**Project Title:** **Online Retail Segmentation**

**Report Submitted by: Wali Jan**

**Introduction**

This project focuses on performing customer segmentation and analyzing transactional data for an online retail store. The dataset contains detailed information on sales transactions, including customer information, product details, and purchase history. The goal is to use SQL queries to uncover insights that can improve business decisions, such as identifying high-value customers, understanding product preferences, and preventing customer churn.

### Methodology

Explain how you set up your SQL environment, mentioning any specific tools (like MySQL Workbench) you used. Describe the schema creation and initial steps such as:

sql

Copy code

CREATE SCHEMA RetailShop;

USE RetailShop;

**SQL Queries Analysis**

**Query Descriptions and Procedural Aspects**

### 1. ****Basic Table Viewing****

SELECT \* FROM online\_retail;

* **Purpose**: To retrieve and review all data available in the online\_retail table.
* **Procedure**:
  + SELECT \*: This command selects all columns available in the table.
  + FROM online\_retail: Specifies the table from which data is being selected.

### 2. ****Column Structure and Data Types****

SELECT COLUMN\_NAME, DATA\_TYPE

FROM INFORMATION\_SCHEMA.COLUMNS

WHERE TABLE\_NAME = 'Online\_Retail';

* **Purpose**: To identify the names and data types of the columns in the Online\_Retail table.
* **Procedure**:
  + SELECT COLUMN\_NAME, DATA\_TYPE: Extracts the names of the columns and their respective data types.
  + FROM INFORMATION\_SCHEMA.COLUMNS: Accesses the metadata stored in the database's information schema, which contains details about all columns in the database.
  + WHERE TABLE\_NAME = 'Online\_Retail': Filters the columns to include only those from the Online\_Retail table.

### 3. ****Total Order Value by Customer****

SELECT CustomerID, SUM(UnitPrice \* Quantity) AS TotalOrderValue

FROM Online\_Retail

GROUP BY CustomerID

ORDER BY TotalOrderValue DESC;

* **Purpose**: To calculate the total money spent by each customer.
* **Procedure**:
  + SUM(UnitPrice \* Quantity) AS TotalOrderValue: Multiplies the unit price by the quantity for each purchase and sums these products to get the total expenditure for each customer.
  + GROUP BY CustomerID: This command groups all transactions by customer, so the sum is calculated for each customer individually.
  + ORDER BY TotalOrderValue DESC: Orders the results in descending order based on the total order value, showing the highest spenders first.

### 4. ****Unique Products per Customer****

SELECT CustomerID, COUNT(DISTINCT StockCode) AS UniqueProducts

FROM Online\_Retail

GROUP BY CustomerID

ORDER BY UniqueProducts DESC;

* **Purpose**: To determine the number of unique products purchased by each customer.
* **Procedure**:
  + COUNT(DISTINCT StockCode) AS UniqueProducts: Counts the number of different StockCode entries for each customer, ensuring no duplicates are counted.
  + GROUP BY CustomerID: Groups the results by customer ID to ensure the count is specific to each customer.
  + ORDER BY UniqueProducts DESC: Sorts the results to show customers with the greatest variety of purchases first.

### 5. ****One-time Purchase Customers****

SELECT CustomerID, COUNT(InvoiceNo) AS PurchaseCount

FROM Online\_Retail

GROUP BY CustomerID

HAVING COUNT(InvoiceNo) = 1;

* **Purpose**: To identify customers who have made only one purchase.
* **Procedure**:
  + COUNT(InvoiceNo) AS PurchaseCount: Counts the number of invoices associated with each customer.
  + GROUP BY CustomerID: This grouping ensures the count is done per customer.
  + HAVING COUNT(InvoiceNo) = 1: Filters the grouped results to only include those customers who have exactly one invoice, indicating a single purchase.

### 6. ****Most Frequent Products****

SELECT StockCode, COUNT(\*) AS Frequency

FROM Online\_Retail

GROUP BY StockCode

ORDER BY Frequency DESC

LIMIT 5;

* **Purpose**: To find the top five most frequently purchased products.
* **Procedure**:
  + COUNT(\*) AS Frequency: Counts the total number of times each StockCode appears in the dataset.
  + GROUP BY StockCode: Groups the data by product code so that each product's frequency can be calculated.
  + ORDER BY Frequency DESC: Sorts the products by their purchase frequency in descending order.
  + LIMIT 5: Limits the output to the top five records for easier analysis.

