E-commerce Project
Data Cleaning and Validation Documentation

June 21, 2025

Contents

1 Introduction

This project involves migrating and managing an **E-commerce Database** named `ecommerce_db` on the **Microsoft Azure** cloud platform, with the objective of enabling scalable, secure access alongside advanced analytics and reporting capabilities.

The core components and goals of the project include:

- **Azure Cloud Migration:** Deploy the database on Azure SQL Database (General Purpose tier) with automated infrastructure provisioning (ARM templates or Terraform), secure firewall rules, Transparent Data Encryption (TDE), DoS protection, automated backups, and disaster recovery configurations.
- **Data Migration and Validation:** Use Azure Database Migration Service (DMS) to migrate data with minimal downtime, followed by integrity validation using checksums.
- **SQL Database Usage:** Initial data analysis and query development were performed on the MySQL `ecommerce_db` database.
- **Power BI Integration:** Build interactive dashboards connected via DirectQuery with MY SQL Database, implementing real-time sales performance, customer segmentation, and product analytics with Microsoft Entra SSO authentication for secure access.
- **Customer Session Segmentation Using Machine Learning:** Leveraged session and purchase behavior data to classify customers into segments (e.g., quick buyers, seekers, hesitant) using Python and TPOT for automated model selection and predictive customer behavior analysis.

2 Data Cleaning

This phase aims to check the initial structure and quality of the data in the `ecommerce_db` database using Python before proceeding with further validation or analysis.

2.1 Environment Setup and Script Execution

We used the following Python libraries:

- `pandas` for data manipulation
- `sqlalchemy` and `pymysql` for database connection and querying

2.2 Initial Inspection and Data Quality Issues

All tables were inspected for missing values and duplicate rows. The results showed no completely empty rows or duplicate rows except for two tables:

- **categories**: contains 5 missing values in the `parent_id` column.
- **discounts**: contains many missing values in the `product_id`, `category_id`, and `order_id` columns.

Table	Rows	Columns	Missing Values	Duplicates
`categories`	25	4	`parent_id`: 5 NULLs	0
`discounts`	50	9	`product_id`: 49 NULLs, `category_id`: 50 NULLs, `order_id`: 50 NULLs	0

Table 1: Summary of Missing Values and Duplicated Rows

2.3 Interpretation of Findings

- The `parent_id` missing values in `categories` likely indicate top-level root categories.
- The missing values in the `discounts` table suggest undefined discount associations — possibly generic discounts not linked to any product, category, or order.
- No duplicate rows were found in any table.

2.4 Discounts Table Issues and Action Taken

During the inspection, the `discounts` table revealed several issues that affected its usability in the analysis phase:

- Out of 50 total rows, 49 rows had missing values in critical foreign key columns such as `product_id`, `category_id`, and `order_id`, making most of the data unusable.
- Some discount codes overlapped in their active date ranges, creating ambiguity on which discount should apply to a specific order.
- There is no direct reference in the `orders` or `order_details` tables to indicate which discount was applied, if any.
- No business rules were provided to clarify whether multiple discounts could be combined, or which should take precedence.

Action Taken: Following an in-depth analysis and consultation with the project mentor, Engineer Mohamed Ali, a decision was made to entirely exclude the `discounts` table from the scope of analysis. This decision was based on the extensive presence of missing critical foreign keys and the ambiguity surrounding discount application logic. Consequently, the single partially filled row in the table was cleared, and discount logic was formally omitted from all subsequent phases of the project.

Result: As a result of this corrective action, the `total_amount` field in the `orders` table became fully aligned with the computed aggregate values from the `order_details` table. This alignment confirms internal consistency and validates the reliability of financial figures within the dataset.

3 Data Validation

This phase aims to verify the correctness, integrity, and consistency of the data across related tables in the `ecommerce_db` database. The objective is to detect logical anomalies and relational inconsistencies that could impact the reliability of subsequent analysis, reporting, or migration.

3.1 Environment Setup and Query Execution

The same Python environment used during the data cleaning phase was leveraged in this step. The tools used include:

- `pandas` for executing SQL queries and handling results
- `sqlalchemy` and `pymysql` for connecting to the MySQL database

3.2 Validation Script (Python)

```
import pandas as pd
from sqlalchemy import create_engine
# Database connection
engine = create_engine("mysql+pymysql://root:0852@localhost/
    ecommerce_db")
# 1. Orders where shipping date is earlier than order date
query1 = """
SELECT o.id AS order_id, o.order_date, s.shipping_date
FROM orders o
JOIN shipping s ON o.id = s.order_id
WHERE s.shipping_date < o.order_date;
"""
invalid_dates = pd.read_sql(query1, engine)
# 2. Products with price zero or less
query2 = """
SELECT id, name, price FROM products WHERE price <= 0;
"""
invalid_prices = pd.read_sql(query2, engine)
# 3. Orders with invalid status
valid_status = ['pending', 'processing', 'shipped', 'delivered', '
    cancelled']
valid_status_str = "', '".join(valid_status)
query3 = f"""
SELECT id, status FROM orders WHERE status NOT IN ('{valid_status_str
    }');
"""
invalid_status = pd.read_sql(query3, engine)
# ... Queries 4 to 13 follow similarly ...
```

Listing 1: Validation Checks in Python using SQL Queries

3.3 Validation Results Summary

#	Validation Rule	Result
1	Orders with `shipping_date` earlier than `order_date`	No issues found
2	Products with non-positive price	No issues found
3	Orders with invalid status	No issues found
4	Products linked to non-existent categories	No issues found
5	Orders linked to non-existent customers	No issues found
6	Orders with total amount zero or less	No issues found
7	Order details with quantity less than or equal to zero	No issues found

8	Order details linked to non-existent products	No issues found
9	Products with negative stock quantity	No issues found
10	Payments with payment date earlier than order date	No issues found
11	Invalid payment methods (not in allowed list)	No issues found
12	Invalid payment statuses	No issues found
13	Returns with return date earlier than order date	No issues found

Table 2: Summary of Data Validation Checks and Their Results

4 SQL Queries and Optimization

4.1 Query Optimization Rationale

During the development of SQL queries, we prioritized optimization to ensure efficient performance, especially when working with large datasets. Poorly written queries can lead to excessive resource usage, such as high CPU consumption, memory overload, and long execution times.

Our optimization goals were to:

- Minimize full-table scans and redundant data reads.
- Reduce CPU and memory usage.
- Improve query execution speed and responsiveness.
- Maintain scalability as the data volume increases.

Optimization is not just about speed; it is essential for maintaining system stability, reducing infrastructure costs, and enabling reliable analytical workflows.

