

Final Project Report

Retail Sales Analytics Using Excel, SQL, Power BI and Python

Project work done by
Thaslima Banu. S

Course
Data Analytics

Introduction:

Retail Sales Analytics aims to analyze sales performance, customer behaviour, product trends, and staff contribution using Excel, SQL, Python and Power BI. The goal is to build an interactive dashboard that supports data-driven decisions for improving profitability and operational efficiency.

Phase 1: Excel-Data Cleaning & Preparation

Q1. Standardize Column Headers – Replaces spaces with underscore.

Q2. Remove Duplicates: No Duplicate records were found while checking by – Data-Remove Duplicates.

Q3. Handle Missing Values

Customer Table:

1. Null values in phone column were replaced with ‘Not Available’ to maintain completeness while retaining customer records.
2. Replaced null values by selecting the column-Home-Find and Replace option-Find-NULL, Replace-Not Available.

customer_id	first_name	last_name	phone	email	street	city	state	zip
1	Debra	Burks	Not Available	debra.burks@yahoo.com	9273 Thorne Ave.	Orchard Park	NY	14215
2	Lashah	Ortiz	Not Available	lashah.ortiz@gmail.com	910 Victoria Street	Carmel	CA	93923
3	Taneeka	Fisher	Not Available	taneeka.fisher@aol.com	368 Pearl Lane	Redondo Beach	CA	90277
4	Daryl	Spence	Not Available	daryl.spence@aol.com	107 River Dr.	Uniondale	NY	11046
5	Charlotte	Rice	(916) 381-6003	charlotte.rice@msn.com	769 West Road	Sacramento	CA	95821
6	Lyndsey	Bean	Not Available	lyndsey.bean@hotmail.com	7014 Market Station Rd.	Fairport	NY	14450
7	Latahia	Hays	(716) 986-3359	latahia.hays@yahoo.com	7014 Market Station Rd.	Buffalo	NY	14215
8	Jessica	McLean	Not Available	jessica.mclean@gmail.com	1000 Main St.	Jackson Heights	NY	11372
9	Genevieve	Baldwin	Not Available	genevieve.baldwin@gmail.com	4550 Spruce Drive	Port Washington	NY	11060
10	Pamela	Newman	Not Available	pamelia.newman@gmail.com	478 Chestnut Ave.	Monroe	NY	10950
11	Deshawn	Mendoza	Not Available	deshawn.mendoza@yahoo.com	9790 Cobblestone Street	Monsey	NY	10952
12	Robby	Sykes	(516) 583-7761	robby.sykes@hotmail.com	486 Rock Maple Street	Hempstead	NY	11550
13	Lashawn	Orritz	Not Available	lashawn.orritz@gmail.com	27 Wellington Drive	Longview	TX	75606
14	Gerry	Esparmoza	Not Available	gerry.esparmoza@hotmail.com	7000 Runaway Court	Forney	TX	75120
15	Unnie	Branch	Not Available	unnie.branch@gmail.com	314 South Columbia Ave.	Plattsburgh	NY	12901
16	Emmie	Sanchez	(212) 945-8823	emmiet.sanchez@hotmail.com	461 Squaw Creek Road	New York	NY	10002
17	Caren	Stephens	Not Available	caren.stephens@msn.com	914 Brook St.	Scarsdale	NY	10583
18	Georgetta	Hardin	Not Available	georgetta.hardin@aol.com	4727 Green Hill Lane	Canarsie	NY	11234
19	Uzzette	Stonley	Not Available	uzzette.stonley@aol.com	100 Green Hill Lane	Orchard Park	NY	14207
20	Aleta	Shepard	Not Available	aleta.shepard@aol.com	684 Howard St.	Sugar Land	TX	77478
21	Tobie	Uttle	Not Available	tobie.little@gmail.com	10 Silver Spear Dr.	Victoria	TX	77904
22								

Orders Table:

3. In Ship_date – blank values are replaced with ‘Not yet shipped’

order_id	customer_id	order_status	order_date	required_by	shipped_date
1	259	4	01-01-16	01-11-16	01-11-16
2	1212	4	01-01-16	01-12-16	1
3	523	4	01-02-16	01-12-16	2
4	175	4	01-03-16	01-12-16	3
5	1334	4	01-04-16	01-12-16	4
6	94	4	01-04-16	01-12-16	5
7	324	4	01-04-16	01-12-16	6
8	1204	4	01-04-16	01-12-16	7
9	60	4	01-05-16	01-12-16	8
10	442	4	01-05-16	01-12-16	9
11	1326	4	01-05-16	01-12-16	10
12	91	4	01-06-16	01-12-16	11
13	873	4	01-08-16	01-12-16	12
14	258	4	01-09-16	01-12-16	13
15	450	4	01-09-16	01-12-16	14
16	552	4	01-12-16	01-15-16	15
17	1175	4	01-12-16	01-14-16	16
18	541	4	01-14-16	01-17-16	17
19	696	4	01-14-16	01-17-16	18
20	923	4	01-14-16	01-17-16	19
21	1250	4	01-15-16	01-18-16	20
22					

Staffs Table:

- Replaced Null value in manager column with 'o'.

The screenshot shows a Microsoft Excel spreadsheet titled 'staffs'. The 'Replace' dialog box is open over the data. The 'Find what' field is set to 'NULL' and the 'Replace with' field is set to 'o'. The 'Replace All' button is visible at the bottom left of the dialog.

Q4. Data Type Conversion.

Products Table:

- List_price changed to number

The screenshot shows a Microsoft Excel spreadsheet titled 'products'. The 'Format Cells' dialog box is open over the data, specifically for the 'list_price' column. The 'Number' tab is selected, and the 'list_price' format is chosen from the dropdown menu. Other options like 'Currency', 'Short Date', and 'Long Date' are also visible.

Orders Table:

- List_price changed to number.
- Quantity to Number
- Standardized order_date, require_date, and shipped_date to Date Format by right clicking the column – Format cells – Date.

The screenshot shows a Microsoft Excel spreadsheet titled 'orders'. The 'Format Cells' dialog box is open over the data, specifically for the 'order_date' column. The 'Date' category is selected, and the 'Wednesday, March 7, 2001' format is chosen from the dropdown menu. Other date formats like 'Wednesday' and '3-7-2001' are also visible.

Q5. Data Validation

1. Standardized order_status values for consistency.
 - i) 1 – Pending.
 - ii) 2 – Processing.
 - iii) 3 – Cancelled.
 - iv) 4 – Shipped/Delivered.

The screenshot shows a Microsoft Excel spreadsheet titled 'orders.csv'. The 'order_status' column is currently selected. A 'Replace' dialog box is open over the spreadsheet, with 'Find what' set to '4' and 'Replace with' set to 'Shipped'. The rest of the spreadsheet contains various order details like order_id, customer_id, order_date, required_date, shipped_date, store_id, and staff_id.

2. Select the order_status column- Data-Data validation-List.

The screenshot shows the same Microsoft Excel spreadsheet as before, but now the 'Data Tools' ribbon tab is active. The 'order_status' column is still selected. The rest of the spreadsheet remains the same with various order details.

Q6. Create Derived Column

Order_Items Table:

1. Created Revenue Column
2. Revenue = List_price*Quantity-Discount.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	order_id	item_id	product_id	quantity	list_price	discount	Revenue											
2	1	1	20	1	599.99	0.20	599.79											
3	1	2	8	2	1799.99	0.07	3599.91											
4	1	3	10	2	1549	0.05	3097.95											
5	1	4	16	2	599.99	0.05	1199.93											
6	1	5	4	1	2899.99	0.20	5799.98											
7	2	1	20	1	599.99	0.07	599.92											
8	2	2	16	2	599.99	0.05	1199.93											
9	3	1	3	1	999.99	0.05	999.94											
10	3	2	20	1	599.99	0.05	599.94											
11	4	1	2	749.99	0.1	1499.88												
12	5	1	10	2	1549	0.05	3097.95											
13	5	2	17	1	429	0.07	428.93											
14	5	3	26	1	599.99	0.07	599.92											
15	6	1	18	1	449	0.07	448.93											
16	6	2	12	2	599.99	0.05	1199.93											
17	6	3	20	1	599.99	0.10	599.89											
18	6	4	3	2	999.99	0.07	1999.91											
19	6	5	9	2	2999.99	0.07	5999.91											
20	7	1	15	1	529.99	0.07	529.92											
21	7	2	3	1	999.99	0.10	999.89											
22	7	3	17	2	429.00	0.10	857.90											

Q7. Merge Lookup Data

1. I used VLOOKUP in order_items, added new column product_name using product_id.
2. =VLOOKUP(C2,products.csv!\$A:\$B,2, FALSE)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O		
1	order_id	item_id	product_id	quantity	list_price	discount	Revenue	product_name									
2	1	1	20	1	599.99	0.20	599.79	Electra Townie Original 7D EQ - Women's - 2016									
3	1	2	8	2	1799.99	0.07	3599.91	Trek Remedy 29 Carbon Frameset - 2016									
4	1	3	10	2	1549	0.05	3097.95	Surly Straggler - 2016									
5	1	4	16	2	599.99	0.05	1199.93	Electra Townie Original 7D EQ - 2016									
6	1	5	4	1	2899.99	0.20	5799.98	Electra Townie Original 7D EQ - 2016									
7	2	1	20	1	599.99	0.07	599.92	Electra Townie Original 7D EQ - Women's - 2016									
8	2	2	16	2	599.99	0.05	1199.93	Electra Townie Original 7D EQ - 2016									
9	3	1	3	1	999.99	0.05	999.94	Surly Wednesday Frameset - 2016									
10	3	2	20	1	599.99	0.05	599.94	Electra Townie Original 7D EQ - Women's - 2016									
11	4	1	2	749.99	0.1	1499.88	Ritchey Timbervolff Frameset - 2016										
12	5	1	10	2	1549.00	0.05	3097.95	Surly Straggler - 2016									
13	5	2	17	1	429.00	0.07	428.93	Pure Cycles Vine 8-Speed - 2016									
14	5	3	26	1	599.99	0.07	599.92	Electra Townie Original 7D EQ - 2016									
15	6	1	18	1	449.00	0.07	448.93	Pure Cycles Western 3-Speed - Women's - 2015/2016									
16	6	2	12	2	599.99	0.05	1199.93	Electra Townie Original 21D - 2016									
17	6	3	20	1	599.99	0.10	599.89	Electra Townie Original 7D EQ - Women's - 2016									
18	6	4	3	2	999.99	0.07	1999.91	Surly Wednesday Frameset - 2016									
19	6	5	9	2	2999.99	0.07	5999.91	Trek Conduit+ - 2016									
20	7	1	15	1	529.99	0.07	529.92	Electra Moto 1 - 2016									
21	7	2	3	1	999.99	0.10	999.89	Surly Wednesday Frameset - 2016									
22	7	3	17	2	429.00	0.10	857.90	Pure Cycles Vine 8-Speed - 2016									

