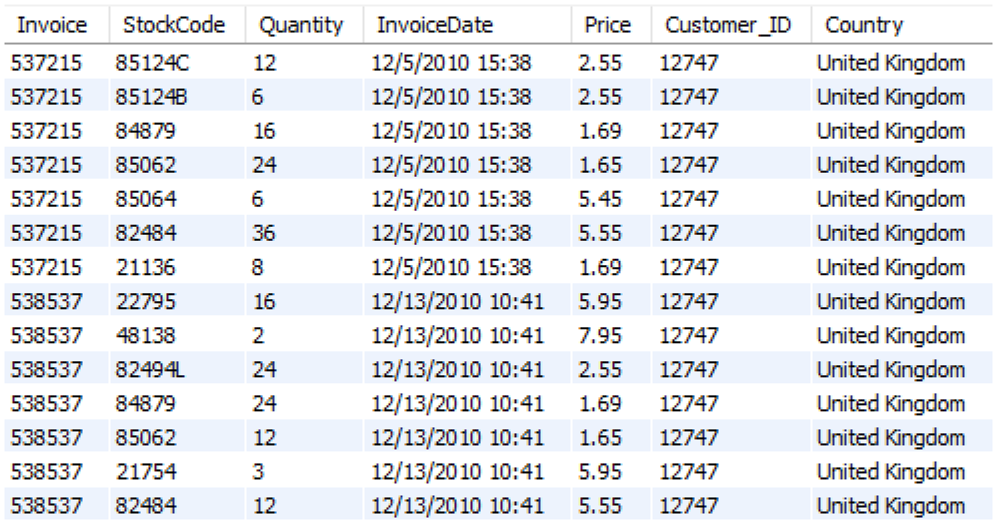
Analytical SQL Project

Mostafa Abdelrahman

ITI

Dataset Background: 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.



Invoice	StockCode	Quantity	InvoiceDate	Price	Customer_ID	Country
537215	85124C	12	12/5/2010 15:38	2.55	12747	United Kingdom
537215	85124B	6	12/5/2010 15:38	2.55	12747	United Kingdom
537215	84879	16	12/5/2010 15:38	1.69	12747	United Kingdom
537215	85062	24	12/5/2010 15:38	1.65	12747	United Kingdom
537215	85064	6	12/5/2010 15:38	5.45	12747	United Kingdom
537215	82484	36	12/5/2010 15:38	5.55	12747	United Kingdom
537215	21136	8	12/5/2010 15:38	1.69	12747	United Kingdom
538537	22795	16	12/13/2010 10:41	5.95	12747	United Kingdom
538537	48138	2	12/13/2010 10:41	7.95	12747	United Kingdom
538537	82494L	24	12/13/2010 10:41	2.55	12747	United Kingdom
538537	84879	24	12/13/2010 10:41	1.69	12747	United Kingdom
538537	85062	12	12/13/2010 10:41	1.65	12747	United Kingdom
538537	21754	3	12/13/2010 10:41	5.95	12747	United Kingdom
538537	82484	12	12/13/2010 10:41	5.55	12747	United Kingdom

you will be required to answer using SQL Analytical functions you have learnt in the course.

PROBLEM-1)

• write at least **5 analytical SQL** queries that tells a story about the data
• write small **description** about the business meaning behind each query

Query1)

```
SELECT
    DISTINCT Customer_ID,
    ROUND(Price * Quantity) AS transaction_amount,
    ROUND(AVG(Price * Quantity) OVER (PARTITION BY Customer_ID)) AS total_sales
FROM
    tableRetail
ORDER BY transaction_amount DESC;
```

Business Description: This query provides insights into the individual transaction amounts and the overall sales performance of each customer. It calculates the transaction amount for each purchase made by customers and their average total sales, aiding in understanding customer spending behavior and identifying high-value customers.

Query2)

```
SELECT distinct MONTH(STR_TO_DATE(SUBSTRING_INDEX(invoicedate, ' ', 1), '%m/%d/%Y'))
AS extracted_month,

round(avg(price*Quantity) over(partition by
MONTH(STR_TO_DATE(SUBSTRING_INDEX(invoicedate, ' ', 1), '%m/%d/%Y')))) avg_sale,

round(sum(price*Quantity) over(partition by
MONTH(STR_TO_DATE(SUBSTRING_INDEX(invoicedate, ' ', 1), '%m/%d/%Y')))) total_sale

FROM tableretail
```

Business Description: This query provides a monthly overview of sales performance, calculating the average and total sales for each month. It aids in identifying seasonal trends in customer purchasing behavior, enabling better inventory management and targeted marketing campaigns aligned with peak sales periods.

