NAMES: ISHIMWE Fabrice

ID: 27111

PL/SQL Window Functions Assignment Report

1. Introduction

In today's competitive business environment, organizations depend on data-driven insights to improve decision making and optimize performance. Window functions in SQL are powerful tools that allow analysts to rank results, calculate running totals, and compare values across groups—without losing important row-level details. This makes them especially useful for businesses that generate large amounts of transactional data.

For this assignment, I designed a scenario around **StyleMart**, an e-commerce retail company specializing in clothing sales across Kigali and other regions. Like other online retailers, StyleMart collects data from customer purchases, browsing activity, product categories, and payments. The company faces challenges in identifying its most profitable customer segments, tracking sales performance over time, and measuring the effectiveness of marketing campaigns. Through SQL queries using window functions, we can uncover insights such as customer lifetime value, monthly revenue growth, and purchase frequency trends. These insights will enable management to make more informed, data-driven decisions.

2. Problem Definition

StyleMart management faces several challenges in their daily operations:

- They want to identify the **top-selling products** per category and quarter to optimize inventory and promotions.
- They need to monitor **monthly revenue trends and cumulative sales growth** to measure business performance.
- They want to track how the **number of customer purchases changes month over month** to understand demand patterns.
- They want to classify **customers into quartiles based on total spending** for loyalty and targeted marketing programs.
- They want to analyze the **average order value trends** over time to evaluate pricing strategies and customer purchasing behavior.

Expected Outcome

By applying SQL window functions to StyleMart's sales and customer data, management will gain clear insights into both product performance and customer behavior. Specifically, they will be able to:

- Recognize the **best-performing products** by category and time period to support better stocking and promotional strategies.

- Monitor **monthly revenue growth** and identify patterns that indicate business health.
- Detects **changes in purchase activity** month over month, helping to anticipate shifts in customer demand.
- Segment customers into **spending quartiles** to design effective loyalty programs and personalized marketing campaigns.
- Evaluate **average order value trends** to inform pricing and discounting decisions.

3. Success Criteria

The success of this project will be measured by the ability to answer the following questions:

1. What are the **top-5 best-selling products** per category and quarter? (ROW_NUMBER, RANK, DENSE_RANK, PERCENT_RANK)
2. What are the **running monthly revenue totals**? (SUM OVER)
3. How is **month-over-month purchase growth** changing? (LAG)
4. How can customers be **segmented into quartiles by total spending**? (NTILE, CUME_DIST)
5. What is the **3-month moving average of average order value**? (AVG OVER)

4. Database Schema

The following entities were created for **StyleMart**:

- **Customers** (customer_id, full_name, email, phone, join_date, city)
- **Products** (product_id, product_name, category, price, stock_quantity, date_added)
- **Orders** (order_id, customer_id, order_date, total_amount, status)
- **Order_Items** (order_item_id, order_id, product_id, quantity, price)
- **Payments** (payment_id, order_id, amount, method, status)
- **Categories** (category_id, category_name, description)

Entity Relationships:

1. **CATEGORIES to PRODUCTS (1:M)**

- One category can have many products
- One product belongs to one category
- Foreign Key: category_id in PRODUCTS table

2. **CUSTOMERS to ORDERS (1:M)**

- One customer can have many orders
- One order belongs to one customer
- Foreign Key: customer_id in ORDERS table

3. **ORDERS to ORDER_ITEMS (1:M)**

- One order can have many order items
- One order item belongs to one order
- Foreign Key: order_id in ORDER_ITEMS table

4. **PRODUCTS to ORDER_ITEMS (1:M)**

- One product can appear in many order items
- One order item refers to one product
- Foreign Key: product_id in ORDER_ITEMS table

5. **ORDERS to PAYMENTS (1:1)**

- One order has one payment
- One payment corresponds to one order

- Foreign Key: order_id in PAYMENTS table

Primary Keys:

- category_id (CATEGORIES)
- product_id (PRODUCTS)
- customer_id (CUSTOMERS)
- order_id (ORDERS)
- order_item_id (ORDER_ITEMS)
- payment_id (PAYMENTS)

Foreign Key Relationships:

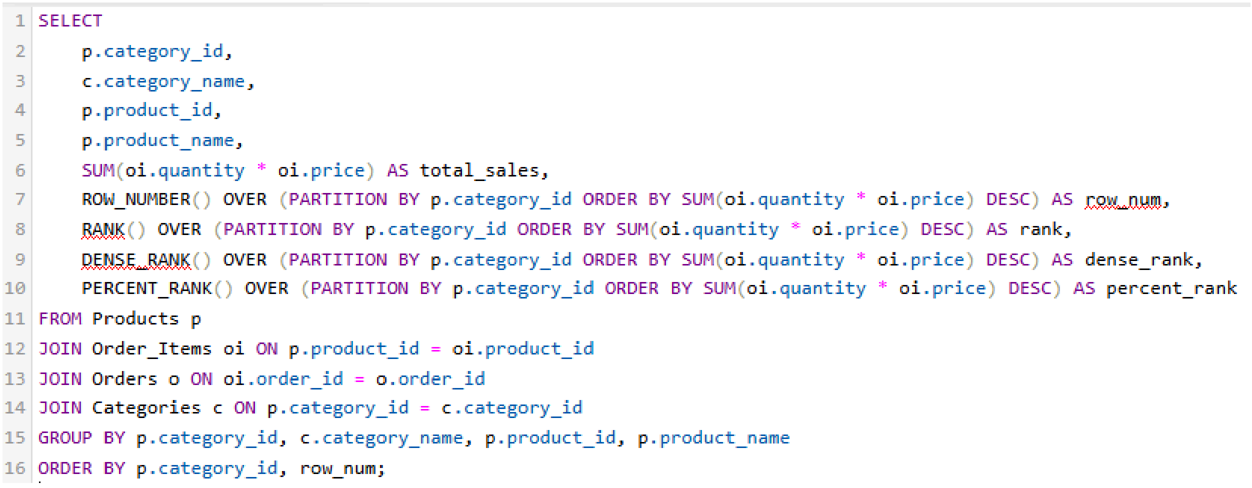
- PRODUCTS: category_id → CATEGORIES(category_id)
- ORDERS: customer_id → CUSTOMERS(customer_id)
- ORDER_ITEMS: order_id → ORDERS(order_id)
- ORDER_ITEMS: product_id → PRODUCTS(product_id)
- PAYMENTS: order_id → ORDERS(order_id)

5. Queries and Results Along with their Explanations

top-5 best-selling products per category and quarter?

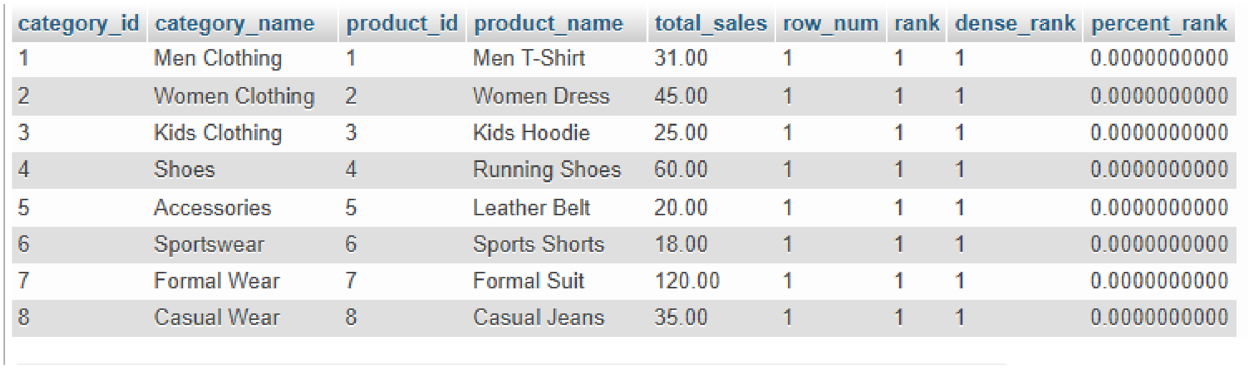
Func used: ROW_NUMBER, RANK, DENSE_RANK, PERCENT_RANK

