PROJECT NAME :- Building SQL Project With E-Commerce Dataset

Project Description

Sales analysis is used to determine how well some products are selling by comparing their sales target with the actual sales. It also helps us to discover insights such as popular and underperforming products, market trends, high-value customers, etc. In this article, I will explain how I performed sales analysis on the dataset containing sales details from an Indian e-commerce website.

I performed this analysis using pgAdmin 4.

Three primary tables are used in this project named as:-

1) list_of_orders
2) order_details
3) sales_target

1)
```
select * from list_of_orders;
```

Data Output | Messages | Notifications

	orderid text	order_date date	customer_name text	state text	city text
1	B-25602	2022-04-01	Pearl	Maharashtra	Pune
2	B-25603	2022-04-03	Jahan	Madhya Pradesh	Bhopal
3	B-25604	2022-04-03	Divsha	Rajasthan	Jaipur
4	B-25605	2022-04-05	Kasheen	West Bengal	Kolkata
5	B-25606	2022-04-06	Hazel	Karnataka	Bangalore
6	B-25607	2022-04-06	Sonakshi	Jammu and Kashmir	Kashmir
7	B-25608	2022-04-08	Aarushi	Tamil Nadu	Chennai
8	B-25609	2022-04-09	Jitesh	Uttar Pradesh	Lucknow
9	B-25610	2022-04-09	Yogesh	Bihar	Patna
10	B-25611	2022-04-11	Anita	Kerala	Thiruvananthapuram
11	B-25612	2022-04-12	Shrichand	Punjab	Chandigarh

Total rows: 499 of 499 Query complete 00:00:00.156 Ln 10, Col 1

2) `select * from order_details;`

Data Output | Messages | Notifications

	orderid text	amount smallint	profit smallint	quantity smallint	category text	sub_category text
1	B-25601	1275	-1148	7	Furniture	Bookcases
2	B-25601	66	-12	5	Clothing	Stole
3	B-25601	8	-2	3	Clothing	Hankerchief
4	B-25601	80	-56	4	Electronics	Electronic Games
5	B-25602	168	-111	2	Electronics	Phones
6	B-25602	424	-272	5	Electronics	Phones
7	B-25602	2617	1151	4	Electronics	Phones
8	B-25602	561	212	3	Clothing	Saree
9	B-25602	119	-5	8	Clothing	Saree
10	B-25603	1355	-60	5	Clothing	Trousers
11	B-25603	24	-30	1	Furniture	Chairs

Total rows: 1000 of 1500 Query complete 00:00:00.178 Ln 23, Col 1

3) `select * from sales_target;`

Data Output | Messages | Notifications

	month_of_order_date text	category text	target smallint
1	Apr-22	Furniture	10400
2	May-22	Furniture	10500
3	Jun-22	Furniture	10600
4	Jul-22	Furniture	10800
5	Aug-22	Furniture	10900
6	Sep-22	Furniture	11000
7	Oct-22	Furniture	11100
8	Nov-22	Furniture	11300
9	Dec-22	Furniture	11400
10	Jan-23	Furniture	11500
11	Feb-23	Furniture	11600

