DATA CLEANING

MYSQL WORKBENCH

Denis Kombe
BS.ICT | THE DATA ANALYST/SCIENTIST

Table of Contents

1. Dataset Overview 2
2. Steps Undertaken 2
    2.1. Creating a Backup Table 2
    2.2. Adding a Sequential Customer ID 2
    2.3. Identifying and Removing Duplicates 3
    2.4. Dealing with Extra Spaces 4
    2.5. Correcting Spelling and Case Issues 5
    2.6. Updating Columns Based on Conditional Logic 6
    2.7. Recalculating the Total Price 6
    2.8. Normalizing the Spend Column 7
    2.9. Standardizing the Conversion Rate 7
    2.10. Date Format Correction 9
    2.11. Handling NULL and Blank Values 9
    2.12. Dropping the Row Number Column 10
Conclusion 10

```
SELECT * FROM sales_data3;
```

Data Cleaning Process for Sales Dataset

1. Dataset Overview

The dataset consists of sales transaction records including various columns such as CustomerID, OrderDate, Salesperson, ProductName, ProductCategory, Quantity, UnitPrice, TotalPrice, City, Country, Channel, Spend, LeadsGenerated, and ConversionRate. The initial dataset contained inconsistencies such as duplicates, extra spaces, incorrect data formats, and case issues that needed to be addressed to ensure data integrity for further analysis.

2. Steps Undertaken

2.1. Creating a Backup Table

```
CREATE TABLE sales_datac
LIKE sales_data3;

INSERT INTO sales_datac
SELECT * FROM sales_data3;
```

Reasoning: A backup table was created to ensure the original data remained intact. This allows for reversion if needed.

2.2. Adding a Sequential Customer ID

Reasoning: Introduced a sequential CustomerID to replace the old CustomerID column. This helps in maintaining unique identifiers.

```
-- Inserting a new column named customer_id and then dropping the CustomerID column
   for irrelevant data.alter
-- Step one: using a MySQL variable to generate and populate the column with
  sequential numbers
SET @seq_num = 0;

UPDATE sales_datac
SET customer_id = (@seq_num := @seq_num + 1)
ORDER BY (SELECT NULL);

-- step two: verying whether the customer_id column was populated correctly
SELECT customer_id FROM sales_datac ORDER BY customer_id;
```

```
-- Step three: dropping the CustomerID column
ALTER TABLE sales_datac
DROP COLUMN CustomerID;

-- Step four: Moving the newly added row "customer_id" to the first position in the
   table
ALTER TABLE sales_datac
MODIFY COLUMN customer_id INT FIRST;

-- Step five: Rename the customer_id back to CustomerID
ALTER TABLE sales_datac
CHANGE customer_id CustomerID INT;
```

2.3. Identifying and Removing Duplicates

Outcome: No duplicates were found in the dataset.

```
-- 1. IDENTIFYING DUPLICATES
SELECT *,
ROW_NUMBER() OVER(
PARTITION BY CustomerID, OrderDate, Salesperson, ProductName, ProductCategory,
  Quantity, UnitPrice_In_Dollars, TotalPrice_In_Dollars, City, Country, Channel,
  Spend, LeadsGenerated, ConversionRate) AS Row_Num
FROM sales_datac;

-- Use a CTE to identify any duplicates
WITH Duplicate_values AS
(
SELECT *,
ROW_NUMBER()  OVER(
PARTITION BY CustomerID, OrderDate, Salesperson, ProductName, ProductCategory,
  Quantity, UnitPrice_In_Dollars, TotalPrice_In_Dollars, City, Country, Channel,
  Spend, LeadsGenerated, ConversionRate) AS Row_Num
FROM sales_datac
)
SELECT * FROM Duplicate_values
WHERE Row_Num > 1;
```

```
-- creating a copy of the sales_datac table and adding the row_num column
CREATE TABLE `sales_datac1` (
  `CustomerID` int DEFAULT NULL,
  `OrderDate` text,
  `Salesperson`  text,
  `ProductName`  text,
  `ProductCategory` text,
  `Quantity` int DEFAULT NULL,
  `UnitPrice_In_Dollars` int DEFAULT NULL,
  `TotalPrice_In_Dollars` int DEFAULT NULL,
  `City` text,
  `Country` text,
  `Channel` text,
  `Spend` double DEFAULT NULL,
  `LeadsGenerated` int DEFAULT NULL,
  `ConversionRate` double DEFAULT NULL,
  `Row_Num` INT
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;


-- we need to populate this table
INSERT INTO sales_datac1
SELECT *,
ROW_NUMBER() OVER(
PARTITION BY CustomerID, OrderDate, Salesperson, ProductName, ProductCategory,
  Quantity, UnitPrice_In_Dollars, TotalPrice_In_Dollars, City, Country, Channel,
  Spend, LeadsGenerated, ConversionRate) AS Row_Num
FROM sales_datac;

SELECT * FROM sales_datac1 WHERE Row_Num > 1;
```

