PROJECT REPORT

Retail Sales and Customer Insights Analysis

Prepared by:

Madiha Khan

TABLE OF CONTENTS

INTRODUCTION

In the fast-paced world of retail, understanding sales performance, customer behavior, and managing inventory effectively are essential for business success. This project, "Retail Sales and Customer Insights Analysis," focuses on analyzing data from a retail company to uncover valuable insights that can help drive smarter business decisions. By using advanced SQL techniques such as Common Table Expressions (CTEs), window functions, subqueries, joins, and aggregation functions, we'll explore key areas like sales trends, customer segments, and product performance.

The goal of this report is to give the retail company a clear view of its performance, highlighting trends and patterns that can guide decisions on everything from inventory management to customer engagement. Through this analysis, we'll showcase how SQL can be used to turn raw data into meaningful insights that help businesses stay ahead of the curve.

DATA OVERVIEW

The dataset used in this analysis consists of several tables that provide comprehensive information about the operations of a retail company. These tables include:

1. **Customers**: This table contains details about the company's customers, including their personal information, registration dates, and purchase history.
2. **Products:** Information about the products sold by the company, including product names, categories, prices, and stock quantities.
3. **Sales:** The sales table records transaction data, detailing customer purchases, product quantities sold, sale dates, and the discounts applied during each sale.
4. **Inventory Movements**: This table tracks inventory changes, whether products are restocked or sold, and the quantity of products moved in and out of stock.

These tables provide the foundation for uncovering insights into sales performance, customer behavior, and inventory management, allowing for detailed analysis of key business metrics.

QUESTIONS & ANALYSIS

01 Total Sales Per Month

Calculate the total sales amount per month, including the number of units sold and the total revenue generated.

```
SELECT
    DATE_FORMAT(sale_date, '%Y-%m') AS sale_month,
    SUM(quantity_sold) AS units_sold,
    ROUND(SUM(total_amount),2) AS revenue_generated
FROM
    sales_data
GROUP BY sale_month
ORDER BY sale_month;
```

sale_month	units_sold	revenue_generated
2023-11	115	22203.43
2023-12	140	30448.4
2024-01	123	22347.63
2024-02	80	15681.57
2024-03	121	24974.2
2024-04	106	24704.07
2024-05	151	32691.2
2024-06	135	27238.63
2024-07	138	32316.64
2024-08	124	30544.39
2024-09	111	21681.36
2024-10	125	30086.27
2024-11	27	6010.65

This query provides insights into the monthly sales performance, showing the total number of units sold and the total revenue generated. It helps in understanding the sales dynamics and enables the company to make data-driven decisions related to inventory management and sales forecasting.

02 Average Discount Per Month

Calculate the average discount applied to sales in each month and assess how discounting strategies impact total sales.

```
SELECT
    DATE_FORMAT(sale_date, '%Y-%m') AS sale_month,
    AVG(discount_applied) AS average_discount,
    ROUND(SUM(total_amount),2) AS total_sales
FROM
    sales_data
GROUP BY sale_month
ORDER BY sale_month;
```

sale_month	average_discount	total_sales
2023-11	10.9091	22203.43
2023-12	9.8000	30448.4
2024-01	9.8684	22347.63
2024-02	11.8519	15681.57
2024-03	9.5122	24974.2
2024-04	10.1316	24704.07
2024-05	9.3396	32691.2
2024-06	8.1818	27238.63
2024-07	11.0000	32316.64
2024-08	11.4634	30544.39
2024-09	9.5238	21681.36
2024-10	10.3947	30086.27
2024-11	9.0000	6010.65

This query helps assess the impact of discounting strategies by calculating the average discount and total sales per month. It provides insights into how discounts influence sales performance, helping the company optimize its pricing strategy.

