

REPORT ABOUT PROJECT

Muhammad Ali
NUST Campus Gilgit

Contents

About Me	2
Online Retail Segmentation Report	4
Introduction	4
Beginner Queries	4
Query 1: Meta Data	4
Query 2: Distribution of Order Values	5
Query 3: Unique Products per Customer.....	6
Query 4: Single Purchase Customers	7
Query 5: Frequently Purchased Product Pairs.....	8
Advanced Queries.....	9
1. Customer Segmentation by Purchase Frequency.....	9
2. Average Order Value by Country	10
3. Customer Churn Analysis	11
4. Product Affinity Analysis.....	12
5. Time-based Analysis.....	13
Conclusion.....	14

ABOUT ME

Name: Muhammad Ali

Project Title: Online Retail Segmentation

GitHub Profile Link : <https://github.com/Muhammad-Ali-56>





ONLINE RETAIL SEGMENTATION REPORT

Introduction

In this report, we analyze a dataset containing customer purchase information. The dataset provides insights into customer behaviors, purchase frequencies, order values, and product preferences.

BEGINNER QUERIES

Query 1: Meta Data

use mining_data;

#

Query 1: Meta Data

show tables;

select project_data;

*select * from project_data;*

We start by exploring the structure of the dataset. The show tables command reveals the tables present in the "mining_data" database. The subsequent queries retrieve and display all rows from the "project_data" table, allowing us to understand the data format and content.

- The **SHOW TABLES** command is used to list all the tables present in the current database (**mining_data** in this case).
- This query provides a quick overview of the available tables in the database, giving insight into the structure of the dataset.

The screenshot displays the MySQL Workbench interface. The SQL Editor at the top contains the following queries:

```
1 use mining_data;
2 #
3 show tables;
4 select project_data;
5 select * from project_data
6 limit 15;
```

The Results Grid below shows the output of the last query, displaying 15 rows of data from the **project_data** table. The columns are InvoiceNo, StockCode, Description, Quantity, InvoiceDate, UnitPrice, CustomerID, and Country.

InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/2010 8:26	2.55	17850	United Kingdom
536365	71053	WHITE METAL LANTERN	6	12/1/2010 8:26	3.39	17850	United Kingdom
536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/2010 8:26	2.75	17850	United Kingdom
536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/2010 8:26	3.39	17850	United Kingdom
536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/2010 8:26	3.39	17850	United Kingdom
536365	22752	SET 7 BABUSHKA NESTING BOXES	2	12/1/2010 8:26	7.65	17850	United Kingdom
536365	21730	GLASS STAR FROSTED T-LIGHT HOLDER	6	12/1/2010 8:26	4.25	17850	United Kingdom
536366	22633	HAND WARMER UNION JACK	6	12/1/2010 8:28	1.85	17850	United Kingdom
536366	22632	HAND WARMER RED POLKA DOT	6	12/1/2010 8:28	1.85	17850	United Kingdom
536367	84879	ASSORTED COLOUR BIRD ORNAMENT	32	12/1/2010 8:34	1.69	13047	United Kingdom
536367	22745	POPPY'S PLAYHOUSE BEDROOM	6	12/1/2010 8:34	2.1	13047	United Kingdom
536367	22748	POPPY'S PLAYHOUSE KITCHEN	6	12/1/2010 8:34	2.1	13047	United Kingdom
536367	22749	FELTCRAFT PRINCESS CHARLOTTE DOLL	8	12/1/2010 8:34	3.75	13047	United Kingdom
536367	22310	IVORY KNITTED MUG COSY	6	12/1/2010 8:34	1.65	13047	United Kingdom
536367	84969	BOX OF 6 ASSORTED COLOUR TEASPOONS	6	12/1/2010 8:34	4.25	13047	United Kingdom

The interface also shows the Schemas pane on the left, the Object Info pane at the bottom left, and the Results Grid pane at the bottom right. The status bar at the bottom indicates the query is completed and the session is active.

Query 2: Distribution of Order Values

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

```
select CustomerID, sum(Quantity * UnitPrice) as Total_Order_Value
```

```
from project_data
```

```
group by CustomerID
```

```
order by Total_Order_Value desc
```

```
limit 15;
```

This query examines the distribution of order values across customers. By calculating the total order value for each customer, we identify the top 15 customers with the highest order values. This insight can help identify the most valuable customers in terms of their purchasing power.