3. In products table added new column category_name using category_id .
4. In order_items table added new column category_name using product_id to create pivot table.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	order_id	item_id	product_id	quantity	list_price	discount	Revenue	product_name	category_name					
2	1	1	20	1	599.99	0.20	599.79	Electra Townie Original 7D EQ - Women's - 2016	Cruisers Bicycles					
3	1	2	8	2	1799.99	0.07	3599.91	Trek Remedy 29 Carbon Frameset - 2016	Cyclocross Bicycles					
4	1	3	10	2	1549.00	0.05	3097.95	Surly Straggler - 2016	Cruisers Bicycles					
5	1	4	16	2	599.99	0.05	1199.93	Electra Townie Original 7D EQ - 2016	Cruisers Bicycles					
6	1	5	4	1	2899.99	0.20	5799.98	Trek Fuel EX 29 - 2016	Mountain Bikes					
7	2	1	20	1	599.99	0.07	599.92	Electra Townie Original 7D EQ - Women's - 2016	Cruisers Bicycles					
8	2	2	16	2	599.99	0.05	1199.93	Electra Townie Original 7D EQ - 2016	Cruisers Bicycles					
9	3	1	3	1	999.99	0.05	999.94	Electra Townie Original 7D EQ - Women's - 2016	Cruisers Bicycles					
10	3	2	20	1	599.99	0.05	599.94	Electra Townie Original 7D EQ - 2016	Cruisers Bicycles					
11	4	1	2	749.99	0.1	1499.88	Ritchey Timbervolff Frameset - 2016	Mountain Bikes						
12	5	1	10	2	1549.00	0.05	3097.95	Surly Straggler - 2016	Cyclocross Bicycles					
13	5	2	17	1	429.00	0.07	428.93	Pure Cycles Vine 8-Speed - 2016	Cruisers Bicycles					
14	5	3	26	1	599.99	0.07	599.92	Electra Townie Original 7D EQ - 2016	Comfort Bicycles					
15	6	1	18	1	449.00	0.07	448.93	Electra Townie Original 3-Speed - Women's - 2015/2016	Off-Road Bicycles					
16	6	2	12	2	549.99	0.05	1099.93	Electra Townie Original 21D - 2016	Cruisers Bicycles					
17	6	3	20	1	599.99	0.10	599.91	Surly Wednesday Frameset - 2016	Mountain Bikes					
18	6	4	3	2	999.99	0.07	1999.91	Surly Wednesday Frameset - 2016	Electric Bikes					
19	6	5	9	2	2999.99	0.07	5999.91	Trek Conduit+ - 2016	Cruisers Bicycles					
20	7	1	15	1	529.99	0.07	529.92	Electra Moto 1 - 2016	Mountain Bikes					
21	7	2	3	1	999.99	0.10	999.89	Surly Wednesday Frameset - 2016	Mountain Bikes					
22	7	3	17	2	429.00	0.10	857.90	Pure Cycles Vine 8-Speed - 2016	Cruisers Bicycles					

Q8. Create Pivot Table

1. In order_items table select all-insert-pivot table.
2. Rows: category_name
3. Values: Revenue
4. Renamed the pivot table and sheet name as ‘Total sales by category’.

category_name	Sum of Revenue
Children Bicycles	327803.83
Comfort Bicycles	438452.02
Cruisers Bicycles	1199007.23
Eco-Friendly Bicycles	7999.11
Electric Bikes	102924.56
Mountain Bikes	3036651.45
Road Bikes	1852516.12
(blank)	
Grand Total	8578491.31

Q9. Sort and Filter Outliers

1. In products table select list_price column -sort-Largest to Smallest

product_id	product_name	brand_id	category_id	model_year	list_price	category_name
155	Trek Domane SLR 9 Disc - 2018	9	7	2018	11999.99	Road Bikes
149	Trek Domane SLR 8 Disc - 2018	9	7	2018	7499.99	Road Bikes
51	Trek Silque SLR 8 Women's - 2017	9	7	2017	6499.99	Road Bikes
156	Trek Domane SL Frameset - 2018	9	7	2018	6499.99	Road Bikes
157	Trek Domane SL Frameset Women's - 2018	9	7	2018	6499.99	Road Bikes
169	Trek Emonda SLR 8 - 2018	9	7	2018	6499.99	Road Bikes
50	Trek Silque SLR 7 Women's - 2017	9	7	2017	5999.99	Road Bikes
56	Trek Domane SLR 6 Disc - 2017	9	7	2017	5499.99	Road Bikes
154	Trek Domane SLR 6 Disc - 2018	9	7	2018	5499.99	Road Bikes
154	Trek Domane SLR 6 Disc Women's - 2018	9	7	2018	5499.99	Road Bikes
177	Trek Domane SLR 6 Disc - 2018	9	7	2018	5499.99	Road Bikes
43	Trek Fuel EX 9.8 27.5 Plus - 2017	9	6	2017	5299.99	Mountain Bikes
47	Trek Remedy 9.8 - 2017	9	6	2017	5299.99	Mountain Bikes
40	Trek Fuel EX 9.8 29 - 2017	9	6	2017	4999.99	Mountain Bikes
58	Trek Madone 9.2 - 2017	9	7	2017	4999.99	Road Bikes
61	Trek Powerfly 8 FS Plus - 2017	9	5	2017	4999.99	Electric Bikes
140	Trek Remedy 9.8 27.5 - 2018	9	6	2018	4999.99	Mountain Bikes
146	Trek Domane SLR 6 - 2018	9	7	2018	4999.99	Road Bikes
153	Trek Domane SL 7 Women's - 2018	9	7	2018	4999.99	Road Bikes
203	Trek Powerfly 7 FS - 2018	9	5	2018	4999.99	Electric Bikes
205	Trek Super Commuter+ 8S - 2018	9	5	2018	4999.99	Electric Bikes

2. To identify pricing outliers (much larger than majority values),
3. List_price values in the products table were sorted in descending order. The highest-priced products were Trek Domane SLR 9 Disc – 2018 with 11999.99 list_price, in Road Bikes category because they are significantly higher than the typical price range of most products.

Q10. Save Final CSVs

1. Click File
2. Select Save AS
3. Choose Other Formats
4. Save the file in CSV format wherever we want.

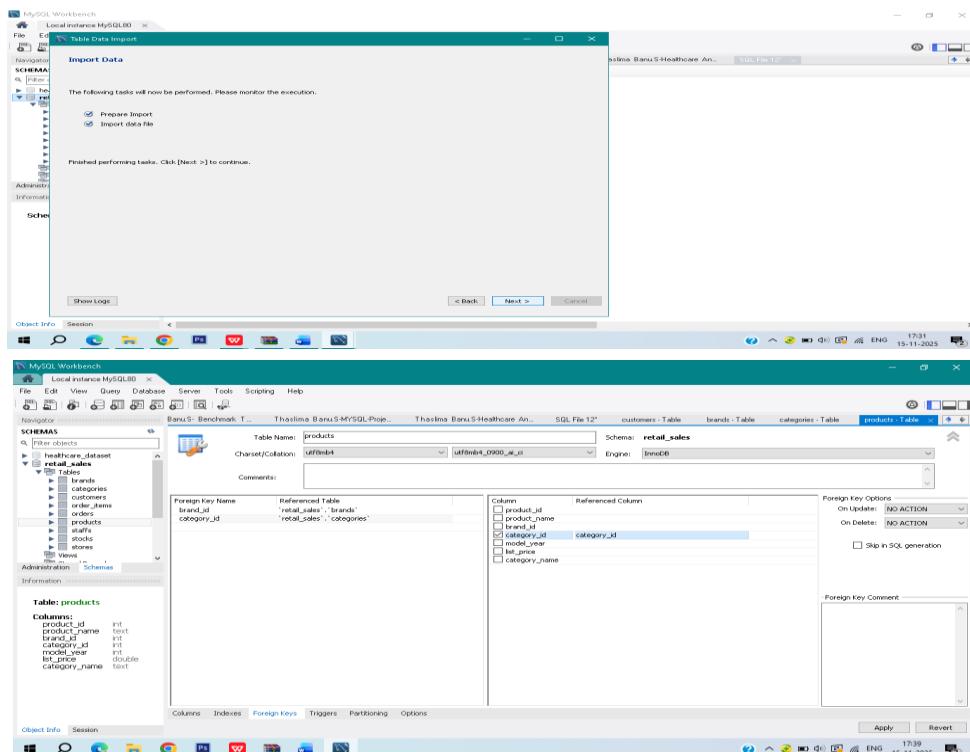
Phase 2: SQL-Database Management and Querying

1. write the query on MySQL Workbench

```
CREATE database Retail_sales;
use retail_Sales;
```

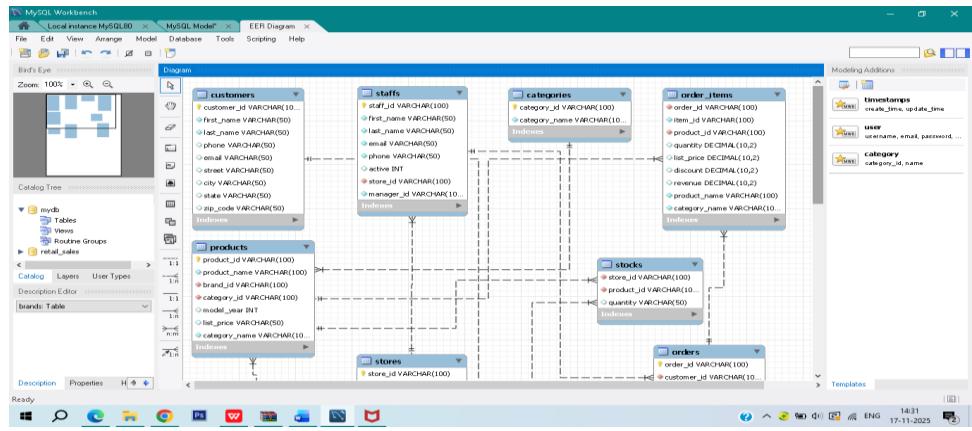
2. Database was created – It will visible on the left side of the workbench under Schemas.