Query3)

```
SELECT
    DISTINCT StockCode,
    ROUND(SUM(Quantity) OVER (PARTITION BY StockCode)) AS total_sale
FROM
    tableRetail
ORDER BY
    total_sale DESC;
```

Business Description: This query delves into the quantity sold for each product stock code, offering crucial insights into the most sought-after products within the inventory. By discerning the top demanded products.

Query4)

```
SELECT
    DISTINCT StockCode,
    total_sale
FROM
    (SELECT
        StockCode,
        SUM(Quantity) OVER (PARTITION BY StockCode) AS total_sale
    FROM
    tableRetail) AS subquery
WHERE
  total_sale < 10
ORDER BY
  total_sale ASC;
```

Business Insight: This query identifies products that have been sold fewer than 10 items. Analyzing products with low sales volume is crucial for inventory management and decision-making. It helps businesses identify slow-moving or unpopular items that may require attention, such as reevaluation of pricing, marketing strategies, or inventory levels. By addressing underperforming products, businesses can optimize resources, improve profitability, and ensure a more balanced product portfolio that aligns with customer demand.

Query5)

```
SELECT distinct (STR_TO_DATE(invoicedate, '%m/%d/%Y')) AS extracted_day,
    round(price*Quantity ,2) sales,
    round(avg(price*Quantity) over()) avg_daily_sales
FROM tableretail;
```

Business Insights: Identify peak and slow sales days to understand customer behavior patterns.Track short-term sales trends to inform resource allocation and promotions.
Set realistic daily sales goals based on data-driven insights.Optimize sales strategies by understanding daily sales performance.Improve overall sales performance through informed decision-making.

Information
Technology
Institute

PROBLEM-2)

On the previous data set:

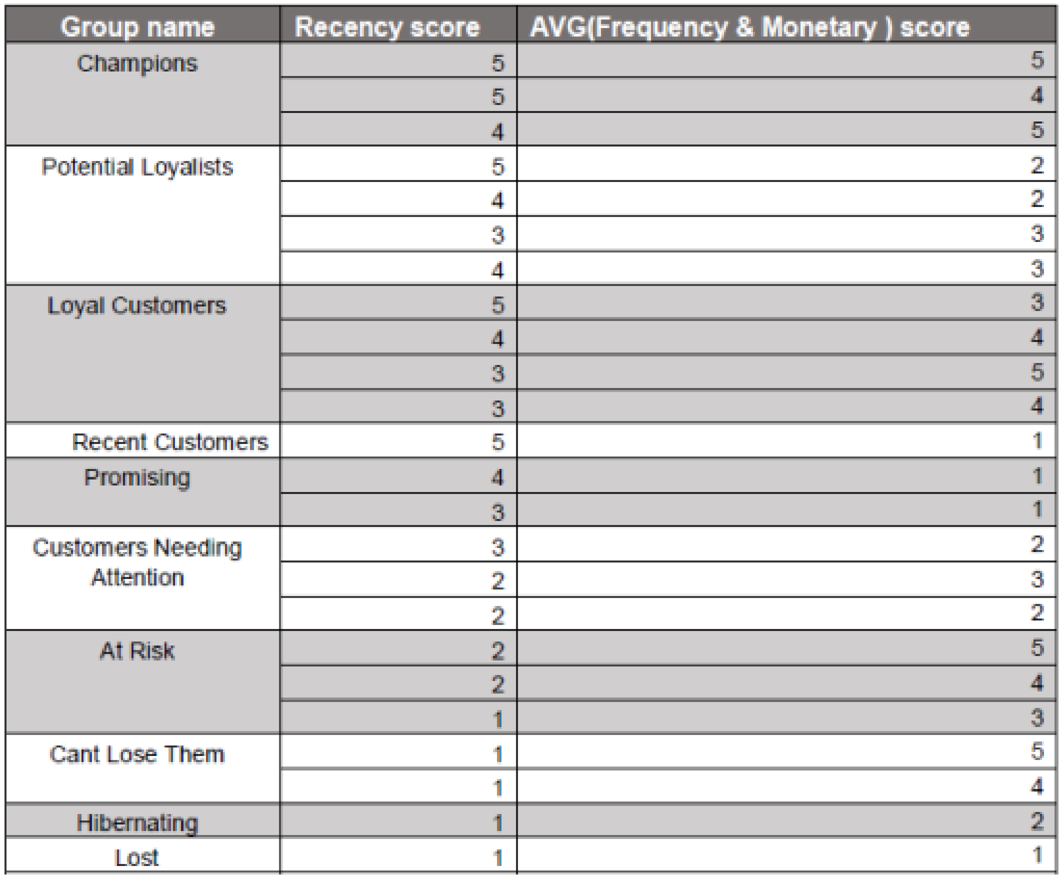
implement a Monetary model for customers behavior for product purchasing and segment each customer based on the below groups Champions - Loyal Customers - Potential Loyalists – Recent Customers – Promising - Customers Needing Attention - At Risk - Cant Lose Them – Hibernating – Lost The customers will 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 of them are indicative to purchase volume anyway)