2.4. Dealing with Extra Spaces

Reasoning: Removed any leading or trailing spaces from various text columns to standardize the entries.

```
-- DEALING WITH WHITE/EXTRA SPACES
-- 1. Salesperson column
SELECT distinct Salesperson
FROM sales_datac1;

SELECT DISTINCT Salesperson, TRIM(Salesperson)
FROM sales_datac1;

-- remove the white spaces
UPDATE sales_datac1
SET Salesperson = TRIM(Salesperson);

-- 2. ProductName column
SELECT DISTINCT ProductName, TRIM(ProductName)
FROM sales_datac1;

UPDATE sales_datac1
SET ProductName = TRIM(ProductName);

-- 3. ProductCategory column
SELECT DISTINCT ProductCategory, TRIM(ProductCategory)
FROM sales_datac1;
```

```
UPDATE sales_datac1
SET ProductName = TRIM(ProductName);

-- 4. City column
SELECT DISTINCT City, TRIM(City)
FROM sales_datac1;

UPDATE sales_datac1
SET City = TRIM(City);

-- 5. Country column
SELECT DISTINCT Country, TRIM(Country)
FROM sales_datac1;

UPDATE sales_datac1
SET Country = TRIM(Country);

-- 6. Channel column
SELECT DISTINCT Channel, TRIM(Channel)
FROM sales_datac1;

UPDATE sales_datac1
SET Channel = TRIM(Channel);
```

2.5. Correcting Spelling and Case Issues

Reasoning: Addressed inconsistencies in the case sensitivity and corrected any misspellings to standardize the data.

```
-- DEALING SPELLING AND CASE ISSUES
UPDATE sales_datac1
SET Salesperson = 'Denis Kombe'
WHERE Salesperson = 'denisKombe';

UPDATE sales_datac1
SET Salesperson = 'Bob Johnson'
WHERE Salesperson = 'bob JohNson';

UPDATE sales_datac1
SET ProductName = 'Maasai Shuka'
WHERE ProductName = 'masaiShuka';

UPDATE sales_datac1
SET Country = 'Tanzania'
WHERE Country = 'TanZania';

-- DEALING WITH SPELLING & (UPPERCASE/LOWERCASE) ISSUES
-- USING THE CASE STATEMENT
-- CONDITIONAL STATEMENT
UPDATE sales_datac1
```

```
SET
Salesperson = CASE
                  WHEN Salesperson = 'denisKombe' THEN 'Denis Kombe'
                  WHEN Salesperson = 'bob JohNson' THEN 'Bob Johnson'
              ELSE Salesperson
            END,
ProductName = CASE
                  WHEN ProductName = 'masaiShuka' THEN 'Maasai Shuka'
              ELSE ProductName
            END,
Country = CASE
              WHEN Country = 'TanZania' THEN 'Tanzania'
          ELSE Country
        END;



SELECT * FROM sales_datac1;

SELECT City, Country
FROM sales_datac1;
```

2.6. Updating Columns Based on Conditional Logic

Reasoning: Updated the City column based on the corresponding Country value to ensure consistency.

```
-- Updating different columns with different conditions in a single SQL UPDATE
  Statement.
-- Using the CASE statement
UPDATE sales_datac1
SET
    City = CASE
                WHEN Country = 'Kenya' THEN 'Nairobi'
                WHEN Country = 'Tanzania' THEN 'Dar es Salaam'
                WHEN Country = 'Nigeria' THEN 'Lagos'
                WHEN Country = 'South Africa' THEN 'Johannesburg'
                WHEN Country = 'Ghana' THEN 'Accra'
                WHEN Country = 'Angola' THEN 'Luanda'
                WHEN Country = 'Ethiopia' THEN 'Addis Ababa'
                ELSE City
            END;
```

