Exploratory Data Analysis (EDA) Steps for Amazon Dataset

Step 1: Data Overview

- **Objective**: Initial exploration of the data tables to understand their structure and content.
- **Actions**:
    - Display all data from each table for initial exploration:
    - Identify distinct payment statuses in the Payments table:
        - SELECT DISTINCT payment_status FROM Payments;
    - Count rows with non-null return dates in the Shipping table:
        - SELECT COUNT(return_date) AS Non_Null_Returns FROM Shipping WHERE return_date IS NOT NULL;

Step 2: Data Quality Checks

- **Objective**: Identify data quality issues like missing values, duplicates, and foreign key mismatches.
- **Actions**:
    - Check for NULL values in critical columns across all tables:
        - Example:
        - SELECT 'Category' AS Table_Name, COUNT(*) AS Total_Rows, COUNT(category_id) AS Non_Null_Ids, COUNT(category_name) AS Non_Null_Names FROM Category
        - UNION
        - SELECT 'Customers', COUNT(*), COUNT(customer_id), COUNT(f_name) FROM Customers;
    - Detect duplicate primary keys in tables:
        - SELECT category_id, COUNT(*) FROM Category GROUP BY category_id HAVING COUNT(*) > 1;
    - Validate foreign key relationships:
        - Example:
        - SELECT p.category_id FROM Products p LEFT JOIN Category c ON p.category_id = c.category_id WHERE c.category_id IS NULL;

Step 3: Descriptive Statistics

- **Objective**: Calculate summary statistics to understand key metrics.
- **Actions**:
    - Analyze basic statistics for product prices:
        - SELECT MIN(price) AS Min_Price, MAX(price) AS Max_Price, AVG(price) AS Avg_Price, STDDEV(price) AS Stddev_Price FROM Products;
    - Evaluate the distribution of quantities in the Order_items table:
        - SELECT MIN(quantity) AS Min_Quantity, MAX(quantity) AS Max_Quantity, AVG(quantity) AS Avg_Quantity FROM Order_items;
    - Count distinct categories and customers:
        - SELECT COUNT(DISTINCT category_name) AS Unique_Categories FROM Category;
        - SELECT COUNT(DISTINCT f_name || ' ' || l_name) AS Unique_Customers FROM Customers;

Step 4: Relationship Exploration

- **Objective**: Understand relationships between key entities.
- **Actions**:
    - Category-Level Analysis: Count products per category:
        - SELECT c.category_name, COUNT(p.product_id) AS Product_Count FROM Category c LEFT JOIN Products p ON c.category_id = p.category_id GROUP BY c.category_name;
    - Customer Behavior: Count the number of orders per customer:
        - SELECT c.f_name || ' ' || c.l_name AS Customer_Name, COUNT(o.order_id) AS Total_Orders FROM Customers c LEFT JOIN Orders o ON c.customer_id = o.customer_id GROUP BY Customer_Name ORDER BY 1 LIMIT 5;
    - Seller Performance: Analyze total orders and average order value per seller:
        - SELECT s.seller_name, COUNT(o.order_id) AS Total_Orders, AVG(oi.price_per_unit * oi.quantity) AS Avg_Order_Value FROM Sellers s LEFT JOIN Orders o ON s.seller_id = o.seller_id LEFT JOIN Order_items oi ON o.order_id = oi.order_id GROUP BY s.seller_name;

Step 5: Trend Analysis

- **Objective**: Analyze trends over time and across geographies.
- **Actions**:
    - Order Trends Over Time: Monthly aggregation of orders:
        - SELECT DATE_TRUNC('month', order_date) AS Month, COUNT(order_id) AS Total_Orders FROM Orders GROUP BY Month ORDER BY Month;
    - State-Wise Customer Distribution:
        - SELECT state, COUNT(customer_id) AS Total_Customers FROM Customers GROUP BY state ORDER BY Total_Customers DESC;

Step 6: Anomalies and Outliers

- **Objective**: Detect anomalies and identify potential data issues.
- **Actions**:
    - Detect products with unusually high or low prices (outliers):
        - SELECT * FROM Products WHERE price < (SELECT AVG(price) - 3 * STDDEV(price) FROM Products) OR price > (SELECT AVG(price) + 3 * STDDEV(price) FROM Products);
    - Identify orders with missing shipping or payment information:
        - SELECT o.order_id, s.delivery_status, p.payment_status FROM Orders o LEFT JOIN Shipping s ON o.order_id = s.order_id LEFT JOIN Payments p ON o.order_id = p.order_id WHERE s.shipping_id IS NULL OR p.payment_id IS NULL;

Step 7: Visualization Preparation

- **Objective**: Summarize data for easy visualization and reporting.
- **Actions**:
    - Product Count and Total Sales by Category:
        - SELECT c.category_name, COUNT(p.product_id) AS Product_Count, SUM(oi.price_per_unit * oi.quantity) AS Total_Sales FROM Category c LEFT JOIN Products p ON c.category_id = p.category_id LEFT JOIN Order_items oi ON p.product_id = oi.product_id GROUP BY c.category_name;