3. Right Click the database created – Table data import wizard – Click Browse – Select the saved csv file - Import tables



Q2.ER Diagram

Relationship Flow	Cardinality	Rationale
Customers - Orders	One-to-many(1:*)	A customer can place many orders.
Staffs – Orders	One-to-many(1:*)	A staff member can process many orders.
Orders – OrderItems	One-to-many(1:*)	One order contains many individual items.
Products - OrderItems	One-to-many(1:*)	One product can be sold across many order items.
Brands - Products	One-to-many(1:*)	One brand can produce many products.
Categories - Products	One-to-One(1:1)	One category can contain many products.
Customer–Customer_Segments	One-to-One(1:1)	Each unique customer is assigned a single, specific segment.



Q3. Inner Join for Order Details

The screenshot shows the MySQL Workbench interface. At the top, there are tabs for 'MySQL Workbench', 'Local Instance MySQL80', 'MySQL Model', and 'EER Diagram'. The 'File', 'Edit', 'View', 'Query', 'Database', 'Server', 'Tools', and 'Help' menu items are visible. Below the menu is a 'Navigator' pane containing 'SCHEMAS' and a tree view of the 'retail_sales' database, including tables like 'customers', 'bands', 'categories', 'products', 'stores', and 'stocks'. The main area displays an 'EER Diagram' with entities 'customers', 'bands', 'categories', 'products', 'stores', 'stocks', and 'orders'. A query editor window is open, showing the following SQL code:

```
75 select o.order_id, o.order_date,
76       p.product_name,
77       c.category_name,
78       oq.quantity, od.discount, or.revenue
79   from orders o
80   inner join order_items oi on o.order_id = oi.order_id
81   inner join products p on oi.product_id = p.product_id
82
83
```

Below the query editor is a 'Result Grid' showing the output of the query. The columns are 'order_id', 'order_date', 'product_name', 'quantity', 'discount', and 'revenue'. The data includes various bicycle models and their prices.

Q4. Total Sales by Store

A screenshot of the MySQL Workbench application. The interface includes a top menu bar with File, Edit, View, Query, Database, Server, Tools, Scripting, Help, and a toolbar with various icons. On the left is a Navigator pane showing the schema tree for 'retail_sales' and a list of tables: customers, brands, categories, products, stores, order_items, orders, and staffs. Below the schema tree is an 'Information' section. The main area contains a query editor with the following SQL code:

```
83 -- Q4
84 select o.store_id,
85   sum(ai.revenue) as Total_Sales
86 from orders o
87 inner join order_items ai on o.order_id = ai.order_id group by o.store_id;
```

The results are displayed in a 'Result Grid' table:

store_id	Total_Sales
1	1790341.05
2	598104.85
3	962546.41

At the bottom, there are tabs for Object Info, Session, and Result 15, along with a status bar showing 'Query Completed' and the date '15-11-2023'. A bottom navigation bar includes icons for Home, Search, File, Edit, View, Insert, Delete, Refresh, Undo, Redo, Save, Print, and Exit.

Q5. Top 5 Selling Products

The screenshot shows the MySQL Workbench interface with the following details:

- Top Bar:** MySQL Workbench, Local instance MySQL80, MySQL Model, EER Diagram.
- Menu Bar:** File, Edit, View, Query, Database, Server, Tools, Scripting, Help.
- Toolbar:** Home, Local, Recent, Refresh, Stop, Run, Save, Open, Import, Export, Paste, Copy, Paste, Find, Replace, Help, Options.
- Navigator:** Schemas (selected), Filter object, healthcare_dataset, retail_sales (selected), tables, brands, categories, customers, order_items, order, products, staffs, stores.
- Information:** Administration, Schemas.
- Table List:** orders (selected).
- Query Editor:** Phase 2*, customers - Table, brands - Table, categories - Table, products - Table, stores - Table, stocks - Table, staffs - Table, staffs - Table, orders - Table. The query is:

```
90 -- Q5
91   select p.product_name,
92          sum(oi.quantity) as Total_Sold
93     from order_items oi
94    inner join products p on oi.product_id = p.product_id group by p.product_id
95   order by Total_Sold desc limit 5;
```
- Result Grid:** Result Grid, Filter Rows, Export, Wrap Cell Content, Fetch rows. The results are:

product_name	Total_Sold
Surly Ice Cream Truck Frameset - 2016	167.00
Electra Cruiser 1 (24-Inch) - 2016	157.00
Electra Townie Original 7D EQ - 2016	156.00
Trek Stache 8 27.5+ - 2016	154.00
Electra Girls Hawaii 1 (20-Inch) - 2015/2016	154.00
- Bottom Bar:** Object Info, Session, Result 16, Read Only.

Q6. Customer Purchase Summary

The screenshot shows the MySQL Workbench interface. The top navigation bar includes tabs for 'MySQL Workbench', 'Local instance MySQL80', 'MySQL Model', and 'EER Diagram'. Below the navigation is a toolbar with various icons for database management. The left sidebar has sections for 'SCHEMAS' (containing 'healthcare_dataset' and 'retail_sales') and 'Administration' (with 'Schemas'). The main area displays an 'EER Diagram' with entities like 'customers', 'brands', 'categories', 'products', 'stores', and 'stocks'. A query editor window is open, showing a complex SELECT statement with joins and aggregate functions. The results grid shows 101 rows of customer data. The bottom status bar indicates 'Query Completed'.

```
07 *--> Q4
08 *--> select c.customer_id,
09      concat(c.first_name, ' ', c.last_name) as Customer_Name,
10      count(distinct o.order_id) as Total_Orders,
11      sum(oil.quantity) as Total_Items,
12      sum(oil.revenue) as Total_Revenue
13
14 from customers c
15 left join orders o on c.customer_id = o.customer_id
16 left join order_items oil on o.order_id = oil.order_id
17 group by c.customer_id
```

customer_id	Customer_Name	Total_Orders	Total_Items	Total_Revenue
1	Debra Ruffo	3	17.00	30044.70
2	Pamela Abshire	3	16.00	37630.42
3	Kelly Franco	1	2.00	5999.86
4	Shobh Reeves	1	3.00	2381.47
5	Lee Roman	1	4.00	4131.76
6	Curtis Lamm	1	3.00	1321.47
7	Shereif Ross	1	4.00	3941.83
8	Jerald Baldwin	1	1.00	299.92
9	Ruthanne Franco	1	5.00	20679.55
10	Martina Madia	1	8.00	7999.86
11	Jimmy Russell	1	6.00	19419.70
12	Bernice Pollard	1	2.00	999.86
13	Deangelo Cadey	1	1.00	349.89
14	Nicole Howell	1	4.00	1479.86

Q7. Segment Customers by Total Spend

The screenshot shows the MySQL Workbench interface. The top navigation bar includes tabs for Local instance MySQL80, MySQL Model, and EER Diagram. Below the navigation is a toolbar with various icons for database management. The left sidebar displays the Schemas tree, showing categories, customers, order_items, orders, products, staffs, stores, and views. The main workspace contains an EER Diagram with entities: An, Phase, customers - Table, brands - Table, categories - Table, products - Table, stores - Table, stocks - Table, staffs - Table, staffs - Table, and orders - Table. A query editor window is open, displaying the following SQL code:

```
108 --- Q7
109 select c.customer_id,
110       sum(oI.revenue) as Total_Revenue,
111       case
112       when sum(oI.revenue) < 5000 then "Low"
113       when sum(oI.revenue) between 5000 and 30000 then "Medium"
114       else "High"
115       end as Segment
116   from customers c
117   inner join orders o on c.customer_id = o.customer_id
118   inner join order_items oI on o.order_id = oI.order_id
119  group by c.customer_id;
```

Below the query editor is a Result Grid showing the output of the query:

customer_id	Total_Revenue	Segment
1	30644.78	High
10	37800.42	High
100	5999.88	Medium
1000	2381.47	Low
1001	4131.76	Low
1002	1333.75	Low
1003	1941.81	Low
1004	2092	Low
1005	20679.55	High
1006	7396.52	Medium
1007	19419.70	Medium
1008	939.88	Low

The bottom status bar indicates "Object Info Session" and "Result 19 x". The taskbar at the bottom shows various application icons, and the system tray indicates the date as "ENG 17.11.2025" and the time as "17:00".

