

SALES ANALYSIS

Mariam Mahmoud Mohamed



AGENDA

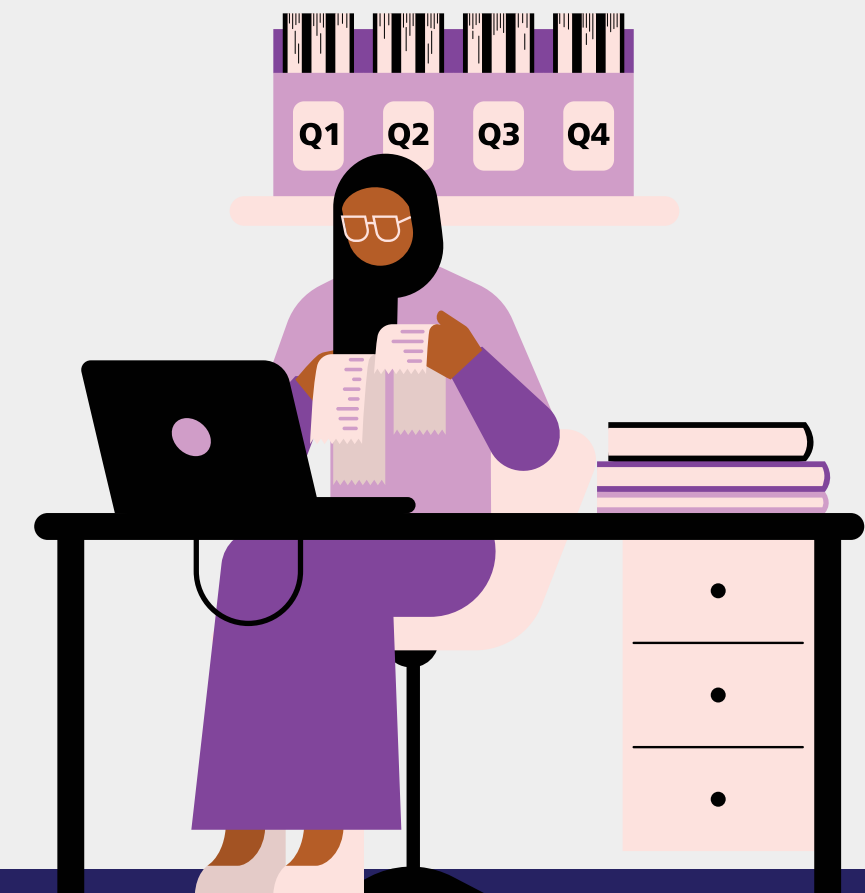
Introduction

Customer Segmentation
Analysis

Product Analysis

Order Analysis

Employee
Performance



INTRODUCTION

- This report aims to analyze customer behavior, product performance, and sales trends using SQL queries. The primary goal is to identify actionable insights, such as customer segmentation, top-performing products, and slow movers.
- Dataset Overview:
- Key tables analyzed: Customers, Orders, Order Details, Products, and Employees.
- Relationships between these tables were leveraged for a comprehensive analysis.



Customer Segmentation



Product Analysis



Order Analysis



Employee Performance



CUSTOMER SEGMENTATION ANALYSIS



Recency Analysis

I Wanted to identify how recently each customer placed an order.

I found that customers with the shortest recency indicate recent engagement with the business.
This can be used to target lapsed customers or reward frequent ones

Query Used:

```
SELECT
    CustomerID,
    MAX(OrderDate) AS LastOrderDate,
    ROUND(julianday('now') - julianday(MAX(OrderDate))) AS Recency
FROM Orders
GROUP BY CustomerID
ORDER BY Recency ASC
```



Frequency Analysis

Objective: To measure how often each customer places orders.

Metric: Count of orders placed by each customer.

Key Findings: High-frequency customers are likely loyal and can be prioritized for retention strategies.

Query Used:

```
--Frequency: Total number of orders (volume).  
SELECT  
    CustomerID,  
    COUNT(OrderID) Frequency  
FROM Orders  
GROUP BY CustomerID  
ORDER BY Frequency DESC
```



Monetary Value Analysis

Objective: To calculate the revenue generated by each customer

Metric: Total revenue per customer

Key Findings: Using total revenue, customers were evaluated to identify those contributing the highest revenue to the business, revealing valuable insights for prioritization

