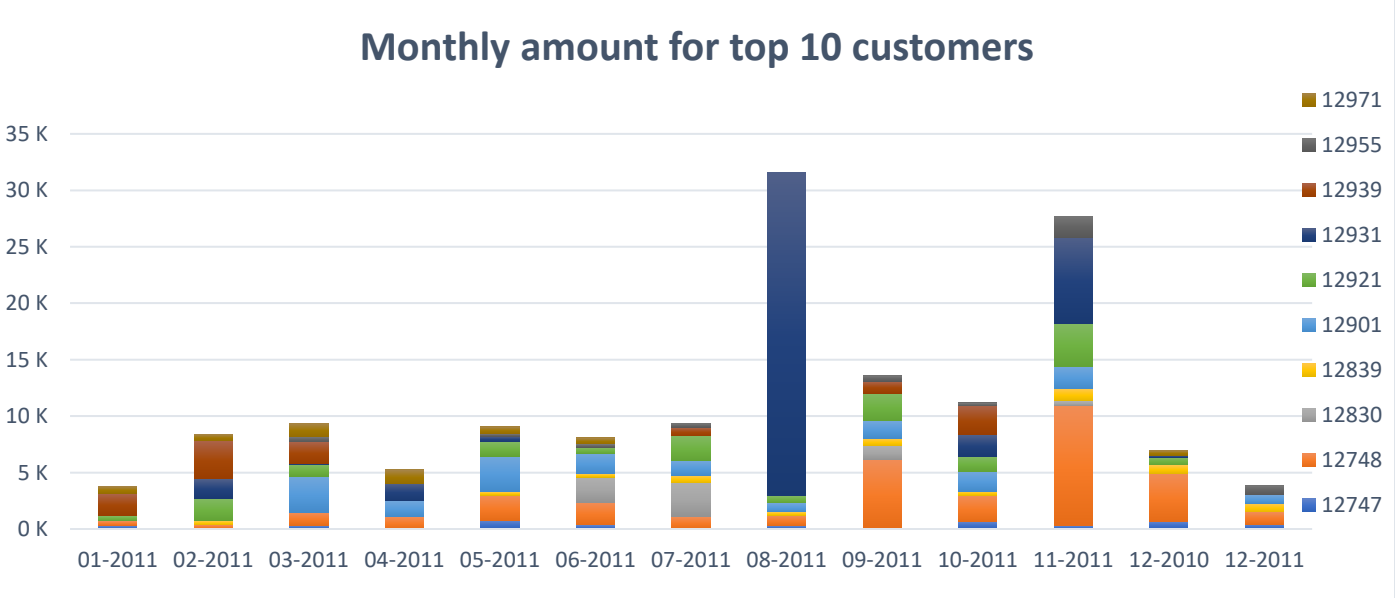
This query figures out how much the top 10 customers spend each month. It groups data by customer and month, calculating how much each customer spends monthly and overall. Then, it ranks customers by their total spending and selects the top 10. This helps businesses see how much their best customers spend each month, which can guide marketing and customer service efforts.

Query:

```
/*
Data Preparation (CTE 'cte'):
-Groups data by customer ID and month.
-Calculates the total amount spent by each customer for each month.
-Also calculates the total amount spent by each customer overall.

Identifying Top Customers (CTE 'top10'):
-Ranks customers based on their total amount spent overall.
-Assigns a rank to each customer using the 'dense_rank()' function.

Main Query:
-Selects customer ID, month, monthly spending (amount_per_month), and total spending
overall (total_amount) from the 'top10' CTE.
-Filters the results to include only the top 10 customers based on their assigned rank.
*/
with cte as (
    select distinct customer_id ,  to_char(to_date(invoicedate,'mm/dd/yyyy hh24:mi'),'mm-yyyy')
as "month" , sum(price*quantity) over(partition by customer_id ,
to_char(to_date(invoicedate,'mm/dd/yyyy hh24:mi'),'mm-yyyy')) Amount_per_month ,
    sum(price*quantity) over(partition by customer_id) total_amount
    from tableretail
),

top10 as (
    select cte.* , dense_rank () over(order by total_amount desc) "rank"
    from cte
)

select customer_id , "month" , amount_per_month , total_amount
from top10
where "rank" between 1 and 10;
```