03 Identifying High-Value Customers

Which customers have spent the most on their purchases? Show their details

```sql
WITH customer_spending as(
SELECT
    c.customer_id, CONCAT(c.first_name, ' ', c.last_name) AS full_name,
    c.email, c.gender,
    SUM(s.total_amount) AS total_spending
FROM customers_data c
JOIN sales_data s
ON c.customer_id = s.customer_id
GROUP BY c.customer_id, c.first_name, c.last_name, c.email, c.gender )

select customer_id, full_name, email, gender,
ROUND(total_spending,2) as total_spending
FROM customer_spending
ORDER BY total_spending DESC
LIMIT 10;       -- using Limit to see the top 10 customers
```

customer_id	full_name	email	gender	total_spending
94	FirstName94 LastName94	customer94@example.com	Female	7880.24
100	FirstName100 LastName100	customer100@example.com	Female	7808.09
92	FirstName92 LastName92	customer92@example.com	Male	7486.66
7	FirstName7 LastName7	customer7@example.com	Male	6712.69
16	FirstName16 LastName16	customer16@example.com	Male	6571.85
60	FirstName60 LastName60	customer60@example.com	Female	6300.79
17	FirstName17 LastName17	customer17@example.com	Female	6177.61
95	FirstName95 LastName95	customer95@example.com	Female	6169.84
2	FirstName2 LastName2	customer2@example.com	Female	6059.56
28	FirstName28 LastName28	customer28@example.com	Male	5992.74

This query identifies the top 10 customers based on their total spending. It provides insights into customer behavior, allowing the company to recognize high-value customers. This information can be used for targeted marketing, loyalty programs, and personalized offers to retain these valuable customers.

04 Identify the Oldest Customers

Find the details of customers born in the 1990s, including their total spending and specific order details.

```sql
SELECT
    c.customer_id,
    CONCAT(c.first_name, ' ', c.last_name) AS full_name,
    c.email,
    c.date_of_birth,
    ROUND(SUM(s.total_amount),2) AS total_spending
FROM
    customers_data c
        JOIN
    sales_data s ON c.customer_id = s.customer_id
WHERE
    c.date_of_birth BETWEEN '1990-01-01' AND '1999-12-31'
GROUP BY
    c.customer_id, c.first_name, c.last_name, c.email, c.date_of_birth;
```

customer_id	full_name	email	date_of_birth	total_spending
2	FirstName2 LastName2	customer2@example.com	1991-08-04	6059.56
6	FirstName6 LastName6	customer6@example.com	1992-09-04	3467.22
10	FirstName10 LastName10	customer10@example.com	1997-05-21	4519.44
16	FirstName16 LastName16	customer16@example.com	1999-07-01	6571.85
17	FirstName17 LastName17	customer17@example.com	1990-02-12	6177.61
24	FirstName24 LastName24	customer24@example.com	1994-03-28	3548.08
26	FirstName26 LastName26	customer26@example.com	1990-11-30	3903.8
27	FirstName27 LastName27	customer27@example.com	1995-06-03	4777.3
32	FirstName32 LastName32	customer32@example.com	1999-01-22	763.8
37	FirstName37 LastName37	customer37@example.com	1994-05-02	2202.22
44	FirstName44 LastName44	customer44@example.com	1995-11-06	3770.69
45	FirstName45 LastName45	customer45@example.com	1996-12-05	2261.47
53	FirstName53 LastName53	customer53@example.com	1997-10-13	2283.53
60	FirstName60 LastName60	customer60@example.com	1994-06-14	6300.79
63	FirstName63 LastName63	customer63@example.com	1994-05-11	2119.8
82	FirstName82 LastName82	customer82@example.com	1995-03-31	1017.56
85	FirstName85 LastName85	customer85@example.com	1995-08-15	3822.25
89	FirstName89 LastName89	customer89@example.com	1993-06-29	3226.09

This query identifies customers born in the 1990s, along with their total spending and personal details. It helps the company analyze purchasing behaviors of older customer segments, enabling more targeted marketing strategies and tailored engagement efforts.