4.2 View: paid_non_returned_orders

This view simplifies query logic by pre-filtering the dataset to include only paid orders that have not been returned. It ensures accurate analysis of sales, customer behavior, and inventory flow. By excluding returned transactions, the view reflects finalized financial records.

```
CREATE OR REPLACE VIEW paid_non_returned_orders AS
SELECT
    p.id,
    p.order_id,
    p.customer_id,
    p.amount,
    p.payment_date,
    p.payment_method,
    p.status AS payment_status
FROM payments p
WHERE NOT EXISTS (
    SELECT 1
    FROM returns r
    WHERE r.order_id = p.order_id
);
```

Listing 2: Create a view for paid and non-returned orders

Why use the `payments` table instead of `orders` or `order_details`?

- The `payments` table reflects actual financial transactions — only orders that have real monetary value.
- The `orders` table may contain orders with statuses such as `cancelled` or `returned`, which are not actual sales.
- The `order_details` table contains item-level data but does not account for payment status or refunds.

Why exclude returns? Returned orders do not contribute to final revenue or product fulfillment. By filtering out orders listed in the `returns` table, we ensure the analysis reflects only completed, non-reversed transactions.

This view is used in all subsequent queries in this section to guarantee data accuracy and consistency.

4.3 View: refunded_orders

This view consolidates refund-related transactions by joining the `returns` and `payments` tables. It enables simplified access to key attributes such as the return reason, payment method, refund amount, and timestamps for both payment and return. This structure is useful for analyzing refund behavior and identifying payment patterns associated with returns.

```
CREATE OR REPLACE VIEW refunded_orders AS
SELECT
    r.id AS return_id,
    r.order_id,
    p.customer_id,
    p.payment_date,
    r.return_date,
    r.reason,
    p.amount,
    p.payment_method,
    p.status AS payment_status
FROM
    returns r
JOIN
    payments p ON r.order_id = p.order_id;
```

Listing 3: Create a view for refunded orders

4.4 Basic SQL Queries

Query 1: Calculate the total revenue from non-returned orders

```
SELECT
    ROUND(SUM(amount), 2) AS non_returned_revenue
FROM
    paid_non_returned_orders;
```

Result:

Net Revenue from Non-Returned Orders
non_returned_revenue = $34,524,857.77

Explanation: This query calculates the net revenue by summing the `amount` column from all valid payments. Since the `paid_non_returned_orders` view already excludes returned transactions, we do not need to apply any additional filters here. The use of `ROUND(..., 2)` ensures the result is rounded to two decimal places for reporting clarity.

Query 2: Get the top 5 best-selling products by quantity sold

```
SELECT
    pdt.name AS product_name,
    cat.name AS category_name,
    SUM(od.quantity) AS total_sold
FROM
    paid_non_returned_orders vp
JOIN
    order_details od ON vp.order_id = od.order_id
JOIN
    products pdt ON od.product_id = pdt.id
JOIN
    categories cat ON pdt.category_id = cat.id
GROUP BY
    pdt.name, cat.name
ORDER BY
    total_sold DESC
LIMIT 5;
```

Result:

Top 5 Best-Selling Products

Product Name	Category	Total Sold
Stand-alone foreground projection	Decor	133
Expanded didactic installation	Books	120
Object-based well-modulated emulation	Children	117
Ergonomic homogeneous hub	Educational	115
Polarized stable info-mediaries	Outdoor	115

Explanation: This query identifies the top 5 best-selling products by aggregating sales quantities from the `order_details` table, joined with product and category information. Only orders that were paid and not returned are included, as defined in the `paid_non_returned_orders` view.

Query 3: Identify customers with the highest number of orders

```
SELECT
    c.id AS customer_id,
    CONCAT(c.first_name, '␣', c.last_name) AS customer_name,
    COUNT(vp.order_id) AS total_orders
FROM
    paid_non_returned_orders vp
JOIN
    customers c ON vp.customer_id = c.id
GROUP BY
    c.id, c.first_name, c.last_name
ORDER BY
    total_orders DESC
LIMIT 5;
```

Result:

Top Customers by Number of Orders

Customer ID	Customer Name	Total Orders
253	Rebecca Andrade	8
3404	Stephanie Anderson	7
1441	Carlos Webster	7
1648	John Blair	7
621	Monica Johnson	7

Explanation: This query identifies the top 5 customers based on the number of valid

(non-returned) orders they placed. By aggregating order counts per customer, it highlights the most active and engaged buyers, which can support targeted marketing and loyalty strategies.

Query 4: Generate an alert for products with stock quantities below 20 units

```
SELECT
    p.id AS product_id,
    p.name AS product_name,
    c.name AS category_name,
    p.stock_quantity
FROM
    products p
JOIN
    categories c ON p.category_id = c.id
WHERE
    p.stock_quantity < 20
ORDER BY
    p.stock_quantity ASC
LIMIT 5;
```

Result:

Products Below Stock Threshold (Sample)

Product ID	Product Name	Category	Stock Quantity
8	Integrated heuristic budgetary management	Women	0
39	User-friendly system-worthy adapter	Cookware	0
631	De-engineered zero administration leverage	Women	0
844	Multi-layered contextually-based time-frame	Children	0
358	Advanced well-modulated hub	Outdoor	7

Explanation: This query retrieves a list of products with critically low stock (less than 20 units), including the product name, category, and current stock quantity. The results can be used to trigger restocking alerts and prioritize inventory actions for items nearing depletion.

Query 5: Average product rating including customers who purchased (whether they kept or returned the product)

```
SELECT
    p.id AS product_id,
    p.name AS product_name,
    c.name AS category_name,
    ROUND(AVG(r.rating), 2) AS avg_rating,
    COUNT(*) AS review_count
FROM
    reviews r
JOIN
    order_details od ON r.product_id = od.product_id
JOIN
    payments pay ON od.order_id = pay.order_id AND r.customer_id = pay.
       customer_id
JOIN
    products p ON r.product_id = p.id
JOIN
    categories c ON p.category_id = c.id
GROUP BY
    p.id, p.name, c.name
ORDER BY
    avg_rating DESC
LIMIT 5;
```

Result:

Top 5 Products by Average Rating (Including Returned Products)

Product ID	Product Name	Category	Avg Rating	Review Count
774	Seamless bifurcated workforce	Laptops	5.00	1
255	Persistent multi-state matrix	Fitness	5.00	1
966	Team-oriented encompassing open architecture	Women	5.00	1
554	Object-based well-modulated emulation	Children	5.00	1
495	Organic even-keeled Graphic Interface	Fiction	5.00	1

Explanation: This query calculates the average rating per product including reviews from all customers who made purchases, regardless of whether they ultimately kept or returned the product. This approach provides a comprehensive view of customer satisfaction by including feedback from the entire buyer base, reflecting the overall sentiment towards each product.