Output:

CUSTOMER_ID	month	AMOUNT_PER_MONTH	TOTAL_AMOUNT
12931	12-2010	177	42055.96
12931	11-2011	7615.9	42055.96
12931	04-2011	1488	42055.96
12931	08-2011	28610	42055.96
12748	02-2011	389.64	33719.73
12748	09-2011	6148.84	33719.73
12748	08-2011	898.24	33719.73
12748	11-2011	10639.23	33719.73
12748	01-2011	418.77	33719.73



Conclusion:

This chart displays each month the amount purchased for the top 10 customers. Understanding their spending habits helps us focus on keeping them happy and coming back. We can use this information to offer them special deals and personalized service, making them feel valued and increasing our sales.

7- Quantity sold and profit for each month:

Description:

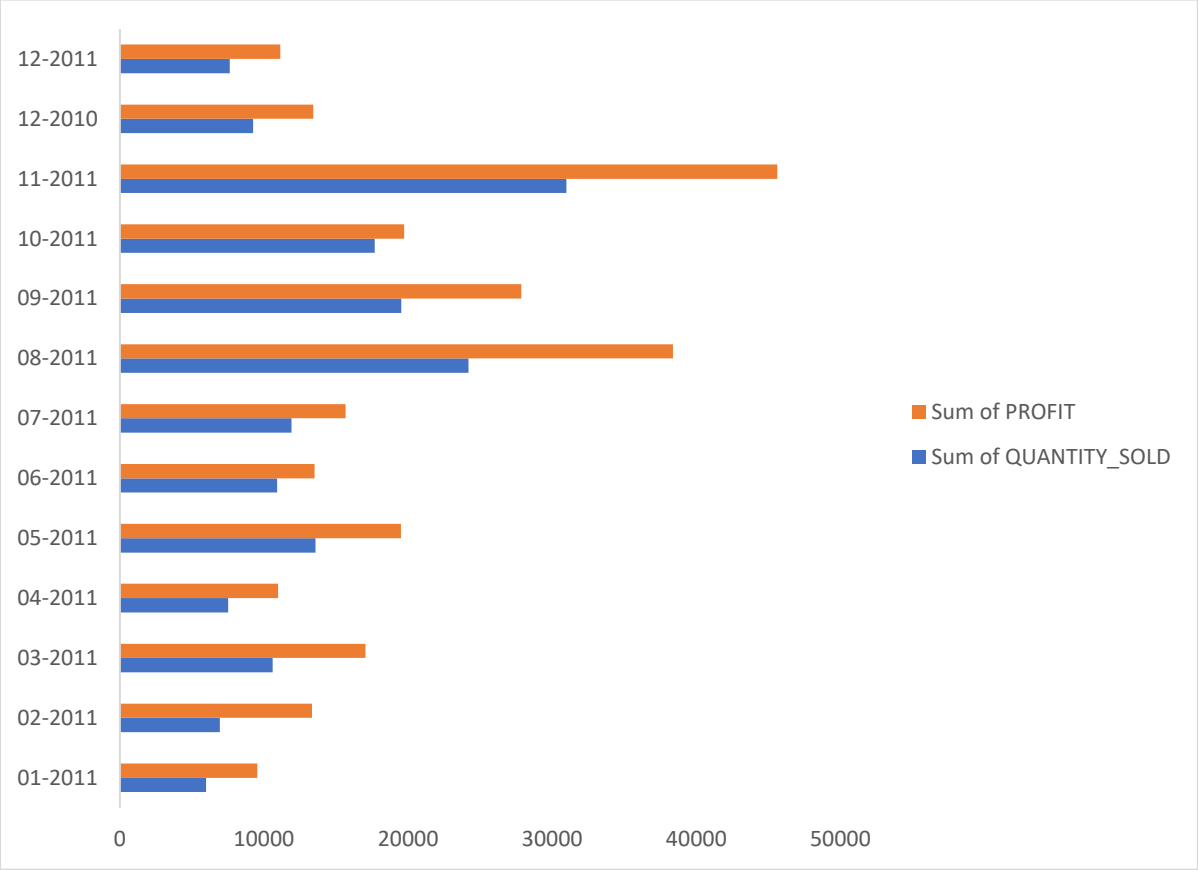
This query sums up how much we sold and earned each month. It helps us see how our sales and profits change over time, month by month. This information can guide our decisions on when to promote products or adjust pricing to maximize profits.