Q8. Staff Performance Analysis

The screenshot shows the MySQL Workbench interface with the following details:

- File Bar:** Local instance MySQL80, MySQL Model, EER Diagram.
- Navigator:** Schemas (customers, brands, categories, products, stores, stocks, staffs, orders, staffs, sys, views, stored procedures, functions).
- Query Editor:** Phase 2*, customers - Table, brands - Table, categories - Table, products - Table, stores - Table, stocks - Table, staffs - Table, staffs - Table, orders - Table, on...
The query is:-- Q8
select s.staff_id,
concat(s.first_name, ' ', s.last_name) as Staff_name,
sum(o.revenue) as Total_Revenue
from staffs s
inner join orders o on s.staff_id = o.staff_id
inner join order_items oi on o.order_id = oi.order_id
group by s.staff_id order by Total_Revenue desc;
- Result Grid:** Shows the results for 9 staff members:| staff_id | Staff_name | Total_Revenue |
| --- | --- | --- |
| 6 | Marceline Boyer | 2938714.12 |
| 7 | Venita Daniel | 2807187.73 |
| 3 | Genna Serrano | 952665.27 |
| 2 | Mireya Copeland | 837375.78 |
| 8 | Kali Vargas | 516667.66 |
| 9 | Layla Terrell | 446860.75 |
- Object Info:** Session
- Query Completed:** Read Only
- System Bar:** Windows, Task View, Start, Taskbar icons, Network, Battery, ENG, 17:12, 17-11-2025.

Q9. Stock Alert Query

The screenshot shows the MySQL Workbench interface with the following details:

- File Bar:** Local instance MySQL80, MySQL Model, EER Diagram.
- Navigator:** Schemas (categories, customers, order_items, orders, products, staffs, stocks, stores, sys, views, stored procedures, functions).
- Query Editor:** Phase 2*, customers - Table, brands - Table, categories - Table, products - Table, stores - Table, stocks - Table, staffs - Table, staffs - Table, orders - Table, on...
The query is:-- Q9
select product_id, store_id,
quantity as Remaining_stock
from stocks
where quantity < 10;
- Result Grid:** Shows the remaining stock for various products across different stores:| product_id | store_id | Remaining_stock |
| --- | --- | --- |
| 2 | 1 | 5 |
| 3 | 1 | 6 |
| 6 | 1 | 0 |
| 7 | 1 | 8 |
| 9 | 1 | 0 |
| 11 | 1 | 8 |
| 14 | 1 | 0 |
| 15 | 1 | 3 |
| 16 | 1 | 4 |
| 17 | 1 | 2 |
| 19 | 1 | 4 |
| 23 | 1 | 9 |
| 31 | 1 | 2 |
| 32 | 1 | 0 |
| 34 | 1 | 2 |
| 39 | 1 | 2 |
| 42 | 1 | 0 |
| 43 | 1 | 2 |
| 44 | 1 | 1 |
| 48 | 1 | 5 |
- Object Info:** Session
- Query Completed:** Read Only
- System Bar:** Windows, Task View, Start, Taskbar icons, Network, Battery, ENG, 17:15, 17-11-2025.

Q10. Final Segmentation Table

The screenshot shows the MySQL Workbench interface with the following details:

- File Bar:** unconnected, Local instance MySQL80, MySQL.
- Navigator:** Schemas (headware_dataset, headware_stores, Tables (categories, customers, order_items, products, staffs, stocks, stores), sys, views, stored procedures, functions).
- Query Editor:** ThakurFinalProject, Local instance MySQL80, MySQL Model, EER Diagram.
The query is:concat(s.first_name, ' ', s.last_name) as Staff_name,
sum(o.revenue) as Total_Revenue
from staffs s
inner join orders o on s.staff_id = o.staff_id
inner join order_items oi on o.order_id = oi.order_id
group by s.staff_id order by Total_Revenue desc;
-- Q9
select product_id, store_id,
quantity as Remaining_stock
from stocks
where quantity < 10;
-- Q10
CREATE TABLE retail_sales.customer_segments (
customer_id INT,
Recency INT,
Frequency INT,
Monetary DECIMAL(10,2),
Segment VARCHAR(50))
select * from retail_sales.customer_segments;
- Object Info:** Session
- Query Completed:** Read Only
- System Bar:** Windows, Task View, Start, Taskbar icons, Network, Battery, ENG, 17:21, 24-11-2025.

Phase 3: Python and ML Tasks

1. Open Command prompt.
2. Type command python -m notebook.
3. It will open Jupyter notebook.
4. Install pandas, numpy, matplotlib by using the word pip before the libraries.
5. Import pandas as pd, numpy as np, matplotlib.pyplot as plt.

Q1. Load Data from SQL

Steps:

1. pip install sqlalchemy
2. from sqlalchemy import create_engine
3. engine=create_engine("mysql+pymysql://root:*****@localhost:3306/Retail_Sales")
4. orders = pd.read_sql ("SELECT * FROM orders", engine)
display(orders)

Orders:

```
[28]: engine = create_engine("mysql+pymysql://root:*****@localhost:3306/Retail_Sales")
[29]: engine
[30]: orders = pd.read_sql("SELECT * FROM orders",engine)
[31]: display(orders)
```

order_id	customer_id	order_status	order_date	required_date	shipped_date	store_id	staff_id	
0	1	259	shipped	2016-01-01	2016-01-03	2016-01-03	1	2
1	10	442	shipped	2016-01-05	2016-01-06	2016-01-06	2	6
2	100	1237	shipped	2016-03-01	2016-03-04	2016-03-03	2	7
3	1000	649	shipped	2017-07-12	2017-07-14	2017-07-15	2	7
4	1001	354	shipped	2017-07-13	2017-07-16	2017-07-15	2	6
...
1610	995	621	shipped	2017-07-11	2017-07-13	2017-07-14	1	2
1611	996	919	shipped	2017-07-11	2017-07-12	2017-07-12	1	2
1612	997	485	shipped	2017-07-11	2017-07-13	2017-07-14	2	7
1613	998	219	shipped	2017-07-12	2017-07-13	2017-07-14	2	7
1614	999	590	shipped	2017-07-12	2017-07-13	2017-07-14	2	6

Order Items:

```
[32]: order_items = pd.read_sql("SELECT * FROM order_items",engine)
[33]: order_items
```

order_id	item_id	product_id	quantity	list_price	discount	revenue	product_name	category_name	
0	1	1	20	1.0	599.99	0.2	599.79	Electra Townie Original 7D EQ - Women's - 2016	Cruisers Bicycles
1	1	2	8	2.0	1799.99	0.1	3599.91	Trek Remedy 29 Carbon FrameSet - 2016	Mountain Bikes
2	1	3	10	2.0	1549.00	0.1	3097.95	Surf Straggler - 2016	Cyclocross Bicycles
3	1	4	16	2.0	599.99	0.1	1199.93	Electra Townie Original 7D EQ - 2016	Cruisers Bicycles
4	1	5	4	1.0	2899.99	0.2	2899.79	Trek Fuel EX 8.29 - 2016	Mountain Bikes
...
4717	1614	2	159	2.0	2299.99	0.1	4599.91	Trek Emonda ALR 6 - 2018	Road Bikes
4718	1614	3	213	2.0	269.99	0.2	539.78	Electra Cruiser 1 - 2016/2017/2018	Cruisers Bicycles
4719	1615	1	197	2.0	2299.99	0.2	4599.78	Trek Verve+ LowStep - 2018	Electric Bikes
4720	1615	2	214	1.0	899.99	0.1	899.92	Electra Tiger Shark 3i - 2018	Cruisers Bicycles
4721	1615	3	182	1.0	2499.99	0.2	2499.79	Trek Domane SL 5 Disc - 2018	Road Bikes

Customers

A screenshot of a Jupyter Notebook interface. The code cell [33] contains the command `customers = pd.read_sql("SELECT * FROM customers", engine)` followed by the resulting DataFrame. The DataFrame has 1445 rows and 9 columns, with columns labeled: customer_id, first_name, last_name, phone, email, street, city, state, and zip_code. The data includes various customer names and addresses from New York.

	customer_id	first_name	last_name	phone	email	street	city	state	zip_code
0	1	Debra	Burks	Not Available	debra.burks@yahoo.com	9273 Thorne Ave.	Orchard Park	NY	14127
1	10	Pamela	Newman	Not Available	pamelia.newman@gmail.com	476 Chestnut Ave.	Monroe	NY	10950
2	100	Kellie	Franco	Not Available	kellie.franco@yahoo.com	444 South Walnut Rd.	Commack	NY	11725
3	1000	Shiloh	Reeves	Not Available	shiloh.reeves@msn.com	818 Kirkland Lane	West Babylon	NY	11704
4	1001	Lee	Roman	(631) 913-6967	lee.roman@gmail.com	4 Canal Ave.	Brentwood	NY	11717
...
1440	995	Amina	Salazar	Not Available	amina.salazar@aol.com	944 Wellington Street	Canandagua	NY	14424
1441	996	Serafina	Clemons	Not Available	serafina.demons@gmail.com	851 Brown Ave.	Shirley	NY	11967
1442	997	Trinidad	McCain	Not Available	trinidad.mccain@msn.com	98 Tunnel Drive	Baldwin	NY	11510
1443	998	Heather	Chaney	Not Available	heather.chaney@yahoo.com	4 Morris Dr.	Jamestown	NY	14701
1444	999	Latoya	Johns	Not Available	latoya.johns@hotmail.com	7914 W. Woodsman St.	Ballston Spa	NY	12020

Q2. Basic EDA (Exploratory Data Analysis)

i) Orders:

Order.describe()

A screenshot of a Jupyter Notebook interface. The code cell [57] contains the command `orders.describe()` followed by the resulting summary statistics for the 'orders' DataFrame. The output shows counts, unique values, top values, and frequency for each column.