- The query groups the data by **CustomerID**.
- It calculates the total order value for each customer by multiplying **Quantity** with **UnitPrice** and summing the results.
- The **GROUP BY** clause ensures that the calculations are performed per customer.
- The **ORDER BY** clause arranges the results in **descending order** of total order value.
- The **LIMIT 15** clause restricts the output to the top 15 customers with the highest order values.

The screenshot shows the MySQL Workbench interface. The SQL editor contains the following query:

```
# Query 2: What is the distribution of order values across all customers in the dataset?
select CustomerID, sum(Quantity * UnitPrice) as Total_Order_Value
from project_data
group by CustomerID
order by Total_Order_Value desc
limit 15;
```

The Result Grid displays the following data:

CustomerID	Total_Order_Value
16029	3702.12
16210	2474.7399999999993
17511	1825.74
17850	1499.3399999999999
13408	1024.6800000000003
12583	855.86
13694	842.12
14307	783.1099999999999
17920	514.4099999999999
13767	507.8800000000001
18074	489.6
16218	471.29999999999995
15311	445.33
14688	444.98
13448	443.96000000000004

The interface also shows the 'project_data' table structure in the left sidebar:

```
Table: project_data
Columns:
InvoiceNo text
StockCode text
Description text
Quantity bigint
InvoiceDate text
UnitPrice double
CustomerID text
```

Query 3: Unique Products per Customer

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

select CustomerID, count(distinct StockCode) as Unique_Product_Count

from project_data

group by CustomerID

order by 2 desc;

Here, we investigate how many unique products each customer has purchased. The results provide an understanding of customers' variety in product preferences. Customers with a high count of unique products may exhibit diverse interests.

- Similar to the previous query, this query groups the data by **CustomerID**.
- It uses the **COUNT(DISTINCT StockCode)** function to count the number of unique products each customer has purchased.
- The results are **ordered in descending order** of unique product count
- The **2** in **ORDER BY** denotes **Unique_Product_count**

MySQL Workbench

Project File - Warning - not su... x

File Edit View Query Database Server Tools Scripting Help

Navigator: Data Mining Project* SQL File 3* x

Limit to 1000 rows

1 # Query 3: How many unique products has each customer purchased?
 2 select CustomerID, count(distinct StockCode) as Unique_Product_Count
 3 from project_data
 4 group by CustomerID
 5 order by 2 desc
 6 limit 15;
 7

Result Grid

CustomerID	Unique_Product_Count
17968	74
14729	69
15862	64
12838	59
17920	54
17908	51
14307	48
15311	35
17897	31
16218	28
15983	28
17511	24
15012	24
12583	20
17850	20

Result 14 x

Output

Query Completed

12:27 AM 8/19/2023

Query 4: Single Purchase Customers

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

select CustomerID, count(distinct InvoiceNo) as Number_Of_Purchases

from project_data

group by CustomerID

having Number_Of_Purchases = 1

order by Number_Of_purchases

limit 10;

Identifying customers who have made only a single purchase can be crucial for targeted marketing efforts. This query lists customers who have made just one purchase, helping to identify potential areas for improvement in customer retention.

- The query groups the data by **CustomerID**.
- It uses the **COUNT(DISTINCT InvoiceNo)** function to count the number of distinct invoices (purchases) each customer has made.
- The **HAVING** clause filters the results to include only those customers who have made exactly **one purchase**.
- The results are **ordered by** the number of purchases and **limited to the top 10**.

The screenshot shows the MySQL Workbench interface. The SQL Editor contains the following query:

```

select CustomerID, count(distinct InvoiceNo) as Number_Of_Purchases
from project_data
group by CustomerID
having Number_Of_Purchases = 1
order by Number_Of_purchases
limit 10;

```

The Result Grid displays the following data:

CustomerID	Number_Of_Purchases
12431	1
12583	1
12748	1
12791	1
12838	1
12855	1
12908	1
12991	1
13005	1
13705	1

The left sidebar shows the database schema with the 'project_data' table selected. The bottom status bar indicates 'Query Completed'.

Query 5: Frequently Purchased Product Pairs

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

```
select t1.CustomerID, t1.InvoiceNo, t1.StockCode as Product_1, t2.StockCode as Product_2, count(*) as Frequency
```

```
from project_data as t1
```

```
join project_data as t2 on t1.InvoiceNo=t2.InvoiceNo and t1.StockCode < t2.StockCode
```

```
group by Product_1, Product_2, InvoiceNO, CustomerID
```

```
order by Frequency desc
```

```
limit 10;
```

By analyzing products commonly purchased together, we can uncover valuable cross-selling opportunities. This query identifies pairs of products frequently purchased together by customers. Such insights can inform product bundling strategies.