Query Used:

```
--Monetary Value: Total amount spent (revenue).  
SELECT  
    o.CustomerID,  
    SUM(od.UnitPrice * od.Quantity) AS MonetaryValue  
FROM Orders o  
JOIN "Order Details" od ON o.OrderID = od.OrderID  
GROUP BY o.CustomerID  
ORDER BY MonetaryValue DESC
```



```

CREATE VIEW RFM_Metrics AS
SELECT
    o.CustomerID,
    ROUND(julianday('now') - julianday(MAX(o.OrderDate))) AS Recency,
    COUNT(DISTINCT o.OrderID) AS Frequency,
    SUM(od.UnitPrice * od.Quantity * (1 - od.Discount)) AS MonetaryValue
FROM Orders o
JOIN "Order Details" od ON o.OrderID = od.OrderID
GROUP BY o.CustomerID;

CREATE VIEW RFM_Ranks AS
SELECT
    CustomerID,
    Recency,
    Frequency,
    MonetaryValue,
    NTile(3) OVER (ORDER BY Recency ASC) AS R,
    NTile(3) OVER (ORDER BY Frequency DESC) AS F,
    NTile(3) OVER (ORDER BY MonetaryValue DESC) AS M
FROM RFM_Metrics;

SELECT
    CustomerID,
    Recency,
    Frequency,
    MonetaryValue,
    R,
    F,
    M,
    CASE
        WHEN R = 1 AND F = 3 AND M = 3 THEN 'Champion'
        WHEN (F = 2 OR F = 3) AND (M = 2 OR M = 3) THEN 'Potential Loyalist'
        ELSE 'At Risk'
    END AS Segment
FROM RFM_Ranks
ORDER BY Segment;

```

Objective

This query segments Northwind's customers based on RFM metrics (Recency, Frequency, and Monetary Value). The segmentation helps categorize customers into actionable groups such as "Champions," "Potential Loyalists," and "At Risk," enabling targeted marketing strategies.

Logic

1. Creating RFM Metrics:

- A view RFM_Metrics is created to calculate the three key RFM metrics for each customer:
 - Recency: Days since the customer's most recent order.
 - Frequency: Total number of distinct orders placed by the customer.
 - Monetary Value: Total revenue generated by the customer, calculated as $\text{UnitPrice} * \text{Quantity} * (1 - \text{Discount})$.

2. Ranking RFM Metrics:

- A second view, RFM_Ranks, ranks customers into three tiers for each RFM metric using the NTILE function:
 - Recency (R): Ranked in ascending order (recent buyers have lower values).
 - Frequency (F): Ranked in descending order (frequent buyers have higher values).
 - Monetary Value (M): Ranked in descending order (high spenders have higher values).

3. Segmenting Customers:

- Customers are segmented based on their RFM ranks:
 - Champion: Customers with the lowest recency ($R = 1$) and the highest frequency ($F = 3$) and monetary value ($M = 3$).
 - Potential Loyalist: Customers with high frequency ($F = 2$ or 3) and high monetary value ($M = 2$ or 3).
 - At Risk: Customers who don't meet the above criteria.

4. Sorting Results:

- The output is sorted by the segment, making it easy to analyze each group.



Results

- The query produces a table with the following columns:
 - CustomerID: Unique identifier for the customer.
 - Recency: Number of days since the last order.
 - Frequency: Total number of distinct orders.
 - MonetaryValue: Total revenue generated.
 - R, F, M: RFM rank tiers for each metric.
 - Segment: The customer's segment based on RFM ranks.
- **Recommendations**
- Champions:
 - These customers are highly engaged and generate significant revenue. Focus on retention strategies like loyalty rewards or exclusive offers.
- Potential Loyalists:
 - These customers show promise of becoming Champions. Encourage higher engagement through personalized communication or targeted campaigns.
- At Risk:
 - These customers may be disengaged. Implement win-back strategies, such as discounts or re-engagement emails, to revive interest.



```

115 -- I want to know each segment count
116 SELECT
117     Segment,
118     COUNT(CustomerID) AS CustomerCount
119 FROM (
120     SELECT
121         CustomerID,
122         CASE
123             WHEN R = 1 AND F = 3 AND M = 3 THEN 'Champion'
124             WHEN (F = 2 OR F = 3) AND (M = 2 OR M = 3) THEN 'Potential Loyalist'
125             ELSE 'At Risk'
126         END AS Segment
127     FROM RFM_Ranks
128 ) Segments
129 GROUP BY Segment
130 ORDER BY CustomerCount DESC;
131

```

Segment	CustomerCount
Potential Loyalist	52
At Risk	37
Champion	4

The number of each segment for better analysis and to help develop strategies according to the number of each segment. For example, the champions, must be considered in how to maintain them and strive to increase their number. Potential Loyalist must be considered in how to transform them into champion. As for those who are at risk, the reasons behind that must be looked at and solved.