	order_id	customer_id	order_status	order_date	required_date	shipped_date	store_id	staff_id
count	1615	1615	1615	1615	1615	1445	1615	1615
unique	1615	1445	4	725	734	675	3	6
top	1	31	shipped	2018-04-12	2018-04-17	2017-10-29	2	6
freq	1	3	1445	9	9	8	1093	553

orders.info()

orders['order_status'].value_counts

A screenshot of a Jupyter Notebook interface. The code cell [58] contains the command `orders.info()` followed by the resulting information about the 'orders' DataFrame. The code cell [59] contains the command `orders['order_status'].value_counts` followed by the resulting count of each order status.

	order_id	customer_id	order_status	order_date	required_date	shipped_date	store_id	staff_id
count	1615	1615	1615	1615	1615	1445	1615	1615
unique	1615	1445	4	725	734	675	3	6
top	1	31	shipped	2018-04-12	2018-04-17	2017-10-29	2	6
freq	1	3	1445	9	9	8	1093	553

orders['order_status'].value_counts

order_status	count
shipped	1610
shipped	1611
shipped	1612
shipped	1613
shipped	1614

Order_items:

order_items.describe()

```
[60]: order_items.describe()
[60]:
   quantity      list_price      discount      revenue
count    4722.000000    4722.000000    4722.000000    4722.000000
mean     1499.9411    1212.707872    0.125476    1816.707181
std      0.500052    1352.798257    0.043578    2228.162327
min      1.000000    89.990000    0.100000    89.790000
25%     1.000000    429.990000    0.100000    539.780000
50%     1.000000    599.990000    0.100000    939.910000
75%     2.000000    1549.000000    0.200000    1999.910000
max     2.000000    1999.990000    0.200000    23999.880000
```

order_items.info()

order_items['product_name'].value_counts()

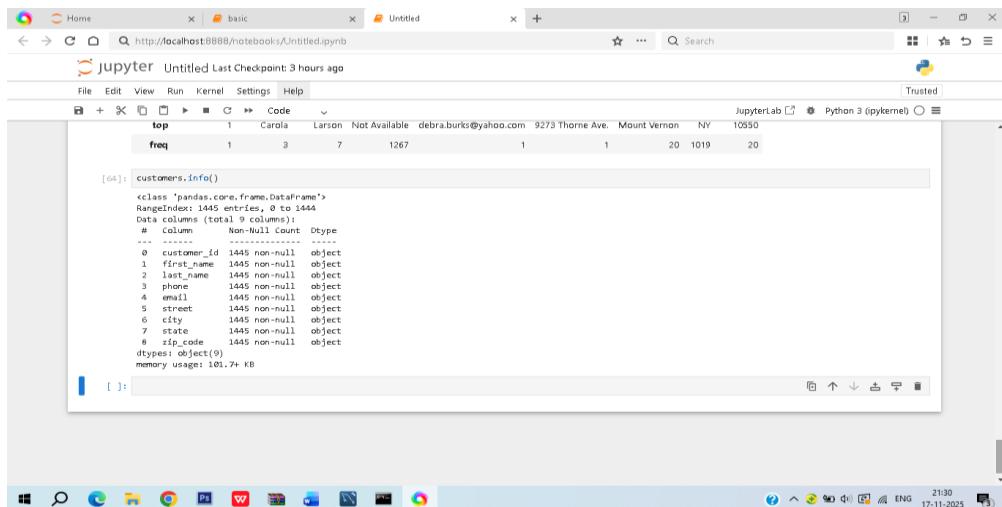
```
[62]: order_items.info()
[62]:
order_items['product_name'].value_counts()
[62]:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4722 entries, 0 to 4721
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype  
--- 
 0   order_id        4722 non-null   object  
 1   item_id         4722 non-null   object  
 2   product_id     4722 non-null   object  
 3   quantity        4722 non-null   float64 
 4   list_price      4722 non-null   float64 
 5   discount        4722 non-null   float64 
 6   revenue         4722 non-null   float64 
 7   product_name   4722 non-null   object  
 8   category_name  4722 non-null   object  
dtypes: float64(4), object(5)
memory usage: 332.1+ KB
[62]: products
[62]:
products['Name'].value_counts()
[62]:
Electra Cruiser 1 (24-Inch) - 2016    193
Electra Townie Original 310 - 2016     193
Electra Townie Original 70 EQ - 2016    185
Electra Girl's Hawaii 1 (16-inch) - 2015/2016 180
Surly Ice Cream Truck Frameset - 2016    110
...
Electra Straight 8 1 (16-inch) - Boy's - 2018    1
Trek Boone 24 7-speed - 2018             1
Trek Precaliber 24 7-speed Girl's - 2018    1
Trek Adorne 9 Frameset - 2018            1
Electra Superbolt 1 20" - 2018           1
Name: count, Length: 276, dtype: int64
```

Customers:

customers. Describe()

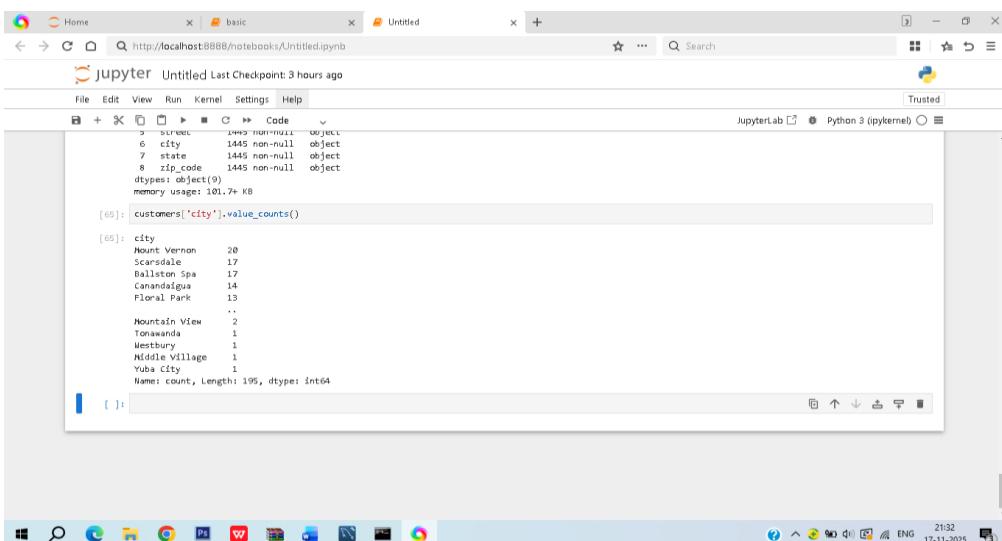
```
[63]: customers.describe()
[63]:
   customer_id  first_name  last_name  phone  email  street  city  state  zip_code
count      1445       1445      1445    1445    1445    1445  1445  1445    1445
unique     1445       1265      753     179    1445    1445    195     3    195
top        1       Carol  Larson  Not Available  debra.burks@yahoo.com  9273 Thorne Ave.  Mount Vernon  NY  10550
freq       1         3         7      1267           1           1      20    1019      20
```

customers.info()



```
[64]: customers.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1445 entries, 0 to 1444
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   customer_id    1445 non-null   object 
 1   first_name     1445 non-null   object 
 2   last_name      1445 non-null   object 
 3   phone          1445 non-null   object 
 4   email          1445 non-null   object 
 5   street          1445 non-null   object 
 6   city            1445 non-null   object 
 7   state           1445 non-null   object 
 8   zip_code        1445 non-null   object 
dtypes: object(9)
memory usage: 101.7+ KB
```

Customers['city'].value_counts()

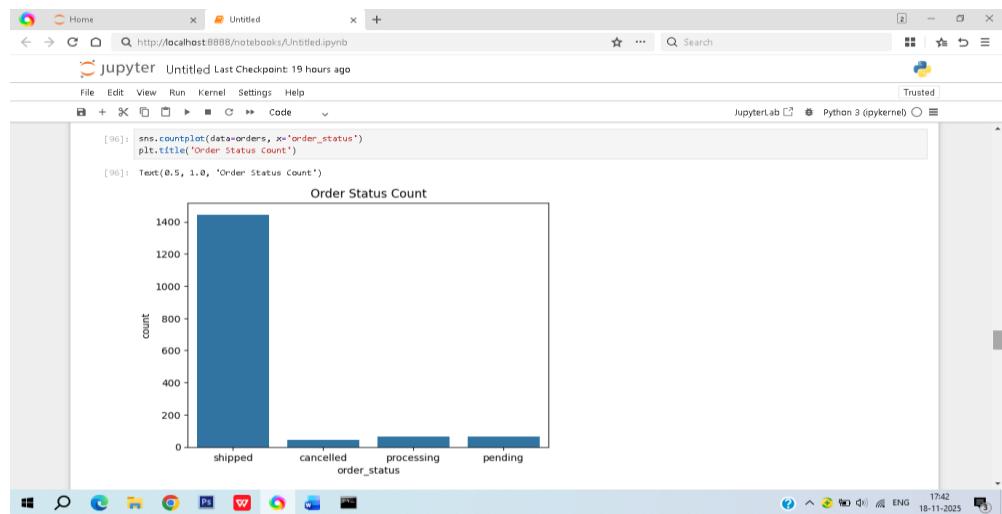


```
[65]: customers['city'].value_counts()
city
Mount Vernon    20
Scarsdale       17
Ballston Spa   17
Canandaigua    14
Floral Park     13
...
Mountain View    2
Towanda         1
Westbury         1
Middle Village    1
Yuba City        1
Name: count, Length: 195, dtype: int64
```

ii) Bar Chart – Order Status Distribution using Seaborn.Countplot

```
sns.countplot(data=orders, x='order_status')
```

```
plt.title('Order Status Count')
```



Q3. Calculate RFM Features for Customers

Steps:

1. Calculate Recency

- i) Determine the max order date for each customer
- ii) Define a reference
- iii) Calculate the difference in days between the Reference Date and the customer's max order date.