05 Customer Segmentation

Use SQL to create customer segments based on their total spending (e.g., Low Spenders, High Spenders)

```sql
WITH customer_spending AS (
    SELECT c.customer_id,
        CONCAT(c.first_name, ' ', c.last_name) AS full_name, c.email,
        ROUND(SUM(s.total_amount), 2) AS total_spending
    FROM customers_data c
        JOIN sales_data s ON c.customer_id = s.customer_id
    GROUP BY c.customer_id, c.first_name, c.last_name, c.email )
SELECT *,
    CASE
        WHEN total_spending <= 1500 THEN 'Low Spender'
        WHEN total_spending <= 4000 THEN 'Medium Spender'
        ELSE 'High Spender'
    END AS Type_of_Spender
FROM customer_spending
ORDER BY customer_id;
```

customer_id	full_name	email	total_spending	Type_of_Spender
1	FirstName1 LastName1	customer1@example.com	4310.44	High Spender
2	FirstName2 LastName2	customer2@example.com	6059.56	High Spender
3	FirstName3 LastName3	customer3@example.com	3913.19	Medium Spender
4	FirstName4 LastName4	customer4@example.com	2152.5	Medium Spender
5	FirstName5 LastName5	customer5@example.com	5336.57	High Spender
6	FirstName6 LastName6	customer6@example.com	3467.22	Medium Spender
7	FirstName7 LastName7	customer7@example.com	6712.69	High Spender
8	FirstName8 LastName8	customer8@example.com	1710.2	Medium Spender
10	FirstName10 LastName10	customer10@example.com	4519.44	High Spender
11	FirstName11 LastName11	customer11@example.com	2406.66	Medium Spender
12	FirstName12 LastName12	customer12@example.com	3845.82	Medium Spender
13	FirstName13 LastName13	customer13@example.com	2029.05	Medium Spender
14	FirstName14 LastName14	customer14@example.com	3937.01	Medium Spender
15	FirstName15 LastName15	customer15@example.com	2657.65	Medium Spender
16	FirstName16 LastName16	customer16@example.com	6571.85	High Spender
17	FirstName17 LastName17	customer17@example.com	6177.61	High Spender
18	FirstName18 LastName18	customer18@example.com	4877.24	High Spender
19	FirstName19 LastName19	customer19@example.com	4006.99	High Spender

28	FirstName28 LastName28	customer28@example.com	5992.74	High Spender
29	FirstName29 LastName29	customer29@example.com	838.96	Low Spender
30	FirstName30 LastName30	customer30@example.com	2855.05	Medium Spender
31	FirstName31 LastName31	customer31@example.com	5188.22	High Spender
32	FirstName32 LastName32	customer32@example.com	763.8	Low Spender
33	FirstName33 LastName33	customer33@example.com	5406.79	High Spender
34	FirstName34 LastName34	customer34@example.com	3971.12	Medium Spender
35	FirstName35 LastName35	customer35@example.com	5656.12	High Spender
36	FirstName36 LastName36	customer36@example.com	2665.24	Medium Spender
37	FirstName37 LastName37	customer37@example.com	2202.22	Medium Spender
38	FirstName38 LastName38	customer38@example.com	4649.1	High Spender
39	FirstName39 LastName39	customer39@example.com	4162.11	High Spender
40	FirstName40 LastName40	customer40@example.com	177.9	Low Spender

This query segments customers into "Low Spenders," "Medium Spenders," and "High Spenders" based on their total spending. It provides valuable insights into customer behavior, enabling the company to tailor marketing strategies and offers for different customer groups effectively.

06 Stock Management

Write a query to find products that are running low in stock (below a threshold like 10 units) and recommend restocking amounts based on past sales performance.

```
WITH average_sales AS (
    SELECT
        p.product_id, p.product_name, p.category, p.stock_quantity,
        AVG(s.quantity_sold) AS avg_sales_per_day
    FROM products_data p
    JOIN sales_data s
    ON p.product_id = s.product_id
    GROUP BY p.product_id, p.product_name, p.category, p.stock_quantity)
```

```sql
SELECT
    product_id, product_name, category, stock_quantity,
    CASE
        WHEN stock_quantity <= 10 THEN "Low in stock"
        ELSE "Sufficient stock"
    END AS restocking_information,
    ROUND(avg_sales_per_day * 7) AS recommended_restocking_amount -- Assuming one week of inventory
FROM
    average_sales
ORDER BY
    stock_quantity;
```