- This query uses a **self-join** on the **project_data** table, creating two different aliases (**t1** and **t2**) to reference the same table.
- The join condition ensures that the rows being joined have the same **InvoiceNo** but different **StockCode** values.
- The query groups the data by **Product_1**, **Product_2**, **InvoiceNo**, and **CustomerID**.
- The **COUNT(*)** function calculates the **frequency** of each product pair being purchased together.
- The **results** are **ordered** in **descending order of frequency** and **limited to the top 1**

The screenshot displays the MySQL Workbench interface. The SQL editor contains the following query:

```
1 # Query 5: Which products are most commonly purchased together by customers in the dataset?
2 select t1.CustomerID, t1.InvoiceNo, t1.StockCode as Product_1, t2.StockCode as Product_2, count(*) as Frequency
3 from project_data as t1
4 join project_data as t2 on t1.InvoiceNo=t2.InvoiceNo and t1.StockCode < t2.StockCode
5 group by Product_1, Product_2, InvoiceNO, CustomerID
6 order by Frequency desc
7 limit 10;
```

The Result Grid shows the following data:

CustomerID	InvoiceNo	Product_1	Product_2	Frequency
17920	536412	21448	22749	15
17920	536412	21448	22243	15
17920	536412	21448	85049E	15
17920	536412	21448	21738	15
17920	536412	21448	22077	15
17920	536412	21448	22273	15
17920	536412	21448	22940	10
17920	536412	21448	22759	10
17920	536412	21448	22327	10
17920	536412	21448	22900	10

The left sidebar shows the Schemas pane with the 'project_data' table selected. The bottom status bar indicates 'Query Completed'.

ADVANCED QUERIES

1. Customer Segmentation by Purchase Frequency

1. Customer Segmentation by Purchase Frequency

```
select CustomerID,
       case
         when count(distinct InvoiceDate) > 10 then 'High'
         when count(distinct InvoiceDate) > 5 then 'Medium'
         else 'Low'
       end as Purchase_Frequency
from project_data
group by CustomerID
order by Purchase_Frequency desc
limit 10;
```

Segmenting customers based on their purchase frequency provides valuable insights into customer engagement. This query categorizes customers as "**High**," "**Medium**," or "**Low**" frequency based on their purchase history.

- Uses a **CASE** statement to categorize customers based on the number of distinct **InvoiceDate** values.
- Results are **ordered by** purchase **frequency**, showing which customers fall into **high**, **medium**, or **low frequency** categories.

The screenshot displays the MySQL Workbench interface. The query editor shows the SQL query for customer segmentation. The results grid shows the output of the query, listing CustomerID and Purchase_Frequency.

CustomerID	Purchase_Frequency
17850	Medium
12431	Low
12583	Low
12748	Low
12791	Low
12838	Low
13047	Low
13255	Low
13408	Low
13448	Low

The interface also shows the Navigator panel on the left with the 'mining_data' schema selected, and the 'project_data' table highlighted. The bottom status bar indicates 'Query Completed'.

2. Average Order Value by Country

2. Average Order Value by Country

```
select Country, avg(Quantity * UnitPrice) as Avarage_Order_Value  
from project_data  
group by Country;
```

Understanding average order values by country can assist in tailoring marketing strategies to different regions. This query calculates the average order value for each country represented in the dataset.

- Groups the data by **Country**.
- Calculates the **average order value** by multiplying **Quantity** with **UnitPrice** and taking the **average**.
- Provides insights into the spending patterns of different countries.

The screenshot displays the MySQL Workbench interface. The SQL Editor window shows the following query:

```
1 # 2. Average Order Value by Country
2 select Country, avg(Quantity * UnitPrice) as Avarage_Order_Value
3 from project_data
4 group by Country;
```

The Results window shows the output of the query as a table with two columns: Country and Avarage_Order_Value. The data is as follows:

Country	Avarage_Order_Value
United Kingdom	23.11303482587075
France	42.793
Australia	25.589285714285715
Netherlands	96.30000000000001

The left sidebar shows the Schemas pane with the 'project_data' table selected under the 'mining_data' database. The bottom status bar indicates 'Query Completed'.

3. Customer Churn Analysis

#

3. Customer Churn Analysis

```

select CustomerID
from project_data
where InvoiceDate <= date_sub(now(), interval 6 month)
group by CustomerID
limit 15;

```

Customer churn analysis is essential for customer retention efforts. This query identifies customers who haven't made purchases in the last six months, potentially indicating churn.