The screenshot shows a Jupyter Notebook interface with a single code cell containing Python code. The code uses the pandas library to group orders by customer_id and find the maximum order_date for each customer. It then calculates the difference in days between this maximum date and today's date.

```
[28]: timestamp('2025-11-28 15:32:59.625807')  
[29]: last_order = orders.groupby('customer_id')['order_date'].max()  
last_order  
[29]: customer_id  
1 2018-11-18  
10 2018-08-23  
100 2018-02-25  
1000 2017-04-11  
1001 2016-08-29  
995 2016-08-14  
996 2016-02-08  
997 2016-09-22  
998 2017-11-09  
999 2018-03-07  
Name: order_date, Length: 1445, dtype: datetime64[ns]  
[30]: recency = (today - last_order).dt.days  
recency  
[30]: customer_id  
1 2563  
10 2650  
100 2829  
1000 3149  
1001 3374  
..  
995 3389  
996 3577  
997 3350  
998 3937  
999 2819  
Name: order_date, Length: 1445, dtype: int64
```

2. Calculate Frequency

To determine the total number of transactions made by each customer's, Count the number of unique order_id associated with each customer_id from the orders table. It will show the dataset showing the Number of Orders placed by each customer.

The screenshot shows a Jupyter Notebook interface with a single code cell containing Python code. The code groups orders by customer_id and counts the number of unique order_ids for each customer.

```
[31]: frequency = orders.groupby('customer_id')['order_id'].nunique()  
frequency  
[31]: customer_id  
1 5  
10 3  
100 1  
1000 1  
1001 1  
..  
995 1  
996 1  
997 1  
998 1  
999 1  
Name: order_id, Length: 1445, dtype: int64  
[40]: order_items['revenue'] = order_items['quantity'] * order_items['list_price'] - order_items['discount']  
[41]: customer_money = orders.merge(order_items, on='order_id')  
customer_money  
[41]: order_id customer_id order_status order_date required_date shipped_date store_id staff_id item_id product_id quantity list_price discount revenue pro
```

3. Calculate Monetary value

To calculate Total net purchase value for each customer, Group the order_items table by customer_id and sum the calculated Total Revenue, it will give the result ,the dataset where each customer is associated with their Total value of all purchases.

The image shows two side-by-side Jupyter Notebook interfaces running on port 8888. Both notebooks are titled "Banu-Final Project Phase 3" and show code execution history.

Top Notebook (Left):

```
[40]: order_items['revenue'] = order_items['quantity'] * order_items['list_price'] - order_items['discount']
[41]: customer_money = orders.merge(order_items, on='order_id')
      customer_money
```

order_id	customer_id	order_status	order_date	required_date	shipped_date	store_id	staff_id	item_id	product_id	quantity	list_price	discount	revenue	
0	1	259	shipped	2016-01-01	2016-01-03	2016-01-03	1	2	1	20	1.0	599.99	0.2	599.79
1	1	259	shipped	2016-01-01	2016-01-03	2016-01-03	1	2	2	8	2.0	1799.99	0.1	3599.88
2	1	259	shipped	2016-01-01	2016-01-03	2016-01-03	1	2	3	10	2.0	1549.00	0.1	3097.90
3	1	259	shipped	2016-01-01	2016-01-03	2016-01-03	1	2	4	16	2.0	599.99	0.1	1199.88
4	1	259	shipped	2016-01-01	2016-01-03	2016-01-03	1	2	5	4	1.0	2899.99	0.2	2899.79

Bottom Notebook (Right):

```
[57]: monetary = customer_money.groupby('customer_id')['revenue'].sum()
monetary
```

customer_id	revenue
1	30644.57
10	37801.84
100	5999.98
1000	2381.97
1001	4131.96
...	...
995	549.97
996	549.99
997	12080.95
998	6209.95
999	6957.97

```
[58]: rfm = pd.DataFrame({'Recency': recency, 'Frequency': frequency, 'Monetary': monetary})
rfm
```

customer_id	Recency	Frequency	Monetary
1	2557	3	30645.87
10	2644	3	37801.84
100	2823	1	5999.98
1000	3143	1	2381.97

Q4. Export Segmentation Results to SQL

rfm.reset_index()

rfm_final = rfm.reset_index()

rfm_final[['customer_id','Recency','Frequency','Monetary','Segment']].to_sql('customer_segments',engine , if_exists='append',index=False)

The image shows a single Jupyter Notebook interface running on port 8888. The notebook is titled "Banu-Final Project Phase 3" and shows code execution history.

```
[45]: rfm.head()
[46]: rfm.reset_index()
rfm_final = rfm.reset_index()
[51]: rfm_final[['customer_id','Recency','Frequency','Monetary','Segment']].to_sql('customer_segments',engine , if_exists='append',index=False)
```

customer_id	Recency	Frequency	Monetary	Segment
1	2563	3	30644.57	Loyal Customer
10	2650	3	37800.34	Loyal Customer
100	2029	1	5999.98	High Value Customer
1000	2149	1	2381.47	At Risk Customer
1001	3374	1	4131.66	At Risk Customer

```

CREATE TABLE retail_sales.customer_segments (
    customer_id INT,
    Recency INT,
    Frequency INT,
    Monetary DECIMAL(10,2),
    Segment VARCHAR(50)
);

select * from retail_sales.customer_segments;

```

customer_id	Recency	Frequency	Monetary	Segment
1	2553	3	3044.57	Loyal Customer
10	2650	3	37900.34	Loyal Customer
100	2829	1	5999.88	High Value Customer
1000	3149	1	2381.47	At Risk Customer
1001	3374	1	4131.66	At Risk Customer
1002	2951	1	1333.67	At Risk Customer
1003	3019	1	3941.76	At Risk Customer
1004	3401	1	299.89	At Risk Customer
1005	2827	1	20679.55	High Value Customer
1006	2965	1	7390.00	High Value Customer
1007	2853	1	13145.54	High Value Customer
1008	2884	1	939.78	At Risk Customer
1009	3005	1	349.89	At Risk Customer
101	3351	1	1479.76	At Risk Customer
1010	2866	1	11279.45	High Value Customer
1011	3400	1	1139.77	At Risk Customer

Phase 4: Power BI – Visualization & Dashboarding

1. In Power BI – Click Get data – More – Search for MySQL Database.
2. Click on Connect.
3. Select Database - Enter Username and Password
4. Click connect.
5. Select the Tables we need to Transform
6. Load the Tables – check errors and rectify it.

The Power Query Editor window displays a complex transformation step. It shows multiple tables being joined or transformed:

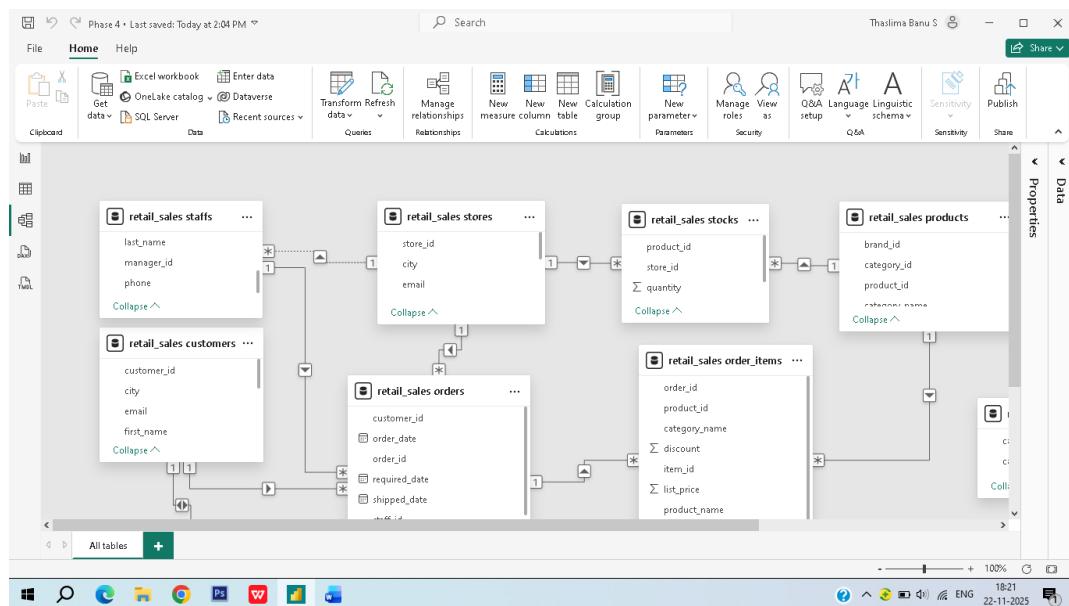
- retail_sales.customers
- retail_sales.order_items
- retail_sales.staffs
- retail_sales.stores

The preview pane shows the resulting data structure with columns such as customer_id, first_name, last_name, phone, email, staff_id, value, table, and store_id. The 'Applied Steps' pane indicates a 'Navigation' step has been applied.