Query:

```
/*
Data Preparation:
-Converts the 'invoicedate' field into a 'month' format (e.g., "mm-yyyy") using the TO_CHAR and TO_DATE functions.
-Partitions the data by year and month to calculate totals.

Calculations:
-Computes the total quantity sold for each month using the SUM function over the partitioned data.
-Calculates the total profit for each month by multiplying quantity sold by price and summing the results over the partitioned data.

Results:
-Retrieves distinct months along with the corresponding quantity sold and profit.
-Orders the results by month.
*/
select distinct  to_char(to_date(invoicedate,'mm/dd/yyyy hh24:mi'),'mm-yyyy') as "month" ,
sum(quantity) over(partition by  to_char(to_date(invoicedate,'mm/dd/yyyy hh24:mi'),'yyyy-mm')) as Quantity_Sold,
    sum(quantity*price) over(partition by  to_char(to_date(invoicedate,'mm/dd/yyyy hh24:mi'),'yyyy-mm')) as Profit
from tableretail
order by "month";
```

Output:

month	QUANTITY_SOLD	PROFIT
01-2011	5972	9541.29
02-2011	6927	13336.84
03-2011	10593	17038.01
04-2011	7511	10980.51
05-2011	13584	19496.18
06-2011	10902	13517.01
07-2011	11908	15664.54
08-2011	24201	38374.64
09-2011	19524	27853.82



Conclusion:

This chart displays each month Total profit and quantity sold. This query provides valuable insights into sales and profitability trends over time, allowing businesses to analyze monthly performance and make informed decisions to optimize revenue and maximize profits.

8- Top 10 products by monthly profit and total profit:

Description:

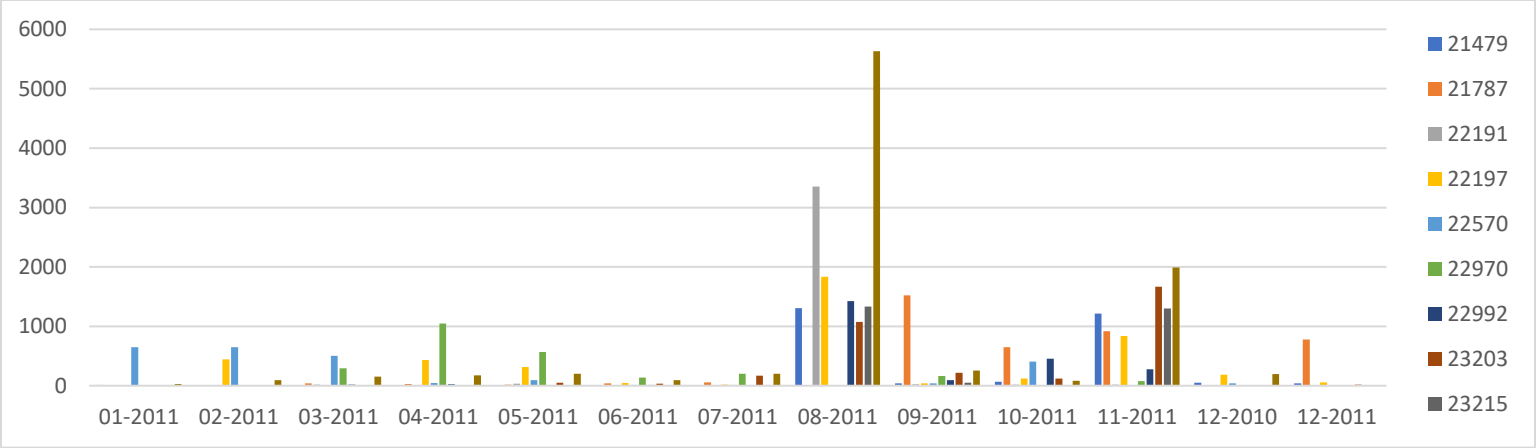
This query finds the top 10 products based on their monthly and total profits. It first calculates the monthly and total profits for each product. Then, it ranks the products by their total profits. Finally, it selects the top 10 products and displays their monthly and total profits. This information helps businesses identify their most profitable products and make strategic decisions to maximize revenue.

