



## **High Impact Skills Development Program in Artificial Intelligence, Data Science, and Blockchain**

**Project Title:** Online Retail Segmentation.

**Module 06:** Data Mining

**Name :** Muhammad Aslam

**Roll-no :** SK23008

**Training center:** Skardu

**Section :** 02

**email :** [maslambaig009@gmail.com](mailto:maslambaig009@gmail.com)

**github link :** <https://github.com/Aslam009/ORS-Data-mining>

### **Beginner Queries:**

#### **1- Define meta data in mysql workbench**

##### **1. InvoiceNo:**

- Data Type: VARCHAR

##### **2. StockCode:**

- Data Type: VARCHAR (to store alphanumeric stock codes)

- Constraints: PRIMARY KEY ( StockCode is unique)

##### **3. Description:**

- Data Type: TEXT

- Constraints: None

##### **4. Quantity:**

- Data Type: INT

- Constraints: (constraint to ensure positive values)

5. InvoiceDate:

- Data Type: DATETIME or TIMESTAMP
- Constraints: None (not null)

6. UnitPrice:

- Data Type: DECIMAL
- Constraints: None (non-negative values)

7. CustomerID:

- Data Type: INT (assuming customer IDs are represented as integers)
- Constraints: FOREIGN KEY (if linked to a Customers table) or None

8. Country:

- Data Type: VARCHAR
- Constraints: None

**2:What is the distribution of order values across all customers in the dataset?**

```
SELECT CustomerID, SUM(Quantity * UnitPrice) AS TotalOrderValue
FROM OnlineRetail
GROUP BY CustomerID
ORDER BY TotalOrderValue DESC;
```

The screenshot shows a SQL IDE interface. The top menu bar includes File, Edit, View, Query, Database, Server, Tools, Scripting, and Help. The left sidebar displays a 'SCHEMAS' tree with nodes for dev\_db, Lab5, onlineRetailSegmentation, Organization, sakila, and sys. The 'onlineRetailSegmentation' node is expanded, showing 'OnlineRetail' under 'Tables'. The main query editor contains the following SQL code:

```

1 • SELECT * FROM onlineRetailSegmentation.OnlineRetail;
2
3 • SELECT CustomerID, SUM(Quantity * UnitPrice) AS TotalOrderValue
4   FROM OnlineRetail
5  GROUP BY CustomerID
6  ORDER BY TotalOrderValue DESC;

```

The 'Result Grid' at the bottom shows the results of the query, limited to 100 rows. The columns are '#', 'CustomerID', and 'TotalOrderValue'. The results are ordered by 'TotalOrderValue' in descending order.

#	CustomerID	TotalOrderValue
1	14646	116135.91999999977
2	18102	106601.55000000002
3	12346	77163.6
4	12415	74310.70999999996
5	17450	61165.74000000001
6	14156	49348.93000000003
7	14911	45223.860000000124
8	15749	44534.3
9	17511	40165.83999999997
10	15098	39916.5
11	16029	36746.26
12	15311	33875.310000000056
13	13694	32310.900000000005
14	15061	31280.619999999998
15	14298	30493.989999999925
16	13089	29530.920000000104
17	15769	27889.72
18	16684	25075.940000000002
19	13798	23632.259999999995

This SQL query get information from customers' total order values from the "OnlineRetail" table. The query calculates the sum of the product of quantity and unit price for each customer's transactions. The results are then grouped by customer and ordered in descending order based on the total order value.

### 3:How many unique products has each customer purchased?

```

SELECT CustomerID, COUNT(DISTINCT StockCode) AS UniqueProductCount
FROM OnlineRetail
GROUP BY CustomerID

```

The screenshot shows a SQL IDE with a query editor and a result grid. The query editor contains the following SQL code:

```

1 • SELECT * FROM onlineRetailSegmentation.OnlineRetail;
2
3 • SELECT CustomerID, COUNT(DISTINCT StockCode) AS UniqueProductCount
4   FROM OnlineRetail
5  GROUP BY CustomerID
  
```

The result grid displays the following data:

#	CustomerID	UniqueProductCount
1	12346	1
2	12347	75
3	12348	22
4	12350	17
5	12352	26
6	12353	4
7	12354	58
8	12355	13
9	12356	53
10	12359	135
11	12360	45
12	12361	10
13	12362	52
14	12363	16
15	12365	22
16	12370	125
17	12372	32
18	12373	14
19	12377	72

This SQL query retrieves information about the count of unique products that each customer has purchased from the "OnlineRetail" table. The query calculates the number of distinct product codes (StockCodes) associated with each customer's transactions. The results are grouped by customer.

