|  |  |
| --- | --- |
| **Final Project Title** | **Customer Purchase Behavior in Retail using Data Analytics** |
| **Skills take away From This Project** | **Data analytics and visualization using Excel, SQL queries, Power BI dashboards, DAX, and interactive reports.** |
| **Domain** | **Retail and E-commerce Analytics** |

**Technical Document: MS Excel, MySQL and MS Power BI**

**By,**

**Naveen Ramesh**

**BADM-WD-T-B5**

**MS Excel: Data Cleaning, Preprocessing, and Formulas in Excel File**

## 1. Introduction

This document provides an analysis of the data cleaning, preprocessing, and formulas used in the provided Excel file (order\_items.xlsm). The file contains multiple sheets, with the order\_items-Dataset sheet serving as the primary dataset.

## 2. Data Cleaning Steps

### 2.1 Handling Missing Values

* **No missing values found, the Null values are replaced with Unknown.**

### 2.2 Duplicate Data

* No duplicate rows were found, ensuring data integrity.

### 2.3 Standardization

* **Date Formatting**: The Order Date field appears correctly formatted, likely as YYYY-MM-DD.
* **Email Consistency**: Checked for valid formatting in Staff Email and customer email.
* **Category and Brand Name Standardization**: Ensured consistency in text-based columns (Category\_name, Brand\_name).

## 3. Data Preprocessing Steps

* **Total Sales Calculation**: Derived using the formula:

Total Sales = quantity \* list\_price

* **Statistical Analysis Fields**:
  + Mean, Median, Mode, and Standard Deviation were applied to Total Sales.
  + These fields summarize central tendencies and variations in order amounts.
* **Pivot Tables & Charts**:
  + Likely created to analyze category-wise sales trends.
  + Includes YoY Growth Analysis and What-If Analysis.
* **Conditional Formatting**:
  + Used for highlighting discounts, sales outliers, or order patterns.
* **Lookup Functions (VLOOKUP)**:
  + Used for retrieving product, category, and store details from related sheets.
  + Helps maintain data consistency by fetching relevant information automatically.
* **What-If Analysis**:
  + Performed to analyze how changes in discount rates affect total sales.
  + Used Data Tables to model different discount scenarios.

### Steps for What-If Analysis:

1. **Create a List of Discount Rates**
   * Create a column with different discount rates (e.g., 5%, 10%, 15%, etc.).
2. **Reference Total Sales Calculation**
   * Use a formula in an adjacent column: =quantity \* list\_price \* (1 - selected\_discount).
3. **Use Data Table for Scenario Analysis**
   * Select the range containing discount values and corresponding total sales calculations.
   * Go to Data → What-If Analysis → Data Table.
   * Set the **Column Input Cell** to the original discount cell in the dataset.
   * Excel will recalculate Total Sales for each discount scenario.

## 4. Formulas Used in the Excel File

| **Column Name** | **Formula Used** | **Purpose** |
| --- | --- | --- |
| **Total Sales** | =quantity \* list\_price | Calculates revenue per order item |
| **Mean** | =AVERAGE(Total Sales range) | Computes average sales per order |
| **Median** | =MEDIAN(Total Sales range) | Finds the middle value of sales |
| **Mode** | =MODE(Total Sales range) | Determines the most frequently occurring sales amount |
| **Standard Deviation** | =STDEV(Total Sales range) | Measures variation in sales values |
| **YoY Growth Analysis** | =(Current Year Sales - Previous Year Sales) / Previous Year | Calculates year-over-year sales growth |
| **Conditional Formatting** | Applied for discount > 10% | Highlights high-discounted orders |
| **Lookup Functions (VLOOKUP)** | =VLOOKUP(value, table range, column index, FALSE) Eg: =VLOOKUP(D2,[categories.xlsx]categories'!A:B, 2, FALSE) | Retrieves related data (e.g., product name, category) from another table |
| **What-If Analysis** | Data Table using different discount rates | Analyzes how changes in discount rates affect total sales |

## 5. Summary

The Excel file implements comprehensive data preprocessing techniques, including data validation, calculated fields, and statistical summaries. Pivot tables and what-if analyses enhance data-driven decision-making. Lookup functions (VLOOKUP) help streamline data retrieval and ensure consistency. The document captures the key steps followed in structuring and analyzing the dataset.