2.7. Recalculating the Total Price

Reasoning: Corrected the TotalPrice column to accurately reflect the product of Quantity and Unit Price.

```
-- updating the totalprice column based on the quantity column and unitprice column
```

```
UPDATE sales_datac1
SET TotalPrice_In_Dollars = Quantity * UnitPrice_In_Dollars;
```

2.8. Normalizing the Spend Column

Reasoning: Standardized the Spend per channel, ensuring the spend values were consistent and meaningful.

```
-- Reducing the decimal points in spend row to 0
SELECT ROUND(Spend, 0)
FROM sales_datac1;

UPDATE sales_datac1
SET Spend = ROUND(Spend, 0);

-- Modifying the Spend column from double to int
ALTER TABLE sales_datac1
MODIFY COLUMN Spend INT;

SELECT * FROM sales_datac1;
SELECT distinct ConversionRate, Channel FROM sales_datac1;

-- set constant spenditure per channel
UPDATE sales_datac1
SET
    Spend = CASE
                WHEN Channel='TikTok' THEN 60
                WHEN Channel='Facebook' THEN 50
                WHEN Channel='Instagram' THEN 55
                WHEN Channel='YouTube' THEN 70
                WHEN Channel='X_Twitter' THEN 65
            ELSE Spend
        END;
```

2.9. Standardizing the Conversion Rate

Reasoning: Ensured all conversion rates fell within a realistic range, enhancing data reliability.

```
-- Updating the conversion rate column values such that they all fall between 0.1
  and 0.9
UPDATE sales_datac1
SET ConversionRate = CASE
                        WHEN ConversionRate < 0.1 THEN 0.1
                        WHEN ConversionRate < 0.2 THEN 0.2
                        WHEN ConversionRate < 0.3 THEN 0.3
                        WHEN ConversionRate < 0.4 THEN 0.4
                        WHEN ConversionRate < 0.5 THEN 0.5
                        WHEN ConversionRate < 0.6 THEN 0.6
                        WHEN ConversionRate < 0.7 THEN 0.7
                        WHEN ConversionRate < 0.8 THEN 0.8
                        WHEN ConversionRate < 0.9 THEN 0.9
                     ELSE ConversionRate
                END;


-- set a unique conversion rate for each channel
UPDATE sales_datac1
SET ConversionRate = CASE
                         WHEN Channel='TikTok' THEN 0.6
                         WHEN Channel='Facebook' THEN 0.9
                         WHEN Channel='Instagram' THEN 0.8
                         WHEN Channel='YouTube' THEN 0.2
                         WHEN Channel='X_Twitter' THEN 0.5
            ELSE ConversionRate
        END;
```

2.10. Date Format Correction

Reasoning: Converted the OrderDate from text to a proper date format, facilitating accurate date-based analysis.

```
-- changing the date to the correct format
SELECT OrderDate, STR_TO_DATE(OrderDate, '%m/%d/%Y')
FROM sales_datac1;

-- update the table
UPDATE sales_datac1
SET OrderDate = STR_TO_DATE(OrderDate, '%m/%d/%Y');

-- Modify the table
ALTER TABLE sales_datac1
MODIFY column OrderDate DATE;

SELECT * FROM sales_datac1;
```

2.11. Handling NULL and Blank Values

```
-- checking for NULL or BLANK values per column
SELECT COUNT(*) AS NullCount, COUNT(*) AS BlankCount
FROM sales_datac1
WHERE ProductCategory IS NULL OR ProductCategory = '';
```

2.12. Dropping the Row Number Column

Reasoning: Removed the Row_Num column after it had served its purpose for identifying duplicates.

```
-- Dropping the Row_Num column
ALTER TABLE sales_datac1
DROP Column Row_Num;
```

Conclusion

The data cleaning process has significantly improved the quality and consistency of the dataset. The cleaned data is now well-structured, free from duplicates, and ready for further analysis like Exploratory Data Analysis which I did.

This process (Data Cleaning) ensured that the insights drawn from this data are accurate and reliable. This will help in forming a strong foundation for any subsequent data-driven decisions.