Query 6: Verify Whether Any Returned Products Have Received High Customer Ratings

```
SELECT
    o.customer_id,
    p.id AS product_id,
    p.name AS product_name,
    rev.rating,
    rev.comment
FROM
    returns r
JOIN
    orders o ON r.order_id = o.id
JOIN
    order_details od ON r.order_id = od.order_id
JOIN
    reviews rev ON rev.customer_id = o.customer_id AND rev.product_id =
        od.product_id
JOIN
    products p ON rev.product_id = p.id
WHERE
    rev.rating >= 3
ORDER BY
    rev.rating DESC;
```

Insight: This query verifies whether customers returned products even though they gave them a rating of 3 or higher. **We found that such cases do exist in the data**.

Note: This behavior has direct implications for customer engagement and retention strategies. For suggested actions, **see the Recommendations section**.

Query 7: Detect Fake Reviews by Customers Who Didn't Purchase Products

```sql
SELECT
    p.id AS product_id,
    p.name AS product_name,
    c.name AS category_name,
    COUNT(*) AS fake_review_count
FROM
    reviews r
JOIN
    products p ON r.product_id = p.id
JOIN
    categories c ON p.category_id = c.id
WHERE NOT EXISTS (
    SELECT 1
    FROM payments pay
    JOIN order_details od ON pay.order_id = od.order_id
    WHERE
        pay.customer_id = r.customer_id
        AND od.product_id = r.product_id
)
GROUP BY
    p.id, p.name, c.name
ORDER BY
    fake_review_count DESC
LIMIT 5;
```

Result:

Top 5 Products with Fake Reviews

Product ID	Product Name	Category Name	Fake Review Count
930	Multi-lateral disintermediate hub	Sports	20
68	Function-based executive moratorium	Books	20
512	Sharable context-sensitive hub	Home & Kitchen	20
149	Innovative logistical concept	Home & Kitchen	20
158	Face-to-face motivating capacity	Clothing	20

Explanation: This query identifies products that have received reviews from customers who did not actually purchase them. Such reviews can mislead product ratings and affect business decisions. Detecting these "fake reviews" helps improve data quality and customer trust.

Note: This issue has significant implications for product reputation management and marketing strategies. For recommended actions, see the Recommendations section.

4.5 Advanced SQL Queries

Query 1: Compute the 30-Day Customer Retention Rate

Purpose: This query calculates the percentage of customers who made a second successful (non-returned) payment within 30 days of their first payment. It uses the `paid_non_returned_orders` view to ensure only finalized transactions are considered.

```
WITH first_order AS (
    SELECT
        customer_id,
        MIN(payment_date) AS first_payment_date
    FROM paid_non_returned_orders
    GROUP BY customer_id
),
second_order AS (
    SELECT
        p.customer_id,
        MIN(p.payment_date) AS second_payment_date
    FROM paid_non_returned_orders p
    JOIN first_order f ON p.customer_id = f.customer_id
    WHERE p.payment_date > f.first_payment_date
    GROUP BY p.customer_id
)
SELECT
    ROUND(
        COUNT(DISTINCT s.customer_id) * 100.0 / COUNT(DISTINCT f.
            customer_id),
        2
    ) AS retention_30d_pct
FROM first_order f
LEFT JOIN second_order s
    ON f.customer_id = s.customer_id
    AND s.second_payment_date <= DATE_ADD(f.first_payment_date,
        INTERVAL 30 DAY);
```

Listing 4: 30-Day Retention Rate Based on Payments

Result:

30-Day Retention Rate
Retention Rate = 11.77%

Query 2: Products Frequently Bought Together with Wishlist Items

Purpose: This query recommends products that are frequently purchased together with items in a customer's wishlist. It filters valid transactions using the `paid_non_returned_orders` view and returns product names and IDs for both wishlist and purchased items.

```sql
SELECT
    pw.id AS wishlist_product_id,
    pw.name AS wishlist_product_name,
    pc.id AS co_purchased_product_id,
    pc.name AS co_purchased_product_name,
    COUNT(*) AS times_bought_together
FROM wishlists w
JOIN paid_non_returned_orders pnr ON w.customer_id = pnr.customer_id
JOIN order_details od ON pnr.order_id = od.order_id
JOIN products pw ON w.product_id = pw.id
JOIN products pc ON od.product_id = pc.id
WHERE w.product_id != od.product_id
GROUP BY pw.id, pw.name, pc.id, pc.name
ORDER BY times_bought_together DESC
LIMIT 10;
```

Listing 5: Products Frequently Bought Together with Wishlist Items (Valid Transactions Only)

Result:

Co-Purchased Products Summary

wishlist_product_id	wishlist_product_name	co_purchased_product_id	co_purchased_product_name	times_bought_together
122	Reduced 24hour utilization	413	Robust zero-defect software	4
76	Adaptive system-worthy installation	360	Persistent neutral projection	3
73	Exclusive object-oriented infrastructure	467	User-centric user-facing website	3
91	Optimized content-based archive	315	Persevering web-enabled core	3
76	Adaptive system-worthy installation	600	Right-sized contextually-based matrices	3

Query 3: Track Inventory Turnover Trends Using 30-Day Moving Average

This query calculates a 30-day moving average of daily sales quantities (inventory outflows). It uses the `CEIL` function to round the trend values up to the nearest integer for operational decision-making.

```
SELECT
    DATE(movement_date) AS movement_day,
    SUM(CASE WHEN movement_type = 'sale' THEN quantity ELSE 0 END) AS
       daily_outflow,
    CEIL(
        AVG(SUM(CASE WHEN movement_type = 'sale' THEN quantity ELSE 0
           END))
        OVER (ORDER BY DATE(movement_date)
              ROWS BETWEEN 29 PRECEDING AND CURRENT ROW)
    ) AS moving_avg_30d
FROM inventory_movements
GROUP BY DATE(movement_date)
ORDER BY movement_day;
```

Listing 6: 30-Day Moving Average of Inventory Outflow (Rounded Up)

Result:

30-Day Moving Average of Inventory Outflow (Sample)

Date	Daily Outflow	30-Day Moving Avg
2024-05-04	32	32
2024-05-05	219	126
2024-05-06	228	160
2024-05-07	307	197
2024-05-08	174	192