Query:

```
/*
Data Preparation (CTE 'cte'):
-Groups data by product code and month.
-Calculates the monthly profit and total profit for each product.

Ranking Products (CTE 'top10products'):
-Ranks products based on their total profit in descending order.

Main Query:
-Selects the product code, month, monthly profit, and total profit from the 'top10products' CTE.
-Filters the results to include only the top 10 products based on their total profit rank.
-Orders the results by total profit and then by month.
*/
with cte as (
    select distinct stockcode , to_char(to_date(invoicedate,'mm/dd/yyyy hh24:mi'),'mm-yyyy') as
"month" , sum(quantity * price) over(partition by stockcode ,
to_char(to_date(invoicedate,'mm/dd/yyyy hh24:mi'),'mm-yyyy')) as monthly_profit ,
    sum(quantity * price) over(partition by stockcode) as total_profit
    from tableretail
),

top10products as(
    select cte.* , dense_rank() over (order by total_profit desc) rnk
    from cte
)

select stockcode , "month" , monthly_profit , total_profit
from top10products
where rnk between 1 and 10
order by total_profit desc , "month";
```

Output:

STOCKCODE	month	MONTHLY_PROFIT	TOTAL_PROFIT
84879	07-2011	202.8	9114.69
84879	08-2011	5633.68	9114.69
84879	09-2011	256.88	9114.69
84879	10-2011	81.12	9114.69
84879	11-2011	1991.81	9114.69
84879	12-2010	198.04	9114.69
22197	02-2011	442.2	4323.1
22197	04-2011	434.55	4323.1
22197	05-2011	313.5	4323.1



Conclusion:

This chart displays each month for each product in the month the total profit for it. This query helps businesses identify their most profitable products both on a monthly basis and overall. It enables them to focus on optimizing sales and marketing strategies for these top-performing products.

Step 2: Implementing RFM Model:

Description:

This query segments customers based on their recency, frequency, and monetary values, assigning each customer to a specific segment. The segmentation helps identify different customer groups and tailor marketing strategies accordingly.

