

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

Project Title: Online Retail Segmentation.

Submitted By: Faiza Ali – Section 01

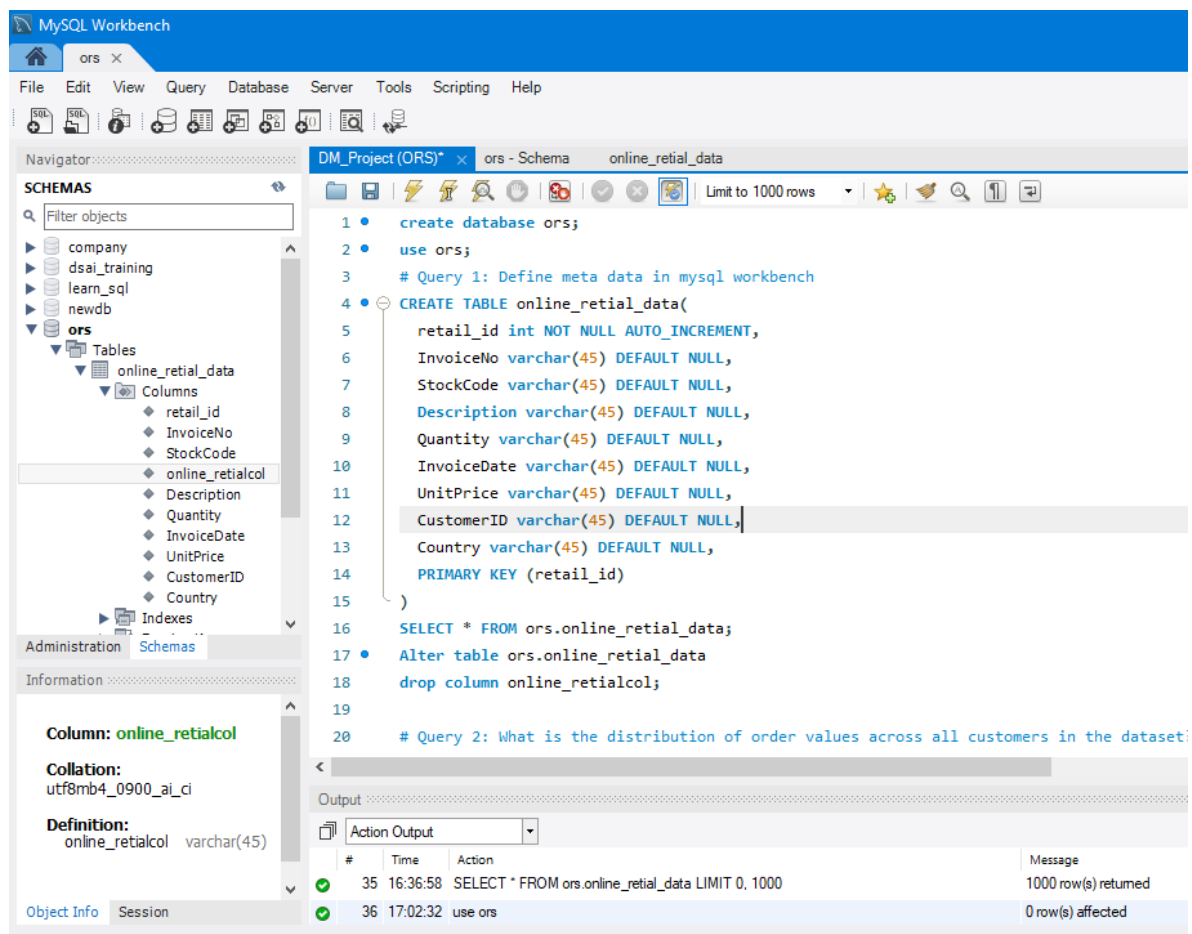
Roll No: GIL23047

GitHub Link: <https://github.com/FaizaAli-Dev/AI-DS-Projects>

Query 1: Meta Data

I have created database with name “ors” a new table with 8 new columns with same name as provided in dataset.

Imported the data from the option table > import wizard, successfully imported 30lk entries to this database, where retail_id is primary Key.