**MySQL Project Technical Document**

**Database Name:** retail\_sales\_db  
**Objective:** The primary objective of this MySQL project is to design, implement, and query a relational database for a retail sales system, providing structured data storage and retrieval.

To create the database and table based on given data, I’ve used the below queries;

### ****Database Schema:****

#### **Database Creation**

sql

CopyEdit

CREATE DATABASE retail\_sales\_db;

USE retail\_sales\_db;

#### **1. Brands Table**

sql

CopyEdit

CREATE TABLE brands (

brand\_id INT PRIMARY KEY,

brand\_name VARCHAR(255) NOT NULL

);

#### **2. Categories Table**

sql

CopyEdit

CREATE TABLE categories (

category\_id INT PRIMARY KEY,

category\_name VARCHAR(255) NOT NULL

);

#### **3. Stores Table**

sql

CopyEdit

CREATE TABLE stores (

store\_id INT PRIMARY KEY,

store\_name VARCHAR(255),

phone VARCHAR(50),

email VARCHAR(255),

street VARCHAR(255),

city VARCHAR(255),

state VARCHAR(255),

zip\_code VARCHAR(50)

);

#### **4. Staffs Table**

sql

CopyEdit

CREATE TABLE staffs (

staff\_id INT PRIMARY KEY,

first\_name VARCHAR(255),

last\_name VARCHAR(255),

email VARCHAR(255),

phone VARCHAR(50),

active BOOLEAN,

store\_id INT,

manager\_id INT,

FOREIGN KEY (store\_id) REFERENCES stores(store\_id),

FOREIGN KEY (manager\_id) REFERENCES staffs(staff\_id) ON DELETE SET NULL

);

#### **5. Customers Table**

sql

CopyEdit

CREATE TABLE customers (

customer\_id INT PRIMARY KEY,

first\_name VARCHAR(255),

last\_name VARCHAR(255),

phone VARCHAR(50),

email VARCHAR(255),

street VARCHAR(255),

city VARCHAR(255),

state VARCHAR(255),

zip\_code VARCHAR(50)

);

#### **6. Products Table**

sql

CopyEdit

CREATE TABLE products (

product\_id INT PRIMARY KEY,

product\_name VARCHAR(255),

brand\_id INT,

category\_id INT,

model\_year INT,

list\_price INT,

FOREIGN KEY (brand\_id) REFERENCES brands(brand\_id),

FOREIGN KEY (category\_id) REFERENCES categories(category\_id)

);

#### **7. Orders Table**

sql

CopyEdit

CREATE TABLE orders (

order\_id INT PRIMARY KEY,

customer\_id INT,

order\_status VARCHAR(50),

order\_date DATE,

required\_date DATE,

shipped\_date DATE,

store\_id INT,

staff\_id INT,

FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id),

FOREIGN KEY (store\_id) REFERENCES stores(store\_id),

FOREIGN KEY (staff\_id) REFERENCES staffs(staff\_id)

);

#### **8. Order Items Table**

sql

CopyEdit

CREATE TABLE order\_items (

order\_id INT,

item\_id INT,

product\_id INT,

quantity INT,

list\_price INT,

discount DECIMAL(10,2),

SalesMade INT,

PRIMARY KEY (order\_id, item\_id),

FOREIGN KEY (order\_id) REFERENCES orders(order\_id) ON DELETE CASCADE,

FOREIGN KEY (product\_id) REFERENCES products(product\_id)

);

#### **9. Stocks Table**

sql

CopyEdit

CREATE TABLE stocks (

store\_id INT,

product\_id INT,

quantity INT,

PRIMARY KEY (store\_id, product\_id),

FOREIGN KEY (store\_id) REFERENCES stores(store\_id),

FOREIGN KEY (product\_id) REFERENCES products(product\_id)

);

This schema defines a **retail sales database** with **nine interrelated tables**, handling **products, sales, customers, orders, and inventory management**

Below are the MySQL tasks and the queries that I’ve used to bring the output based on the questions;

## ****SQL Questions, Queries, and Explanations****

### ****1. Retrieve Customers Who Placed Orders****

**Question:** Retrieve all details of customers who have placed orders.  
**Query:**

SELECT customers.\*, orders.order\_id, orders.order\_status

FROM customers

INNER JOIN orders ON customers.customer\_id = orders.customer\_id;

**Explanation:** This query uses an INNER JOIN to combine the customers and orders tables, selecting only customers who have placed an order.

