/\*

==========================================================

Project: Retail Sales EDA & Advanced Analytics in SQL

Author: Kalandhar

Description:

End-to-end SQL project covering data cleaning,

exploratory data analysis (EDA), advanced analytics,

segmentation, and customer-level reporting.

Database: MySQL

Notes:

- This script is organized into sections with clear

comments. Run sections independently as needed.

==========================================================

\*/

-- =========================

-- DATABASE SETUP / CLEANING

-- =========================

-- Create / use EDA database

CREATE DATABASE IF NOT EXISTS EDA;

USE EDA;

-- Example: inspect raw table

SELECT \* FROM orders\_dataset;

-- Fix date format (example): convert d-m-Y to Y-m-d

SELECT STR\_TO\_DATE(order\_date, '%d-%m-%Y') AS order\_date\_fixed

FROM orders\_dataset

LIMIT 10;

UPDATE orders\_dataset

SET order\_date = DATE\_FORMAT(STR\_TO\_DATE(order\_date, '%d-%m-%Y'), '%Y-%m-%d')

WHERE order\_date IS NOT NULL AND STR\_TO\_DATE(order\_date, '%d-%m-%Y') IS NOT NULL;

-- Rename tables to standardized names (run when safe)

RENAME TABLE customers\_dataset TO customers;

RENAME TABLE products\_dataset TO products;

RENAME TABLE orders\_dataset TO orders;

-- -------------------------

-- Basic Database Exploration

-- -------------------------

-- How many tables exist?

SHOW TABLES;

-- Describe table structures

DESCRIBE customers;

DESCRIBE orders;

DESCRIBE products;

-- Data types and nullability for a table

SELECT COLUMN\_NAME, DATA\_TYPE, IS\_NULLABLE

FROM INFORMATION\_SCHEMA.COLUMNS

WHERE TABLE\_NAME = 'orders' AND TABLE\_SCHEMA = DATABASE();

SELECT COLUMN\_NAME, DATA\_TYPE, IS\_NULLABLE

FROM INFORMATION\_SCHEMA.COLUMNS

WHERE TABLE\_NAME = 'products' AND TABLE\_SCHEMA = DATABASE();

SELECT COLUMN\_NAME, DATA\_TYPE, IS\_NULLABLE

FROM INFORMATION\_SCHEMA.COLUMNS

WHERE TABLE\_NAME = 'customers' AND TABLE\_SCHEMA = DATABASE();

-- Count NULLs per column example (order\_date)

SELECT COUNT(\*) AS null\_order\_date

FROM orders

WHERE order\_date IS NULL;

-- Check duplicate rows based on logical key (customer\_id, order\_id)

SELECT customer\_id, order\_id, COUNT(\*) AS cnt

FROM orders

GROUP BY customer\_id, order\_id

HAVING cnt > 1;

-- Show duplicate rows

SELECT \*

FROM orders

WHERE (customer\_id, order\_id) IN (

SELECT customer\_id, order\_id

FROM orders

GROUP BY customer\_id, order\_id

HAVING COUNT(\*) > 1

)

ORDER BY order\_id;

-- Show create table statements (to inspect primary/foreign keys)

SHOW CREATE TABLE orders;

SHOW CREATE TABLE customers;

SHOW CREATE TABLE products;

-- ---------------------------------------------

-- Dimensions Exploration (Categorical Columns)

-- ---------------------------------------------

-- Unique categories / regions / product ids

SELECT DISTINCT category FROM products;

SELECT DISTINCT region FROM customers;

SELECT DISTINCT product\_id FROM orders;

-- Frequency of categories (most/least frequent)

SELECT category, COUNT(\*) AS total\_sales

FROM orders

NATURAL JOIN products

GROUP BY category

ORDER BY total\_sales DESC;

-- ------------------

-- Date Exploration

-- ------------------

-- First and last order date, range in years/months

SELECT

MIN(order\_date) AS first\_order,

MAX(order\_date) AS last\_order,

TIMESTAMPDIFF(YEAR, MIN(order\_date), MAX(order\_date)) AS order\_range\_years,

TIMESTAMPDIFF(MONTH, MIN(order\_date), MAX(order\_date)) AS order\_range\_months

FROM orders;

-- Highest sales month with year

SELECT YEAR(order\_date) AS year, MONTH(order\_date) AS month, SUM(total\_amount) AS total\_sales

FROM orders

GROUP BY year, month

ORDER BY total\_sales DESC

LIMIT 1;

-- Lowest sales month with year

SELECT YEAR(order\_date) AS year, MONTH(order\_date) AS month, SUM(total\_amount) AS total\_sales