Total rows: 36 of 36 Query complete 00:00:00.314 Ln 16, Col 1

Query 1: The marketing department is running a sales campaign and they target the customer with different sales materials. They categorized customers into groups based on the RFM model. **Show the number and percentage for each customer segment as the final result. Order the results by the percentage of customers.**

```sql
--Ques1)
-- Step 1: Combine `order_details` with `list_of_orders` temporarily
CREATE VIEW all_orders_1 AS
SELECT o.orderid, o.amount, o.profit, o.quantity, o.category, o.sub_category, l.order_date, l.customer_name, l.state, l.city
FROM order_details AS o
INNER JOIN list_of_orders AS l
ON o.orderid = l.orderid;

-- Step 2: Manually create customer segments into groups based on RFM model
CREATE VIEW customers_group AS
WITH rfm_table AS (
    SELECT
        MAX(order_date) AS latest_order_date,
        customer_name,
        -- Calculate recency using age and extract days
        EXTRACT(DAY FROM AGE(TO_DATE('2023-03-31', 'YYYY-MM-DD'), MAX(order_date))) AS recency,
        COUNT(DISTINCT orderid) AS frequency,
        SUM(amount) AS monetary,
        NTILE(5) OVER (ORDER BY EXTRACT(DAY FROM AGE(TO_DATE('2023-03-31', 'YYYY-MM-DD'), MAX(order_date))) DESC) AS R,
        NTILE(5) OVER (ORDER BY COUNT(DISTINCT orderid) ASC) AS F,
        NTILE(5) OVER (ORDER BY SUM(amount) ASC) AS M
    FROM
        all_orders_1
    GROUP BY
        customer_name
)
SELECT *,
    CASE
        WHEN (R >= 4 AND R <= 5) AND (((F + M) / 2) >= 4 AND ((F + M) / 2) <= 5) THEN 'Champions'
        WHEN (R >= 2 AND R <= 5) AND (((F + M) / 2) >= 3 AND ((F + M) / 2) <= 5) THEN 'Loyal Customers'
        WHEN (R >= 3 AND R <= 5) AND (((F + M) / 2) >= 1 AND ((F + M) / 2) <= 3) THEN 'Potential Loyalist'
        WHEN (R >= 4 AND R <= 5) AND (((F + M) / 2) >= 0 AND ((F + M) / 2) <= 1) THEN 'New Customers'
        WHEN (R >= 3 AND R <= 4) AND (((F + M) / 2) >= 0 AND ((F + M) / 2) <= 1) THEN 'Promising'
        WHEN (R >= 2 AND R <= 3) AND (((F + M) / 2) >= 2 AND ((F + M) / 2) <= 3) THEN 'Customers Needing Attention'
        WHEN (R >= 2 AND R <= 3) AND (((F + M) / 2) >= 0 AND ((F + M) / 2) <= 2) THEN 'About to Sleep'
        WHEN (R >= 0 AND R <= 2) AND (((F + M) / 2) >= 2 AND ((F + M) / 2) <= 5) THEN 'At Risk'
        WHEN (R >= 0 AND R <= 1) AND (((F + M) / 2) >= 4 AND ((F + M) / 2) <= 5) THEN 'Cannot Lose Them'
        WHEN (R >= 1 AND R <= 2) AND (((F + M) / 2) >= 1 AND ((F + M) / 2) <= 2) THEN 'Hibernating'
        WHEN (R >= 0 AND R <= 2) AND (((F + M) / 2) >= 0 AND ((F + M) / 2) <= 2) THEN 'Lost'
    END AS customer_segments
FROM
    rfm_table;

-- Step 3: Return the number & percentage of each customer segment
SELECT
    customer_segments,
    COUNT(DISTINCT customer_name) AS num_of_customers,
    ROUND(COUNT(DISTINCT customer_name) * 100.0 / NULLIF((SELECT COUNT(*) FROM customers_group), 0), 2) AS percent_customers
FROM customers_group
GROUP BY customer_segments
ORDER BY percent_customers DESC;
```

SQL code for query 1

For the sake of convenience, a view called **"all_orders_1"** was created by joining the **"order_details"** and *"***list_of_orders***"*. Another view called *"***customers_group***"* was created by using a subquery.

In the inner query, to find the recency for each customer, I used **EXTRACT AGE** function to find the difference between the 31st of March 2023 (the final date in the dataset) and the latest transaction date for each customer.

The **NTILE()** function is used to separate the customers into groups based on the recency, frequency, and monetary of the customers. We rank these customers from 1–5 using RFM values. Since we do not favour long customer inactivity and this function starts from 1, we order the recency in descending order, so that

the inactive customers would be given lower recency. We prefer customers to buy frequently and those who spend more, so we order frequency and monetary features in ascending order. In this way, those customers who buy less often or spend less would be given a lower value.

In the outer query, I used the **CASE** statement to group the customers into different segments based on the criteria as per listed in the table below.