- Filters the data to include only customers **who haven't made purchases** in the last 6 months.
- Groups the data by **CustomerID**.
- Identifies customers who might have churned (**stopped purchasing**) based on their recent activity.
- **Limiting up to 15** due to large Data.

The screenshot displays the MySQL Workbench interface. The SQL Editor window shows the following query:

```

1 #
2 select CustomerID
3 from project_data
4 where InvoiceDate <= date_sub(now(), interval 6 month)
5 group by CustomerID
6 limit 15;

```

The Result Grid shows the output of the query, listing 15 CustomerIDs:

CustomerID
17850
13047
12583
13748
15100
15291
14688
17809
15311
14527
16098
18074
17420
16029
16250

The left sidebar shows the Schemas pane with the 'mining_data' database selected, containing tables like 'project_data'. The bottom status bar indicates 'Query Completed'.

4. Product Affinity Analysis

#

4. Product Affinity Analysis

select t1.CustomerID, t1.InvoiceNo, t1.StockCode as Product_1, t2.StockCode as Product_2, count(*) as Frequency

from project_data as t1

join project_data as t2 on t1.InvoiceNo=t2.InvoiceNo and t1.StockCode < t2.StockCode

group by Product_1, Product_2, InvoiceNO, CustomerID

order by Frequency desc

limit 10;

Similar to Query 5, this query further explores product affinity by examining pairs of products commonly purchased together by customers.

- ❖ This query uses a **self-join** on the **project_data** table, creating two different aliases (**t1** and **t2**) to reference the same table.
- ❖ The join condition ensures that the rows being joined have the same **InvoiceNo** but different **StockCode** values.
- ❖ The query groups the data by **Product_1**, **Product_2**, **InvoiceNo**, and **CustomerID**.
- ❖ The **COUNT(*)** function calculates the frequency of each product pair being purchased together.
- ❖ The results are **ordered in descending order of frequency** and **limited to the top 10**

The screenshot shows the MySQL Workbench interface. The SQL Editor contains the following query:

```
# 4. Product Affinity Analysis
select t1.CustomerID, t1.InvoiceNo, t1.StockCode as Product_1, t2.StockCode as Product_2, count(*) as Frequency
from project_data as t1
join project_data as t2 on t1.InvoiceNo=t2.InvoiceNo and t1.StockCode < t2.StockCode
group by Product_1, Product_2, InvoiceNO, CustomerID
order by Frequency desc
limit 10;
```

The Result Grid shows the following data:

CustomerID	InvoiceNo	Product_1	Product_2	Frequency
17920	536412	21448	22749	15
17920	536412	21448	22243	15
17920	536412	21448	85049E	15
17920	536412	21448	21738	15
17920	536412	21448	22077	15
17920	536412	21448	22273	15
17920	536412	21448	22940	10
17920	536412	21448	22759	10
17920	536412	21448	22327	10
17920	536412	21448	22900	10

The left sidebar shows the Schemas pane with the 'mining_data' database selected, containing tables 'project_data', 'sakila', 'sys', and 'world'. The bottom status bar indicates 'Query Completed'.

5. Time-based Analysis

5. Time-based Analysis

```
select date_format(InvoiceDate,'d-m-y-h-m') as Month_,
       count(distinct CustomerID) as Unique_Customers,
       sum(Quantity * UnitPrice) as Total_Revenue
from project_data
group by Month_
order by Month_;
```

Analyzing sales trends over time is crucial for understanding seasonality and making informed business decisions. This query presents a time-based analysis by grouping data by months and summarizing unique customer counts and total revenue for each month.

- Uses the **DATE_FORMAT** function to extract month, day, year, and time components from **InvoiceDate**.
- Groups the data by formatted **date**, providing insights into monthly trends in terms of **unique customers** and **total revenue**

The screenshot displays the MySQL Workbench interface. The SQL Editor window shows the following query:

```
# 5. Time-based Analysis
select date_format(InvoiceDate,'d-m-y-h-m') as Month_,
       count(distinct CustomerID) as Unique_Customers,
       sum(Quantity * UnitPrice) as Total_Revenue
from project_data
group by Month_
order by Month_;
```

The Results window shows the output of the query:

Month_	Unique_Customers	Total_Revenue
51	51	24635.310000000092

The left sidebar shows the Schemas pane with the 'project_data' table selected. The bottom status bar indicates 'Query Completed'.

CONCLUSION

The analysis of the dataset provides valuable insights into customer behavior, purchase patterns, and potential business opportunities. The various queries shed light on customer segmentation, product affinity, and time-based trends. These insights can guide marketing strategies, customer retention efforts, and product recommendations to enhance business success.

The End...