**4: Which customers have only made a single purchase from the company?**

```

SELECT CustomerID, COUNT(DISTINCT InvoiceNo) AS PurchaseCount
FROM OnlineRetail
GROUP BY CustomerID
HAVING PurchaseCount = 1;
  
```

The screenshot shows a SQL IDE interface. The left sidebar displays a database schema tree with 'onlineRetailSegmentation' selected. The main editor shows a SQL query: `SELECT * FROM onlineRetailSegmentation.OnlineRetail;` followed by a multi-line query: `SELECT CustomerID, COUNT(DISTINCT InvoiceNo) AS PurchaseCount FROM OnlineRetail GROUP BY CustomerID HAVING PurchaseCount = 1;`. The bottom pane shows the 'Result Grid' with 19 rows of data. The columns are '#', 'CustomerID', and 'PurchaseCount'. The data shows 19 unique customers, each with a 'PurchaseCount' of 1.

#	CustomerID	PurchaseCount
1	12346	1
2	12350	1
3	12353	1
4	12354	1
5	12355	1
6	12360	1
7	12361	1
8	12363	1
9	12373	1
10	12380	1
11	12394	1
12	12401	1
13	12402	1
14	12406	1
15	12409	1
16	12421	1
17	12422	1
18	12426	1
19	12427	1

This SQL query get information from customers who have a single purchase from the "OnlineRetail" table. The query calculates the count of distinct invoice numbers (InvoiceNos) associated with each customer's transactions and filters the results to include only customers with a purchase count of 1.

**5: Which products are most commonly purchased together by customers in the dataset?**

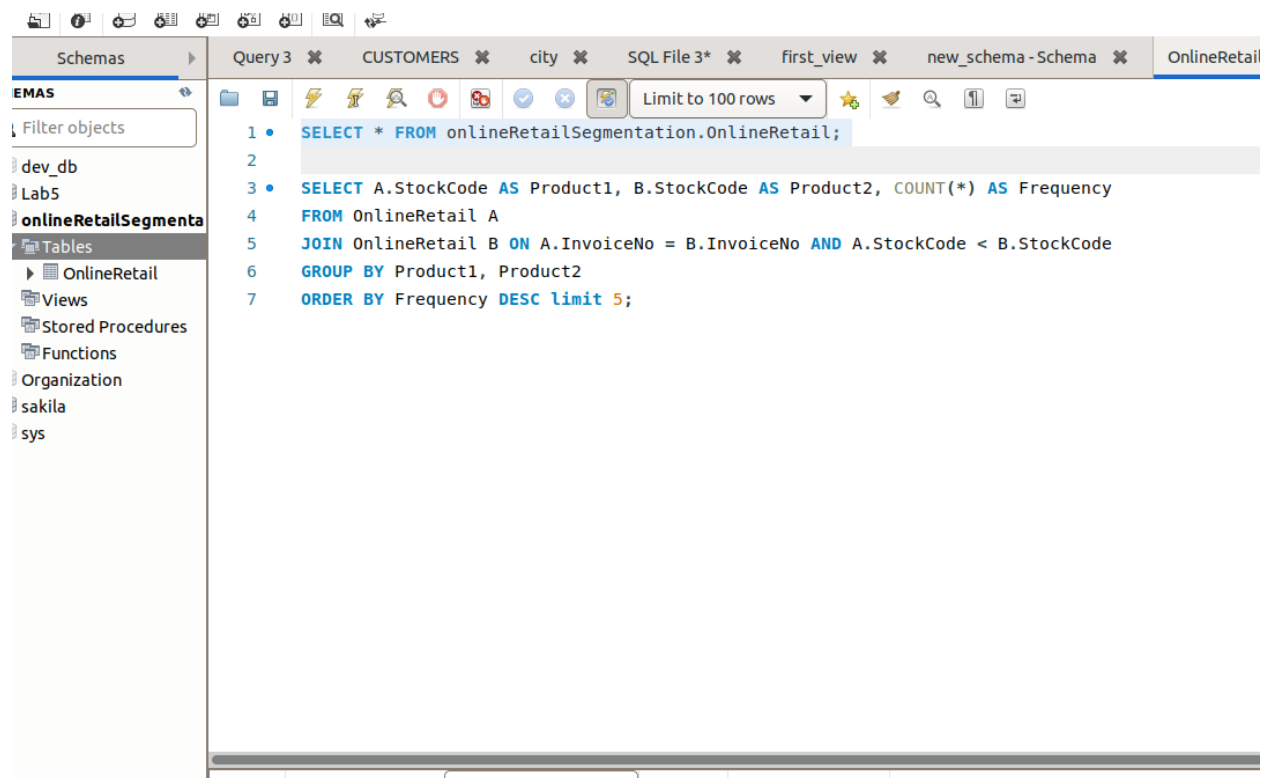
**SELECT A.StockCode AS Product1, B.StockCode AS Product2, COUNT(\*) AS Frequency**

