E-Commerce Customer & Order Analytics (SQLite)

Author: Hatem Elgenedy

README

Overview

This SQL project analyzes e-commerce customer and order data using SQLite to generate key business KPIs and performance metrics. The project focuses on understanding customer behavior, sales performance, profitability, and operational efficiency through structured SQL queries.

The database is designed to support exploratory analysis as well as production-ready KPI reporting.

Database

- Engine: SQLite
- File: E-Commerce Customer & Order Analytics.db
- Data includes customer, order, product, and transaction-level information

Key KPIs & Metrics

- Total Revenue
- Total Profit
- Profit Margin (%)
- Total Orders
- Total Customers
- Average Order Value (AOV)
- Average Items per Order
- Average Orders per Customer
- Average Shipping Time
- Percentage of Orders Shipped Within SLA

Sample Business Questions Answered

- How much revenue and profit does the business generate?
- What is the average customer value and purchasing frequency?
- How efficient is the shipping and fulfillment process?
- Which customers and orders contribute most to profitability?

Skills Demonstrated

- SQL (SQLite)
- Aggregations & Grouping
- Joins & Subqueries
- KPI Design & Business Metrics
- Analytical Thinking
- Data Exploration & Validation

How to Use

- Download the .db file
- Open it using any SQLite-compatible tool (DB Browser for SQLite, SQLiteStudio, or Python)
- Run analytical queries to explore KPIs and insights

Notes

This project demonstrates SQL analytics skills commonly required for Data Analyst and Business Intelligence roles, with a strong focus on translating raw data into meaningful business insights.

SQL Code

```sql
CREATE TABLE IF NOT EXISTS "Orders-Raw" (
  "Row ID" INTEGER,
  "Order ID" TEXT,
  "Order Date" TEXT,
  "Ship Date" TEXT,
  "Ship Mode" TEXT,
  "Customer ID" TEXT,
  "Customer Name" TEXT,
  "Segment" TEXT,
  "Country/Region" TEXT,
  "City" TEXT,
  "State" TEXT,
  "Postal Code" INTEGER,
  "Region" TEXT,
  "Product ID" TEXT,
  "Category" TEXT,
  "Sub-Category" TEXT,
  "Product Name" TEXT,
  "Sales" REAL,
  "Quantity" INTEGER,
  "Discount" REAL,
  "Profit" REAL
);

CREATE TABLE IF NOT EXISTS kpi_summary (
  metric TEXT PRIMARY KEY,
  value REAL
);

CREATE VIEW IF NOT EXISTS v_category_performance AS
SELECT
  Category,
  ROUND(SUM(Sales), 2) AS total_revenue,
  ROUND(SUM(Profit), 2) AS total_profit
FROM "Orders-Raw"
GROUP BY Category;

CREATE VIEW IF NOT EXISTS v_discount_impact AS
SELECT
  CASE
    WHEN Discount <= 0.10 THEN '0-10%'
    WHEN Discount <= 0.20 THEN '10-20%'
    WHEN Discount <= 0.30 THEN '20-30%'
    WHEN Discount <= 0.40 THEN '30-40%'
    ELSE '40%+'
  END AS discount_bucket,
  ROUND(SUM(Sales), 2) AS total_revenue,
  ROUND(SUM(Profit), 2) AS total_profit
FROM "Orders-Raw"
GROUP BY discount_bucket;

CREATE VIEW IF NOT EXISTS v_region_category_profit AS
SELECT
```

```
  Region,
  Category,
  ROUND(SUM(Sales), 2) AS total_revenue,
  ROUND(SUM(Profit), 2) AS total_profit
FROM "Orders-Raw"
GROUP BY Region, Category;

CREATE VIEW IF NOT EXISTS v_region_performance AS
SELECT
  Region,
  ROUND(SUM(Sales), 2) AS total_revenue,
  ROUND(SUM(Profit), 2) AS total_profit
FROM "Orders-Raw"
GROUP BY Region;

CREATE VIEW IF NOT EXISTS v_revenue_by_year AS
SELECT
  SUBSTR("Order Date", -2) AS order_year,
  ROUND(SUM(Sales), 2) AS total_revenue
FROM "Orders-Raw"
GROUP BY order_year
ORDER BY order_year;

CREATE VIEW IF NOT EXISTS v_yoy_revenue_growth AS
WITH yearly AS (
  SELECT
    SUBSTR("Order Date", -2) AS order_year,
    SUM(Sales) AS total_revenue
  FROM "Orders-Raw"
  GROUP BY SUBSTR("Order Date", -2)
)
SELECT
  order_year,
  ROUND(total_revenue, 2) AS total_revenue,
  ROUND(
    (total_revenue - LAG(total_revenue) OVER (ORDER BY order_year)) * 100.0
    / NULLIF(LAG(total_revenue) OVER (ORDER BY order_year), 0),
    2
  ) AS yoy_growth_pct
FROM yearly;

CREATE VIEW IF NOT EXISTS v_shipping_performance AS
SELECT
  "Ship Mode",
  ROUND(
    AVG(
      julianday(
        '20' || SUBSTR("Ship Date", 7, 2) || '-' ||
        printf('%02d', CAST(SUBSTR("Ship Date", 1, instr("Ship Date", '/') - 1)
AS INT)) || '-' ||
        printf('%02d', CAST(SUBSTR("Ship Date", instr("Ship Date", '/') + 1,
          instr(SUBSTR("Ship Date", instr("Ship Date", '/') + 1), '/') - 1) AS
INT))
      )
      -
      julianday(
        '20' || SUBSTR("Order Date", 7, 2) || '-' ||
        printf('%02d', CAST(SUBSTR("Order Date", 1, instr("Order Date", '/') -
1) AS INT)) || '-' ||
        printf('%02d', CAST(SUBSTR("Order Date", instr("Order Date", '/') + 1,
          instr(SUBSTR("Order Date", instr("Order Date", '/') + 1), '/') - 1)
AS INT))
      )
    ),
    2
  ) AS avg_shipping_days
FROM "Orders-Raw"
WHERE "Order Date" IS NOT NULL
  AND "Ship Date" IS NOT NULL
GROUP BY "Ship Mode"
ORDER BY avg_shipping_days;
```