RECENCY

+



FREQUENCY

+



MONETARY



IDEAL CUSTOMER
SEGMENTS



Query of order value:

```
WITH REV AS (  
    SELECT  
        o.CustomerID,  
        AVG(od.UnitPrice * od.Quantity * (1 - od.Discount)) AS AverageRevenue  
    FROM Orders o  
    JOIN "Order Details" od ON od.OrderID = o.OrderID  
    GROUP BY o.CustomerID  
)  
RankedRev AS (  
    SELECT  
        CustomerID,  
        AverageRevenue,  
        Ntile(3) OVER (ORDER BY AverageRevenue DESC) AS RevenueRank  
    FROM REV  
)  
SELECT  
    CustomerID,  
    AverageRevenue,  
    RevenueRank,  
    CASE  
        WHEN RevenueRank = 1 THEN 'High-Value'  
        WHEN RevenueRank = 2 THEN 'Medium-Value'  
        ELSE 'Low-Value'  
    END AS RankRev  
FROM RankedRev  
ORDER BY AverageRevenue DESC;
```

Order Value Analysis: Customer Revenue Segmentation

Objective:

This query identifies and segments customers into three categories (High-Value, Medium-Value, and Low-Value) based on their average order revenue. This segmentation helps Northwind understand customer value distribution and tailor its strategies to maximize revenue.

Logic

1. Calculate Average Revenue:

- The first step in the query computes the average revenue per customer. This is done by calculating the total revenue for each order ($\text{UnitPrice} * \text{Quantity} * (1 - \text{Discount})$) and averaging it over all orders placed by a customer.
- This step is implemented in the Common Table Expression (CTE) named REV.

2. Ranking Customers:

- Customers are ranked into three tiers using the NTILE function. This function divides customers into equal-sized groups (in this case, three) based on their average revenue in descending order.
- This logic is encapsulated in the second CTE, RankedRev.

3. Assigning Labels:

- A CASE statement assigns labels to the revenue ranks:
 - High-Value for the top third of customers.
 - Medium-Value for the middle third.
 - Low-Value for the bottom third.

4. Sorting Results:

- The query sorts the output by AverageRevenue in descending order, ensuring the most valuable customers appear first.

Order Value Analysis: Customer Revenue Segmentation

Results

- The query produces a table with the following columns:
 - CustomerID: The unique identifier for each customer.
 - AverageRevenue: The average revenue generated by the customer.
 - RevenueRank: The rank of the customer based on average revenue.
 - RankRev: The segment label (High-Value, Medium-Value, Low-Value).

Recommendations

- High-Value Customers:
 - These customers contribute the most revenue. Northwind can prioritize retention efforts, such as personalized offers, loyalty programs, and premium services.
- Medium-Value Customers:
 - These customers represent potential for growth. Incentives like discounts or upselling can encourage higher spending.
- Low-Value Customers:
 - These customers might be less engaged or price-sensitive. Northwind can evaluate cost-effective strategies to increase their value or streamline services to optimize profitability.

PRODUCT PERFORMANCE ANALYSIS



High Revenue Products

Objective: This query identifies the top 10 products generating the highest revenue for Northwind. Understanding which products contribute the most to revenue helps the company optimize inventory, marketing efforts, and pricing strategies

```
--High Revenue Value: Identify the top 10 revenue generator products
SELECT
    p.productid,
    p.productname,
    SUM(od.UnitPrice * od.Quantity * (1 - od.Discount)) AS Revenue
FROM
    Products p
INNER JOIN
    "Order Details" od
ON
    p.ProductID=od.ProductID
GROUP BY 1,2
ORDER BY 3 DESC
LIMIT 10
```



High Revenue Products

Logic

1. ***Calculate Revenue:***

- The query calculates total revenue for each product
- This computation ensures that revenue reflects actual sales after discounts.

2. ***Joining Tables:***

- The Products table is joined with the "Order Details" table using the ProductID field to link products with their order details.

3. ***Grouping and Summarizing:***

- Revenue is summed (SUM) for each product (GROUP BY ProductID, ProductName) to calculate total revenue across all orders.

4. ***Sorting and Limiting:***

- Results are sorted in descending order by revenue (ORDER BY 3 DESC) to prioritize the highest-earning products.
- The LIMIT 10 clause restricts the output to the top 10 products.

High Revenue Products

Results

- The query produces a list of the top 10 revenue-generating products with the following columns:
 - ProductID: Unique identifier for the product.
 - ProductName: Name of the product.
 - Revenue: Total revenue generated by the product.

Recommendations

- Inventory Management:
 - Prioritize stocking and ensuring availability of these high-revenue products to prevent lost sales opportunities.
- Targeted Marketing:
 - Focus promotional efforts on these products to drive even higher sales.
- Product Development:
 - Analyze attributes of these products (e.g., pricing, demand, quality) to replicate success in other product lines.

