Insights Into Customer Purchasing Behavior

Customer Transaction Analysis

- Get the **Top 10 products by sales**, we want to check which 10 products caused the highest sales as this means that we need to support our stores always with these products as they are high in demand.

```
SELECT *
FROM -- use row_number for ranking then filter the result with condition in order to get the TOP 10
PRODUCT BY SALES

 (SELECT DISTINCT StockCode,
            TotalSales,
            row_number() OVER (
                        ORDER BY TotalSales DESC) Ranked
  FROM -- Get the total sales of the products by multiplying the price of the product with its quantity

   (SELECT DISTINCT StockCode,
             sum(Quantity * price) OVER (PARTITION BY StockCode) AS TotalSales
    FROM tableRetail))
WHERE Ranked <= 10
ORDER BY Ranked;
```

- I wanted to know what products that are **always sold together** and **how many times** these phenomena occur so we can adjust their places in our store or make offers on them both to increase sales and customer satisfaction "**it was done using MySQL"**

```
SELECT DISTINCT t1.StockCode AS Product1,
          t2.StockCode AS Product2,
          COUNT(*) OVER (PARTITION BY t1.StockCode,
                           t2.StockCode) AS TimesSoldTogether -- The PARTITION BY clause is used to
partition the dataset by the unique combinations of Product1 and Product2.
FROM tableRetail t1
JOIN tableRetail t2 ON t1.Invoice = t2.Invoice
AND t1.StockCode < t2.StockCode
ORDER BY TimesSoldTogether DESC
LIMIT 20;
```

- Now I want to know if there is a certain **time** where there is high sales in it so I noticed that the sales are higher at the morning and then it started to decrease in the night , so in order to get more customers in this duration and decrease the stress on hour employees we could make more offers at night so increase the presence of customers in this timing and encourage them to visit us more. **"It was done using MySQL"**

```
-- Calculate SALES for EACH HOUR of the day
SELECT DATE_FORMAT(LowestSalesTime, '%H:00') AS HOUR,
    round(SUM(Quantity * Price), 0) AS Sales
FROM
 (SELECT STR_TO_DATE(InvoiceDate, '%m/%d/%Y %H:%i') AS LowestSalesTime,
      Quantity,
      Price
  FROM tableRetail) t
GROUP BY HOUR
ORDER BY sales DESC ;
```

- From the previous data we can also make offers on the products that show least sales, but we will choose the time of the offer to be the hours that have the highest sales from the previous example which is the afternoon timing “Exactly at 13:00 pm”, so this will increase our revenue and help us to sell more products.

```
SELECT *
FROM
 (SELECT DISTINCT price,
          quantity,
          StockCode,
          TotalSales,
          ROW_NUMBER() OVER (
                    ORDER BY TotalSales ASC) AS Ranked
 FROM
  (SELECT DISTINCT price,
           quantity,
           StockCode,
           SUM(Quantity * price) OVER (PARTITION BY StockCode) AS TotalSales
   FROM tableRetail))
WHERE Ranked <= 10
ORDER BY Ranked ASC;
```

- Identify the customers with the highest Revenue and their contribution in the total revenue so we can deal with those customers in another way giving them different offers and different treatment on visiting the store.

```
SELECT *
FROM
 (SELECT DISTINCT price,
          quantity,
          StockCode,
          TotalSales,
          ROW_NUMBER() OVER (
                    ORDER BY TotalSales ASC) AS Ranked
 FROM
  (SELECT DISTINCT price,
           quantity,
            StockCode,
            SUM(Quantity * price) OVER (PARTITION BY StockCode) AS TotalSales
  FROM tableRetail))
WHERE Ranked <= 10
ORDER BY Ranked ASC;
```

Monetary Model:

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 - Can’t 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)