**FROM OnlineRetail A**

**JOIN OnlineRetail B ON A.InvoiceNo = B.InvoiceNo AND A.StockCode < B.StockCode**

**GROUP BY Product1, Product2**

## ORDER BY Frequency DESC limit 5;



This SQL query is to find out the frequency of pairs of products that are commonly purchased together in the "OnlineRetail" table. The query performs a self-join on the "OnlineRetail" table to compare different products within the same invoices.

## Advance Queries :

### 1: Customer Segmentation by Purchase Frequency

```
SELECT CustomerID,  
       CASE  
         WHEN PurchaseCount > 5 THEN 'High Frequency'  
         WHEN PurchaseCount > 2 THEN 'Medium Frequency'  
         ELSE 'Low Frequency'  
       END AS PurchaseFrequencySegment  
FROM (  
       SELECT CustomerID, COUNT(DISTINCT InvoiceNo) AS PurchaseCount  
       FROM OnlineRetail
```

GROUP BY CustomerID

) AS CustomerPurchaseCounts;

The screenshot shows a SQL IDE interface. The left pane displays a schema tree with 'dev\_db' expanded, showing 'Lab5' and 'onlineRetailSegmentation'. The 'onlineRetailSegmentation' table is selected. The main editor shows a SQL query:

```
1 • SELECT * FROM onlineRetailSegmentation.OnlineRetail;
2
3 • SELECT CustomerID,
4       CASE
5         WHEN PurchaseCount > 5 THEN 'High Frequency'
6         WHEN PurchaseCount > 2 THEN 'Medium Frequency'
7         ELSE 'Low Frequency'
8       END AS PurchaseFrequencySegment
9 FROM (
10  SELECT CustomerID, COUNT(DISTINCT InvoiceNo) AS PurchaseCount
11  FROM OnlineRetail
12  GROUP BY CustomerID
13 ) AS CustomerPurchaseCounts;
```

The 'Result Grid' at the bottom shows the following data:

#	CustomerID	PurchaseFrequencySegment
1	12346	Low Frequency
2	12347	Medium Frequency
3	12348	Medium Frequency
4	12350	Low Frequency
5	12352	Medium Frequency
6	12353	Low Frequency
7	12354	Low Frequency
8	12355	Low Frequency
9	12356	Low Frequency
10	12359	Medium Frequency
11	12360	Low Frequency
12	12361	Low Frequency
13	12362	Low Frequency
14	12363	Low Frequency
15	12365	Low Frequency
16	12370	Medium Frequency

Query Completed

This query selects data from the subquery and applies a CASE statement to categorize customers based on their purchase frequency.