Label each customer based on the below values



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

Here's the query:

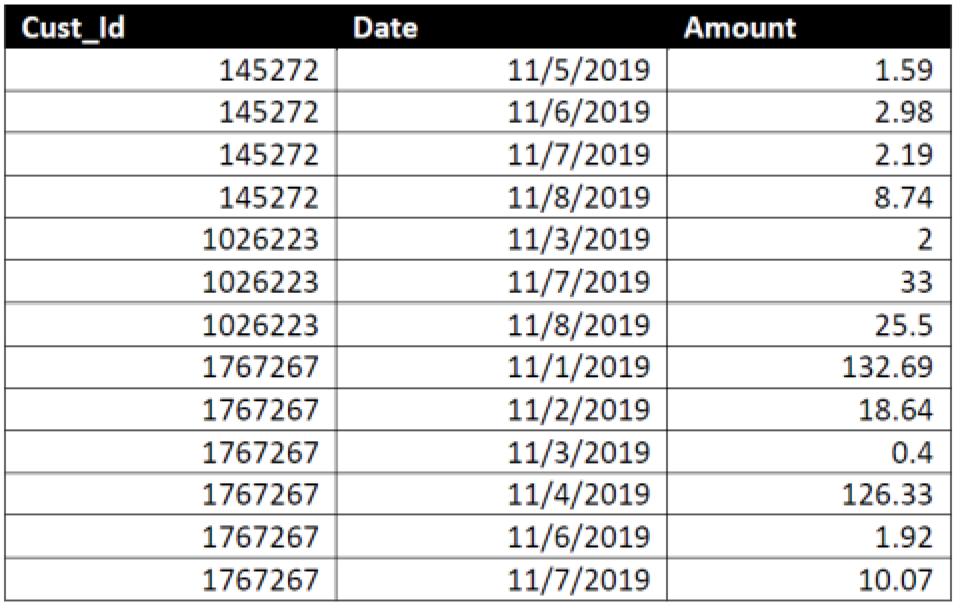
```
with t1 as (
(select distinct count(*) over(partition by Customer_ID) frequency, customer_id,
sum(price*Quantity) over(partition by Customer_ID) total_customer_sale,
max(STR_TO_DATE(SUBSTRING_INDEX(InvoiceDate, ' ', 1), '%m/%d/%Y' )) over(partition by
customer_id) recent_purchase
from tableretail)
),
t2 as (
select distinct customer_id, STR_TO_DATE('2011/29/12', '%Y/%d/%m') - recent_purchase
recency, frequency,
round(percent_rank() over(order by total_customer_sale),2) monetary,
ntile(5) over(order by STR_TO_DATE('2011/29/12', '%Y/%d/%m') - recent_purchase desc)
r_score,
ntile(5) over(order by frequency) f_score,
round(((round(percent_rank() over(order by total_customer_sale),2)+.01)*5 + ntile(5)
over(order by frequency))/2) fm_score
from  t1 )

select Customer_ID, recency, frequency, monetary, r_score, fm_score,
case when (r_score = 5 and fm_score = 5) or (r_score = 4 and fm_score = 5) or (r_score = 5
and fm_score = 4) then 'Champions'
when (r_score = 5 and fm_score = 2) or (r_score = 4 and fm_score = 2) or (r_score = 4 and
fm_score = 3) or (r_score = 3 and fm_score = 3) then 'Potenial Loyalist'
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 Customer'
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 = 4) or (r_score = 1 and fm_score = 5)  then 'Can''t Lose
Them'
when (r_score = 1 and fm_score = 2) then 'Hybernating'
when (r_score = 1 and fm_score = 1) then 'Lost'
end cust_segment
from t2;
```

This query segments customers into different categories based on their purchase history. Here's a breakdown of the steps involved:

1. **Create CTEs (t1 and t2):**
   - **t1:** Calculates various customer metrics like total purchase amount, recent purchase date, and purchase frequency for each customer.
   - **t2**: It takes the data from t1 and calculates additional scores and segments for each customer.
2. **Calculate customer scores:**
   - **Monetary score:** This score (between 0 and 1) reflects a customer's ranking based on their total spending compared to others.
   - **Recency score:** This score (between 1 and 5) indicates how recently a customer made a purchase, with 5 being the most recent.
   - **Frequency score:** This score (between 1 and 5) represents a customer's purchase frequency compared to others, with 5 being the most frequent.
   - **FM Score:** This combines the monetary and frequency scores (between 1 and 10) to get a holistic view of customer value.
3. **Segment customers:**
   - Based on the recency and FM scores, the query assigns each customer a segment label like "Champions," "Loyal Customer," "At Risk," etc. These labels represent different customer categories based on their purchase behavior.