High Sales Volume Products

Objective

This query identifies the top 10 products based on the frequency of distinct orders they appear in. By determining which products are ordered most often, Northwind can focus on ensuring these products are always in stock and marketed effectively.

```
--High Sales Volume: Determine the top 10 most frequently ordered products.  
SELECT  
    p.ProductName,  
    COUNT(DISTINCT orderid) OrderFrequency  
FROM  
    "Order Details" od  
INNER JOIN  
    Products p ON od.ProductID = p.ProductID  
GROUP BY  
    p.ProductName  
ORDER BY  
    OrderFrequency DESC  
LIMIT 10;
```

High Sales Volume Products

Logic

1. Counting Distinct Orders:

- The query counts the number of distinct orders (COUNT(DISTINCT orderid)) in which each product appears. This gives a measure of how frequently each product is ordered, regardless of the quantity in each order.

2. Joining Tables:

- The Order Details table is joined with the Products table using the ProductID field to link order details to their respective products.

3. Grouping and Summarizing:

- Products are grouped by their names (GROUP BY p.ProductName) to calculate order frequency for each product.

4. Sorting and Limiting:

- Results are sorted in descending order by order frequency (ORDER BY OrderFrequency DESC) to prioritize the most frequently ordered products.
- The LIMIT 10 clause restricts the output to the top 10 products.

High Sales Volume Products

Results

- The query outputs a table with:
 - ProductName: Name of the product.
 - OrderFrequency: Number of distinct orders in which the product appears.

Recommendations

- Inventory Management:
 - Ensure adequate stock levels for these frequently ordered products to meet customer demand and avoid stockouts.
- Marketing and Promotions:
 - Promote these products more aggressively, as they are already popular with customers.
- Product Analysis:
 - Analyze the features or appeal of these products to replicate their success in other product lines.

Slow Movers

Objective

This query identifies the 5 products with the lowest order frequency. By analyzing these slow-moving products.

```
--Slow Movers: Identify products with low sales volume
SELECT
    p.ProductID,
    p.ProductName,
    COUNT(DISTINCTorderid) OrderFrequency
FROM "Order Details" od
JOIN Products p ON od.ProductID = p.ProductID
GROUP BY p.ProductID, p.ProductName
ORDER BY OrderFrequency ASC
LIMIT 5;
```

Slow Movers

Logic

1. Counting Distinct Orders:

- The query calculates the number of distinct orders (COUNT(DISTINCT orderid)) in which each product appears. This indicates how often each product is ordered, regardless of quantity.

2. Joining Tables:

- The Order Details table is joined with the Products table using the ProductID field to associate order details with their respective products.

3. Grouping and Summarizing:

- Products are grouped by their unique identifiers and names (GROUP BY p.ProductID, p.ProductName) to calculate the order frequency for each product.

4. Sorting and Limiting:

- Results are sorted in ascending order by order frequency (ORDER BY OrderFrequency ASC) to prioritize the least frequently ordered products.
- The LIMIT 5 clause restricts the output to the 5 products with the lowest order frequency.

Slow Movers

Recommendations

- Inventory Management:
 - Review stock levels and consider reducing inventory for these slow-moving products to minimize holding costs.
- Marketing Strategy:
 - Investigate reasons for low demand and consider promotional campaigns to increase visibility and sales of these products.
- Product Lifecycle Decisions:
 - Evaluate whether to discontinue these products if they consistently underperform, reallocating resources to more popular items.

ORDER ANALYSIS



Seasonality : Monthly Revenue Trends

Objective

This query analyzes the monthly order trends by counting the number of orders placed in each month. It provides insights into seasonality or long-term order patterns, helping Northwind plan inventory and marketing campaigns effectively.

Logic

1. Extracting Month Information:

- strftime function to extract the year and month ('%Y-%m') from the OrderDate. This groups orders by month, providing a time-based aggregation.

2. Counting Orders:

- The query counts the total number of orders (COUNT(orderid)) placed in each month.

3. Grouping and Sorting:

- Orders are grouped by the extracted month (GROUP BY 1) to calculate the monthly order count.
- Results are sorted chronologically by the month (ORDER BY Month) to ensure the data is presented in a time-ordered sequence.

• Recommendation

• Seasonal Trends:

- Identify periods of high or low demand, which can guide inventory planning, staffing, and promotional efforts.

• Sales Forecasting:

- Use historical monthly trends to predict future order volumes and prepare accordingly.