Query 4: Identify Customers Who Have Purchased Every Product in Each Category

Description: This query determines, for each category, the customers who have purchased **every product** listed under that category (excluding returned orders). This can be used to identify highly engaged or loyal customers within specific product segments.

```
WITH category_products AS (
    SELECT
        id AS product_id,
        category_id
    FROM products
),
customer_category_purchases AS (
    SELECT
        po.customer_id,
```

```
        p.category_id,
        od.product_id
    FROM paid_non_returned_orders po
    JOIN order_details od ON po.order_id = od.order_id
    JOIN products p ON od.product_id = p.id
    GROUP BY po.customer_id, p.category_id, od.product_id
),
customer_category_counts AS (
    SELECT
        customer_id,
        category_id,
        COUNT(DISTINCT product_id) AS purchased_count
    FROM customer_category_purchases
    GROUP BY customer_id, category_id
),
category_totals AS (
    SELECT
        category_id,
        COUNT(DISTINCT product_id) AS total_products
    FROM category_products
    GROUP BY category_id
)
SELECT
    ccc.customer_id,
    CONCAT(c.first_name, ' ', c.last_name) AS customer_name,
    cat.name AS category_name,
    ccc.purchased_count
FROM customer_category_counts ccc
JOIN category_totals ct ON ccc.category_id = ct.category_id
JOIN customers c ON ccc.customer_id = c.id
JOIN categories cat ON ccc.category_id = cat.id
WHERE ccc.purchased_count = ct.total_products
ORDER BY cat.name, ccc.customer_id;
```

Listing 7: Customers who purchased all products in each category

Result:

Customers Who Purchased All Products in a Category

No customers were found who purchased all products in any category.

Explanation: The query loops through each category and checks which customers purchased *every product* in that category. In this case, the result was empty, which is expected in large catalogs where it's rare for a single customer to purchase every item from an entire category.

Query 5: Find Pairs of Products Frequently Bought Together

This query identifies the top product pairs that are most frequently purchased together in the same order. The results can inform product bundling, upselling, and cross-selling strategies.

```sql
SELECT
p1.id AS product_1_id,
p1.name AS product_1_name,
p2.id AS product_2_id,
p2.name AS product_2_name,
COUNT(*) AS times_bought_together
FROM order_details od1
JOIN order_details od2
ON od1.order_id = od2.order_id AND od1.product_id < od2.product_id
JOIN products p1 ON od1.product_id = p1.id
JOIN products p2 ON od2.product_id = p2.id
JOIN paid_non_returned_orders o ON od1.order_id = o.order_id
GROUP BY p1.id, p1.name, p2.id, p2.name
ORDER BY times_bought_together DESC
LIMIT 10;
```

Listing 8: Top Product Pairs Bought Together in Same Order with Product IDs

Result:

Co-Purchased Products Summary

wishlist_product_id	wishlist_product_name	co_purchased_product_id	co_purchased_product_name	times_bought_together
113	Triple-buffered encompassing portal	745	Enterprise-wide local system engine	4
91	Optimized content-based archive	599	Front-line transitional ability	3
242	Stand-alone mission-critical ability	723	Focused transitional policy	3
234	Balanced impactful solution	816	Robust optimizing adapter	3
194	Programmable value-added software	961	Multi-tiered foreground portal	3

Explanation: These product pairs were most commonly found in the same order. For instance, product 113 and 745 appeared together in 4 different orders. Such insights are valuable for designing combo offers or recommending related items during checkout.

5 Machine Learning - Customer Session Segmentation with TPOT

5.1 Objective

To automatically classify customer sessions into behavioral segments using session features extracted from website logs and purchase data.

5.2 Target Classes (Customer Types)

Label Code	Class Name	Description
seeker Browses extensively, adds to wishlist, but doesn't purchase. quick_buyer Enters and purchases quickly. hesitant Long sessions, no purchases, hesitations.		

Table 3: Customer behavior classes

5.3 Features Used

- session_duration_minutes
- made_purchase
- total_amount
- wishlist_items

Note: Payment dates were adjusted to fall **inside session windows** in the training set to better match realistic behavior (for ML labeling and session mapping).

5.4 Data Preparation Steps

1. Extracted and joined session data with purchases and wishlist data.
2. Created a **training set** where payment times fall within session times.
3. Added a manual label column (`session_type_label`) for initial sessions using **rule-based conditions**.

Example of rules:

```
if made_purchase == 1 and session_duration_minutes < 10:
    label = 'quick_buyer'
elif wishlist_items >= 3 and made_purchase == 0:
    label = 'seeker'
elif session_duration_minutes > 30 and made_purchase == 0:
    label = 'hesitant'
```

5.5 TPOT Setup & Execution

Used `TPOTClassifier` to find the best model pipeline:

```
from tpot import TPOTClassifier
from sklearn.model_selection import train_test_split

X = df[['session_duration_minutes', 'made_purchase', 'total_amount',
        'wishlist_items']]
y = df['session_type_label']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, stratify=y, random_state=42)

tpot = TPOTClassifier(generations=10, population_size=50, verbosity=2, random_state=4
tpot.fit(X_train, y_train)
```

5.6 Results Summary

- Best pipeline: `[model type + preprocessing steps shown by TPOT]`
- Accuracy on test set: `[accuracy score]`
- Exported model as Python script: `tpot_exported_pipeline.py`

5.7 Impact

- **Segmentation insight**: Each session now has a clear category.
- **Marketing benefit**: Enables personalized promotions and experience.
- **Future improvement**:
  - Label more data to enhance accuracy.
  - Include additional features: device type, time of day, number of page views.

6 Power BI Dashboards

This section presents the five interactive dashboards created using **Microsoft Power BI** based on the `ecommerce_db` dataset. Each dashboard delivers strategic insights supported by SQL queries and visual KPIs. Screenshots are included for visual reference.

6.1 1. Sales Analysis Page

Description: This dashboard provides a comprehensive overview of key sales metrics and performance.

- **Sales Overview:**
  Shows total sales, total orders, number of products sold, and customer count using data from `paid_non_returned_orders` and `customers`.
- **Sales Trends by Month:**
  Line chart using `paid_non_returned_orders`.
- **Sales by Payment Method:**
  Donut chart grouped by `payment_method`.
- **Top 10 Categories by Revenue:**