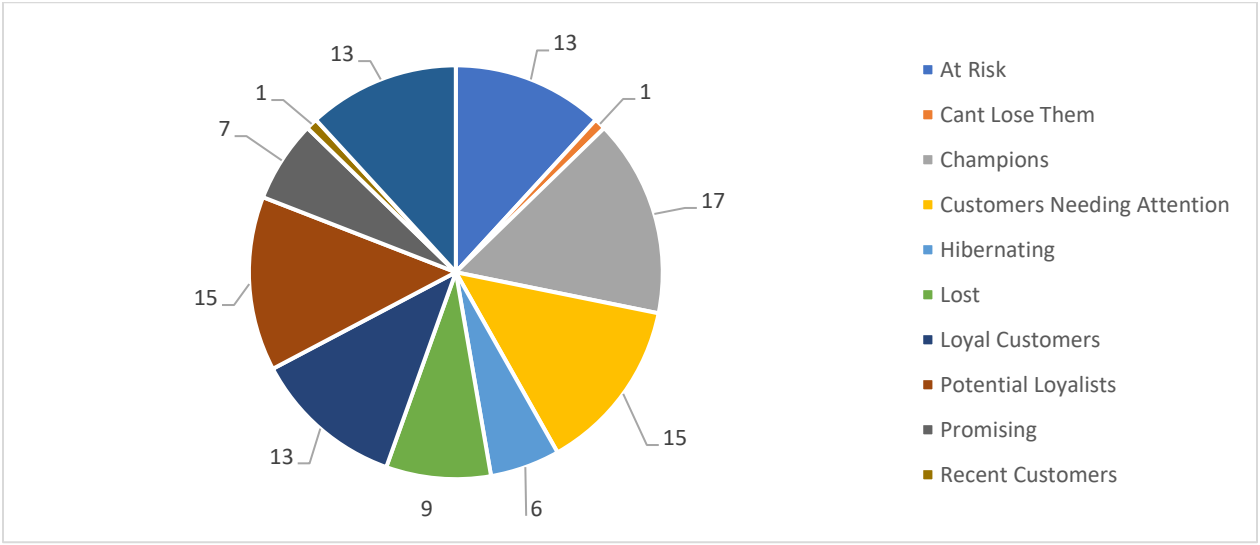
Query:

```
with rfm as(
    select customer_id , round((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,
            sum(quantity * price) as Monetary
    from tableretail
    group by customer_id
),

rfm_score as(
    select customer_id , recency , frequency , monetary,
            ntile(5) over(order by recency desc) r_score,
            ntile(5) over(order by frequency) f_score,
            ntile(5) over(order by monetary) m_score
    from rfm
),


fm_score as (
    select customer_id , recency , frequency , monetary , r_score , ntile(5) over(order by
(f_score+m_score)/2) fm_score
    from rfm_score
    group by customer_id , recency , frequency , monetary , r_score , f_score , m_score
)

select customer_id , recency , frequency , monetary , r_score , fm_score,
        CASE
            WHEN r_score = 5 AND fm_score = 5 THEN 'Champions'
            WHEN r_score = 5 AND fm_score = 4 THEN 'Champions'
            WHEN r_score = 5 AND fm_score = 5 THEN 'Champions'

            WHEN r_score = 5 AND fm_score = 2 THEN 'Potential Loyalists'
            WHEN r_score = 4 AND fm_score = 2 THEN 'Potential Loyalists'
            WHEN r_score = 3 AND fm_score = 3 THEN 'Potential Loyalists'
            WHEN r_score = 4 AND fm_score = 3 THEN 'Potential Loyalists'

            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 = 5 THEN 'Loyal Customers'
            WHEN 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 THEN 'Promising'
        WHEN r_score = 3 AND fm_score = 1 THEN 'Promising'

        WHEN r_score = 3 AND fm_score = 2 THEN 'Customers Needing Attention'
        WHEN r_score = 2 AND fm_score = 3 THEN 'Customers Needing Attention'
        WHEN r_score = 2 AND fm_score = 2 THEN 'Customers Needing Attention'

        WHEN r_score = 2 AND fm_score = 5 THEN 'At Risk'
        WHEN r_score = 2 AND fm_score = 4 THEN 'At Risk'
        WHEN r_score = 1 AND fm_score = 3 THEN 'At Risk'

        WHEN r_score = 1 AND fm_score = 5 THEN 'Cant Lose Them'
        WHEN 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 'Uncategorized'
    END AS cust_segment
from fm_score
order by customer_id desc;
```

Output:

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	Uncategorized
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 Attention
12963	8	8	1856.63	5	5	Champions
12962	7	2	266.39	5	1	Recent Customers
12957	9	8	4017.54	4	5	Uncategorized



Step 2: Daily Purchasing Transactions:

1- Maximum number of consecutive days a customer made purchases:

Description:

This code calculates the maximum number of consecutive days a customer made purchases using the daily transaction data in the "dailycustomers" table.

```
with cte as(
    select cust_id , calendar_dt , lag(calendar_dt) over(partition by cust_id order by calendar_dt)
lag_cal , case
                                    when lag(calendar_dt) over(partition by cust_id order by
calendar_dt)  + 1 = calendar_dt
                                    then 0
                                    else 1
                                    end as check_cons_day
    from customers
),

--sum the check cons day partition by cust_id and using order by to use rows between
unbounded preceding and current row
flagged_partition as (
    SELECT
        cte.*,
        SUM(check_cons_day) OVER (PARTITION BY cust_id ORDER BY calendar_dt) AS
flag_partition
    FROM cte
)

--counting partition by the flagpartition and then get max of that count
select distinct cust_id , max(temp_consecutive_days) over(partition by cust_id)
max_consecutive_number
from(
    select flagged_partition.* , count(*) over(partition by cust_id , flag_partition)
temp_consecutive_days
    from flagged_partition
);
```

Output:

CUST_ID	MAX_CONSECUTIVE_NUMBER
26592	35
66688	5
175749	2
625527	19
710264	12
924314	2
1073321	3
1444226	3
1490919	3

2- Number of days or transactions it takes a customer to reach a spent threshold of 250 L.E:

Description:

The query calculates the number of days or transactions it takes for a customer to reach a spent threshold of 250 L.E using CTEs and window functions.

```
/*
Common Table Expression (CTE 'cte'):
-Calculates the cumulative sum of transaction amounts (amt_le) for each customer (cust_id)
over time (calendar_dt).
-Assigns a row number (rnk) to each transaction within each customer's transaction history,
indicating the order in which transactions occur.
-Determines the first transaction date (first_dt) for each customer.

First Query:
-Selects distinct cust_id, calculating the number of transactions (num_transactions) and the
number of days (num_days) from the first transaction to the current transaction.
-Filters transactions where the cumulative amount spent (sum_amt) is greater than or equal to
250.
-Orders the results by cust_id.

Second Query (for Average):
-Uses the results from the first query.
Calculates the average number of transactions (avg_num_transactions) and the average number
of days (avg_num_days) for customers whose cumulative spending exceeds or equals 250.
*/

with cte as (
   select cust_id ,calendar_dt, amt_le ,sum(amt_le) over(partition by cust_id order by
calendar_dt) as sum_amt , row_number() over(partition by cust_id order by calendar_dt) rnk ,
min(calendar_dt) over(partition by cust_id) first_dt
   from customers
)

-- to get each num trasactions and num days for each customer use this query
select distinct cust_id , min(rnk)over(partition by cust_id) num_transactions, min(calendar_dt -
first_dt)over(partition by cust_id) num_days
from cte
where sum_amt >= 250
order by cust_id asc;
```

Output:

CUST_ID	NUM_TRANSACTIONS	NUM_DAYS
26592	5	4
45234	17	59
60045	9	8
66688	4	32
113502	7	13
151293	8	48
217534	2	1
232210	5	21
259866	1	0

For Average use same cte's but change last query by this query

```
select avg(num_transactions) avg_num_transactions , avg(num_days) avg_num_days
from(
    select distinct cust_id , min(rnk)over(partition by cust_id) num_transactions, min(calendar_dt
- first_dt)over(partition by cust_id) num_days
    from cte
    where sum_amt >= 250
    order by cust_id asc
);
```

Output:

AVG_NUM_TRANSACTIONS	AVG_NUM_DAYS
6.25507350304769	11.3541054141269