Q2. Create Relationships Between Tables

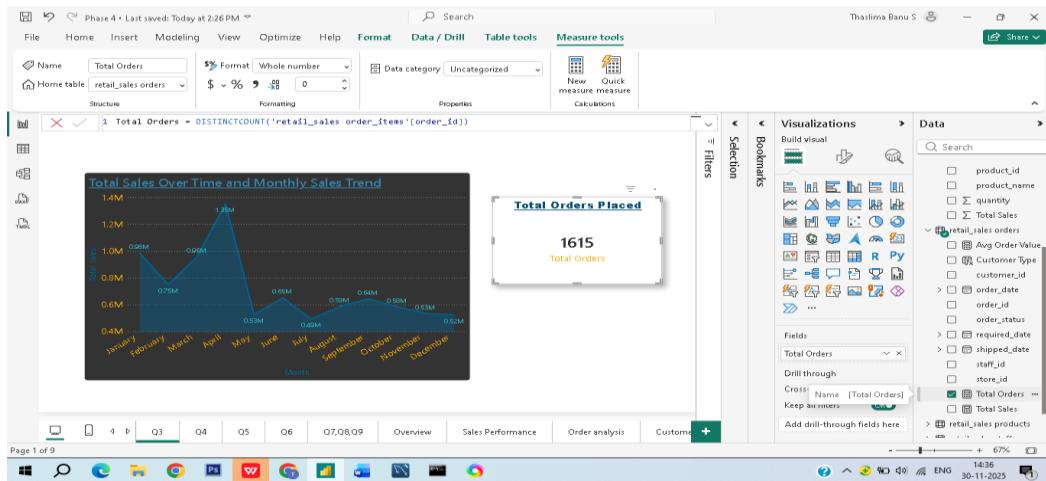
Relationship Flow	Cardinality	Rationale
Customers - Orders	One-to-many(1:*)	A customer can place many orders.
Staffs - Orders	One-to-many(1:*)	A staff member can process many orders.
Orders – OrderItems	One-to-many(1:*)	One order contains many individual items.
Products - OrderItems	One-to-many(1:*)	One product can be sold across many order items.
Brands - Products	One-to-many(1:*)	One brand can produce many products.
Categories - Products	One-to-One(1:1)	One category can contain many products.
Customer CustomerSegments	One-to-One(1:1)	Each unique customer is assigned a single, specific segment.

Click on Model View – to view the ER Diagram

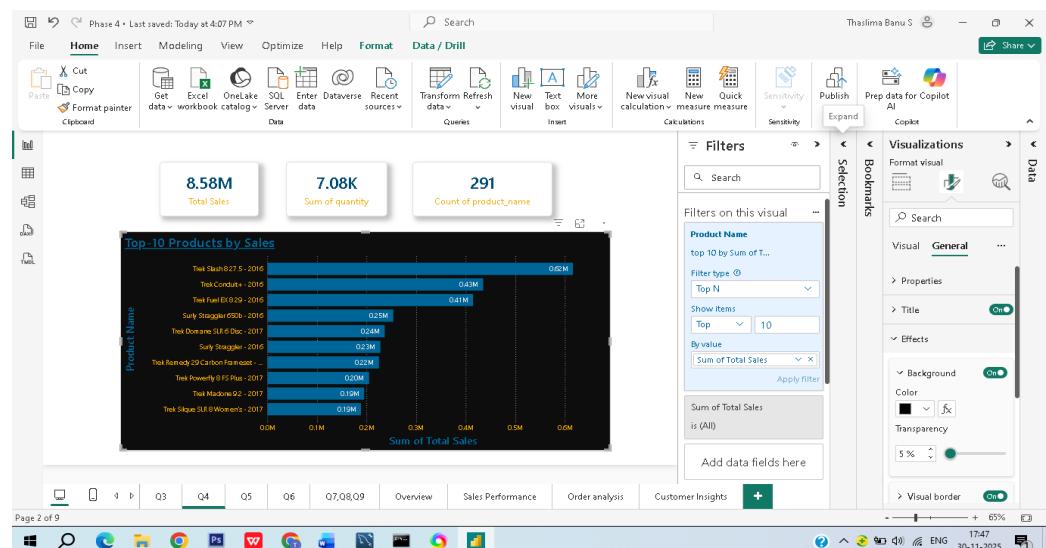


Q3. Sales Overview Report

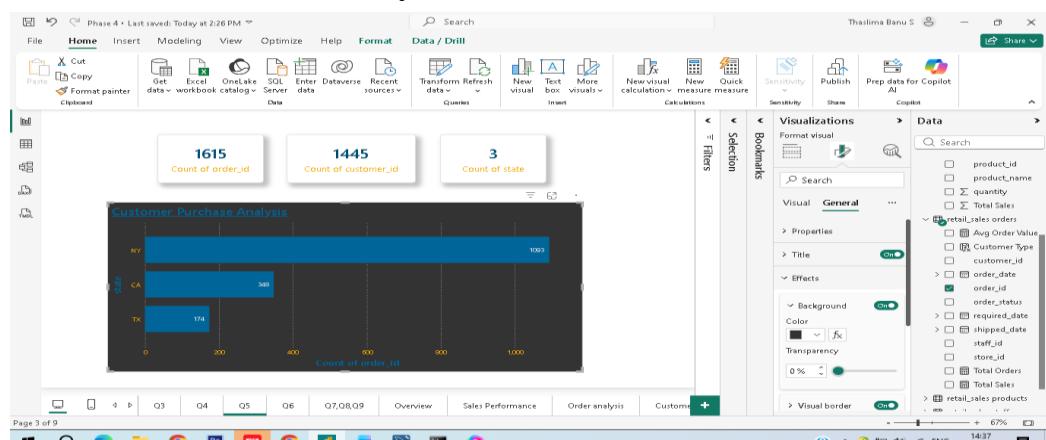
To Calculate TotalOrders Placed – click new measure – Enter the Dax formula
 - Total Orders = DISTINCTCOUNT('retail_sales_orders'[order_id])



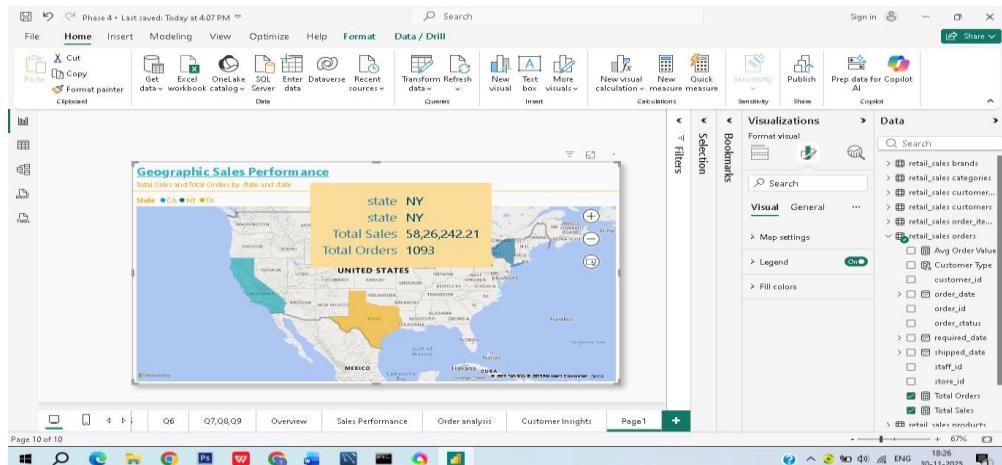
Q4. Top Products by Sales



Q5. Customer Purchase Analysis



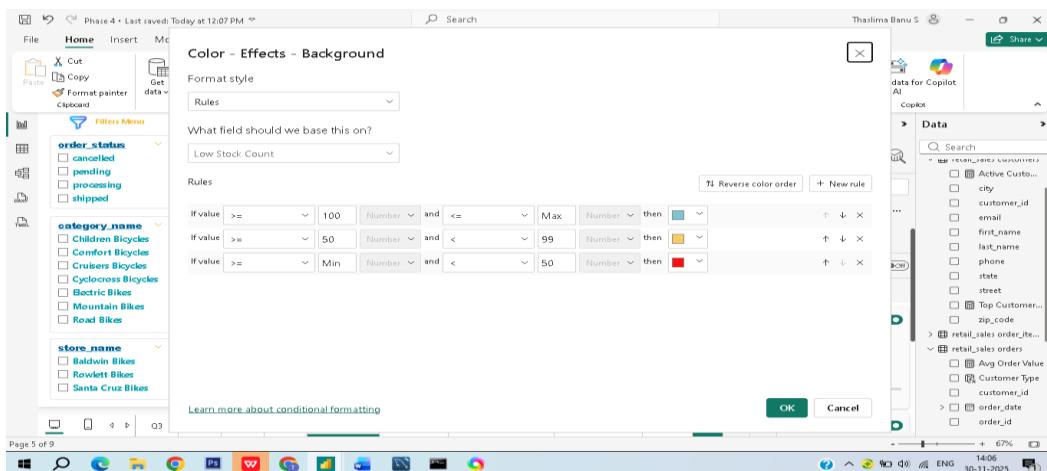
Q6. Sales by Store Map



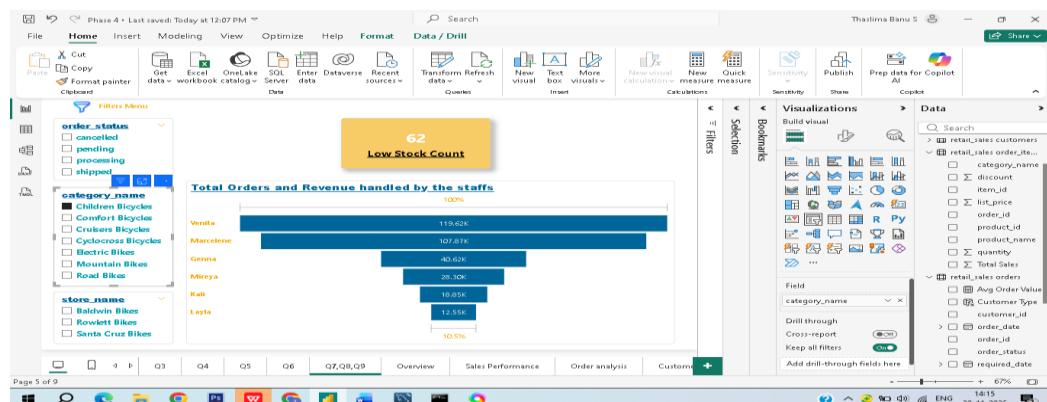
Q7. Low Stock Alert Dashboard

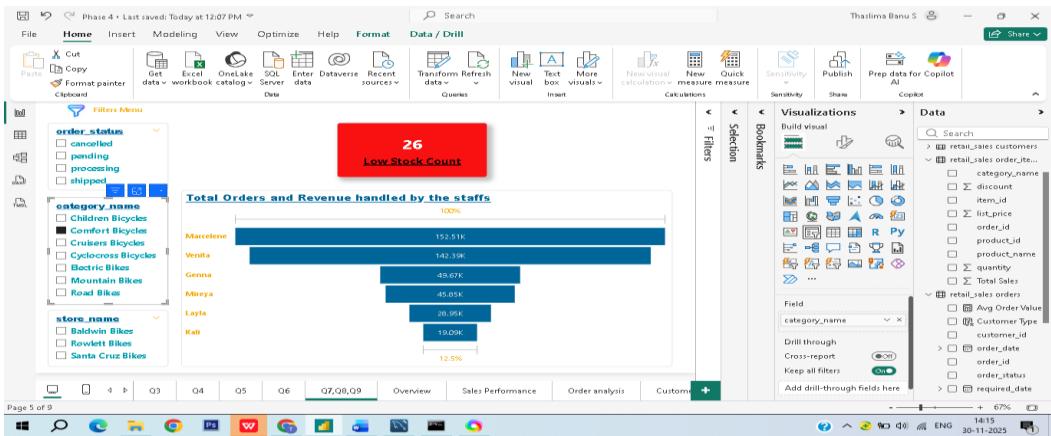
Apply Conditional Formatting

- Select the card – in Format Visuals – Select format your visuals – Go for General – Effects – Background - Click on the fx