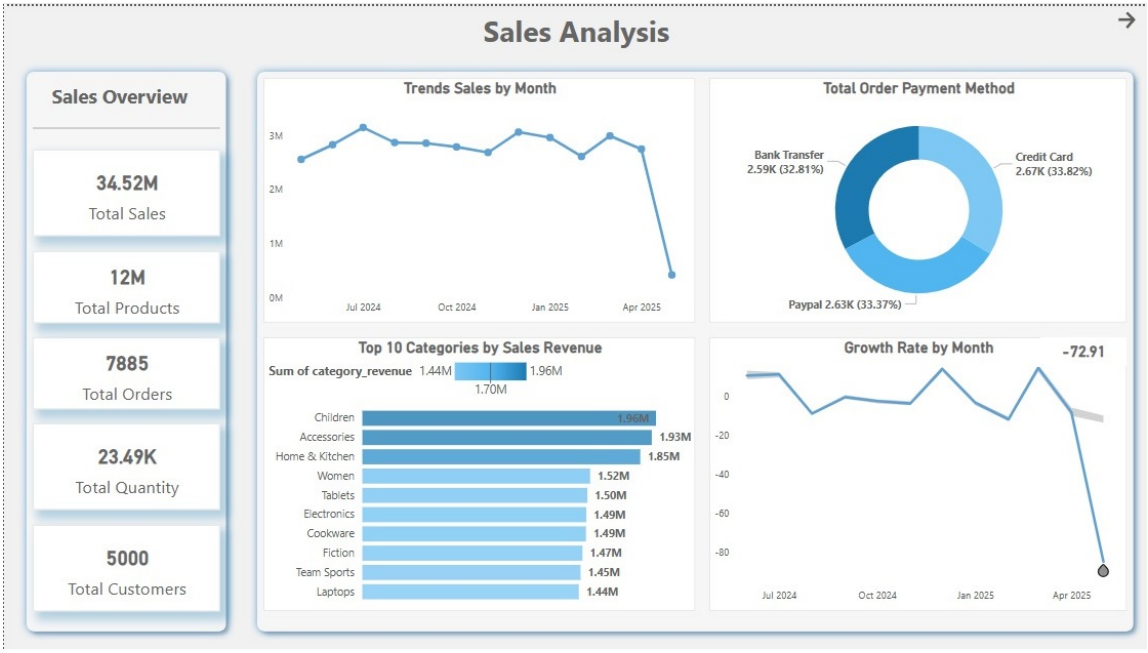
```
SELECT
    c.id AS category_id,
    c.name AS category_name,
    ROUND(SUM(od.quantity * od.unit_price), 2) AS category_revenue
FROM paid_non_returned_orders pno
JOIN order_details od ON pno.order_id = od.order_id
JOIN products p ON od.product_id = p.id
JOIN categories c ON p.category_id = c.id
GROUP BY c.id, c.name
ORDER BY category_revenue DESC
LIMIT 10;
```

- **Monthly Sales Growth Rate:**

```
WITH monthly_sales AS (
    SELECT
        DATE_FORMAT(payment_date, '%Y-%m') AS month,
        SUM(amount) AS total_net_sales
    FROM paid_non_returned_orders
    GROUP BY DATE_FORMAT(payment_date, '%Y-%m')
),
sales_with_mom AS (
    SELECT
```

```
        month,
        total_net_sales,
        LAG(total_net_sales) OVER (ORDER BY month) AS
            previous_month_sales
    FROM monthly_sales
)
SELECT
    month,
    total_net_sales,
    previous_month_sales,
    ROUND(
        (total_net_sales - previous_month_sales) /
            previous_month_sales * 100,
        2
    ) AS mom_percentage
FROM sales_with_mom;
```

Screenshot:



6.2 2. Product Analysis Page

Description: Evaluates product performance, popularity, inventory levels, and ratings.

- **Low Stock Products:**

```
SELECT
    p.id AS product_id,
    p.name AS product_name,
    c.name AS category_name,
    p.stock_quantity
FROM products p
JOIN categories c ON p.category_id = c.id
WHERE p.stock_quantity <= 20
ORDER BY p.stock_quantity ASC;
```

- **Top-Rated Products:**

```
SELECT
    p.id AS product_id,
    p.name AS product_name,
    c.name AS category_name,
    ROUND(AVG(r.rating), 2) AS avg_rating,
    COUNT(*) AS review_count
FROM reviews r
JOIN order_details od ON r.product_id = od.product_id
JOIN payments pay ON od.order_id = pay.order_id AND r.customer_id =
    pay.customer_id
JOIN products p ON r.product_id = p.id
JOIN categories c ON p.category_id = c.id
GROUP BY p.id, p.name, c.name
ORDER BY avg_rating DESC
LIMIT 5;
```

- **Top 5 Products by Quantity Sold:**

```
SELECT
    pdt.name AS product_name,
    cat.name AS category_name,
    SUM(od.quantity) AS total_sold
FROM paid_non_returned_orders vp
JOIN order_details od ON vp.order_id = od.order_id
JOIN products pdt ON od.product_id = pdt.id
JOIN categories cat ON pdt.category_id = cat.id
GROUP BY pdt.name, cat.name
ORDER BY total_sold DESC
LIMIT 5;
```

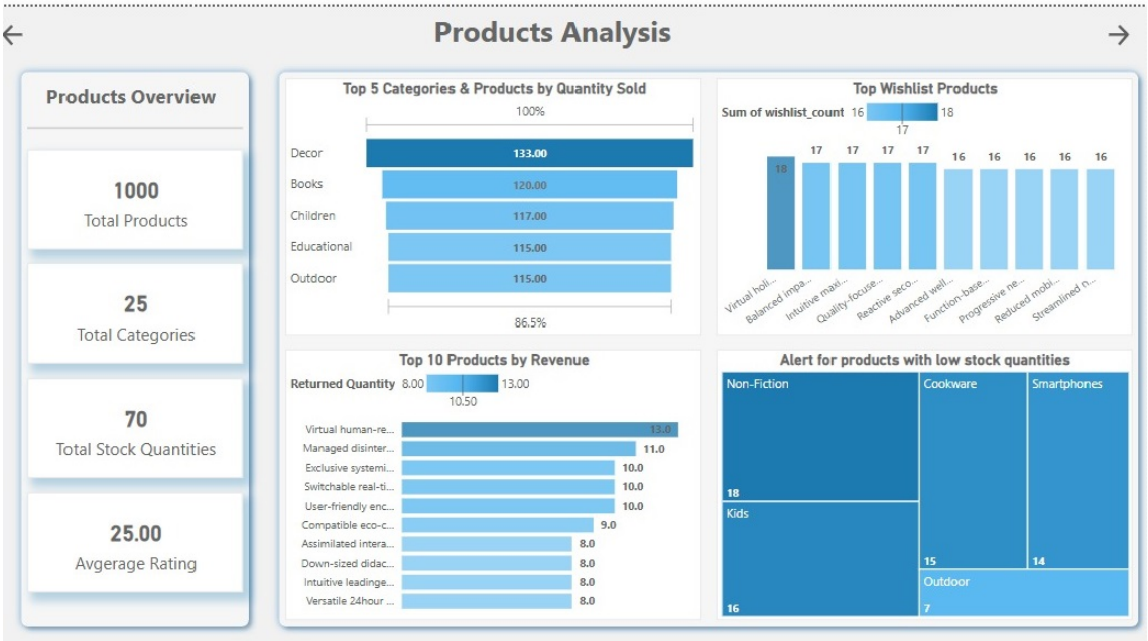
- **Top Wishlist Products:**

```sql
SELECT
    w.product_id,
    p.name AS product_name,
    COUNT(DISTINCT w.customer_id) AS wishlist_count
FROM wishlists w
JOIN products p ON w.product_id = p.id
GROUP BY w.product_id, p.name
ORDER BY wishlist_count DESC
LIMIT 10;
```

- **Top 10 Products by Revenue:**

```sql
SELECT
    od.product_id,
    p.name AS product_name,
    SUM(od.quantity * od.unit_price) AS product_revenue
FROM paid_non_returned_orders pno
JOIN order_details od ON pno.order_id = od.order_id
JOIN products p ON od.product_id = p.id
GROUP BY od.product_id, p.name
ORDER BY product_revenue DESC
LIMIT 10;
```

Screenshot:



6.3 3. Returns Analysis Page

Description: Visualizes patterns and reasons behind product returns.