• Marketing Strategies:

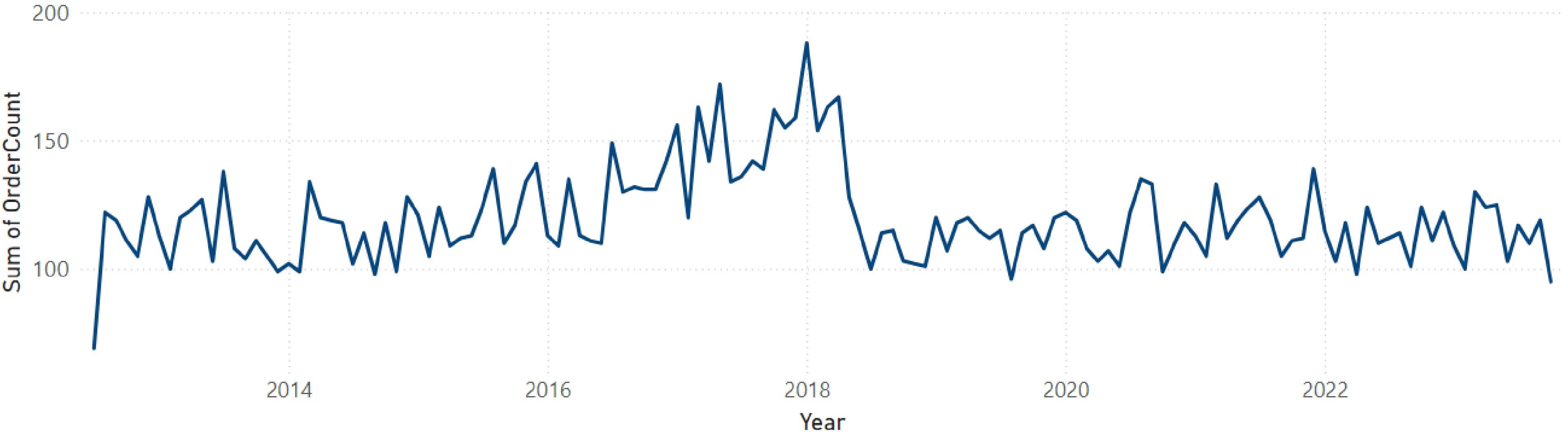
- Target low-demand months with special promotions to increase order volume.

```
--Seasonality: Identify any seasonal fluctuations
SELECT
    strftime('%Y-%m', OrderDate) AS Month,
    COUNT(DISTINCT orderid) OrderCount
FROM Orders
GROUP BY 1
ORDER BY Month;
```

Seasonality : Monthly Revenue Trends



seasonal fluctuations in order volume.



Day-of-the-Week Analysis

Objective

This query analyzes the distribution of orders across the days of the week. Understanding which days are most popular for placing orders

```
--Day-of-the-Week Analysis: Determine the most popular order days.
SELECT
  CASE
    WHEN strftime('%w', OrderDate) = '0' THEN 'Sunday'
    WHEN strftime('%w', OrderDate) = '1' THEN 'Monday'
    WHEN strftime('%w', OrderDate) = '2' THEN 'Tuesday'
    WHEN strftime('%w', OrderDate) = '3' THEN 'Wednesday'
    WHEN strftime('%w', OrderDate) = '4' THEN 'Thursday'
    WHEN strftime('%w', OrderDate) = '5' THEN 'Friday'
    WHEN strftime('%w', OrderDate) = '6' THEN 'Saturday'
  END AS DayOfWeekName,
  COUNT(DISTINCT orderid) OrderCount
FROM
  Orders
GROUP BY 1
```

Day-of-the-Week Analysis

Logic

1. Identifying Day of the Week:

- The query uses the strftime function with the %w format specifier to extract the weekday as a numerical value (0 = Sunday, 1 = Monday, ..., 6 = Saturday) from the OrderDate.
- A CASE statement maps these numerical values to their corresponding day names.

2. Counting Orders:

- The query counts the number of distinct orders (COUNT(DISTINCT orderid)) placed on each day of the week.

