Data Cleaning and Analysis

This documentation outlines the processes and SQL queries used to clean and analyze sales data. It covers table creation, data loading, data quality checks, and various analyses to ensure the data is accurate and ready for further insights.

1. Creating the 'salesdata' Table

We create the 'salesdata' table with appropriate data types and constraints. This ensures data integrity and uniqueness for each transaction. It also allows for efficient storage and querying of sales data.

```
CREATE TABLE salesdata (
    TransactionID INT PRIMARY KEY AUTO_INCREMENT,
    TransactionDate DATE,
    CustomerID INT,
    CustomerAge INT,
    CustomerGender VARCHAR(10),
    ProductName VARCHAR(100),
    StoreType VARCHAR(50),
    City VARCHAR(50),
    Region VARCHAR(50),
    PaymentMethod VARCHAR(50),
    TransactionAmount DECIMAL(10, 2),
    Discount DECIMAL(5, 2),
    ReturnStatus VARCHAR(10),
    FeedbackScore INT
);
```

2. Loading Data from CSV

Data is imported from a CSV file into the 'salesdata' table. This step is essential to populate the database with raw sales data for further cleaning and analysis.

```
LOAD DATA INFILE 'salesdata.csv'
INTO TABLE salesdata
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
```

IGNORE 1 ROWS

(@TransactionDate, CustomerID, CustomerAge, CustomerGender, ProductName, StoreType, City, Region, PaymentMethod, TransactionAmount, Discount, ReturnStatus, FeedbackScore)

SET TransactionDate = NULLIF(@TransactionDate, ''),

TransactionAmount = NULLIF(TransactionAmount, ''),

Discount = NULLIF(Discount, '');

3. Viewing Records

This query is used to view all the records in the 'salesdata' table. It helps verify that data is successfully loaded.

SELECT * FROM salesdata;

4. Checking for NULL or Empty Values

This query identifies records with missing or empty values in any column. It is a crucial step to detect incomplete data entries.

SELECT * FROM salesdata WHERE TransactionDate IS NULL OR CustomerID IS NULL OR CustomerAge IS NULL OR CustomerGender = '' OR ProductName = '' OR StoreType = '' OR City = '' OR Region = '' OR PaymentMethod = '' OR TransactionAmount IS NULL OR Discount IS NULL;

5. Summary of Missing Values

This query provides a summary of missing values for each column. It helps quantify the extent of missing data.

SELECT 'TransactionDate' AS ColumnName, COUNT(*) AS MissingValues FROM salesdata WHERE TransactionDate IS NULL

UNION ALL

SELECT 'CustomerID', COUNT(*) FROM salesdata WHERE CustomerID IS NULL

UNION ALL

SELECT 'CustomerAge', COUNT(*) FROM salesdata WHERE CustomerAge IS NULL

UNION ALL

SELECT 'CustomerGender', COUNT(*) FROM salesdata WHERE CustomerGender = ''

UNION ALL

SELECT 'ProductName', COUNT(*) FROM salesdata WHERE ProductName = ''

UNION ALL

SELECT 'StoreType', COUNT(*) FROM salesdata WHERE StoreType = ''

UNION ALL

SELECT 'City', COUNT(*) FROM salesdata WHERE City = ''

```
UNION ALL

SELECT 'Region', COUNT(*) FROM salesdata WHERE Region = ''

UNION ALL

SELECT 'PaymentMethod', COUNT(*) FROM salesdata WHERE PaymentMethod = ''

UNION ALL

SELECT 'TransactionAmount', COUNT(*) FROM salesdata WHERE TransactionAmount IS NULL

UNION ALL

SELECT 'Discount', COUNT(*) FROM salesdata WHERE Discount IS NULL;
```

6. Forward Filling Missing Transaction Dates

This query fills missing transaction dates with the previous non-null value. It is especially useful when dates follow a sequential pattern.

```
SET @prev_date = NULL;

UPDATE salesdata

SET TransactionDate = COALESCE(TransactionDate, @prev_date),

   @prev_date = COALESCE(TransactionDate, @prev_date);
```

7. Identifying Duplicate Records

This query identifies customers with multiple transactions. It helps detect potential duplicate records.

```
SELECT CustomerID, COUNT(*)

FROM salesdata

GROUP BY CustomerID

HAVING COUNT(*) > 1;
```

8. Replacing Empty Strings with 'Unknown'

This query replaces empty strings in categorical columns with 'Unknown'. It standardizes data and avoids null-related issues.

```
UPDATE salesdata SET PaymentMethod = 'Unknown' WHERE PaymentMethod = '';

UPDATE salesdata SET StoreType = 'Unknown' WHERE StoreType = '';

UPDATE salesdata SET Region = 'Unknown' WHERE Region = '';

UPDATE salesdata SET CustomerGender = 'Unknown' WHERE CustomerGender = '';

UPDATE salesdata SET ProductName = 'Unknown' WHERE ProductName = '';
```