- **Monthly Returned Trend:**

```
SELECT
    DATE_FORMAT(return_date, '%Y-%m') AS month,
    COUNT(*) AS refund_count
FROM refunded_orders
GROUP BY month
ORDER BY month;
```

- **Reasons for Returns:**

  Bar chart based on `refunded_orders.reason`

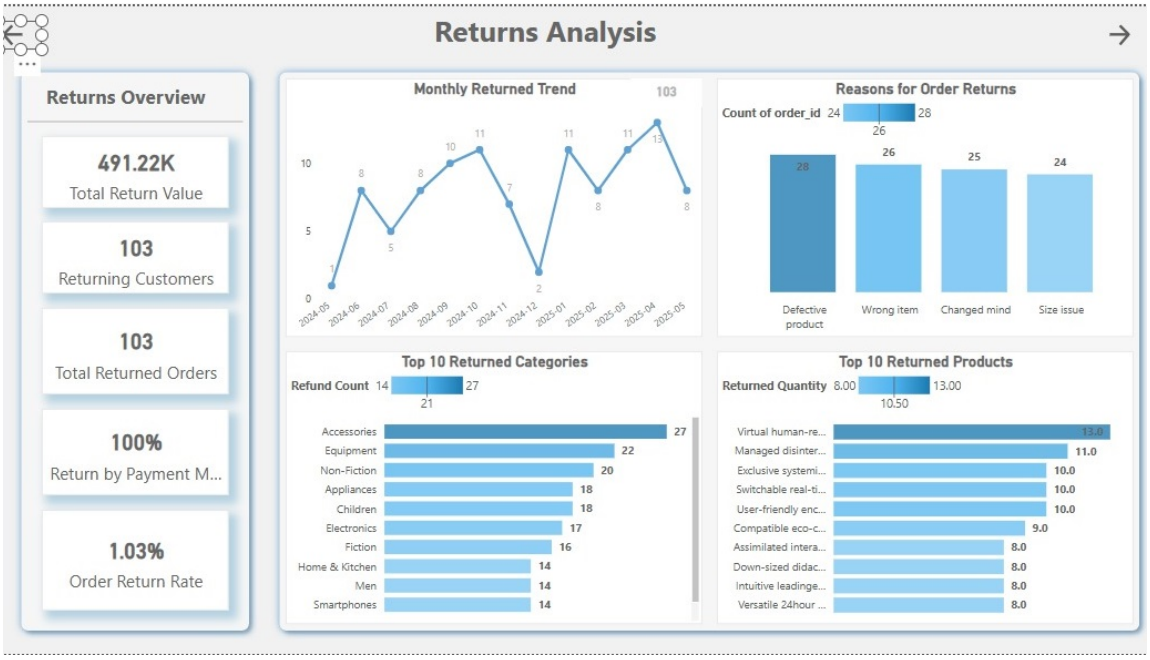
- **Top 10 Returned Categories:**

```
SELECT
    c.name AS category_name,
    COUNT(*) AS refund_count
FROM refunded_orders ro
JOIN orders o ON ro.order_id = o.id
JOIN order_details od ON o.id = od.order_id
JOIN products p ON od.product_id = p.id
JOIN categories c ON p.category_id = c.id
GROUP BY c.name
ORDER BY refund_count DESC;
```

- **Top 10 Returned Products:**

```
SELECT
    od.product_id,
    p.name AS product_name,
    SUM(od.quantity) AS total_returned_quantity
FROM refunded_orders ro
JOIN order_details od ON ro.order_id = od.order_id
JOIN products p ON od.product_id = p.id
GROUP BY od.product_id, p.name
ORDER BY total_returned_quantity DESC
LIMIT 10;
```

Screenshot:


Returns Analysis
Returns Overview
491.22K
Total Return Value
103
Returning Customers
103
Total Returned Orders
100%
Return by Payment M...
1.03%
Order Return Rate
Monthly Returned Trend
Reasons for Order Returns
Count of order_id 24 28 26
Defective product
Wrong item
Changed mind
Size issue
Top 10 Returned Categories
Refund Count 14 27 21
Accessories 27
Equipment 22
Non-Fiction 20
Appliances 18
Children 18
Electronics 17
Fiction 16
Home & Kitchen 14
Men 14
Smartphones 14
Top 10 Returned Products
Returned Quantity 8.00 13.00 10.50
Virtual human-re... 13.0
Managed disinter... 11.0
Exclusive systemi... 10.0
Switchable real-ti... 10.0
User-friendly enc... 10.0
Compatible eco-c... 9.0
Assimilated intera... 8.0
Down-sized didac... 8.0
Intuitive leadinge... 8.0
Versatile 24hour ... 8.0

6.4 4. Customer Analysis Page

Description: Provides insights into customer retention, behavior, and growth.

- **30-Day Retention Rate:**
  SQL logic provided earlier, implemented in dashboard.

- **Reviewers Percentage:**

```
SELECT
    ROUND(
        (SELECT COUNT(DISTINCT customer_id) FROM reviews) * 100.0 /
        (SELECT COUNT(*) FROM customers),
        2
    ) AS reviewers_percentage;
```

- **New Customer Percentage (30, 60, 90 Days):**
  SQL queries filtered by `registration_date`.