product_id	product_name	category	stock_quantity	restocking_information	recommended_restocking_amount
16	Product16	Home	7	Low in stock	22
12	Product12	Sports	8	Low in stock	17
41	Product41	Clothing	9	Low in stock	19
15	Product15	Electronics	10	Low in stock	21
39	Product39	Sports	12	Sufficient stock	21
24	Product24	Sports	13	Sufficient stock	21
17	Product17	Clothing	14	Sufficient stock	25
1	Product1	Home	15	Sufficient stock	20
29	Product29	Sports	18	Sufficient stock	25
28	Product28	Home	18	Sufficient stock	25
3	Product3	Sports	19	Sufficient stock	23
32	Product32	Sports	19	Sufficient stock	22
6	Product6	Sports	20	Sufficient stock	24
37	Product37	Electronics	21	Sufficient stock	14
36	Product36	Sports	21	Sufficient stock	18

This query identifies products with low stock levels (below 10 units) and categorizes them for restocking purposes. It helps ensure inventory is maintained, preventing stockouts and supporting smooth operations. It also recommends restocking amounts based on average daily sales over time. By forecasting a week's worth of inventory needs, it helps optimize stock levels, prevent shortages, and maintain operational efficiency.

07 Inventory Movements Overview

Create a report showing the daily inventory movements (restock vs. sales) for each product over a given period.

```sql
SELECT
    movement_date,
    product_id,
    SUM(CASE WHEN movement_type = 'IN' THEN quantity_moved ELSE 0 END) AS restock_quantity,
    SUM(CASE WHEN movement_type = 'OUT' THEN quantity_moved ELSE 0 END) AS sales_quantity
FROM
    inventory_movements_data
WHERE movement_date BETWEEN '2024-01-01' AND '2024-01-31'
GROUP BY
    movement_date, product_id
ORDER BY
    movement_date, product_id;
```

movement_date	product_id	restock_quantity	sales_quantity
2024-01-01	14	0	4
2024-01-01	25	0	3
2024-01-01	41	1	0
2024-01-02	1	1	0
2024-01-04	14	14	0
2024-01-05	15	0	9
2024-01-14	40	0	1
2024-01-19	40	9	9
2024-01-20	37	0	15
2024-01-22	28	3	0
2024-01-25	30	11	0
2024-01-31	9	4	0

This query shows the daily stock changes for each product, separating restocked items from sold items. It helps track inventory usage and ensures the company can manage stock effectively to avoid shortages or overstocking.

08 Rank Products

Rank products in each category by their prices

```
SELECT
    category,
    product_id,
    product_name,
    price,
    RANK() OVER (PARTITION BY category ORDER BY price DESC) AS price_rank
FROM
    products_data;
```

category	product_id	product_name	price	price_rank
Clothing	41	Product41	355.61	1
Clothing	26	Product26	329.88	2
Clothing	18	Product18	313.92	3
Clothing	17	Product17	294.81	4
Clothing	14	Product14	280.73	5
Clothing	25	Product25	235.95	6
Clothing	35	Product35	232.72	7
Clothing	2	Product2	143.95	8
Clothing	34	Product34	123.8	9
Clothing	42	Product42	68.46	10
Clothing	49	Product49	52.44	11
Electronics	13	Product13	487.12	1
Electronics	8	Product8	450.01	2
Electronics	37	Product37	418.53	3

This query ranks products within each category based on their prices, from highest to lowest. It helps identify premium products in each category, which can be useful for pricing strategies and promotional campaigns.