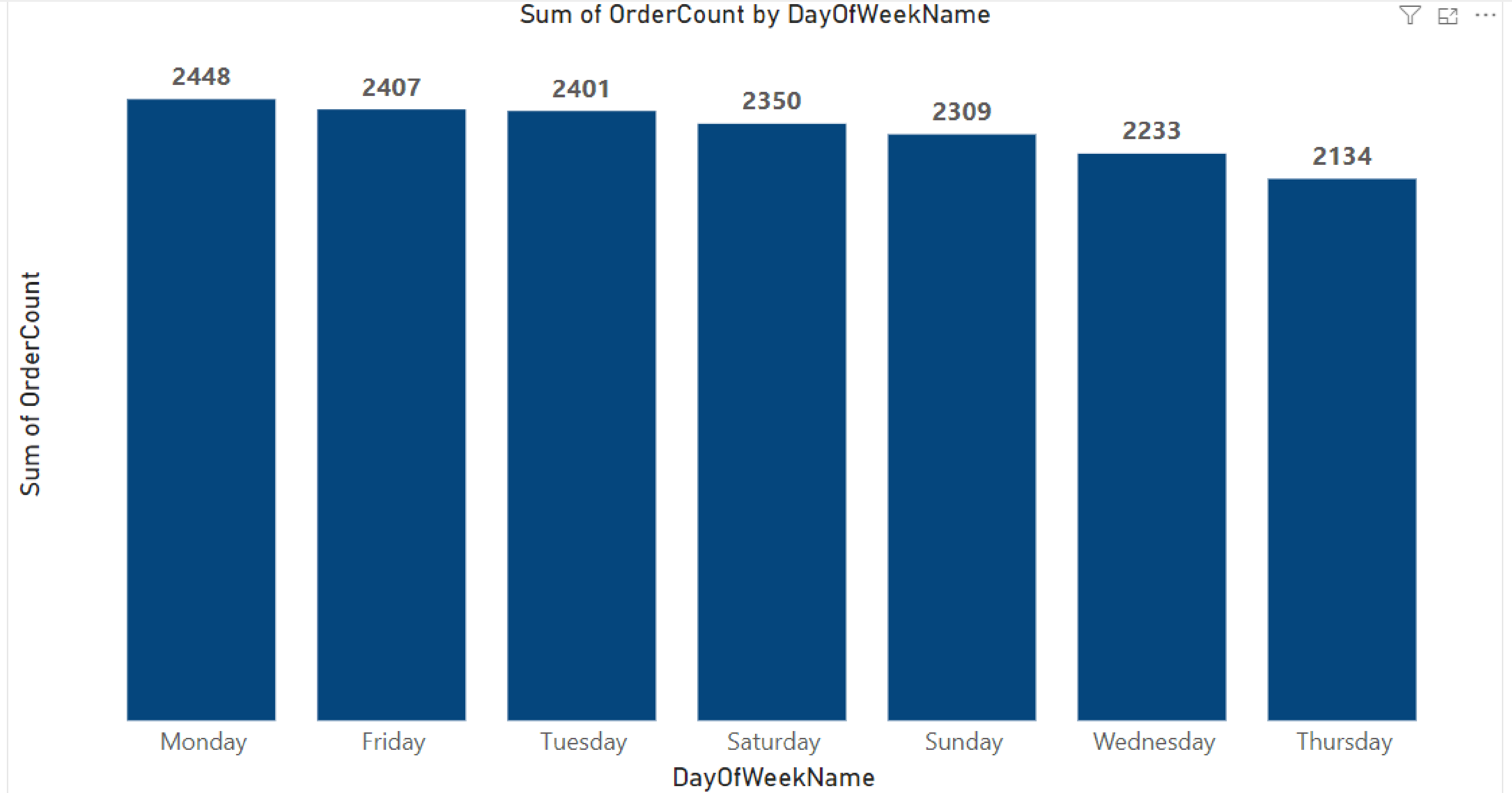
3. Grouping Results:

- Results are grouped by the day name (GROUP BY 1) to calculate the order count for each day.

Recommendation

- Operational Planning:
 - Allocate resources like staff and inventory based on the busiest days.
- Promotional Campaigns:
 - Target promotions or discounts on slower days to balance the workload and increase order volume.
- Customer Behavior Insights:
 - Understand customer purchasing habits and tailor marketing strategies accordingly.