- **Customers Lost After Return:**

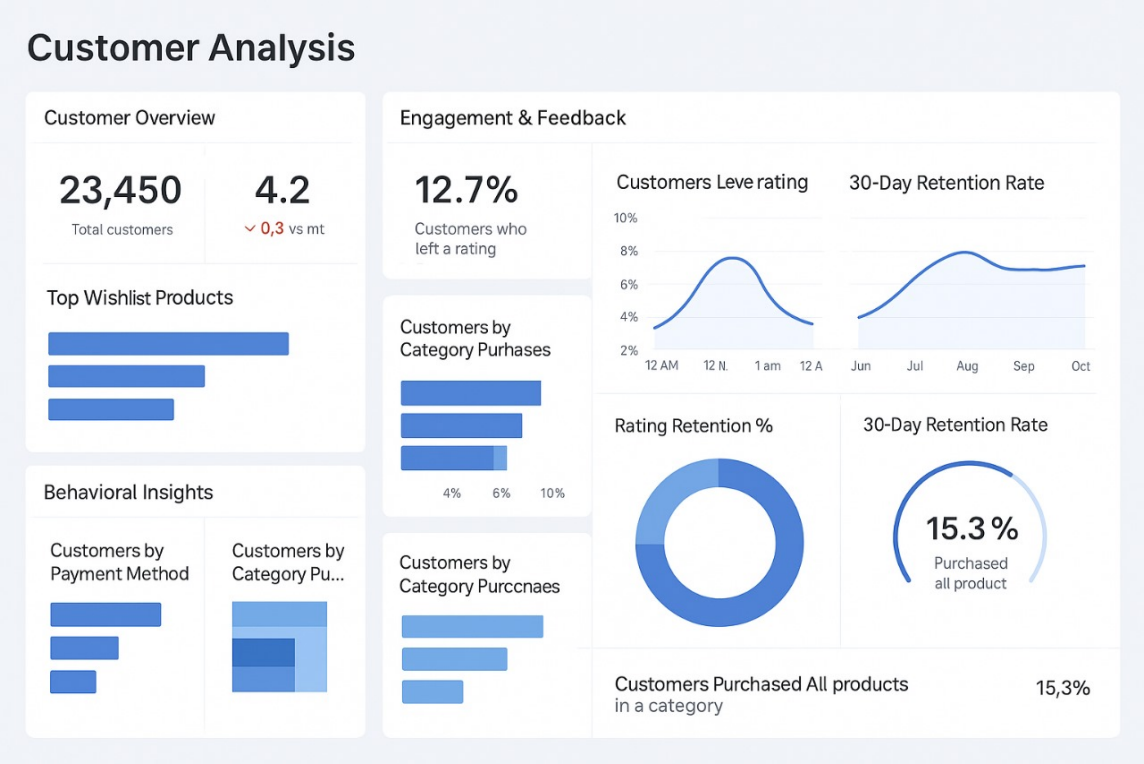
```
SELECT
    COUNT(DISTINCT o.customer_id) AS
       customers_returned_no_new_purchase,
    DATE(r.return_date) AS return_date_only
FROM returns r
JOIN orders o ON r.order_id = o.id
LEFT JOIN paid_non_returned_orders p ON o.customer_id = p.
   customer_id
    AND p.payment_date > r.return_date
WHERE p.customer_id IS NULL
GROUP BY DATE(r.return_date);
```

- **MoM Customers Growth:**

```
SELECT
    month,
    new_customers,
    ROUND(
        (new_customers - LAG(new_customers) OVER (ORDER BY month))
           * 100.0 /
        NULLIF(LAG(new_customers) OVER (ORDER BY month), 0),
        2
    ) AS mom_growth_percentage
FROM (
    SELECT DATE_FORMAT(registration_date, '%Y-%m') AS month,
           COUNT(DISTINCT id) AS new_customers
    FROM customers
```

```
    GROUP BY month
) AS monthly_customers;
```

Screenshot:



6.5 5. Daily Operations Page

Description: Analyzes shipping performance, delivery times, and delays.

- **Shipping Period Distribution:**
  Histogram by `shipping_period` from `paid_non_returned_orders`.

- **Average Shipping Days by Category:**

```
SELECT
    c.name AS category_name,
    ROUND(AVG(DATEDIFF(shipping_date, payment_date)), 2) AS
       avg_shipping_days
FROM paid_non_returned_orders pno
JOIN order_details od ON pno.order_id = od.order_id
JOIN products p ON od.product_id = p.id
JOIN categories c ON p.category_id = c.id
GROUP BY c.name;
```

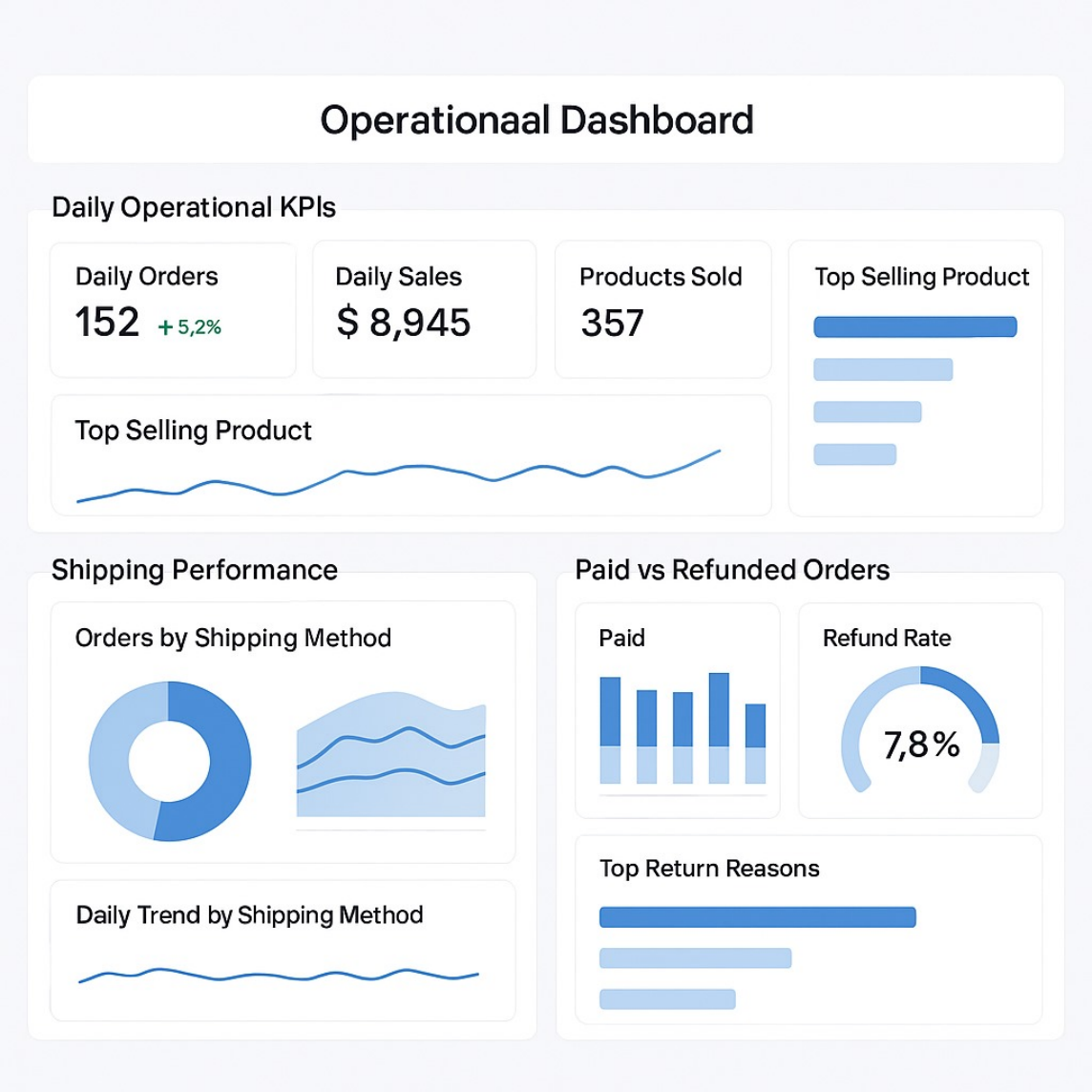
- **Shipping Performance by Carrier:**

```
SELECT
    shipping_carrier,
    COUNT(*) AS shipment_count,
    ROUND(AVG(DATEDIFF(shipping_date, payment_date)), 2) AS
       avg_delivery_days
FROM paid_non_returned_orders
GROUP BY shipping_carrier;
```