PROBLEM-3)

Given the following data set as csv file



Cust_Id	Date	Amount
145272	11/5/2019	1.59
145272	11/6/2019	2.98
145272	11/7/2019	2.19
145272	11/8/2019	8.74
1026223	11/3/2019	2
1026223	11/7/2019	33
1026223	11/8/2019	25.5
1767267	11/1/2019	132.69
1767267	11/2/2019	18.64
1767267	11/3/2019	0.4
1767267	11/4/2019	126.33
1767267	11/6/2019	1.92
1767267	11/7/2019	10.07

1)

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

Information Technology Institute

Here's the query:

```
WITH source_cte AS (
  SELECT
    cust_id,
    calendar_dt,
    LEAD(calendar_dt) OVER(ORDER BY cust_id, calendar_dt) AS lead_cal,
    LEAD(calendar_dt) OVER(ORDER BY cust_id, calendar_dt) - calendar_dt AS streak,
    CASE
      WHEN COALESCE(difference1, 2) != 1 THEN 'start new streak'
      WHEN difference2 != 1 THEN 'end streak'
      ELSE 'inside'
    END AS label
  FROM (
    SELECT
      cust_id,
      calendar_dt,
      LAG(calendar_dt) OVER(ORDER BY cust_id, calendar_dt) AS lagged_date,
      calendar_dt - LAG(calendar_dt) OVER(ORDER BY cust_id, calendar_dt) AS difference1,
      LEAD(calendar_dt) OVER(ORDER BY cust_id, calendar_dt) - calendar_dt AS difference2
    FROM transactions
  )
  WHERE (difference2 != 1 OR COALESCE(difference1, 2) != 1)
)

SELECT DISTINCT
  cust_id,
  MAX(streak) OVER(PARTITION BY cust_id) AS max_streak
FROM source_cte
WHERE label = 'start new streak' AND streak > 0
ORDER BY cust_id;
```

Information
Technology
Institute

Breakdown of the provided SQL code:

1. source_cte **(Common Table Expression):**

a. Subquery:

- This subquery calculates several columns for the main CTE:
  - lagged_date: The previous calendar date for each customer, using the LAG window function.
  - difference1: The difference between the current and previous calendar date.
  - difference2: The difference between the current and following calendar date.

b. Main query within source_cte**:**

- Selects cust_id, calendar_dt, and the results from the subquery.
- Uses a WHERE clause to filter rows where either difference2 (difference with next date) or COALESCE(difference1, 2) (coalesced difference with previous date) is not equal to 1. This selects rows potentially marking the beginning or end of streaks.
- Calculates streak: The difference between the current and next calendar date, indicating the streak length.
- Assigns labels ('start new streak', 'end streak', or 'inside') based on the calculated differences and the previous difference (using COALESCE to handle potential null values).

2. Main Query:

- Selects distinct cust_id and the maximum streak for each customer.
- Uses MAX function with a partition by cust_id to find the maximum value of streak within each customer group.
- Filters the source_cte for rows with the label 'start new streak' and a streak greater than 0 (to exclude potential initial gaps).
- Orders the results by cust_id.

Overall, this code identifies customer IDs, their longest streaks of consecutive days, and potentially marks the beginning of each new streak for further analysis.

2)

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

From what I understood, I needed to count the number of transactions or days on which the customer made transactions until they reached a total spending of 250 L.E. in the store.

Here's the query:

```
SELECT DISTINCT ROUND(AVG(trans_count) OVER(),2) avg_tans_count
from (SELECT DISTINCT cust_id, COUNT(*) OVER(PARTITION BY cust_id) + 1 AS trans_count
FROM (
   SELECT cust_id, calendar_dt,
         SUM(AMT_LE) OVER(PARTITION BY cust_id ORDER BY calendar_dt) AS running_total
   FROM transactions
) t1
WHERE running_total < 250 AND cust_id IN (
   SELECT cust_id
   FROM (
      SELECT cust_id, calendar_dt,
            SUM(AMT_LE) OVER(PARTITION BY cust_id ORDER BY calendar_dt) AS
running_total
      FROM transactions
   ) t2
   GROUP BY cust_id
   HAVING MAX(running_total) >= 250
));
```

Here's the breakdown of the query:

1. **Calculated running total:** The query first calculates the running total of transactions for each customer by adding up their transaction amounts in chronological order.

2. **Identified high-spending customers:** Then, it identifies customers who have ever spent a total of $250 or more by finding the maximum running total for each customer.

3. **Filtered transactions:** The query keeps only the transactions from the high-spending customers identified in the previous step.

4. **Counted transactions:** For each high-spending customer, it counts the number of transactions they made.