FROM orders

GROUP BY year, month

ORDER BY total\_sales

LIMIT 1;

-- -------------------------

-- Measures / KPI Exploration

-- -------------------------

-- Total sales

SELECT ROUND(SUM(total\_amount)) AS total\_sales FROM orders;

-- Total items sold

SELECT SUM(quantity) AS total\_quantity FROM orders;

-- Average product price (from products table)

SELECT ROUND(AVG(price),2) AS avg\_price FROM products;

-- Total number of orders (two variants)

SELECT COUNT(order\_id) AS total\_orders FROM orders;

SELECT COUNT(DISTINCT order\_id) AS total\_distinct\_orders FROM orders;

-- Total number of products

SELECT COUNT(product\_name) AS total\_products FROM products;

SELECT COUNT(DISTINCT product\_name) AS total\_distinct\_products FROM products;

-- Total number of customers

SELECT COUNT(customer\_id) AS total\_customers FROM customers;

-- Total number of customers who placed orders

SELECT COUNT(DISTINCT customer\_id) AS customers\_with\_orders FROM orders;

-- Consolidated KPI report

SELECT 'total\_sales' AS measure\_name, ROUND(SUM(total\_amount)) AS value FROM orders

UNION ALL

SELECT 'total\_quantity', SUM(quantity) FROM orders

UNION ALL

SELECT 'avg\_price', ROUND(AVG(price),2) FROM products

UNION ALL

SELECT 'total\_distinct\_orders', COUNT(DISTINCT order\_id) FROM orders

UNION ALL

SELECT 'total\_distinct\_products', COUNT(DISTINCT product\_name) FROM products

UNION ALL

SELECT 'total\_customers', COUNT(customer\_id) FROM customers

UNION ALL

SELECT 'customers\_with\_orders', COUNT(DISTINCT customer\_id) FROM orders;

-- ------------------------

-- Magnitude / Comparison

-- ------------------------

-- Total customers by region

SELECT region, COUNT(customer\_id) AS total\_customers

FROM customers

GROUP BY region

ORDER BY total\_customers DESC;

-- Total products by category

SELECT category, COUNT(product\_id) AS total\_products

FROM products

GROUP BY category

ORDER BY total\_products DESC;

-- Average cost (price) by category

SELECT category, ROUND(AVG(price),2) AS avg\_cost

FROM products

GROUP BY category

ORDER BY avg\_cost DESC;

-- Total revenue by category

SELECT category, ROUND(SUM(total\_amount),2) AS total\_revenue

FROM orders

NATURAL JOIN products

GROUP BY category

ORDER BY total\_revenue DESC;

-- Total revenue by customer

SELECT customer\_id, customer\_name, ROUND(SUM(total\_amount),2) AS total\_revenue

FROM customers

NATURAL JOIN orders

GROUP BY customer\_id, customer\_name

ORDER BY total\_revenue DESC;

-- Distribution of items sold across regions

SELECT region, SUM(quantity) AS total\_sold\_items

FROM customers

NATURAL JOIN orders

GROUP BY region

ORDER BY total\_sold\_items DESC;

-- ----------------

-- Ranking Analysis

-- ----------------

-- Top 5 products by revenue

SELECT product\_name, SUM(total\_amount) AS total\_revenue,

DENSE\_RANK() OVER (ORDER BY SUM(total\_amount) DESC) AS rank\_products

FROM products

NATURAL JOIN orders

GROUP BY product\_name

ORDER BY total\_revenue DESC

LIMIT 5;

-- Bottom 5 products by revenue

SELECT product\_id, SUM(total\_amount) AS total\_revenue,

DENSE\_RANK() OVER (ORDER BY SUM(total\_amount) ASC) AS rank\_products

FROM products

NATURAL JOIN orders

GROUP BY product\_id

ORDER BY total\_revenue ASC

LIMIT 5;

-- Top 10 customers by revenue

SELECT c.customer\_id, c.customer\_name, SUM(o.total\_amount) AS revenue,

RANK() OVER (ORDER BY SUM(o.total\_amount) DESC) AS ranks

FROM customers AS c

JOIN orders AS o ON c.customer\_id = o.customer\_id

GROUP BY c.customer\_id, c.customer\_name

ORDER BY revenue DESC

LIMIT 10;

-- Bottom 3 customers by number of orders

SELECT customer\_name, COUNT(order\_id) AS total\_orders,

ROW\_NUMBER() OVER (ORDER BY COUNT(order\_id)) AS ranks

FROM customers

NATURAL JOIN orders

GROUP BY customer\_name

ORDER BY total\_orders ASC

LIMIT 3;