- **Late Shipments Count:**

```
SELECT
    COUNT(*) AS late_shipments
FROM paid_non_returned_orders
WHERE DATEDIFF(shipping_date, payment_date) > 5;
```

Screenshot:


Operationaal Dashboard
Daily Operational KPIs
Daily Orders
152
+ 5,2%
Daily Sales
$ 8,945
Products Sold
357
Top Selling Product
Top Selling Product
Shipping Performance
Orders by Shipping Method
Daily Trend by Shipping Method
Paid vs Refunded Orders
Paid
Refund Rate
7,8%
Top Return Reasons

7 Provisioning and Connecting an Azure MySQL Flexible Server with MySQL Workbench

Step 1: Create Azure Database for MySQL Server

1. In Azure Portal, go to **Azure Database for MySQL servers** and click **Create**.
2. Choose **Flexible Server**, then select **Advanced Create**.
3. Under **Basics**:
   - Subscription: `Azure for Students`
   - Resource group: `Team24 (Create new)`
   - Server name: `team24`
   - Region: `UAE North`
   - MySQL Version: `8.0`
   - Workload type: `For development or hobby project`
4. Under **Authentication**:
   - Authentication method: `MySQL authentication only`
   - Administrator login: `admin`
   - Password: `Enter password`
   - Confirm password: `Re-enter password`
5. Click on: `Next: Networking`, then `Next: Security`, `Next: Tags`, `Review + create`, then **Create**.

Step 2: Configure Networking

- Go to `Settings` → `Networking`
- Enable: `Allow public access to this resource through the internet using a public IP address`
- Under Firewall Rules:
  - Click `+ Add current client IP address (156.198.254.74)`
  - Check `Allow public access from any Azure service`
- Click `Save`

Step 3: Connect with MySQL Workbench

1. Open `MySQL Workbench`, click on `+` to add new connection.
2. Connection Name: `Team24`
3. Hostname: `team24.mysql.database.azure.com`, Port: `3306`
4. Username: `ahmed`
5. Click `Store in Vault` for password and enter it.
6. Click `Test Connection`, then click `OK` if successful.

8 Recommendations

Based on the insights gained during data validation and analysis, the following key recommendations are proposed to improve data quality, reporting accuracy, and business intelligence capabilities:

8.1 Improve Discount Data Quality

The `discounts` table contains multiple missing foreign keys (e.g., product_id, category_id, order_id), making it difficult to determine which discount was applied to which order. This reduces the reliability of sales performance analysis.

- Define clear discount rules and ensure consistent application logic.
- Enforce foreign key constraints to link discounts to specific products or categories.
- Consider removing ambiguous or incomplete discount records to improve data trustworthiness.

8.2 Impact of Missing Age and Gender Data on Identifying Active Customer Segments

The absence of both age and gender data in the database significantly limits our ability to identify the most active customer segments in product reviews. This gap prevents us from deriving meaningful demographic-based insights.

- Without age and gender data, it is challenging to determine which demographic groups are more likely to engage in reviewing products.
- Age-related insights could have helped understand generational preferences, while gender data could have revealed differences in purchasing behavior and feedback patterns.
- The lack of these critical data points reduces the depth of analysis and limits the ability to tailor marketing strategies or improve customer engagement.

8.3 Engaging Customers Who Returned Products with Positive Reviews

Analysis of customer reviews revealed that some customers who returned products still provided positive ratings and feedback. This indicates that while the product may have qualities appreciated by customers, issues such as fit, expectations, or delivery could have influenced their decision to return.

- It is recommended to proactively reach out to these customers to understand the specific reasons for returns.
- Collect detailed feedback to identify common pain points that could be addressed through product improvements or clearer communication.
- Use this insight to tailor marketing messages, improve sizing guides, or enhance after-sales support.
- Monitoring and addressing return causes can reduce return rates, improve customer satisfaction, and boost brand loyalty.

Implementing these actions will enhance customer experience and contribute to continuous product and service quality improvement.

8.4 Handling Fake Reviews

- Implement verification mechanisms to ensure reviews come from verified purchasers, such as linking reviews to completed payments.
- Monitor and flag suspicious review patterns, like multiple reviews from customers who never purchased the product.
- Use machine learning models to detect anomalous review behavior and potential fake reviews.
- Engage with customers who leave reviews without purchase to understand their intent and improve customer relations.
- Regularly audit and clean the reviews dataset to maintain data integrity for accurate product ratings.

8.5 Impact of Missing Delivery Date Data on Logistics and Customer Satisfaction Analysis

The absence of delivery date data in the database creates a significant gap in analyzing logistics performance and customer satisfaction. This missing information limits our ability to evaluate delivery timelines and their impact on customer experience.

- Without delivery date data, it is impossible to measure delivery efficiency or identify delays in the supply chain.
- Delivery timelines are critical for understanding customer satisfaction, as late deliveries can lead to negative feedback or returns.

- The lack of this data prevents us from correlating delivery performance with product reviews or return rates.

8.6 Impact of Missing Product Stock-In Date on Sales Timing Insights

The absence of data indicating when a product was first added to inventory significantly limits our ability to measure the time interval between product availability and its first sale. This restricts valuable operational and marketing insights.

- Without a clear stock-in date, we cannot determine how long a product remains unsold in inventory before the first transaction occurs.
- This metric is important for evaluating product performance, optimizing stock turnover, and identifying slow-moving items.
- It also limits our ability to study customer adoption trends and assess the effectiveness of initial marketing efforts.

Recommendations:

- Introduce a dedicated `stock_in_date` field in the product or inventory table to capture the first availability date for each product.
- Automate the logging of this date when a new product is first registered or imported into the system.
- Ensure this field is non-nullable and consistently recorded to support future analysis of product performance and shelf-life metrics.