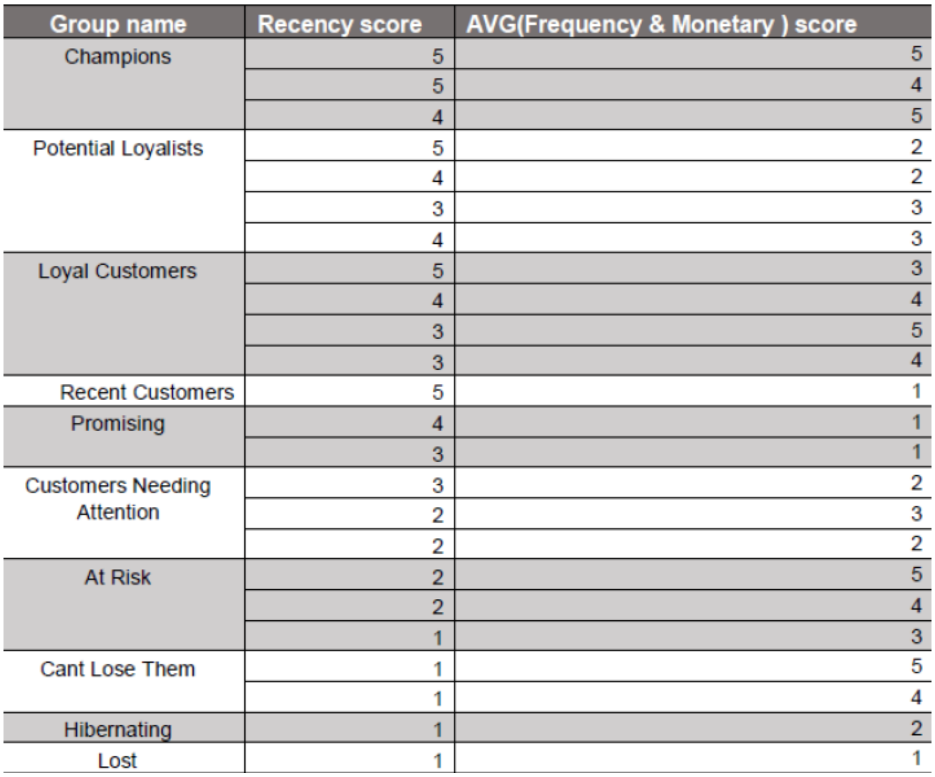
• **Frequency** => how many times the customer has bought from our store

• **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 are indicative to purchase volume anyway)

Label each customer as the following:



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

```
-- Customer Segmentation was done to the customers on certain conditions on the combination of r_score and
fm_avg
SELECT DISTINCT customer_id,
            Recency,
            Monetary,
            FREQUENCY,
            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
                    OR r_score = 2
                    AND fm_score = 1 THEN 'Hibernating'
                WHEN r_score = 1
                    AND fm_score <= 1 THEN 'Lost'
            END cust_segment
FROM -- Rule was done by giving range on the scale from 0 to 1 in order to rank RECENCY  from 1 to 5
producing r_score
  (SELECT DISTINCT customer_id,
               Recency,
               Monetary,
               FREQUENCY,
               CASE
                  WHEN Recency BETWEEN 0 AND 0.2 THEN 1
                  WHEN Recency BETWEEN 0.2 AND 0.4 THEN 2
                  WHEN Recency BETWEEN 0.4 AND 0.6 THEN 3
                  WHEN Recency BETWEEN 0.6 AND 0.8 THEN 4
                  WHEN Recency BETWEEN 0.8 AND 1 THEN 5
               END AS r_score,
               CASE -- Rule was done by giving range on the scale from 0 to 1 in order to rank Fm_avg from 1
to 5
                  WHEN Fm_avg BETWEEN 0 AND 0.2 THEN 1
                  WHEN Fm_avg BETWEEN 0.2 AND 0.4 THEN 2
                  WHEN Fm_avg BETWEEN 0.4 AND 0.6 THEN 3
                  WHEN Fm_avg BETWEEN 0.6 AND 0.8 THEN 4
                  WHEN Fm_avg BETWEEN 0.8 AND 1 THEN 5
               END AS Fm_score
   FROM
    (SELECT DISTINCT customer_id,
                 Recency,
                 Monetary,
                 FREQUENCY,
                 round((Monetary + FREQUENCY)/ 2, 2) AS Fm_avg
     FROM /* In This Part I made Normalization for RECENCY,MONETARY and FREQUENCY in order to
scale the dataset from 0 to 1 in order to facilitate the comparison using the following equation
x_normalized = (x - min(x)) / (max(x) - min(x)) where x is the original value*/
      (SELECT DISTINCT customer_id,
                   round((Get_Recency - MIN(Get_Recency) OVER ()) / (MAX(Get_Recency) OVER () -
MIN(Get_Recency) OVER ()), 2) AS Recency,
                   round((Get_Monetary - MIN(Get_Monetary) OVER ()) / (MAX(Get_Monetary) OVER () -
MIN(Get_Monetary) OVER ()), 2) AS Monetary,
                   round((Get_FREQUENCY - MIN(Get_FREQUENCY) OVER ()) /
(MAX(Get_FREQUENCY) OVER () - MIN(Get_FREQUENCY) OVER ()), 2) AS FREQUENCY
       FROM -- In order to calculate RECENCY it was needed to get the Most Recent purchase date in the
dataset it was done by MAX function over the whole dataset and then subtract it from the last purchase done
by the customer
```

```
        (SELECT DISTINCT customer_id,
                    round(Max(orderdate) over() - Last_purchase_for_customer, 0) Get_Recency,
                    Get_Monetary,

                    Get_FREQUENCY

         FROM

           (SELECT DISTINCT customer_id,

                      orderdate,

                      last_value(orderdate) OVER (PARTITION BY customer_id

                                          ORDER BY orderdate ROWS BETWEEN UNBOUNDED preceding
AND UNBOUNDED FOLLOWING) Last_purchase_for_customer,

                                         Get_Monetary,

                                         Get_FREQUENCY

           FROM

             (SELECT DISTINCT customer_id,

                        TO_DATE(Invoicedate, 'MM/DD/YYYY HH24:MI') orderdate,

                        SUM (Quantity * price) OVER (PARTITION BY customer_id) Get_Monetary,

                                            count(DISTINCT invoice) OVER (PARTITION BY customer_id)
Get_FREQUENCY

             FROM tableRetail) sub1) sub2) sub3) sub4) sub5) sub6

ORDER BY customer_id;
```