### ****2. Calculate Total Sales for Each Store****

**Question:** Calculate the total sales for each store.  
**Query:**

SELECT stores.store\_name, SUM(order\_items.quantity \* order\_items.list\_price) AS total\_sales

FROM order\_items

JOIN orders ON order\_items.order\_id = orders.order\_id

JOIN stores ON orders.store\_id = stores.store\_id

GROUP BY store\_name;

**Explanation:** This query joins orders, order\_items, and stores tables to calculate total sales per store using SUM and GROUP BY.

### ****3. Find Products Never Ordered****

**Question:** Find the products that have never been ordered.  
**Query:**

SELECT product\_id, product\_name

FROM products

WHERE product\_id NOT IN (SELECT DISTINCT product\_id FROM order\_items)

ORDER BY product\_id ASC;

**Explanation:** This query uses a subquery to filter out products that appear in order\_items, identifying unpurchased products.

### ****4. Retrieve Staff Names and Their Managers****

**Question:** Retrieve the names and email addresses of staff along with their managers.  
**Query:**

SELECT CONCAT(s.first\_name, ' ', s.last\_name) AS StaffName, s.email,

CONCAT(m.first\_name, ' ', m.last\_name) AS ManagerName

FROM staffs s

LEFT JOIN staffs m ON s.manager\_id = m.staff\_id;

**Explanation:** A self-join is used to map employees (staffs) to their managers.

### ****5. Rank Stores Based on Sales****

**Question:** Rank stores based on their total sales.  
**Query:**

SELECT orders.store\_id, SUM(order\_items.quantity \* order\_items.list\_price) AS total\_sales,

RANK() OVER (ORDER BY SUM(order\_items.quantity \* order\_items.list\_price) DESC) AS store\_Rank

FROM order\_items

JOIN orders ON order\_items.order\_id = orders.order\_id

GROUP BY orders.store\_id;

**Explanation:** A window function (RANK) is used to assign a rank to stores based on their total sales.

### ****6. Calculate Days Each Order Took to Ship****

**Question:** Calculate the number of days each order took to ship.  
**Query:**

SELECT order\_id, order\_date, shipped\_date,

DATEDIFF(shipped\_date, order\_date) AS days\_to\_ship

FROM orders;

**Explanation:** The DATEDIFF function calculates the difference between the order date and shipped date.

### ****7. Categorize Orders Based on Status****

**Question:** Categorize orders based on their status.  
**Query:**

SELECT order\_id, order\_status,

CASE

WHEN order\_status = '1' THEN 'pending'

WHEN order\_status = '2' THEN 'processing'

WHEN order\_status = '3' THEN 'rejected'

WHEN order\_status = '4' THEN 'completed'

END AS status\_category

FROM orders;

**Explanation:** A CASE statement is used to categorize orders based on their order\_status value.

### ****8. Retrieve Orders with Product Names and Store Names****

**Question:** Retrieve all orders along with the product names and the store names.  
**Query:**

SELECT orders.order\_id, products.product\_name, stores.store\_name

FROM orders

JOIN stores ON orders.store\_id = stores.store\_id

JOIN order\_items ON orders.order\_id = order\_items.order\_id

JOIN products ON order\_items.product\_id = products.product\_id;

**Explanation:** This complex join retrieves orders along with corresponding product names and store names.

### ****9. Create and Use Temporary Table for Sales Calculation****

**Question:** Create a temporary table to store intermediate sales calculations.  
**Query:**

CREATE TEMPORARY TABLE TempJoin AS

SELECT order\_id, SUM(quantity \* list\_price) AS total\_by\_order

FROM order\_items

GROUP BY order\_id;

SELECT \* FROM TempJoin ORDER BY order\_id ASC;

DROP TEMPORARY TABLE TempJoin;

**Explanation:** A temporary table TempJoin is created to store order-level sales calculations for further analysis.

### ****10. Stored Procedure to Update Stock Quantities****

**Question:** Write a stored procedure to update stock quantities based on new shipments.  
**Query:**

DELIMITER $$

DROP PROCEDURE IF EXISTS UpdateStock;

CREATE PROCEDURE UpdateStock(IN p\_product\_id INT, IN p\_quantity\_received INT)

BEGIN

UPDATE stocks

SET quantity = quantity + p\_quantity\_received

WHERE product\_id = p\_product\_id;

END$$

DELIMITER ;

CALL UpdateStock(3, 50);

CALL UpdateStock(4, 100);

**Explanation:** This stored procedure updates stock levels when new shipments are received, modifying the stocks table dynamically.