## 2: Average Order Value by Country

SELECT Country, AVG(TotalOrderValue) AS AverageOrderValue

FROM (

SELECT Country, InvoiceNo, SUM(Quantity \* UnitPrice) AS TotalOrderValue

FROM OnlineRetail

GROUP BY Country, InvoiceNo

) AS CountryOrderValues

GROUP BY Country

ORDER BY AverageOrderValue DESC;

The screenshot shows a SQL IDE interface with a menu bar (File, Edit, View, Query, Database, Server, Tools, Scripting, Help) and a toolbar. The 'Schemas' panel on the left lists databases like dev\_db, Lab5, and onlineRetailSegmentation. The main query editor displays a SQL query that calculates the average order value for each country. The 'Result Grid' at the bottom shows the output of the query, which is a table with columns for Country and AverageOrderValue. The results are sorted in descending order of AverageOrderValue.

**Query:**

```

1 • SELECT * FROM onlineRetailSegmentation.OnlineRetail;
2
3 • SELECT Country, AVG(TotalOrderValue) AS AverageOrderValue
4 FROM (
5     SELECT Country, InvoiceNo, SUM(Quantity * UnitPrice) AS TotalOrderValue
6     FROM OnlineRetail
7     GROUP BY Country, InvoiceNo
8 ) AS CountryOrderValues
9 GROUP BY Country
10 ORDER BY AverageOrderValue DESC;

```

**Result Grid:**

#	Country	AverageOrderValue
1	Netherlands	2983.5165
2	Australia	2850.2425000000003
3	Singapore	2565.6624999999995
4	Japan	1948.6118181818179
5	Lebanon	1693.8800000000003
6	Denmark	1559.3319999999999
7	Greece	1219.43
8	Brazil	1143.6000000000001
9	Sweden	1098.8853333333332
10	ELIRE	1064.5371739130435
11	Switzerland	1025.8186363636362
12	United Ar...	889.24
13	Channel I...	870.3766666666666
14	Cyprus	852.027
15	Malta	833.1600000000001
16	Israel	779.1
17	Spain	774.9645714285717
18	Finland	739.4399999999999
19	Norway	710.4162500000001

Result 9

Query Completed

This SQL query calculates the average order value for each country in the "OnlineRetail" dataset. It involves nested subqueries to first calculate the total order value for each invoice within each country, and then it calculates the average order value for each country.

### 3:Customer Churn Analysis

Identify customers who haven't made a purchase in a specific period (e.g., last 6 months) to assess churn.

```

SELECT CustomerID
FROM OnlineRetail
GROUP BY CustomerID
HAVING MAX(InvoiceDate) <= DATE_SUB(CURDATE(), INTERVAL 6 MONTH);

```



The screenshot shows a SQL IDE interface with a menu bar (File, Edit, View, Query, Database, Server, Tools, Scripting, Help) and a toolbar. The left pane displays a 'SCHEMAS' tree with folders for 'dev\_db', 'Lab5', 'onlineRetailSegmenta', 'Tables', 'Views', 'Stored Procedures', 'Functions', 'Organization', 'sakila', and 'sys'. The 'onlineRetailSegmenta' folder is expanded, showing 'OnlineRetail'. The main query editor contains the following SQL code:

```

1 • SELECT * FROM onlineRetailSegmentation.OnlineRetail;
2
3 • SELECT CustomerID
4 FROM OnlineRetail
5 GROUP BY CustomerID
6 HAVING MAX(InvoiceDate) <= DATE_SUB(CURDATE(), INTERVAL 6 MONTH);

```

The 'Result Grid' at the bottom shows the results of the query, with columns '#', 'CustomerID', and 'OnlineRetail 10'. The data is as follows:

#	CustomerID	OnlineRetail 10
1	17850	
2	13047	
3	12583	
4	13748	
5	15100	
6	15291	
7	14688	
8	17809	
9	15311	
10	16098	
11	18074	
12	17420	
13	16029	
14	16250	
15	12431	
16	17511	
17	13705	
18	13747	
19	13408	

The status bar at the bottom indicates 'Query Completed'.