9. Imputing Missing Customer Ages

This query replaces missing CustomerAge values with the average age. It prevents data loss while maintaining data consistency.

```
SET @avg_age = (SELECT AVG(CustomerAge) FROM salesdata WHERE CustomerAge IS NOT NULL);

UPDATE salesdata

SET CustomerAge = @avg_age

WHERE CustomerAge IS NULL;
```

10. Customer Transaction Counts

This query counts the number of transactions per customer. It is useful for customer segmentation and sales behavior analysis.

```
SELECT CustomerID, COUNT(*) AS TransactionCount

FROM salesdata

GROUP BY CustomerID;
```

11. Customer Transaction Counts (Excluding CustomerID = 0)

This query retrieves the number of transactions made by each customer, excluding records where CustomerID is 0. This is useful for focusing on valid customer data and identifying active customers.

```
sql

CopyEdit

SELECT CustomerID, COUNT(*)

FROM salesdata

WHERE CustomerID != 0

GROUP BY CustomerID;
```

12. Checking for Invalid Transaction Amounts

This query identifies records with negative transaction amounts. Such values may indicate data entry errors.

```
SELECT * FROM salesdata WHERE TransactionAmount < 0;
```

13. Creating a View of Positive Transactions

This query creates a view containing only positive transactions. It simplifies further analysis by filtering out invalid records.

```
CREATE VIEW positive_salesdata AS

SELECT * FROM salesdata WHERE TransactionAmount > 0;
```

14. Descriptive Statistics

This query computes basic statistical measures for transaction amounts. It helps understand the distribution and variability of sales data.

```
SELECT    'TransactionAmount' AS column_name,
    COUNT(TransactionAmount) AS count,
    MIN(TransactionAmount) AS min,
    MAX(TransactionAmount) AS max,
    AVG(TransactionAmount) AS mean,
    STD(TransactionAmount) AS std
FROM positive_salesdata
UNION ALL
SELECT    'Quantity', COUNT(Quantity), MIN(Quantity), MAX(Quantity), AVG(Quantity), STD(Quantity)
FROM positive_salesdata
UNION ALL
SELECT    'DiscountPercent', COUNT(DiscountPercent), MIN(DiscountPercent), MAX(DiscountPercent),
AVG(DiscountPercent), STD(DiscountPercent)
FROM positive_salesdata
UNION ALL
SELECT    'CustomerAge', COUNT(CustomerAge), MIN(CustomerAge), MAX(CustomerAge),
AVG(CustomerAge), STD(CustomerAge)
FROM positive_salesdata
UNION ALL
SELECT    'LoyaltyPoints', COUNT(LoyaltyPoints), MIN(LoyaltyPoints), MAX(LoyaltyPoints),
AVG(LoyaltyPoints), STD(LoyaltyPoints)
FROM positive_salesdata
UNION ALL
SELECT    'FeedbackScore', COUNT(FeedbackScore), MIN(FeedbackScore), MAX(FeedbackScore),
AVG(FeedbackScore), STD(FeedbackScore)
FROM positive_salesdata
UNION ALL
SELECT    'ShippingCost', COUNT(ShippingCost), MIN(ShippingCost), MAX(ShippingCost),
AVG(ShippingCost), STD(ShippingCost)
FROM positive_salesdata
```

```
UNION ALL

SELECT     'DeliveryTimeDays', COUNT(DeliveryTimeDays), MIN(DeliveryTimeDays),
MAX(DeliveryTimeDays), AVG(DeliveryTimeDays), STD(DeliveryTimeDays)

FROM positive_salesdata;
```

15. Unique Value Counts for Categorical Columns

These queries provide counts of unique values in key categorical columns. They help analyze customer and product diversity.

```
SELECT PaymentMethod, COUNT(*) FROM positive_salesdata GROUP BY PaymentMethod;

SELECT City, COUNT(*) FROM positive_salesdata GROUP BY City;

SELECT StoreType, COUNT(*) FROM positive_salesdata GROUP BY StoreType;

SELECT CustomerGender, COUNT(*) FROM positive_salesdata GROUP BY CustomerGender;

SELECT ProductName, COUNT(*) FROM positive_salesdata GROUP BY ProductName;

SELECT Region, COUNT(*) FROM positive_salesdata GROUP BY Region;

SELECT Returned, COUNT(*) FROM positive_salesdata GROUP BY Returned;

SELECT IsPromotional, COUNT(*) FROM positive_salesdata GROUP BY IsPromotional;
```

16.Date Range Analysis

This query finds the minimum and maximum transaction dates. It determines the data coverage period.

```
SELECT MIN(TransactionDate) AS StartDate, MAX(TransactionDate) AS EndDate FROM salesdata;
```