The screenshot displays the MySQL Workbench interface. On the left, the 'SCHEMAS' pane shows the 'ors' database selected, with the 'online_retail_data' table and its columns visible. The main editor shows the following SQL script:

```
1 • create database ors;
2 • use ors;
3 • # Query 1: Define meta data in mysql workbench
4 • CREATE TABLE online_retail_data(
5     retail_id int NOT NULL AUTO_INCREMENT,
6     InvoiceNo varchar(45) DEFAULT NULL,
7     StockCode varchar(45) DEFAULT NULL,
8     Description varchar(45) DEFAULT NULL,
9     Quantity varchar(45) DEFAULT NULL,
10    InvoiceDate varchar(45) DEFAULT NULL,
11    UnitPrice varchar(45) DEFAULT NULL,
12    CustomerID varchar(45) DEFAULT NULL,
13    Country varchar(45) DEFAULT NULL,
14    PRIMARY KEY (retail_id)
15 )
16 SELECT * FROM ors.online_retail_data;
17 • Alter table ors.online_retail_data
18 drop column online_retailcol;
19
20 # Query 2: What is the distribution of order values across all customers in the dataset;
```

The 'Output' pane at the bottom shows the execution results:

#	Time	Action	Message
35	16:36:58	SELECT * FROM ors.online_retail_data LIMIT 0, 1000	1000 row(s) returned
36	17:02:32	use ors	0 row(s) affected

Query 2: Distribution of order values across all customers

In this modified query, we're calculating the total order value (Quantity * UnitPrice) for each customer's orders and then summing those values. This query would give us insight into the distribution of order values across all customers in the dataset.

The screenshot shows the MySQL Workbench interface. The left sidebar displays the database schema, including the 'ors' database and the 'online_retail_data' table. The main editor window contains the following SQL query:

```
# Query 2: What is the distribution of order values across all customers in the dataset?
SELECT CustomerID, SUM(Quantity * UnitPrice) AS TotalOrderValue
FROM ors.online_retail_data
GROUP BY CustomerID;
```

The query results are displayed in a table with two columns: CustomerID and TotalOrderValue. The results are sorted by TotalOrderValue in descending order.

CustomerID	TotalOrderValue
17850	5391.2100000000009
13047	366.63000000000001
12583	855.86
13748	204
15100	569.4
15291	328.8
14688	444.98
17809	1251.5
15311	1857.83000000000008
14527	460.539999999999974
16098	430.599999999999997
18074	489.6
17420	130.85
16029	4271.52

The bottom status bar indicates that the query was completed successfully, returning 1000 rows.

Query 3: Number of unique products has each customer purchased

In the following query, I'm determining the count of distinct products that each customer has purchased. Using the 'GROUP BY' clause with the 'CustomerID' column, I group the data for each unique customer. The 'COUNT(DISTINCT StockCode)' function then calculates the number of different products (StockCodes) bought by each customer. This query offers insight into the variety of products bought by individual customers in the 'online_retail_data' table.

The screenshot displays the MySQL Workbench interface. The left sidebar shows the 'SCHEMAS' tree with 'ors' selected, and the 'online_retail_data' table is highlighted. The 'Columns' list for 'online_retail_data' is visible, including 'retail_id', 'InvoiceNo', 'StockCode', 'online_retailcol', 'Description', 'Quantity', 'InvoiceDate', 'UnitPrice', 'CustomerID', and 'Country'. The 'Information' tab shows details for the 'online_retailcol' column: 'Column: online_retailcol', 'Collation: utf8mb4_0900_ai_ci', and 'Definition: online_retailcol varchar(45)'. The main query editor shows the following SQL query:

```
# Query 3: How many unique products has each customer purchased?
SELECT CustomerID, COUNT(DISTINCT StockCode) AS UniqueProductsPurchased
FROM online_retail_data
GROUP BY CustomerID;
```

The 'Result Grid' shows the results of the query, with columns 'CustomerID' and 'UniqueProductsPurchased'. The results are as follows:

CustomerID	UniqueProductsPurchased
1984	1
12347	31
12370	82
12386	8
12395	12
12427	10
12429	20
12431	14
12433	79
12434	3
12441	11
12471	47
12472	77
12474	1
12476	2
174R1	10

The 'Output' tab shows the message: '38 17:11:55 SELECT CustomerID, count(Quantity) FROM ors.online_retail_data GROUP BY CustomerID... 760 row(s) returned'.