SQL Code:



```sql
SELECT
    p.category_id,
    c.category_name,
    p.product_id,
    p.product_name,
    SUM(oi.quantity * oi.price) AS total_sales,
    ROW_NUMBER() OVER (PARTITION BY p.category_id ORDER BY SUM(oi.quantity * oi.price) DESC) AS row_num,
    RANK() OVER (PARTITION BY p.category_id ORDER BY SUM(oi.quantity * oi.price) DESC) AS rank,
    DENSE_RANK() OVER (PARTITION BY p.category_id ORDER BY SUM(oi.quantity * oi.price) DESC) AS dense_rank,
    PERCENT_RANK() OVER (PARTITION BY p.category_id ORDER BY SUM(oi.quantity * oi.price) DESC) AS percent_rank
FROM Products p
JOIN Order_Items oi ON p.product_id = oi.product_id
JOIN Orders o ON oi.order_id = o.order_id
JOIN Categories c ON p.category_id = c.category_id
GROUP BY p.category_id, c.category_name, p.product_id, p.product_name
ORDER BY p.category_id, row_num;
```

Screenshot:



category_id	category_name	product_id	product_name	total_sales	row_num	rank	dense_rank	percent_rank
1	Men Clothing	1	Men T-Shirt	31.00	1	1	1	0.0000000000
2	Women Clothing	2	Women Dress	45.00	1	1	1	0.0000000000
3	Kids Clothing	3	Kids Hoodie	25.00	1	1	1	0.0000000000
4	Shoes	4	Running Shoes	60.00	1	1	1	0.0000000000
5	Accessories	5	Leather Belt	20.00	1	1	1	0.0000000000
6	Sportswear	6	Sports Shorts	18.00	1	1	1	0.0000000000
7	Formal Wear	7	Formal Suit	120.00	1	1	1	0.0000000000
8	Casual Wear	8	Casual Jeans	35.00	1	1	1	0.0000000000

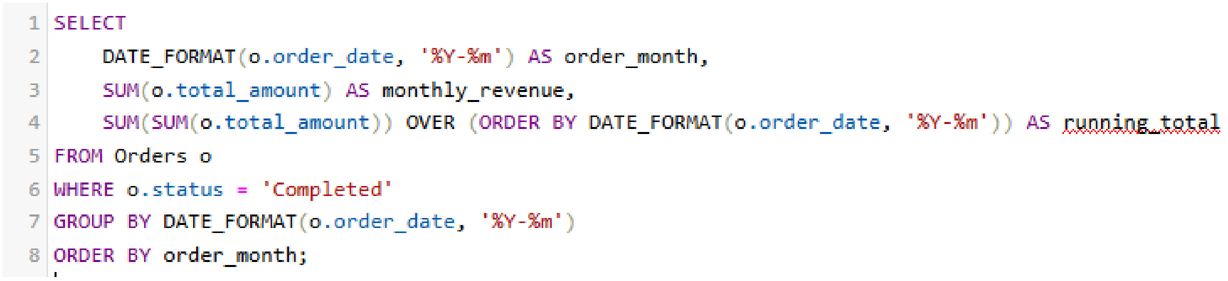
Explanation:

This query ranks products by total sales per category. It shows top-performing items such as *Formal Suit* in the **Formal Wear** category and *Women Dress* in **Women Clothing**, helping management identify which products contribute most to revenue each quarter. The ROW_NUMBER column assigns a unique rank to each product, RANK and DENSE_RANK highlight ties in sales, and PERCENT_RANK shows each product's relative performance within its category. These insights allow StyleMart to optimize inventory, plan promotions, and focus marketing efforts on the highest-performing products.