Daily Purchasing Customers’ Transactions:

1-The maximum number of consecutive days a customer made purchases

```
-- in the first query named CTE I want to divid the data into segments in order to get the consecutive days and
this was done by making row_number then subtraction from the date
-- In the second query named count_days we will count the value of the difference to count the number of
consecutive days for the customer showing the start date and the end date in the query and filtration to
exclude 1 as we want the consective not single transaction
-- I the last query the Maximum number of consecutice days for the customer was calculated
WITH CTE AS
  (SELECT cust_id,
        purchase_date,
        row_number() OVER (PARTITION BY cust_id
                    ORDER BY purchase_date), EXTRACT (DAY
                                           FROM (purchase_date -(row_number() OVER (PARTITION BY
cust_id
                                                                        ORDER BY purchase_date)))) Difference
  FROM purchase_transactions),
   get_count AS
  (SELECT cust_id,
        start_PD,
        End_PD,
        count_days
  FROM
   (SELECT cust_id,
         COUNT (Difference) AS count_days,
             MIN (purchase_date) start_PD,
                MAX (purchase_date) End_PD
    FROM CTE
    GROUP BY cust_id,
          difference)
  WHERE count_days >= 2 )
SELECT cust_id,
     max(count_days) Maximum_Consecutive_days
FROM get_count
GROUP BY cust_id;
```

2-The Average Number of Days it takes the customer to reach a threshold of 250 LE

```sql
-- The first query Count_purchase The running total is calculated using the sum() window function to compute
the cumulative sum of the purchase amounts for each customer. The purchase count is calculated using the
row_number() window function to assign a unique count value to each transaction, within each customer ID
partition.
-- The second Query  threshold_set selects the customer ID and the purchase count value that the first
transaction that pushes the running total over or equal to 250 dollars. This is done using the FIRST_VALUE()
window function over the count_purchase CTE, which returns the first value of the count_purchase column
for each customer ID partition that meets the specified ordering criteria.
-- The main SELECT statement calculates the average value of the count_to_250 column, rounded to the
nearest integer.
WITH count_purchase AS
  (SELECT cust_id,
          purchase_date,
          amount,
          running_total,
          row_number() OVER (PARTITION BY cust_id
                             ORDER BY purchase_date) AS count_purchase
   FROM
     (SELECT cust_id,
             purchase_date,
             amount,
             sum(amount) OVER (PARTITION BY cust_id
                               ORDER BY purchase_date) AS running_total
      FROM purchase_transactions)),
     threshold_set AS
  (SELECT DISTINCT cust_id,
                   FIRST_VALUE(count_purchase) OVER (PARTITION BY cust_id
                                                     ORDER BY purchase_date,
                                                              running_total) AS count_to_250
   FROM count_purchase
   WHERE running_total >= 250 )
SELECT round(AVG(count_to_250), 0) Average_transactions
FROM threshold_set;
```