|  |
| --- |
| Close-up of market graph analysis |
| Customers Transactions Analysis |
| |  |  |  | | --- | --- | --- | | Omar Ashraf |  |  | |

**Step 1: Exploring Dataset:**

**Overview**

This report provides insights into customer transactions, offering businesses actionable recommendations to optimize marketing strategies, inventory management, customer retention, and pricing based on data analysis, thereby fostering growth, and enhancing customer satisfaction.

**Objective**

The objective of this report is to assist businesses in leveraging customer transaction data to gain valuable insights and practical recommendations for enhancing sales, optimizing inventory management, improving customer satisfaction, and setting appropriate pricing strategies.

**Dataset**

The dataset comprises 12,858 rows of transactional records, each containing details such as the invoice number, stock code, quantity, invoice date, price, customer ID, and country. These columns provide information on unique invoice numbers, product codes, purchase quantities, transaction dates, item prices, customer identifiers, and transaction locations, respectively.

**A table with numbers and time

Description automatically generated**

**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:**

A table with numbers and text

Description automatically generated

**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:**

A screenshot of a table

Description automatically generated

**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:**

A screenshot of a spreadsheet

Description automatically generated

**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:**

A screenshot of a data

Description automatically generated

**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:**

A table with numbers and letters

Description automatically generated

**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:**

A screenshot of a spreadsheet

Description automatically generated

**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:**

A screenshot of a table

Description automatically generated

**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:**

A screenshot of a spreadsheet

Description automatically generated

**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'

**Output:**

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;

A table with numbers and letters

Description automatically generated

**Step 2: Daily Purchasing Transactions:**

The Customers dataset contains 574396 rows of daily purchasing transactions data for customers. Each row represents a purchase made by a customer and includes information such as the customer ID, purchasing date, and the amount.

**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:**

A screenshot of a computer

Description automatically generated

**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:**

A screenshot of a table

Description automatically generated

**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:**