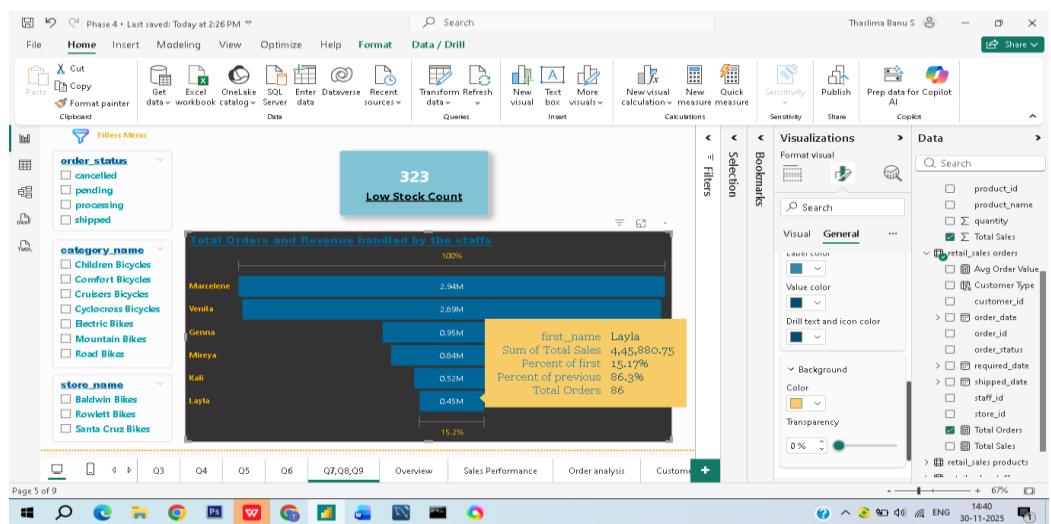


Q8. Interactive Filters and Slicers

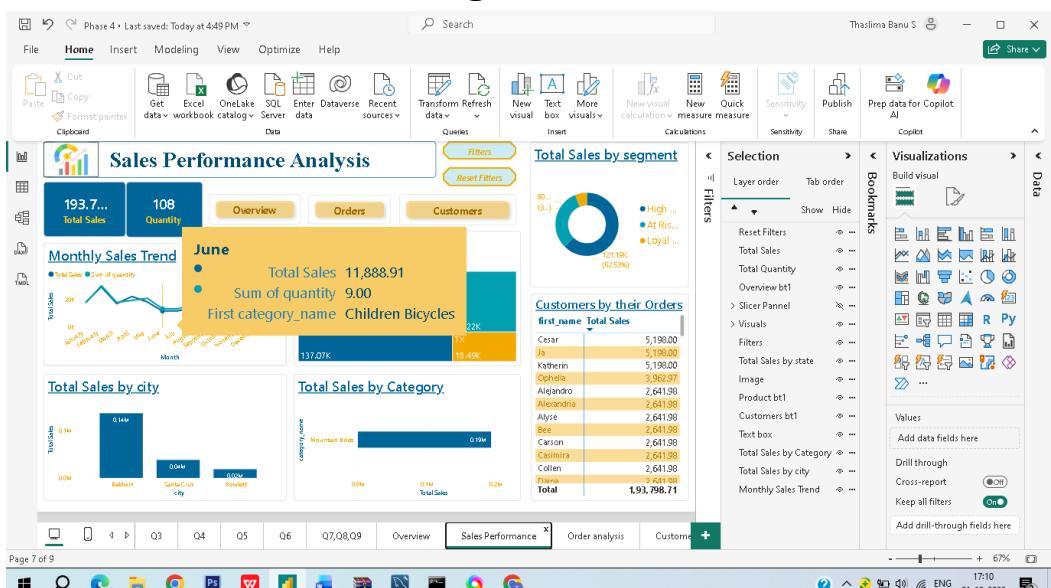


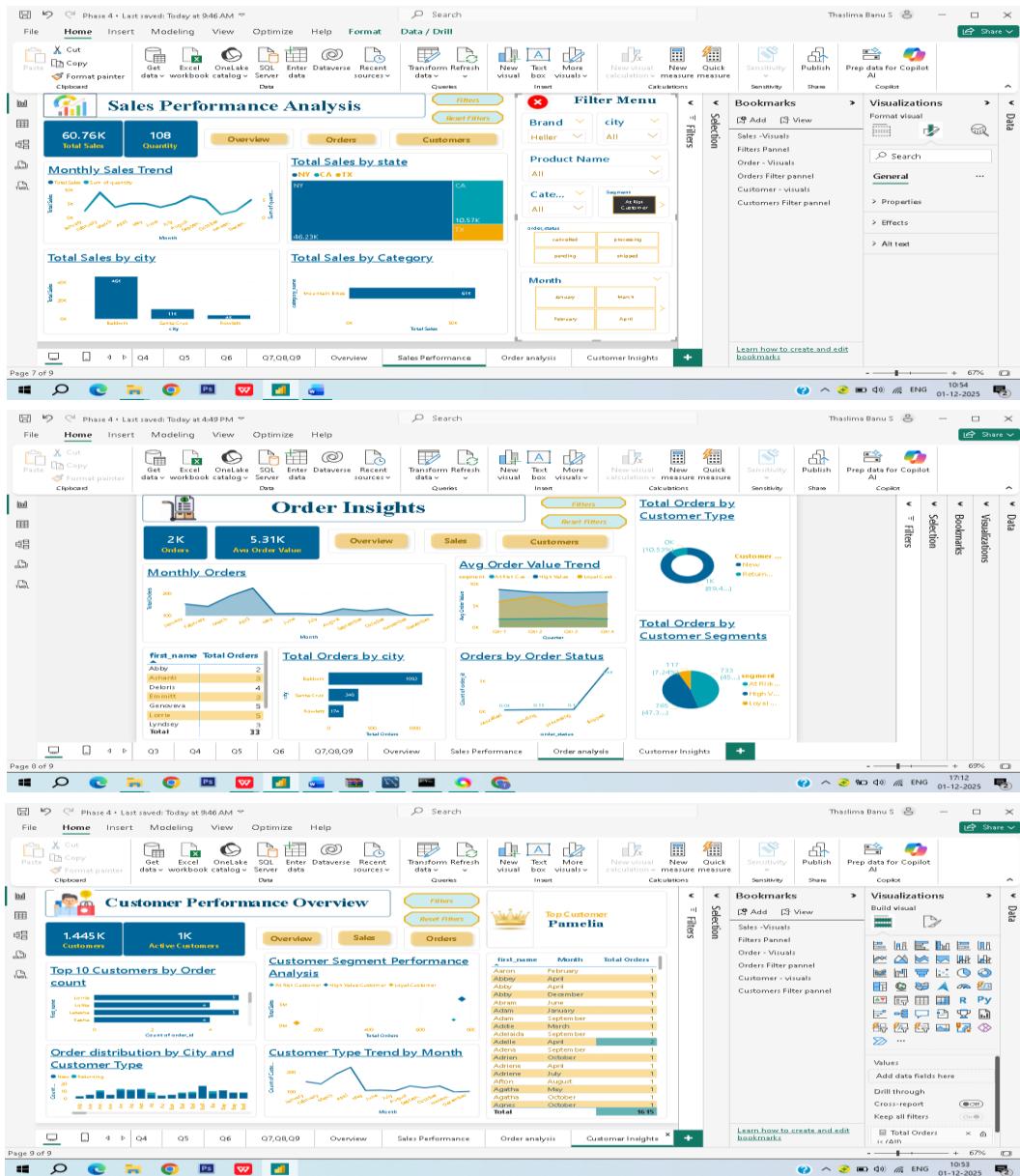


Q9. Staff Performance Report



Q10. Consolidated Dashboard Page





Conclusion:

This Retail Sales Analytics project integrated multiple analytical tools such as Excel, SQL, Python, and Power BI to analyze sales performance and customer behaviour for a retail business. The dataset was cleaned and standardized in Excel, and a structured database was created in MySQL to run meaningful analytical queries, including top-selling products, sales by store, customer segmentation, and staff performance. Python was used for exploratory data analysis and RFM customer segmentation, enabling a deeper understanding of purchasing patterns. Finally, Power BI visualizations provided an interactive dashboard offering insights on total orders, top products, sales trends, store performance, low stock alerts, and customer contribution.

Overall, this project demonstrates the complete data analytics workflow—from raw data to actionable insights—supporting better decision-making and business improvement.