## ****Conclusion****

This MySQL project implements a robust database structure for a retail sales system. It efficiently handles various database operations, including complex queries, aggregate functions, joins, subqueries, window functions, and stored procedures. The database ensures data integrity and supports efficient data retrieval for business insights.

# Power BI Data Model Structure & Visualization Techniques

## 1. Power BI Data Model Structure

### Tables and Relationships:

- Orders Table (order\_id) → Order Items Table (order\_id)

- Order Items Table (product\_id) → Products Table (product\_id)

- Products Table (category\_id) → Categories Table (category\_id)

- Products Table (brand\_id) → Brands Table (brand\_id)

- Order Items Table (store\_id) → Stores Table (store\_id)

- Order Items Table (order\_id) → Customers Table (customer\_id)

- Stocks Table (product\_id) → Products Table (product\_id)

- Stocks Table (store\_id) → Stores Table (store\_id)

- Staffs Table (store\_id) → Stores Table (store\_id)

## 2. Visualization Techniques Applied

### Basic Visualization: Bar Chart

Question: Create a bar chart to show total sales by product category.

Implementation: Import Orders, Order Items, and Products tables into Power BI. Create a Bar Chart with Product Category on the X-axis and Total Sales on the Y-axis.

DAX Formula:

Total Sales = SUM(Order\_Items[SalesMade])

### 3.Slicers

Question: Add slicers to filter sales data by store and date range.

Implementation: Add Slicer visuals to allow users to filter by Store Name and Date. Connect the slicers to the Sales table using relationships.

### 4.Calculated Columns: Profit Margin

Question: Create a calculated column to show the profit margin for each order item.

Implementation: Calculates the profit percentage for each order item.

DAX Formula:

Profit Margin = (Order\_Items[SalesMade] - Order\_Items[Cost\_Price]) / Order\_Items[SalesMade]

### 5.Measures: Average Order Value

Question: Create a measure to calculate the average order value.

Implementation: Divides the total sales by the number of orders to get the average order value.

DAX Formula:

Average Order Value = DIVIDE(SUM(Order\_Items[SalesMade]), COUNT(Order\_Items[Order\_ID]))

### 6.Conditional Formatting

Question: Apply conditional formatting to highlight top-performing stores.

Implementation: Use Conditional Formatting in the Sales Amount column. Apply a color scale where the highest sales are highlighted in green and the lowest in red.

### 7.Hierarchies: Date Hierarchy

Question: Create a hierarchy for drill-down analysis of sales data by year, month, and day.

Implementation: Create a Date Table with Year, Month, and Day columns. Use this hierarchy in a Line Chart to enable drill-down functionality.

### 8.Dashboards: Sales Metrics Dashboard

Question: Create an interactive dashboard to display key sales metrics.

Implementation: Combine Bar Charts, Line Charts, KPI Cards, and Slicers. Use a KPI visual to display total sales and performance trends.

### 9.Power Query: Data Cleaning & Transformation

Question: Clean and transform data from multiple tables before loading into Power BI.

Implementation: Remove duplicates, handle missing values, and ensure data types are correct using Power Query Editor. Merge queries to join Order Items with Products and Stores tables.

### 10.Custom Visuals: Geographic Sales Data

Question: Utilize custom visuals to display geographic sales data.

Implementation: Import a Map Visual to show sales distribution by store location. Use Latitude and Longitude data for precise mapping.

### 11.Key Performance Indicators (KPIs)

Question: Create KPIs to track sales targets versus actual sales.

Implementation: Use KPI visual to compare actual sales vs. predefined targets. Display trends using a goal-based metric visual.

DAX Formula:

Sales Target KPI = 100000

## Conclusion

This document provides a structured approach to building a Power BI report using best practices in data modeling, visualization, and DAX queries. Implementing these techniques enhances analytical capabilities, enabling effective decision-making.