Query 4: Customers, who have only made a single purchase from the company

In this query, I'm identifying customers who have made a solitary purchase from the company. By utilizing the 'GROUP BY' clause on the 'CustomerID' column, the data is grouped based on individual customers. The 'COUNT(DISTINCT InvoiceNo)' function calculates the number of unique invoices for each customer. The 'HAVING' clause then filters the results to include only customers with exactly one unique invoice. This query reveals customers who have made a single purchase from the company in the 'online_retail_data' table."

The screenshot displays the MySQL Workbench interface. The left sidebar shows the 'Navigator' pane with a tree view of databases and tables. The 'online_retail_data' table is selected, and its columns are listed: retail_id, InvoiceNo, StockCode, Description, Quantity, InvoiceDate, UnitPrice, CustomerID, and Country. The main editor shows a SQL query for Query 4:

```
# Query 4: Which customers have only made a single purchase from the company?
SELECT CustomerID, COUNT(DISTINCT InvoiceNo) AS NumberOfPurchases, SUM(Quantity) AS TotalQuantityPurchased
FROM online_retail_data
GROUP BY CustomerID
HAVING COUNT(DISTINCT InvoiceNo) = 1;
```

The 'Result Grid' shows the output of the query, displaying columns CustomerID, NumberOfPurchases, and TotalQuantityPurchased. The results are as follows:

CustomerID	NumberOfPurchases	TotalQuantityPurchased
12347	1	319
12370	1	917
12386	1	214
12395	1	528
12427	1	79
12429	1	454
12431	1	107
12434	1	-13
12441	1	121
12474	1	-4
12481	1	85
12494	1	9
12540	1	300
12557	1	400
12583	1	449
12586	1	-2

The bottom status bar indicates 'Query Completed' and shows the execution time as 17:13:16, with a message stating '760 row(s) returned'.

Query 5: Products that are most commonly purchased together by customers

In this query, I'm identifying products that are frequently purchased together by customers within the dataset. By utilizing the 'GROUP BY' clause with 'InvoiceNo', I group the data by each unique transaction. The 'GROUP_CONCAT(DISTINCT Description)' function combines distinct product descriptions within each transaction. The 'COUNT(*)' function calculates the number of times these combined products appear together. The 'HAVING' clause filters the results to include only transactions with more than one product. This query unveils the most commonly co-purchased products in the 'online_retail_data' table, presenting them alongside the count of occurrences, and limited to the top 25 results.

The screenshot displays the MySQL Workbench interface. The left sidebar shows the 'SCHEMAS' pane with a tree view of the database structure, including 'company', 'dsai_training', 'learn_sql', 'newdb', and 'ors'. The 'ors' schema is selected, showing tables like 'online_retail_data'. The 'Columns' pane for 'online_retail_data' lists fields such as 'retail_id', 'InvoiceNo', 'StockCode', 'online_retailcol', 'Description', 'Quantity', 'InvoiceDate', 'UnitPrice', 'CustomerID', and 'Country'. The 'Information' pane shows details for the 'online_retailcol' column: 'Collation: utf8mb4_0900_ai_ci' and 'Definition: online_retailcol varchar(45)'. The main editor shows the following SQL query:

```
# Query 5: Which products are most commonly purchased together by customers in the dataset?
SELECT GROUP_CONCAT(DISTINCT Description) AS Products,
COUNT(*) AS Count_Products
FROM online_retail_data
GROUP BY InvoiceNo
HAVING COUNT(*) > 1
```

The 'Result Grid' shows the top 25 results of the query, ordered by 'Count_Products' in descending order. The results are as follows:

Products	Count_Products
CREAM CUPID HEARTS COAT HANGER, GLASS S...	7
HAND WARMER, RED POLKA DOT, HAND WARM...	2
ASSORTED COLOUR BIRD ORNAMENT, BOX OF ...	12
BLUE COAT RACK PARIS FASHION, JAM MAKIN...	4
SET 2 TEA TOWELS I LOVE LONDON, ALARM C...	20
HAND WARMER, RED POLKA DOT, HAND WARM...	2
CREAM CUPID HEARTS COAT HANGER, EDWAR...	16
CREAM CUPID HEARTS COAT HANGER, EDWAR...	16
HOT WATER BOTTLE TEA AND SYMPATHY, RED ...	2
HAND WARMER, RED POLKA DOT, HAND WARM...	2
60 TEATIME FAIRY CAKE CASES, BLUE 3 PIECE ...	19
AIRLINE LOUNGE, METAL SIGN, BALLOON ART M...	35
3 TIER CAKE TIN GREEN AND CREAM, 3 TIER C...	12
CLASSIC METAL BIRDCAVE PLANT HOLDER, CO...	13

The 'Output' pane at the bottom shows the execution of the query, with a message indicating '525 row(s) returned'.