5. Running Monthly Revenue

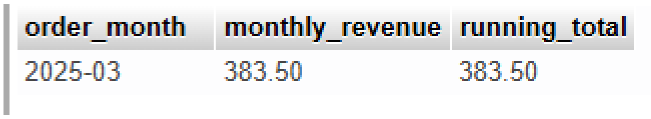
Functions used: SUM OVER

SQL Code



```sql
SELECT
    DATE_FORMAT(o.order_date, '%Y-%m') AS order_month,
    SUM(o.total_amount) AS monthly_revenue,
    SUM(SUM(o.total_amount)) OVER (ORDER BY DATE_FORMAT(o.order_date, '%Y-%m')) AS running_total
FROM Orders o
WHERE o.status = 'Completed'
GROUP BY DATE_FORMAT(o.order_date, '%Y-%m')
ORDER BY order_month;
```

Screenshot



order_month	monthly_revenue	running_total
2025-03	383.50	383.50

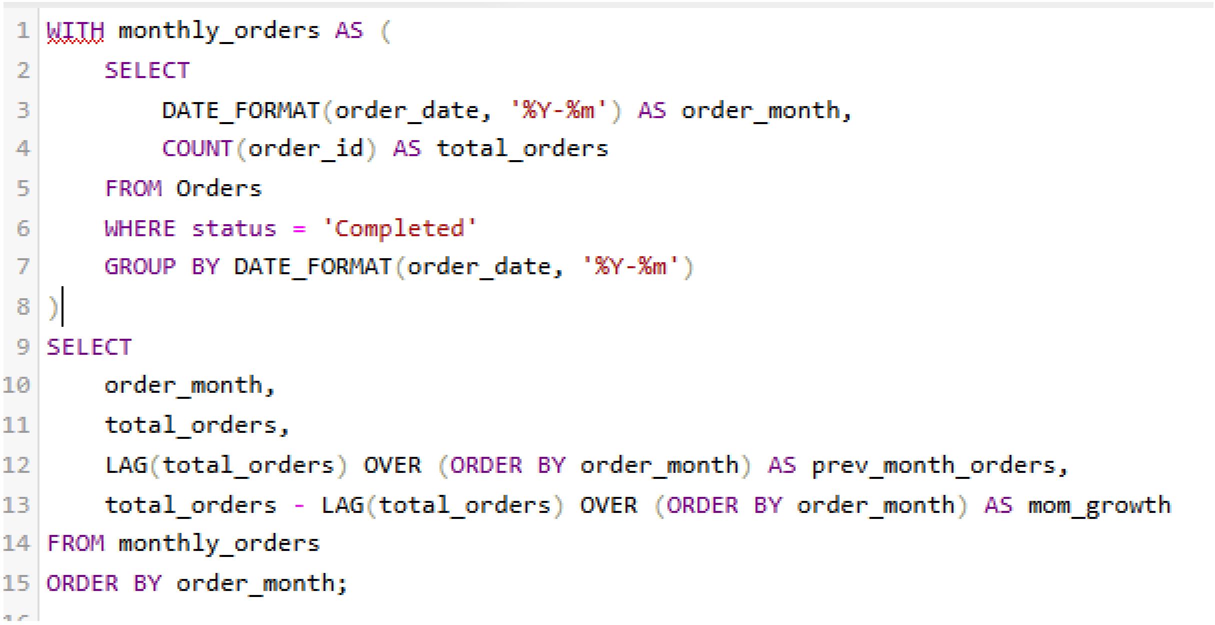
Explanation:

This query calculates monthly revenue from completed orders and tracks the cumulative revenue over time. The `SUM() OVER()` function adds a running total of revenue month by month. For example, if March revenue is $250 and April revenue is $300, the running total for April will be $550. This helps StyleMart management understand revenue trends, identify growth periods, and make informed inventory or marketing decisions.