Segment Name	Range of R values	Range of F and M Average
Champions	4 - 5	4 - 5
Loyal Customers	2 - 5	3 - 5
Potential Loyalist	3 - 5	1 - 3
New Customers	4 - 5	0 - 1
Promising	3 - 4	0 - 1
Customers Needing Attetion	2 - 3	2 - 3
About to Sleep	2 - 3	0 - 2
At Risk	0 - 2	2 - 5
Can't Lose Them	0 - 1	4 - 5
Hibernating	1 - 2	1 - 2
Lost	0 - 2	0 - 2

Data Output Messages Notifications

	customer_segments text	num_of_customers bigint	percent_customers numeric
1	Loyal Customers	111	33.43
2	Potential Loyalist	86	25.90
3	At Risk	61	18.37
4	Champions	39	11.75
5	Customers Needing Attention	24	7.23
6	Hibernating	6	1.81
7	About to Sleep	5	1.51

Result of query 1

RFM model helps the marketers to determine the customers' lifetime value, based on the recency (freshness of customer activity), frequency (number of customer transactions), and monetary (total transaction amount) value of the customers.

- Customers who made recent purchases are more responsive to promotions.
- Customers are more engaged and satisfied if they buy more frequently.
- Monetary value is used to differentiate the purchasing power of the customers.

Based on the result, almost 45% of the customers were loyal customers (spend well and often) and champions (spend well and often, as well as make a recent purchase). To retain these customers, the sellers may constantly ask for reviews and feedback to create personalized communication. Besides, the sellers can give rewards, special offers, discounts, or products that they are likely to be interested in so they feel valued.

To convert the 25.90% of potential loyalists to loyal customers or champions, the seller may offer a loyalty program to keep them engaged. To retain those who have bought a long time ago, only a few times, and have spent little (hibernating), the sellers can send standard communication for offering relevant products and good deals.

Showing the importance of buying and creating personalized product recommendations may help the sellers to retain customers who need attention (those who have recently purchased, but are still not sure whether they will make their next purchase from the company).

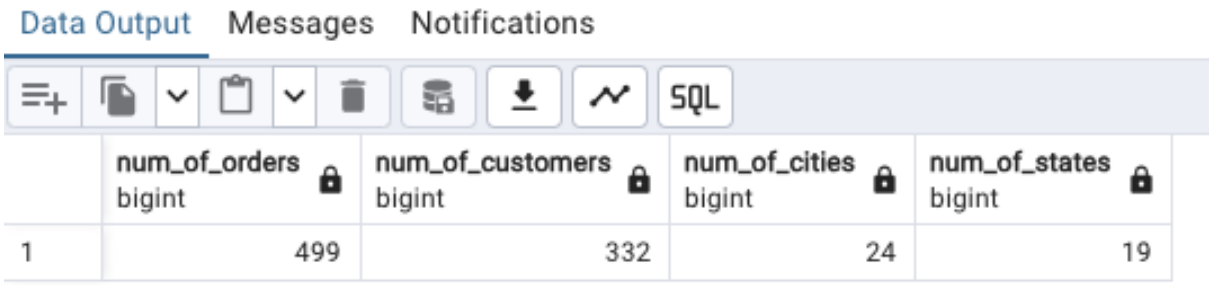
Finally, the sellers could reconnect with the customers who are at risk (those who have spent very little money and buy frequently but have not bought for a long time) by sending them personalized communications and other messages containing good deals.

Query 2: **Find the number of orders, customers, cities, and states.**

```sql
--Ques2)
--number of orders, customers, cities, states
SELECT COUNT(DISTINCT orderid) AS num_of_orders,
       COUNT(DISTINCT customer_name) AS num_of_customers,
       COUNT(DISTINCT city) AS num_of_cities,
       COUNT(DISTINCT state) AS num_of_states
FROM all_orders_1;
```

SQL code for query 2

The **COUNT DISTINCT** function is used to return the number of unique values in the column of *"orderid"*, *"customer_name"*, *"city"*, and *"state"*.



	num_of_orders bigint	num_of_customers bigint	num_of_cities bigint	num_of_states bigint
1	499	332	24	19

Result of query 2

Based on the result, there are 499 orders and 332 customers from 24 different cities and 19 states from April 2022 to March 2023.