09 Average Order Size

What is the average order size in terms of quantity sold for each product?

```
SELECT
    product_id,
    AVG(quantity_sold) AS average_order_size
FROM
    sales_data
GROUP BY
    product_id;
```

product_id	average_order_size
1	2.8571
2	2.1429
3	3.3077
4	2.7692
5	2.7143
6	3.3846
7	4.2500
8	3.2727
9	3.0000
10	2.9286
11	2.2143
12	2.4286
13	3.6000

This query calculates the average number of units sold for each product. It helps the company understand the typical order size for each item, which can assist in inventory planning and sales forecasting. By knowing the average order size, the company can better manage stock levels and optimize product availability.

10 Recent Restock Product

Which products have seen the most recent restocks

```
SELECT
    p.product_id, p.product_name, p.category,
    MAX(m.movement_date) AS last_restock_date
FROM Inventory_Movements_data m
join products_data p
on p.product_id = m.product_id
WHERE movement_type = 'IN'
GROUP BY p.product_id
ORDER BY last_restock_date DESC
LIMIT 10;
```

product_id	product_name	category	last_restock_date
24	Product24	Sports	2024-11-01
22	Product22	Sports	2024-10-09
40	Product40	Electronics	2024-10-06
5	Product5	Sports	2024-09-27
1	Product1	Home	2024-09-26
13	Product13	Electronics	2024-09-17
46	Product46	Home	2024-09-14
31	Product31	Sports	2024-09-13
20	Product20	Electronics	2024-09-03
44	Product44	Electronics	2024-08-29

This query shows which products were restocked most recently. It helps the company keep track of which items are being restocked and ensures popular products are available for customers. This information can help with inventory planning and stocking decisions.

Challenges:

Dynamic Pricing Simulation

Challenge students to analyze how price changes for products impact sales volume, revenue, and customer behavior

```
WITH price_impact AS (
    SELECT s.product_id, DATE_FORMAT(s.sale_date, '%Y-%m') AS sale_month,
    AVG(p.price) AS avg_price, SUM(s.quantity_sold) AS total_units_sold,
    SUM(s.total_amount) AS total_revenue
    FROM sales_data s
    JOIN products_data p
    ON s.product_id = p.product_id
    GROUP BY s.product_id, DATE_FORMAT(s.sale_date, '%Y-%m')


Analysis AS (
    SELECT product_id, sale_month, avg_price, total_units_sold, total_revenue,
    CASE
    WHEN total_units_sold > 0 THEN ROUND(total_revenue / total_units_sold, 2)
    ELSE 0
    END AS revenue_per_unit
    FROM price_impact
)
SELECT *
FROM Analysis
ORDER BY product_id, sale_month;
```

product_id	sale_month	avg_price	total_units_sold	total_revenue	revenue_per_unit
1	2023-11	429.6	5	2040.6	408.12
1	2023-12	429.6	1	429.6	429.6
1	2024-01	429.6	9	3393.84	377.09
1	2024-04	429.6	1	386.64	386.64
1	2024-05	429.6	3	1095.48	365.16
1	2024-06	429.6	5	2083.56	416.71
1	2024-07	429.6	4	1460.64	365.16
1	2024-08	429.6000000000001	8	2792.3999999999996	349.05
1	2024-09	429.6	2	773.28	386.64
1	2024-11	429.6	2	773.28	386.64
2	2024-03	143.95	2	244.71	122.36
2	2024-04	143.95	1	129.56	129.56
2	2024-05	143.95	1	136.75	136.75
2	2024-06	143.95	5	683.76	136.75

This query analyzes the impact of product pricing on sales performance by calculating the average price, total units sold, and revenue per unit for each product on a monthly basis. It provides insights into how pricing strategies influence customer purchasing behavior and revenue generation. By examining trends in revenue per unit and total units sold, the company can optimize pricing decisions to maximize sales and profitability.