17. Total Sales for Each Year and Month

This query calculates the total sales amount for each year and month, helping identify high-performing months.

```
SELECT YEAR(TransactionDate) AS year, MONTHNAME(TransactionDate) AS month,
SUM(TransactionAmount) AS total_sales

FROM positive_salesdata

GROUP BY year, month

ORDER BY total_sales DESC;
```

18. Total Sales for Each City

This query calculates the total sales amount for each city to identify the most profitable cities.

```
SELECT City, SUM(TransactionAmount) AS total_sales

FROM positive_salesdata

GROUP BY City
```

```
ORDER BY total_sales DESC;
```

19. Total Sales for Each Product

This query calculates the total sales amount for each product to identify the best-selling products.

```
SELECT ProductName, SUM(TransactionAmount) AS total_sales

FROM positive_salesdata

GROUP BY ProductName

ORDER BY total_sales DESC;
```

20. Total Sales and Transactions for Each Store Type

This query calculates the total sales amount and the number of transactions for each store type to analyze the performance of different store formats.

```
SELECT StoreType, SUM(TransactionAmount) AS total_sales, COUNT(*) AS transactions

FROM positive_salesdata

GROUP BY StoreType

ORDER BY total_sales DESC;
```

21. Total Sales and Transactions for Each Payment Method

This query calculates the total sales amount and the number of transactions for each payment method to understand customer preferences.

```
SELECT PaymentMethod, SUM(TransactionAmount) AS total_sales, COUNT(*) AS transaction_count

FROM positive_salesdata

GROUP BY PaymentMethod

ORDER BY total_sales DESC;
```

22. Total Sales and Transactions by Discount Range

This query categorizes discounts into ranges and calculates total sales and transactions for each range to analyze the impact of discounts on sales.

```
SELECT CASE

    WHEN DiscountPercent = 0 THEN 'No Discount'

    WHEN DiscountPercent > 0 AND DiscountPercent <= 10 THEN '0-10%'

    WHEN DiscountPercent > 10 AND DiscountPercent <= 20 THEN '10-20%'

    ELSE '>20%'

  END AS discount_range,

  SUM(TransactionAmount) AS total_sales,
```

```
    COUNT(*) AS transactions
FROM positive_salesdata
GROUP BY discount_range
ORDER BY total_sales DESC;
```

23. Total Sales and Transactions by Customer Age Group

This query groups customers into age ranges and calculates total sales and transactions for each group.

```
SELECT CASE
        WHEN CustomerAge < 18 THEN 'Under 18'
        WHEN CustomerAge BETWEEN 18 AND 25 THEN '18-25'
        WHEN CustomerAge BETWEEN 26 AND 35 THEN '26-35'
        WHEN CustomerAge BETWEEN 36 AND 50 THEN '36-50'
        ELSE 'Above 50'
    END AS age_group,
    SUM(TransactionAmount) AS total_sales,
    COUNT(*) AS transaction_count
FROM positive_salesdata
GROUP BY age_group
ORDER BY total_sales DESC;
```

24. Total Sales and Transactions by Return Status

This query calculates total sales and transactions based on whether an item was returned.

```
SELECT Returned, SUM(TransactionAmount) AS total_sales, COUNT(*) AS transactions
FROM positive_salesdata
GROUP BY Returned;
```

25. Month-over-Month Sales Growth

This query calculates the month-over-month sales growth percentage to track sales performance over time.

```
SELECT YEAR(TransactionDate) AS year, MONTH(TransactionDate) AS month,
SUM(TransactionAmount) AS total_sales,
    LAG(SUM(TransactionAmount)) OVER (ORDER BY YEAR(TransactionDate),
MONTH(TransactionDate)) AS previous_month_sales,
```

```
    ROUND(((SUM(TransactionAmount) - LAG(SUM(TransactionAmount)) OVER (ORDER BY
YEAR(TransactionDate), MONTH(TransactionDate))) / LAG(SUM(TransactionAmount)) OVER (ORDER
BY YEAR(TransactionDate), MONTH(TransactionDate))) * 100, 2) AS month_over_month_growth

FROM positive_salesdata

GROUP BY year, month

ORDER BY year, month;
```

26. Total Sales and Transactions for Each Region

This query calculates the total sales amount and the number of transactions for each region.

```
SELECT Region, SUM(TransactionAmount) AS total_sales, COUNT(*) AS transactions

FROM positive_salesdata

GROUP BY Region

ORDER BY total_sales DESC;
```

27. Exporting Cleaned Data

This query exports cleaned data to a CSV file. It enables sharing and further processing of refined sales data.

```
SELECT * FROM positive_salesdata INTO OUTFILE 'cleaned_salesdata.csv' FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n';
```