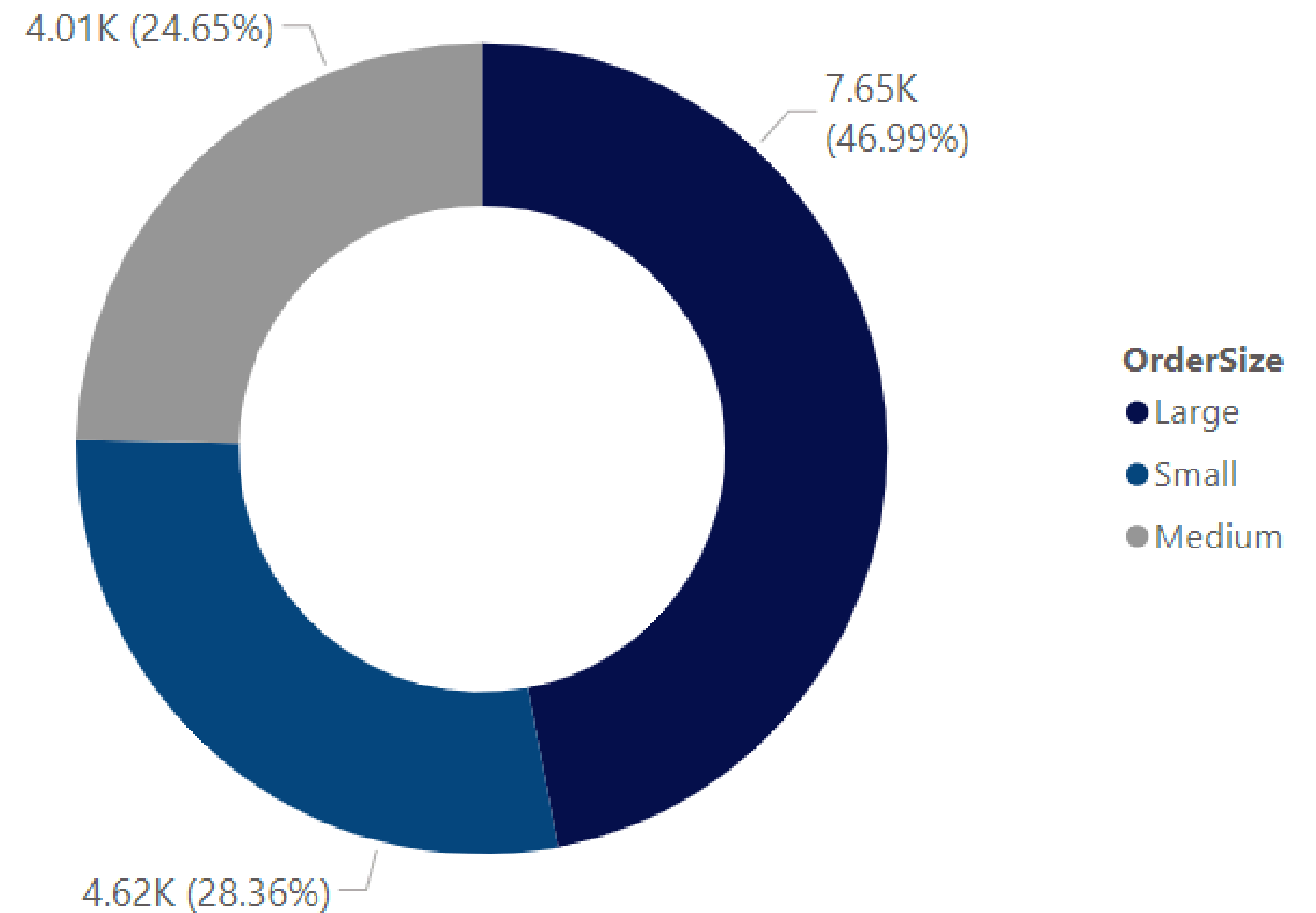
Day-of-the-Week Analysis



Analyze the distribution of order quantities

```
SELECT
 orderid,
  SUM(quantity) OrderQty,
  CASE
    WHEN SUM(Quantity) <= 500 THEN 'Small'
    WHEN SUM(Quantity) BETWEEN 501 AND 1000 THEN 'Medium'
    ELSE 'Large'
  END AS OrderSize,
  COUNT(OrderID) AS NumberOfOrders
FROM "Order Details"
GROUP BY 1
ORDER BY 2 DESC
```

Count of QuantityDistribution



Analyze the distribution of order quantities



Objective

To categorize each order based on the total quantity of items (OrderQty) into size-based segments: Small, Medium, and Large. This classification helps identify order patterns and understand customer purchasing behavior.

Logic

- Calculate Total Quantity per Order (OrderQty):
 - For each OrderID, sum up the Quantity of items from the Order Details table.
- Classify Orders by Size (OrderSize):
 - Use a CASE statement to assign each order to one of the following categories based on its total quantity:
 - Small: Orders with total quantity ≤ 500 .
 - Medium: Orders with total quantity between 501 and 1000.
 - Large: Orders with total quantity > 1000 .
- Count the Number of Orders (NumberOfOrders):
 - For each size category, count how many orders fall into it.
- Grouping and Sorting:
 - Group By OrderID:
 - Each OrderID is treated as a distinct entity, and calculations are performed on a per-order basis.
 - Sort Results by Total Quantity:
 - The results are ordered by OrderQty in descending order to highlight the largest orders first.

Analyze the distribution of order quantities

Recommendations

1. Small Orders (≤ 500 items):

- Encourage bulk purchases with discounts or free shipping.
- Offer product bundles to increase order size.

2. Medium Orders (501–1000 items):

- Upsell complementary products to push orders into the "Large" category.
- Implement loyalty programs to promote repeat purchases.

3. Large Orders (> 1000 items):

- Provide personalized offers and priority service for high-value customers.
- Secure long-term contracts with frequent large-order customers.

EMPLOYEE PERFORMANCE



Total Revenue Generated.

Objective

This query analyzes the revenue generated by employees, providing insights into the performance of employees in terms of sales. The analysis helps in understanding which employees contribute the most to overall sales and aids in incentive planning and resource allocation.

```
SELECT
    Emp.EmployeeID,
    CONCAT(firstname, ' ', lastname) EmployeeName,
    SUM(od.UnitPrice * od.Quantity * (1 - od.Discount)) AS Revenue
FROM
    Employees Emp
INNER JOIN
    Orders o
ON
    Emp.EmployeeID=o.EmployeeID
INNER JOIN
    "Order Details" od
ON
    od.OrderID=o.OrderID
GROUP BY 1,2
ORDER BY 3 DESC
```

Total Revenue Generated.