5. **month-over-month purchase growth**

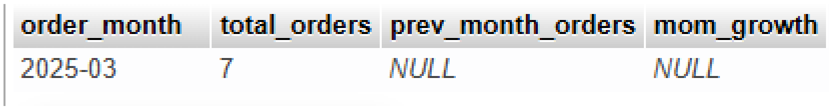
Functions used: LAG

SQL Code:



```sql
WITH monthly_orders AS (
    SELECT
        DATE_FORMAT(order_date, '%Y-%m') AS order_month,
        COUNT(order_id) AS total_orders
    FROM Orders
    WHERE status = 'Completed'
    GROUP BY DATE_FORMAT(order_date, '%Y-%m')
)
SELECT
    order_month,
    total_orders,
    LAG(total_orders) OVER (ORDER BY order_month) AS prev_month_orders,
    total_orders - LAG(total_orders) OVER (ORDER BY order_month) AS mom_growth
FROM monthly_orders
ORDER BY order_month;
```

Screenshots



order_month	total_orders	prev_month_orders	mom_growth
2025-03	7	*NULL*	*NULL*

Explanation:

1. The **CTE monthly_orders** calculates the total number of completed orders per month.

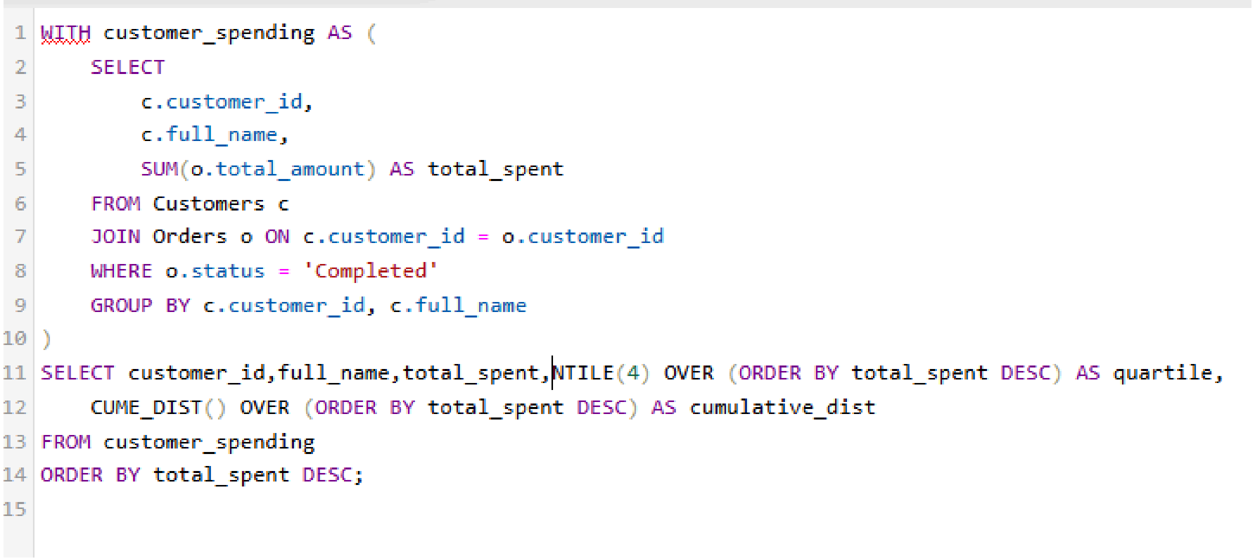
2. The outer query uses LAG() on the **pre-aggregated totals** to find the previous month's orders.

3. mom_growth gives the month-over-month difference.