**From Excel to Power BI**

**1. Load Summary Statistics Table**

**Excel Formulas:**

* **Mean (Average):** =AVERAGE(Last Price)
* **Median:** =MEDIAN(Last Price)
* **Mode:** =MODE.SNGL(Last Price)
* **Standard Deviation:** =STDEV.P(Last Price)

**Power BI:**

Imported the table in the Power BI

Used Card visualization to show the values accordingly.

**Visualizations:**

* Card visualizations for mean, median, mode, and standard deviation.
* Dashboard integration for product pricing overview.

**2. Load Pivot Table Data**

**Excel Pivot Table:**

* Created a pivot table summarizing total sales by store and product category.
* Imported the table in the Power BI

**Power BI Visualizations:**

* **Bar Chart:** Total sales by store.
* **Pie Chart:** Distribution of sales by product category.
* **Slicers:** Store and category filtering.

**3. Load Top 10 Products Data**

**Excel Formula for Top 10 Products:**

* Created Pivot table for Product ID and total sales.
* Imported the table in the Power BI

**Power BI Table Visualization:**

* Conditional formatting to highlight top 10 products.
* Dashboard integration for quick insights.

**4. Load Cleaned Customer Data**

**Power BI Map Visualization:**

* Imported the customer data table in the Power BI
* Map based on customer locations (City, State).
* Slicers for filtering by state and zip code.

**From SQL to Power BI**

**1. Load Customer Orders Data**

**MySQL Query:**

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

FROM orders

GROUP BY customer\_id;

Exported the output from the MySql workbench and imported the file in Power BI.

**Visualizations:**

* Clustered column chart for orders per customer.
* Slicer for filtering by order status.

**2. Load Total Sales by Store Data**

**MySQL Query:**

-- PRINT STORE ID & TOTAL SALES

SELECT orders.store\_id, SUM(order\_items.quantity\*order\_items.list\_price) as total\_sales FROM order\_items

JOIN orders ON order\_items.order\_id = orders.order\_id

GROUP BY orders.store\_id ORDER BY total\_sales DESC;

-- PRINT STORE NAME & TOTAL SALES

SELECT stores.store\_name, SUM(order\_items.quantity\*order\_items.list\_price) as total\_sales FROM order\_items

JOIN orders ON order\_items.order\_id = orders.order\_id

JOIN stores ON orders.store\_id = stores.store\_id

GROUP BY store\_name;

Exported the output from the MySql workbench and imported the file in Power BI.

**Power BI Visualization:**

* Bar chart for sales by store.
* Dashboard for sales performance.

**3. Load Products Never Ordered Data**

**MySQL Query:**

SELECT product\_id, product\_name FROM products WHERE product\_id NOT IN

(SELECT DISTINCT product\_id FROM order\_items) ORDER BY product\_id ASC;

Exported the output from the MySql workbench and imported the file in Power BI.

**Power BI Visualization:**

* Table of products never ordered.
* Filters for product category and brand.

**4. Load Staff and Manager Data**

**MySQL Query:**

SELECT

CONCAT(s.first\_name, ' ', s.last\_name) AS StaffName,

s.email,

CONCAT(m.first\_name, ' ', m.last\_name) AS ManagerName

FROM staffs s

LEFT JOIN staffs m ON s.manager\_id = m.staff\_id;

Exported the output from the MySql workbench and imported the file in Power BI.

**Power BI Matrix Visualization:**

* Staff hierarchy in an HR dashboard.

**5. Load Ranked Stores Data**

**MySQL Query:**

SELECT orders.store\_id, SUM(order\_items.quantity \* order\_items.list\_price) AS total\_sales,

RANK() OVER (ORDER BY SUM(order\_items.quantity \* order\_items.list\_price) DESC) AS store\_Rank

FROM order\_items JOIN orders ON order\_items.order\_id = orders.order\_id GROUP BY orders.store\_id;

Exported the output from the MySql workbench and imported the file in Power BI.

**Power BI Table Visualization:**

* Ranked list of stores.
* Conditional formatting for top-performing stores.

**Conclusion:**

This document provides a structured approach to loading data from Excel and SQL into Power BI, using Excel formulas, SQL queries, and Power BI DAX queries for analysis and visualization.