### 7. ****Customer Segmentation by Purchase Frequency****

SELECT CustomerID, COUNT(InvoiceNo) AS PurchaseFrequency,

CASE

WHEN COUNT(InvoiceNo) > 10 THEN 'High'

WHEN COUNT(InvoiceNo) BETWEEN 5 AND 10 THEN 'Medium'

ELSE 'Low'

END AS FrequencySegment

FROM Online\_Retail

GROUP BY CustomerID;

* **Purpose**: To categorize customers based on how frequently they make purchases.
* **Procedure**:
  + COUNT(InvoiceNo) AS PurchaseFrequency: Counts the number of purchases (invoices) per customer.
  + CASE: This statement classifies customers into 'High', 'Medium', or 'Low' frequency segments based on the number of invoices.
  + GROUP BY CustomerID: Ensures that the frequency count and segmentation are calculated per customer.

### 8. ****Average Order Value by Country****

SELECT Country, AVG(UnitPrice \* Quantity) AS AverageOrderValue

FROM Online\_Retail

GROUP BY Country

ORDER BY AverageOrderValue DESC;

* **Purpose**: To determine the average order value for transactions from each country.
* **Procedure**:
  + **Calculation of Average Order Value**: Multiplies the UnitPrice by the Quantity for each transaction, then calculates the average of these products for each country.
  + **Grouping**: Data is grouped by the Country field, ensuring that the average is calculated separately for each country.
  + **Sorting**: Results are ordered in descending order based on the average order value, allowing for quick identification of countries with higher average spending.

### 9. ****Customer Churn Analysis****

SELECT CustomerID

FROM Online\_Retail

WHERE InvoiceDate < DATE\_SUB(NOW(), INTERVAL 6 MONTH)

GROUP BY CustomerID;

* **Purpose**: To identify customers who have not made a purchase in the past six months, potentially indicating customer churn.
* **Procedure**:
  + **Date Filtering**: Uses the InvoiceDate to filter out transactions that occurred more than six months ago from the current date (NOW()), utilizing DATE\_SUB to calculate the time interval.
  + **Grouping**: Groups results by CustomerID to ensure each customer is listed once if they meet the condition, identifying those who may have churned.

### 10. ****Product Affinity Analysis****

SELECT a.StockCode AS ProductA, b.StockCode AS ProductB, COUNT(\*) AS Frequency

FROM Online\_Retail a

JOIN Online\_Retail b ON a.InvoiceNo = b.InvoiceNo AND a.StockCode <> b.StockCode

GROUP BY a.StockCode, b.StockCode

ORDER BY Frequency DESC

LIMIT 10;

* **Purpose**: To determine which pairs of products are most frequently purchased together.
* **Procedure**:
  + **Self-Join**: The table Online\_Retail is joined to itself to compare rows where the InvoiceNo is the same, indicating a single transaction, but where StockCode differs, meaning different products in the same purchase.
  + **Counting Pairs**: Counts each occurrence of the product pairs across all invoices.
  + **Grouping and Ordering**: Results are grouped by both ProductA and ProductB, ordered by the frequency of occurrence, and limited to the top 10 most frequent pairs.

### 11. ****Time-based Sales Analysis****

sql

Copy code

SELECT YEAR(InvoiceDate) AS Year, MONTH(InvoiceDate) AS Month, SUM(UnitPrice \* Quantity) AS TotalSales

FROM Online\_Retail

GROUP BY YEAR(InvoiceDate), MONTH(InvoiceDate)

ORDER BY Year, Month;

* **Purpose**: To analyze total sales, segmented by month and year, to observe trends and patterns over time.
* **Procedure**:
  + **Time Extraction and Grouping**: Extracts the year and month from InvoiceDate using YEAR() and MONTH() functions, grouping results by these units to analyze sales data over time.
  + **Sales Calculation**: Calculates the total sales for each period by multiplying UnitPrice by Quantity and summing up these values within each group.
  + **Ordering**: Orders the results chronologically by year and month to facilitate trend analysis.

The End