Customer Purchase Pattern

Analyze purchase patterns using time-series data and window functions to find high-frequency buying behavior

```sql
WITH Purchase_Patterns AS (
    SELECT
        c.customer_id,
        CONCAT(c.first_name, ' ', c.last_name) AS full_name,
        DATE_FORMAT(s.sale_date, '%Y-%m') AS Purchase_month,
        COUNT(s.sale_id) AS Purchase_count,
        SUM(s.total_amount) AS Spent_amount,
        ROW_NUMBER() OVER (PARTITION BY c.customer_id ORDER BY SUM(s.total_amount) DESC) AS customer_rank
    FROM customers_data c
    JOIN sales_data s
    ON c.customer_id = s.customer_id
    GROUP BY c.customer_id, full_name, Purchase_month
)

SELECT
    customer_id, full_name, Purchase_month,
    Purchase_count, Spent_amount
FROM  Purchase_Patterns
WHERE customer_rank = 1
ORDER BY Purchase_count DESC, Spent_amount DESC
LIMIT 10;
```

customer_id	full_name	Purchase_month	Purchase_count	Spent_amount
94	FirstName94 LastName94	2024-10	4	6395.280000000001
83	FirstName83 LastName83	2024-08	3	3164.2799999999997
2	FirstName2 LastName2	2024-03	3	2961.39
5	FirstName5 LastName5	2024-07	3	2683.41
35	FirstName35 LastName35	2024-06	3	2625.95
100	FirstName100 LastName100	2024-07	3	2365.54
39	FirstName39 LastName39	2023-12	3	2323.92
19	FirstName19 LastName19	2024-10	3	2266.16
24	FirstName24 LastName24	2023-12	3	2038.38
70	FirstName70 LastName70	2024-05	3	2019.4

This query identifies customers with high-frequency buying behavior by analyzing their monthly purchase patterns. It calculates the total number of purchases and the amount spent by each customer in their most active month. By highlighting the top 10 customers with the highest purchase counts and spending, this analysis helps the company understand its most engaged and valuable customers. These insights can be used to develop targeted loyalty programs, personalized offers, and strategies to retain high-value customers.

Predictive Analysis

Use past data to predict which customers are most likely to churn and recommend strategies to retain them.

```
WITH customer_activity AS (
    SELECT
        c.customer_id, MAX(s.sale_date) AS last_purchase_date,
        COUNT(s.sale_id) AS total_orders, SUM(s.total_amount) AS total_spent
    FROM customers_data c
    LEFT JOIN sales_data s ON c.customer_id = s.customer_id
    GROUP BY c.customer_id
),

churn_prediction AS (
    SELECT
        customer_id,
        DATEDIFF(CURDATE(), last_purchase_date) AS days_since_last_purchase,
        total_orders, total_spent,
        CASE
            WHEN DATEDIFF(CURDATE(), last_purchase_date) > 180 THEN 'High Risk'
            WHEN DATEDIFF(CURDATE(), last_purchase_date) BETWEEN 90 AND 180 THEN 'Moderate Risk'
            ELSE 'Low Risk'
        END AS churn_risk
    FROM customer_activity

SELECT
    customer_id, days_since_last_purchase,
    total_orders, total_spent, churn_risk
FROM churn_prediction
ORDER BY churn_risk DESC, total_spent DESC;
```

customer_id	days_since_last_purchase	total_orders	total_spent	churn_risk
50	139	4	1637.02	Moderate Risk
55	130	5	1367.75	Moderate Risk
90	141	3	1355.04	Moderate Risk
81	129	3	1326.1100000000001	Moderate Risk
32	116	1	763.8	Moderate Risk
43	145	3	621.6199999999999	Moderate Risk
21	137	1	491.22	Moderate Risk
94	19	7	7880.24	Low Risk
100	51	13	7808.090000000001	Low Risk
92	38	10	7486.66	Low Risk
7	17	8	6712.6900000000005	Low Risk
16	39	8	6571.849999999999	Low Risk
60	21	8	6300.79	Low Risk

This query assesses customer churn risk by analyzing purchase activity. It calculates the number of days since each customer's last purchase, total orders placed, and total spending. Based on the inactivity period, customers are categorized into High Risk, Moderate Risk, or Low Risk for churn. The output helps focus on keeping customers by finding those who might stop using the service. This insight helps the company create strategies like special offers or loyalty programs to keep valuable customers and reduce the chances of losing them.