Query 6: Customer Segmentation by Purchase Frequency

Group customers into segments based on their purchase frequency, such as high, medium, and low frequency customers. This can help you identify your most loyal customers and those who need more attention.

In this query, I'm segmenting customers based on their purchase frequency to identify different loyalty levels. By utilizing the 'GROUP BY' clause on the 'CustomerID' column, I group the data for each unique customer. The 'CASE' statement categorizes customers into segments: those with 10 or more distinct invoices are labeled as 'High' purchase frequency, those with 5 to 9 invoices as 'Medium', and the rest as 'Low'. This query allows me to assess customer loyalty and prioritize attention based on purchase behavior in the 'online_retail_data' table.

The screenshot displays the MySQL Workbench interface. On the left, the 'SCHEMAS' pane shows the database structure, including the 'online_retail_data' table. The main editor shows the following SQL query:

```
44 # Subqueries
45 # Query 6: Customer Segmentation by Purchase Frequency
46 # identify your most loyal customers and those who need more attention.
47 SELECT CustomerID,
48        CASE
49          WHEN COUNT(DISTINCT InvoiceNo) >= 10 THEN 'High'
50          WHEN COUNT(DISTINCT InvoiceNo) >= 5 THEN 'Medium'
51          ELSE 'Low'
52        END AS Purchase_Segment
53 FROM online_retail_data
54 GROUP BY CustomerID;
```

The 'Result Grid' shows the output of the query, with columns 'CustomerID' and 'Purchase_Segment'. The results are as follows:

CustomerID	Purchase_Segment
12347	High
12370	Low
12386	Low
12395	Low
12427	Low
12429	Low
12431	Low
12433	Low

The 'Output' pane at the bottom shows the query execution details, including the time taken (17:15:50) and the number of rows returned (25 row(s) returned).

Query 7: Average Order Value by Country

Calculate the average order value for each country to identify where your most valuable customers are located

In this query, I'm determining the average order value for each country to identify the most valuable customer locations. I first calculate the order values for each transaction by multiplying the 'Quantity' with 'UnitPrice', then group the data by both 'Country' and 'InvoiceNo'. Next, I aggregate these per-transaction order values into 'Country' groups and compute the average. By ordering the results in descending order of average order values, this query reveals the countries with the highest average order values in the 'online_retail_data' table.

The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'SCHEMAS' tree with 'ors' selected, showing the 'online_retail_data' table. The main editor displays the following SQL query:

```
# Query 7: Average Order Value by Country
# Calculate the average order value for each country to identify where your most valuable customers are located
SELECT Country,
AVG(Order_Values) AS Avg_Order_Values
FROM (
  SELECT Country, InvoiceNo, SUM(Quantity * UnitPrice) AS Order_Values
  FROM online_retail_data
  GROUP BY Country, InvoiceNo
) AS total_Orders
GROUP BY Country
ORDER BY Avg_Order_Values DESC;
```

The 'Result Grid' shows the following data:

Country	Avg_Order_Values
Japan	2568.3566666666666
Norway	1893.5600000000004
Cyprus	1590.82
Denmark	1281.5000000000002
Iceland	711.79
France	564.2438461538463
Spain	558.5766666666667
EIRE	535.9441666666668
Lithuania	415.265

The bottom status bar indicates 'Query Completed' and '760 row(s) returned'.

Query 8: Customer Churn Analysis

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

In this query we focus on customer churn analysis by identifying customers who haven't made a purchase in a defined period, like the last 6 months. It selects 'CustomerID' values from the 'online_retail_data' table where the 'InvoiceDate' is not within the specified timeframe. By grouping the data by 'CustomerID' and using the 'HAVING' clause with 'MAX(InvoiceDate)' to check against the interval, the query helps pinpoint customers who might have stopped purchasing within the last 6 months.

