Analysis of E-commerce Dataset

Using Python and MySQL Workbench

-Sathwika Reddy

Introduction

- The primary purpose of this project is to perform a comprehensive analysis of an e-commerce dataset to extract valuable insights that can drive business decisions.
- The dataset, downloaded from Kaggle, consists of multiple CSV files containing various aspects of e-commerce operations such as orders, customers, products, Order items, Payments, Geographical locations, Sellers.

Steps undertaken in the project include:

1. **Data Collection**: Downloaded the dataset from Kaggle and organized it into a folder named "Ecommerce."
2. **Data Loading**: Utilized Python and Pandas to read the CSV files and load the data into MySQL Workbench.
3. **Data Analysis**: Conducted detailed data analysis using SQL to uncover trends, customer behavior, and product performance.

Purpose of Loading the Dataset Using Python for Process Optimization

Efficient Data Handling:

Python, with libraries like Pandas, handles large datasets efficiently and provides powerful data manipulation capabilities.

Automated Workflow:

Python scripts automate data loading and preprocessing, making the process repeatable, reducing human error, and saving time.

Integration with Databases:

Python easily connects to databases like MySQL, ensuring smooth data integration and enabling complex, customized data analysis.

Scalability and Flexibility:

Python solutions can scale with increasing data volumes and adapt to different data sources and formats.

Data Analysis Approach

```
create database ecommerce;
use ecommerce;
-- Basic Queries
-- -1. List all unique cities where customers are located.
select * from customers;
select distinct customer_city from customers;

-- 2. Count the number of orders placed in 2017.
select * from orders;

select count(order_id) as total_orders_in_2017 from orders
where year(order_purchase_timestamp)=2017;
```

```sql

    -- 3. Find the total sales per category.

    select * from products;
    select * from order_items;

    select p1.product_category,sum(o.price) as total_price from products p1
    join order_items o
    on p1.product_id=o.product_id
    group by p1.product_category;

    -- 4. Calculate the percentage of orders that were paid in installments.
    select * from payments;

    select (sum(case when payment_installments>=1 then 1 else 0 end)/count(*) )*100 as percentage from payments;

```

```sql
-- 5. Count the number of customers from each state.

select * from customers;

select customer_state ,count(customer_id)
from customers group by customer_state;

-- Calculate the number of orders per month in 2018.
select * from orders;
select month(order_purchase_timestamp) as month,count(order_id) as total_orders from orders
where year(order_purchase_timestamp)=2018
group by month(order_purchase_timestamp)
order by month(order_purchase_timestamp) asc;
```

```sql
-- Find the average number of products per order, grouped by customer city.

select * from order_items;
select * from orders;
select * from customers;

with count_per_order as(
select orders.order_id, orders.customer_id, count(order_items.order_id) as oc
from orders join order_items
on orders.order_id = order_items.order_id
group by orders.order_id, orders.customer_id
)
 select customers.customer_city, round(avg(count_per_order.oc),2) average_orders
from customers join count_per_order
on customers.customer_id = count_per_order.customer_id
group by customers.customer_city order by average_orders desc;

```

```sql
-- Calculate the percentage of total revenue contributed by each product category.

select * from order_items;
select * from products;

select * from payments;

select upper(products.product_category) category,
round((sum(payments.payment_value)/(select sum(payment_value) from payments))*100,2) sales_percentage
from products join order_items
on products.product_id = order_items.product_id
join payments
on payments.order_id = order_items.order_id
group by category order by sales_percentage desc;
```

```sql
-- Identify the correlation between product price and the number of times a product has been purchased.

select products.product_category,
count(order_items.product_id) as total_count,
round(avg(order_items.price),2) as avg_price
from products join order_items
on products.product_id = order_items.product_id
group by products.product_category;

-- Calculate the total revenue generated by each seller, and rank them by revenue.

select *, dense_rank() over(order by revenue desc) as rn from
(select order_items.seller_id, sum(payments.payment_value)
revenue from order_items join payments
on order_items.order_id = payments.order_id
group by order_items.seller_id) as a;
```

```sql

-- 1.Calculate the moving average of order values for each customer over their order history.
WITH cte AS (
    SELECT
        orders.customer_id,
        orders.order_purchase_timestamp,
        payments.payment_value AS payment,
        AVG(payments.payment_value) OVER (
            PARTITION BY orders.customer_id
            ORDER BY orders.order_purchase_timestamp
            ROWS BETWEEN 2 PRECEDING AND CURRENT ROW
        ) AS mov_avg
    FROM
        payments
    JOIN
        orders ON payments.order_id = orders.order_id
)
SELECT
    customer_id,
    order_purchase_timestamp,
    payment,
    mov_avg
FROM
    cte
ORDER BY
    customer_id,
    order_purchase_timestamp;
```

```sql

    -- 2. Calculate the cumulative sales per month for each year.
select * from orders;
select * from payments;
WITH sales_data AS (
    SELECT
        YEAR(orders.order_purchase_timestamp) AS years,
        MONTH(orders.order_purchase_timestamp) AS months,
        ROUND(SUM(payments.payment_value), 2) AS payment
    FROM orders
    JOIN payments ON orders.order_id = payments.order_id
    GROUP BY
        YEAR(orders.order_purchase_timestamp),
        MONTH(orders.order_purchase_timestamp)
    ORDER BY
        YEAR(orders.order_purchase_timestamp),
        MONTH(orders.order_purchase_timestamp)),
cumulative_sales_data AS (
    SELECT years,months,payment,
        SUM(payment) OVER (ORDER BY years, months) AS cumulative_sales
    FROM sales_data)
SELECT years, months, payment, cumulative_sales
FROM cumulative_sales_data
ORDER BY years, months;