#### 4:Product Affinity Analysis

**Determine which products are often purchased together by calculating the correlation between product purchases.**

```

SELECT
    DATE_FORMAT(InvoiceDate, '%Y-%m') AS Month,
    SUM(TotalOrderValue) AS TotalSales
FROM (
    SELECT
        InvoiceDate,
        SUM(Quantity * UnitPrice) AS TotalOrderValue
    FROM
        OnlineRetail
    GROUP BY
        InvoiceNo, InvoiceDate

```

) AS InvoiceTotals

GROUP BY

Month

ORDER BY

Month;

The screenshot shows a SQL IDE interface with a query editor and a results grid. The query editor contains the following SQL code:

```
1 • SELECT * FROM onlineRetailSegmentation.OnlineRetail;  
2  
3 • SELECT  
4     DATE_FORMAT(InvoiceDate, '%Y-%m') AS Month,  
5     SUM(TotalOrderValue) AS TotalSales  
6 FROM (  
7     SELECT  
8         InvoiceDate,  
9         SUM(Quantity * UnitPrice) AS TotalOrderValue  
10    FROM  
11        OnlineRetail  
12   GROUP BY  
13       InvoiceNo, InvoiceDate  
14 ) AS InvoiceTotals  
15 GROUP BY  
16     Month  
17 ORDER BY  
18     Month;
```

The results grid shows the following data:

#	Month	TotalSales
1	2008-01	3740974.221000003

Query Completed

This SQL query is used to summarize the total sales for each month in the "OnlineRetail" table. The query involves nested subqueries to first calculate the total order value for each invoice and then aggregates those values on a monthly basis. Here's a breakdown of each part of the query:

## 5:Time-based Analysis

**Explore trends in customer behavior over time, such as monthly or quarterly sales patterns.**

```
SELECT
    p1.StockCode AS Product1,
    p2.StockCode AS Product2,
    COUNT(DISTINCT p1.InvoiceNo) AS CoPurchaseCount
FROM
    OnlineRetail p1
JOIN
    OnlineRetail p2 ON p1.InvoiceNo = p2.InvoiceNo AND p1.StockCode <
    p2.StockCode
GROUP BY
    Product1, Product2
HAVING
    CoPurchaseCount > 10 -- Adjust the threshold as needed
ORDER BY
    CoPurchaseCount DESC
```

File Edit View Query Database Server Tools Scripting Help

Schemas Query 3 CUSTOMERS city SQL File 3\* first\_view new\_schema - Schema OnlineReta

Limit to 100 rows

```

1 • SELECT * FROM onlineRetailSegmentation.OnlineRetail;
2
3 • SELECT
4     p1.StockCode AS Product1,
5     p2.StockCode AS Product2,
6     COUNT(DISTINCT p1.InvoiceNo) AS CoPurchaseCount
7 FROM
8     OnlineRetail p1
9 JOIN
10    OnlineRetail p2 ON p1.InvoiceNo = p2.InvoiceNo AND p1.StockCode < p2.StockCode
11 GROUP BY
12     Product1, Product2
13 HAVING
14     CoPurchaseCount > 10 -- Adjust the threshold as needed
15 ORDER BY
16     CoPurchaseCount DESC;

```

Result Grid Filter Rows: Export: Wrap Cell Content: Fetch rows:

#	Product1	Product2	CoPurchaseCount
1	22697	22699	289
2	22386	85099B	236
3	22469	22470	235
4	22423	22699	230
5	20725	22383	228
6	22697	22698	223
7	20725	22384	220
8	20725	20727	219
9	21733	85123A	217
10	22698	22699	216
11	20725	20728	209
12	22720	22722	204
13	20725	22382	203

Result 13

Query Completed

This SQL query is used to identify pairs of products that are commonly purchased together in the "OnlineRetail" table. The query performs a self-join on the "OnlineRetail" table to compare different products within the same invoices and calculates the count of distinct invoices where each pair of products was co-purchased.

The End