The screenshot displays the MySQL Workbench interface. The left sidebar shows the 'SCHEMAS' pane with a tree view of the database structure, including tables like 'company', 'dsai_training', 'learn_sql', 'newdb', and 'ors'. The 'ors' schema is selected, showing the 'online_retail_data' table. The main editor window shows a SQL query for product affinity analysis, which is Query 9. The query uses self-joins on the 'online_retail_data' table to calculate the correlation between product purchases. The results are displayed in a 'Result Grid' with columns 'Pro1', 'Pro2', and 'Correlation'. The bottom status bar indicates that the query was completed successfully, returning 11 rows.

```
75
76 # Query 9: Product Affinity Analysis
77 # Determine which products are often purchased together by calculating the correlation between product purchases.
78 • SELECT p1.Description AS Pro1,
79       p2.Description AS Pro2,
80       COUNT(DISTINCT o1.InvoiceNo) AS Correlation
81 FROM online_retail_data o1
82 JOIN online_retail_data o2 ON o1.InvoiceNo = o2.InvoiceNo AND o1.Description < o2.Description
83 JOIN online_retail_data p1 ON o1.InvoiceNo = p1.InvoiceNo AND p1.Description = o1.Description
84 JOIN online_retail_data p2 ON o2.InvoiceNo = p2.InvoiceNo AND p2.Description = o2.Description
85 GROUP BY p1.Description, p2.Description
86 ORDER BY Correlation DESC;
```

Pro1	Pro2	Correlation
PAPER CHAIN KIT 50'S CHRISTMAS	PAPER CHAIN KIT VINTAGE CHRISTMAS	72
KNITTED UNION FLAG HOT WATER BOTTLE	RED WOOLLY HOTTIE WHITE HEART.	63
HEART OF WICKER LARGE	HEART OF WICKER SMALL	54
KNITTED UNION FLAG HOT WATER BOTTLE	WHITE HANGING HEART T-LIGHT HOLDER	52
RED WOOLLY HOTTIE WHITE HEART.	WHITE HANGING HEART T-LIGHT HOLDER	52
CHOCOLATE HOT WATER BOTTLE	SCOTTIE DOG HOT WATER BOTTLE	52
RED WOOLLY HOTTIE WHITE HEART.	WHITE SKULL HOT WATER BOTTLE	51
RED WOOLLY HOTTIE WHITE HEART.	SCOTTIE DOG HOT WATER BOTTLE	51

Result 16 x

Output

Action Output

Time Action Message

49 14:14:47 SELECT YEAR(InvoiceDate) AS SalesYear, MONTH(InvoiceDate) AS SalesMonth, ... 11 row(s) returned

Query Completed

Query 9: Time-based Analysis

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

Here we are analyzing how customer behavior changes over time. By summing up the total price of each transaction and grouping them by their invoice dates. Then, these transaction totals are further aggregated by year and month. This helps reveal trends in sales over months and years, providing insights into customer spending patterns in the 'online_retail_data' table.

The screenshot displays the MySQL Workbench interface. The left sidebar shows the 'SCHEMAS' panel with a tree view containing 'company', 'dsai_training', 'learn_sql', 'newdb', and 'ors'. Under 'ors', there are 'Tables' and 'Indexes'. The 'online_retail_data' table is selected, showing columns: retail_id, InvoiceNo, StockCode, online_retailcol, Description, Quantity, InvoiceDate, UnitPrice, CustomerID, and Country. The 'Columns' panel for 'online_retailcol' shows its definition as 'varchar(45)' with 'utf8mb4_0900_ai_ci' collation.

The main editor shows a SQL query:

```
89 # Explore trends in customer behavior over time, such as monthly or quarterly sales patter
90 SELECT YEAR(InvoiceDate) AS SalesYear,
91        MONTH(InvoiceDate) AS SalesMonth,
92        SUM(T_Price) AS TotalSales
93 FROM (
94     SELECT InvoiceDate, SUM(Quantity * UnitPrice) AS T_Price
95     FROM online_retail_data
96     GROUP BY InvoiceDate
97 ) AS T_Invoice
98 GROUP BY SalesYear, SalesMonth
99 ORDER BY SalesYear, SalesMonth;
```

The 'Result Grid' shows the output of the query:

SalesYear	SalesMonth	TotalSales
2012	1	58635.560000000002
2012	2	46207.280000000003
2012	3	45620.460000000003
2012	5	31383.950000000008
2012	6	53860.18
2012	7	45059.050000000002
2012	8	44189.839999999998
2012	9	52532.130000000005

The bottom status bar indicates 'Query Completed'.