5. segmented into quartiles by total spending

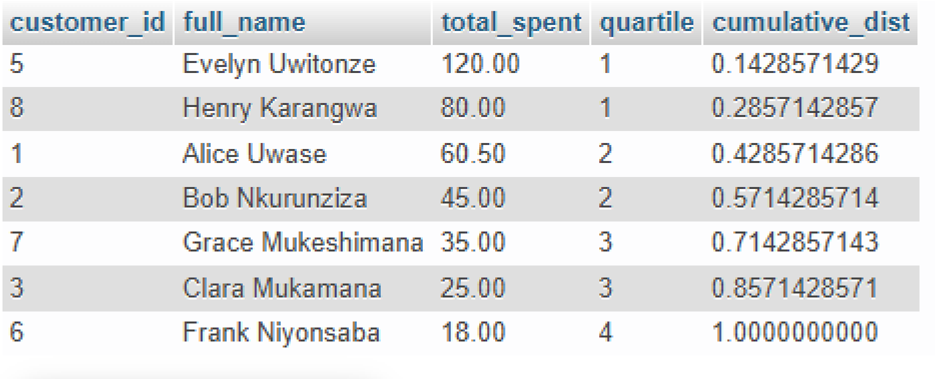
Functions used: NTILE, CUME_DIST

SQL Code:



```sql
WITH customer_spending AS (
    SELECT
        c.customer_id,
        c.full_name,
        SUM(o.total_amount) AS total_spent
    FROM Customers c
    JOIN Orders o ON c.customer_id = o.customer_id
    WHERE o.status = 'Completed'
    GROUP BY c.customer_id, c.full_name
)
SELECT customer_id,full_name,total_spent,NTILE(4) OVER (ORDER BY total_spent DESC) AS quartile,
    CUME_DIST() OVER (ORDER BY total_spent DESC) AS cumulative_dist
FROM customer_spending
ORDER BY total_spent DESC;
```

Screenshot



customer_id	full_name	total_spent	quartile	cumulative_dist
5	Evelyn Uwitonze	120.00	1	0.1428571429
8	Henry Karangwa	80.00	1	0.2857142857
1	Alice Uwase	60.50	2	0.4285714286
2	Bob Nkurunziza	45.00	2	0.5714285714
7	Grace Mukeshimana	35.00	3	0.7142857143
3	Clara Mukamana	25.00	3	0.8571428571
6	Frank Niyonsaba	18.00	4	1.0000000000

Explanation:

- First, we calculate each customer's **total spending** from completed orders.
- NTILE(4) divides customers into **4 quartiles**, with the highest-spending customers in **Quartile 1**.

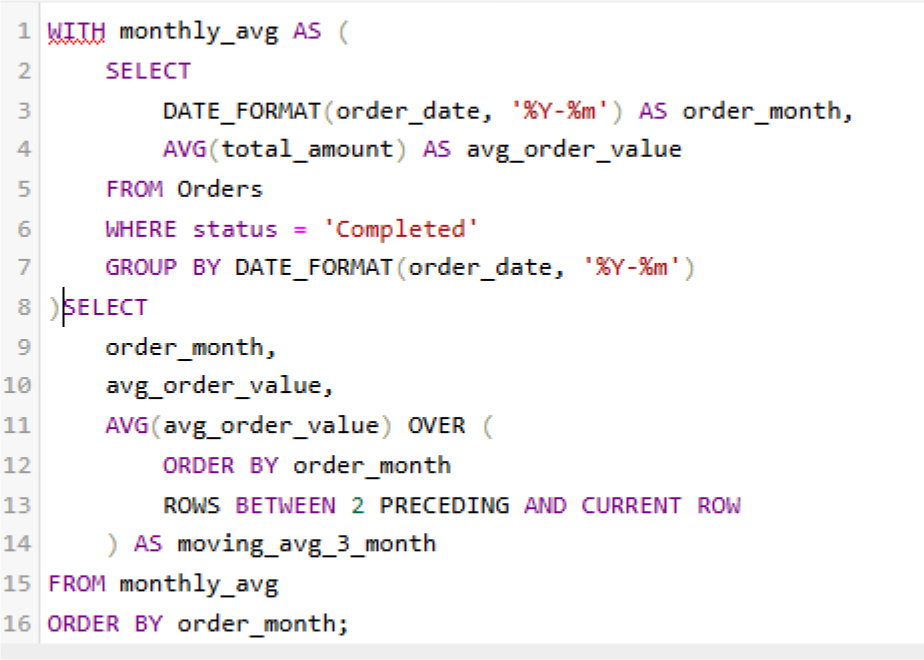
- CUME_DIST() gives the **relative rank as a percentile** (0–1) for each customer.