Logic

1. Aggregate Employee Revenue:

- The query calculates the total revenue for each employee by multiplying the UnitPrice, Quantity, and (1 - Discount) for each item in the Order Details. The SUM function aggregates the total revenue per employee.

2. Join Tables:

- The Employees table is joined with the Orders table to link each order to an employee. The Order Details table is then joined to capture the specific details (e.g., price, quantity, discount) of each order.

3. Group by Employee:

- The query groups the data by EmployeeID and EmployeeName, which ensures that each row corresponds to a unique employee and their respective total revenue.

4. Sort Results:

- The results are sorted in descending order of revenue to highlight the top-performing employees.

Total Revenue Generated.

Recommendations

- Employee Performance Evaluation: By analyzing the revenue contribution of each employee, the business can assess individual performance and tailor incentive programs accordingly.
- Sales Strategy: Identifying high-revenue employees can help in understanding best sales practices that can be shared across teams.

Total Sales Volume (Number of orders processed)

Objective

This query analyzes the number of distinct orders handled by each employee. It helps identify how actively employees are involved in processing orders.

```
SELECT
    Emp.EmployeeID,
    CONCAT(firstname, ' ', lastname) EmployeeName,
    COUNT(DISTINCT orderid) NumberOfOrders
FROM
    Employees Emp
INNER JOIN
    Orders o
ON
    Emp.EmployeeID = o.EmployeeID
GROUP BY 1,2
ORDER BY 3 DESC
```

Logic

1. Count Distinct Orders:

- The query counts the number of unique orders (COUNT(DISTINCT orderid)) processed by each employee. This is done by joining the Employees and Orders tables based on the EmployeeID and grouping the data by EmployeeID and EmployeeName.

2. Join Employees and Orders Tables:

- The Employees table is joined with the Orders table on the EmployeeID to associate each order with the employee who processed it.

3. Group by Employee:

- The query groups the data by EmployeeID and EmployeeName to ensure that the result is aggregated per employee.


4. Sort by Number of Orders:

- The result is ordered by the number of distinct orders in descending order to highlight the employees with the highest number of orders.



Total Sales Volume (Number of orders processed)

Business Implications

- Employee Performance Monitoring: This analysis helps assess how many orders each employee is processing. Employees with a higher number of orders may be more efficient or handling more customer requests.
 - Workload Distribution: Understanding the order distribution among employees helps in balancing workloads and identifying potential areas for improvement or support.
 - Operational Insights: This data can provide insights into the operational capacity of employees and help in workforce planning.
- 

Average order value.

Objective

This query aims to analyze the average order value and the number of orders processed by each employee. This analysis helps in understanding the employee's contribution in terms of revenue generated per order, which can assist in performance evaluation and resource allocation.

```
SELECT
    Emp.EmployeeID,
    CONCAT(firstname, ' ', lastname) EmployeeName,
    AVG(od.UnitPrice * od.Quantity * (1 - od.Discount)) AS AverageOrderValue,
    COUNT(DISTINCT o.orderid) NumberOfOrders
FROM
    Employees Emp
INNER JOIN
    Orders o
ON
    Emp.EmployeeID=o.EmployeeID
INNER JOIN
    "Order Details" od
ON
    od.OrderID=o.OrderID
GROUP BY 1,2
ORDER BY 3 DESC
```


Average order value.

Logic

1. Calculating Average Order Value:

- The query calculates the average order value by multiplying the unit price by the quantity and applying the discount, then averaging it across all orders processed by each employee.

2. Joining Tables:

- The Employees table is joined with the Orders table on EmployeeID to link employees with the orders they processed.
- The Orders table is further joined with the Order Details table on OrderID to get details like unit price, quantity, and discount for each order.

3. Grouping by Employee:

- The data is grouped by EmployeeID and EmployeeName to calculate the metrics for each employee.

4. Sorting:

- The results are sorted by average order value in descending order (ORDER BY 3 DESC) to highlight employees with the highest average revenue per order.



Average order value.

Business Implications

- Employee Performance Evaluation: This analysis helps identify employees who are generating higher revenue per order, which can be used for performance evaluation and bonus allocation.
- Sales Strategy: Employees with a higher average order value might be handling high-value customers or products. This insight can guide sales strategies and training.
- Workload and Revenue Insights: By analyzing both the number of orders processed and the average value per order, businesses can gain insights into how efficiently employees are generating revenue.

THANK YOU