**-- =====================**

**-- ADVANCED ANALYTICS DB**

**-- =====================**

CREATE DATABASE IF NOT EXISTS advanced\_analytics;

USE advanced\_analytics;

-- Example conversions and renames for advanced dataset (gold schema)

SELECT STR\_TO\_DATE(order\_date, '%d-%m-%Y') AS order\_date\_fixed

FROM `gold.fact\_sales`

LIMIT 10;

UPDATE `gold.fact\_sales`

SET order\_date = DATE\_FORMAT(STR\_TO\_DATE(order\_date, '%d-%m-%Y'), '%Y-%m-%d')

WHERE order\_date IS NOT NULL AND STR\_TO\_DATE(order\_date, '%d-%m-%Y') IS NOT NULL;

ALTER TABLE `gold.fact\_sales`

MODIFY COLUMN order\_date DATE;

UPDATE `gold.fact\_sales`

SET order\_date = NULL

WHERE order\_date = '' OR order\_date = '0000-00-00';

-- Rename gold schema tables to standardized names (run as needed)

RENAME TABLE `gold.dim\_products` TO products;

RENAME TABLE `gold.fact\_sales` TO sales;

RENAME TABLE `gold.dim\_customers` TO customers;

-- -------------------------

-- Changes Over Time (Trends)

-- -------------------------

-- Sales performance over years

SELECT

YEAR(order\_date) AS order\_year,

COUNT(DISTINCT customer\_key) AS total\_customer,

SUM(quantity) AS total\_quantity,

SUM(sales\_amount) AS total\_revenue

FROM sales

WHERE YEAR(order\_date) IS NOT NULL

GROUP BY YEAR(order\_date)

ORDER BY YEAR(order\_date);

-- Seasonality: month-level

SELECT

YEAR(order\_date) AS order\_year,

MONTH(order\_date) AS order\_month,

COUNT(DISTINCT customer\_key) AS total\_customer,

SUM(quantity) AS total\_quantity,

SUM(sales\_amount) AS total\_revenue

FROM sales

WHERE YEAR(order\_date) IS NOT NULL

GROUP BY YEAR(order\_date), MONTH(order\_date)

ORDER BY YEAR(order\_date), MONTH(order\_date);

-- --------------------

-- Cumulative Analysis

-- --------------------

-- Monthly total sales and running total

SELECT order\_month, total\_sales,

SUM(total\_sales) OVER (ORDER BY order\_month) AS running\_total

FROM (

SELECT DATE\_FORMAT(order\_date, '%Y-%m') AS order\_month, SUM(sales\_amount) AS total\_sales

FROM sales

WHERE YEAR(order\_date) IS NOT NULL

GROUP BY DATE\_FORMAT(order\_date, '%Y-%m')

) t

ORDER BY order\_month;

-- --------------------

-- Performance Analysis

-- --------------------

WITH monthly\_product\_sales AS (

SELECT

DATE\_FORMAT(order\_date, '%Y-%m') AS order\_month,

p.product\_name,

SUM(s.sales\_amount) AS current\_sales

FROM sales AS s

LEFT JOIN products AS p ON s.product\_key = p.product\_key

WHERE p.product\_name IS NOT NULL

GROUP BY DATE\_FORMAT(order\_date, '%Y-%m'), p.product\_name

)

SELECT

order\_month,

product\_name,

current\_sales,

AVG(current\_sales) OVER (PARTITION BY product\_name) AS avg\_sales,

current\_sales - AVG(current\_sales) OVER (PARTITION BY product\_name) AS avg\_diff,

CASE

WHEN current\_sales - AVG(current\_sales) OVER (PARTITION BY product\_name) > 0 THEN 'above avg'

WHEN current\_sales - AVG(current\_sales) OVER (PARTITION BY product\_name) < 0 THEN 'below avg'

ELSE 'avg'

END AS avg\_change,

LAG(current\_sales) OVER (PARTITION BY product\_name ORDER BY order\_month) AS pm\_sales,

CASE

WHEN current\_sales - LAG(current\_sales) OVER (PARTITION BY product\_name ORDER BY order\_month) > 0 THEN 'increasing'

WHEN current\_sales - LAG(current\_sales) OVER (PARTITION BY product\_name ORDER BY order\_month) < 0 THEN 'decreasing'

ELSE 'no change'

END AS pm\_change

FROM monthly\_product\_sales

ORDER BY product\_name, order\_month;

-- ---------------------------------

-- Part-to-Whole Analysis (Proportions)