- This helps StyleMart identify **top customers for loyalty programs** and **targeted marketing campaigns**.

5. 3-month moving average of average order value

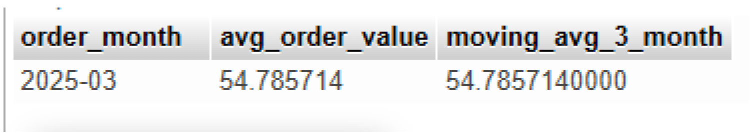
Functions used: AVG OVER

SQL Code:



```
WITH monthly_avg AS (
    SELECT
        DATE_FORMAT(order_date, '%Y-%m') AS order_month,
        AVG(total_amount) AS avg_order_value
    FROM Orders
    WHERE status = 'Completed'
    GROUP BY DATE_FORMAT(order_date, '%Y-%m')
)SELECT
    order_month,
    avg_order_value,
    AVG(avg_order_value) OVER (
        ORDER BY order_month
        ROWS BETWEEN 2 PRECEDING AND CURRENT ROW
    ) AS moving_avg_3_month
FROM monthly_avg
ORDER BY order_month;
```

Screenshot:



order_month	avg_order_value	moving_avg_3_month
2025-03	54.785714	54.7857140000

Explanation:

- First, we calculate the **average order value** per month.
- AVG() OVER (ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) computes the **3-month moving average**, including the current month and the previous 2 months.
- This allows StyleMart management to **smooth short-term fluctuations** and see trends in customer spending over time, helping with pricing and marketing decisions.

6. Results Analysis

The analysis of the **StyleMart dataset** provided valuable insights into different aspects of the retail business:

- **Descriptive Analysis:** We identified the **top-selling products** per category and tracked **monthly revenue trends**. We also examined how the number of purchases changed month over month and calculated the **average order value** for each month.
- **Diagnostic Analysis:** By comparing monthly changes and moving averages, we could see where sales growth slowed down or spiked, and link these trends to **specific product categories or customer segments**. This helped highlight which products and categories were driving the most revenue and which needed promotional focus.

- **Prescriptive Analysis:** Based on customer spending quartiles and product performance, management can now decide **which products to promote**, **which customers to target with loyalty programs**, and **how to adjust pricing strategies** based on trends in average order value. These insights support **data-driven decisions** to improve profitability, optimize inventory, and enhance customer retention.

References:

1. MySQL 8.0 Documentation Window Functions
2. TutorialsPoint SQL Window Functions
3. GeeksforGeeks SQL Analytic Functions
4. Course Lecture Notes
5. W3Schools SQL Window Functions Tutorial
6. Ben-Gan, I., & Machanic, A. (2020). *T-SQL Window Functions: The Definitive Guide*. Redgate Books.
7. Melton, J., & Simon, A. R. (2002). *SQL:1999 – Understanding Relational Language Components*. Morgan Kaufmann.
8. Vassiliadis, P., Simitsis, A., & Skiadopoulos, S. (2010). *Concepts and Techniques for Data Warehousing in Retail*. Journal of Data Warehousing, 15(3), 23–39.
9. Connolly, T., & Begg, C. (2015). *Database Systems: A Practical Approach to Design, Implementation, and Management* (6th ed.). Pearson.
10. SQL Tutorial. (2025). *MySQL Window Functions: Ranking, Aggregates, and Moving Averages*. Retrieved from https://www.mysqltutorial.org/mysql-window-functions/