```

```sql
-- 3. Calculate the year-over-year growth rate of total sales.

WITH a AS (
    SELECT
        YEAR(orders.order_purchase_timestamp) AS years,
        ROUND(SUM(payments.payment_value), 2) AS payment
    FROM orders
    JOIN payments ON orders.order_id = payments.order_id
    GROUP BY
        YEAR(orders.order_purchase_timestamp)
    ORDER BY
        YEAR(orders.order_purchase_timestamp)
),
percentage_change AS (
    SELECT years,payment,
        ((payment - LAG(payment, 1) OVER (ORDER BY years)) /
        LAG(payment, 1) OVER (ORDER BY years)) * 100 AS percentage_change
    FROM a)
SELECT years, payment, percentage_change
FROM percentage_change
ORDER BY years;
```

```sql
-- 4. Identify the top 3 customers who spent the most money in each year.

WITH customer_payments AS (
    SELECT
        YEAR(orders.order_purchase_timestamp) AS years,
        orders.customer_id,
        SUM(payments.payment_value) AS payment,
        DENSE_RANK() OVER (
            PARTITION BY YEAR(orders.order_purchase_timestamp)
            ORDER BY SUM(payments.payment_value) DESC
        ) AS d_rank
    FROM orders
    JOIN payments ON payments.order_id = orders.order_id
    GROUP BY YEAR(orders.order_purchase_timestamp),orders.customer_id
)
SELECT years, customer_id, payment, d_rank
FROM customer_payments
WHERE d_rank <= 3
ORDER BY years, d_rank;
```

Key Findings

Sales Trends:
Monthly/Yearly Sales Growth: Identified periods with significant increases or decreases in sales.
Seasonal Patterns: Observed peaks in sales during certain months, indicating seasonal trends.

Customer Behavior:
Top Customers: Identified customers who make the most frequent or highest value purchases.
Average Order Value: Calculated the average value of customer orders to understand purchasing power.
Customer Segmentation: Grouped customers into segments based on purchasing behavior and demographics.

Product Performance:
Best-Selling Products: Determined which products have the highest sales volume.
Product Categories: Analyzed which categories perform best in terms of sales and customer preference.
Inventory Turnover: Assessed how quickly products move through inventory.

Revenue Analysis:
Total Revenue: Calculated total revenue generated over different periods.
Revenue by Category: Analyzed revenue contributions from different product categories.

Order Analysis:
Order Volume: Monitored the number of orders placed over time.
Order Status: Evaluated the distribution of order statuses (e.g., completed, canceled, returned).

Customer Retention:
Repeat Purchase Rate: Measured the percentage of customers who make repeat purchases.
Customer Lifetime Value (CLV): Estimated the long-term value of customers based on purchasing behavior.

Geographical Insights:
Top Regions: Identified regions or locations with the highest sales.
Regional Preferences: Analyzed product preferences across different regions

Conclusion:

These key findings provide a comprehensive view of the e-commerce platform's performance, customer behavior, and product dynamics. By leveraging this data, stakeholders can make informed decisions to optimize operations, enhance customer satisfaction, and drive business growth.