-- ---------------------------------

WITH category\_sales AS (

SELECT p.category, SUM(s.sales\_amount) AS total\_sales

FROM products AS p

JOIN sales AS s ON p.product\_key = s.product\_key

GROUP BY p.category

)

SELECT

category,

total\_sales,

SUM(total\_sales) OVER () AS overall\_sales,

ROUND((total\_sales / SUM(total\_sales) OVER ()) \* 100, 2) AS percentage\_of\_total

FROM category\_sales

ORDER BY total\_sales DESC;

-- ------------------

-- Data Segmentation

-- ------------------

-- Segment products into cost ranges

WITH product\_segment AS (

SELECT product\_key, product\_name, cost,

CASE

WHEN cost < 100 THEN 'below 100'

WHEN cost BETWEEN 100 AND 500 THEN '100-500'

WHEN cost BETWEEN 500 AND 1000 THEN '500-1000'

ELSE 'above 1000'

END AS cost\_segment

FROM products

)

SELECT cost\_segment, COUNT(product\_name) AS count\_products

FROM product\_segment

GROUP BY cost\_segment

ORDER BY cost\_segment DESC;

-- Customer segmentation based on spending & lifespan

WITH customer\_spends AS (

SELECT c.customer\_key,

SUM(s.sales\_amount) AS total\_spending,

MIN(order\_date) AS first\_order,

MAX(order\_date) AS last\_order,

TIMESTAMPDIFF(MONTH, MIN(order\_date), MAX(order\_date)) AS life\_span

FROM customers AS c

JOIN sales AS s ON c.customer\_key = s.customer\_key

GROUP BY c.customer\_key

)

SELECT cus\_seg, COUNT(customer\_key) AS total\_customer

FROM (

SELECT customer\_key,

CASE

WHEN life\_span >= 12 AND total\_spending >= 5000 THEN 'VIP'

WHEN life\_span >= 12 AND total\_spending <= 4000 THEN 'REGULAR'

ELSE 'NEW'

END AS cus\_seg

FROM customer\_spends

) t

GROUP BY cus\_seg

ORDER BY total\_customer DESC;

-- ------------

-- Reporting

-- ------------

/\*

Customer Report

Purpose:

- Consolidates key customer metrics and behaviors

Highlights:

1. Essential fields: name, lifespan, transactions

2. Segments customers into VIP / REGULAR / NEW

3. Aggregates metrics: total orders, total sales, total quantity, total products, lifespan

4. KPIs: recency, average order value, average monthly spend

\*/

WITH base\_query AS (

SELECT

o.order\_number,

o.customer\_key,

CONCAT(c.first\_name, ' ', c.last\_name) AS customer\_name,

c.customer\_number,

o.product\_key,

o.order\_date,

o.sales\_amount,

o.quantity

FROM sales o

JOIN customers c ON o.customer\_key = c.customer\_key

JOIN products p ON o.product\_key = p.product\_key

),

customer\_segmentation AS (

SELECT

customer\_key,

customer\_number,

customer\_name,

COUNT(DISTINCT order\_number) AS total\_orders,

SUM(sales\_amount) AS total\_sales,

SUM(quantity) AS total\_quantity,

COUNT(DISTINCT product\_key) AS total\_products,

MAX(order\_date) AS last\_order\_date,

TIMESTAMPDIFF(MONTH, MIN(order\_date), MAX(order\_date)) AS lifespan

FROM base\_query

GROUP BY customer\_key, customer\_number, customer\_name

)

SELECT

customer\_key,

customer\_number,

customer\_name,

CASE

WHEN lifespan >= 12 AND total\_sales >= 5000 THEN 'VIP'

WHEN lifespan >= 12 AND total\_sales <= 4000 THEN 'REGULAR'

ELSE 'NEW'

END AS cus\_seg,

last\_order\_date,

TIMESTAMPDIFF(MONTH, last\_order\_date, CURDATE()) AS recency,

total\_orders,

total\_sales,

total\_quantity,

total\_products,

lifespan,

CASE WHEN total\_orders = 0 THEN 0 ELSE total\_sales / total\_orders END AS avg\_order\_value,

CASE WHEN lifespan = 0 THEN total\_sales ELSE total\_sales / lifespan END AS avg\_monthly\_spend

FROM customer\_segmentation

ORDER BY total\_sales DESC;

-- Example: create a view for the report (optional)

-- CREATE VIEW report\_customers AS <above SELECT statement>;

-- To preview report view

-- SELECT \* FROM report\_customers LIMIT 100;

/\* End of script \*/