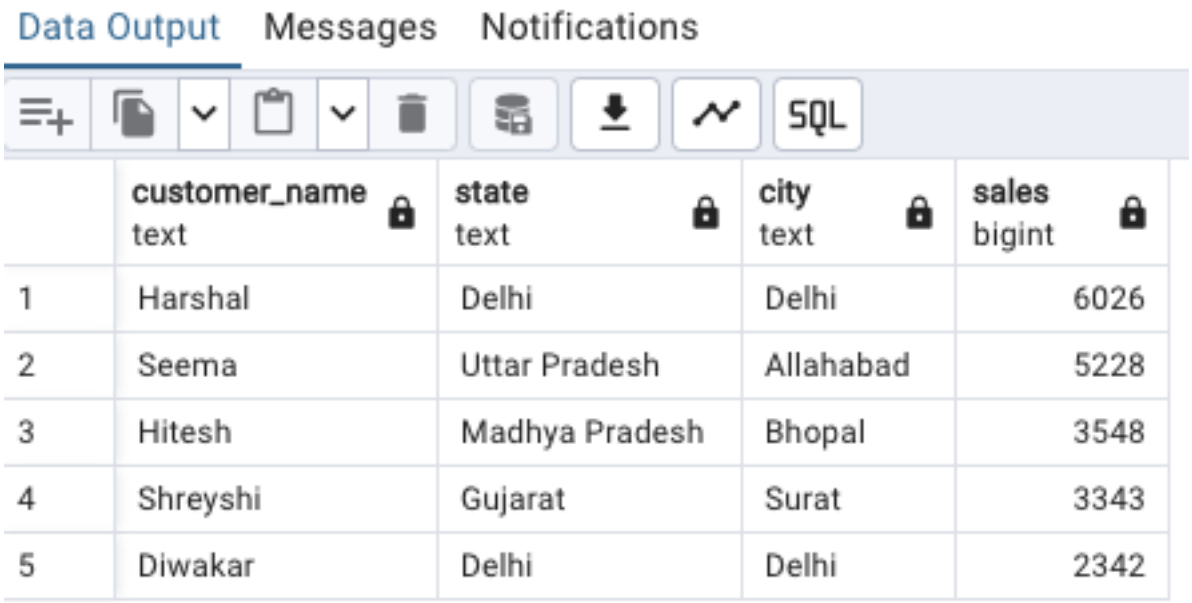
***Query 3*:** Find the new customers who made purchases in the year 2023. Only shows the top 5 new customers and their respective cities and states. Order the result by the amount they spent.

```
--Ques3)
--top 5 new customers
SELECT customer_name, state, city, SUM(amount) AS sales
FROM all_orders_1
WHERE customer_name NOT IN (
    SELECT DISTINCT customer_name
    FROM list_of_orders
    WHERE EXTRACT(YEAR FROM order_date) = 2022
)
AND EXTRACT(YEAR FROM order_date) = 2023
GROUP BY customer_name, state, city
ORDER BY sales DESC
LIMIT 5;
```

SQL code for query 3

The subquery placed after the **IN** operator returns a list of customers' names who made purchases in the year 2022. To find new customers in the year 2023,

simply use **NOT** before the **IN** operator to eliminate those old customers from the result. **YEAR** function is used to extract the year from the *"order_date"*.



Data Output | Messages | Notifications

	customer_name text	state text	city text	sales bigint
1	Harshal	Delhi	Delhi	6026
2	Seema	Uttar Pradesh	Allahabad	5228
3	Hitesh	Madhya Pradesh	Bhopal	3548
4	Shreyshi	Gujarat	Surat	3343
5	Diwakar	Delhi	Delhi	2342

Result of query 3

According to the result, two new customers come from Delhi. This suggests that the people from Delhi may have higher purchasing power.

Query 4: **Find the top 10 profitable states & cities so that the company can expand its business. Determine the number of products sold and the number of customers in these top 10 profitable states & cities.**

```sql
--Oues4
--number of customers, quantitien sold and profit mads & quantity sold in state & city
SELECT state,
       city,
       COUNT(DISTINCT customer_name) AS num_of_customers,
       SUM(profit) AS total_profit,
       SUM(quantity) AS total_quantity
       FROM all_orders_1
       GROUP BY state, city
       ORDER BY total_profit DESC
       LIMIT 10;
```

SQL code for query 4

I used the **GROUP BY** clause to group the customers based on the state and city. The **SUM()** function is used to add the profit made. Since the **ORDER BY** clause will sort the rows of the result set in ascending order, so I specified **DESC** to sort the profit in descending order. Then, I used

the **LIMIT** function to return the results of the top 10 profitable states and cities.

Data Output | Messages | Notifications

	state text	city text	num_of_customers bigint	total_profit bigint	total_quantity bigint
1	Maharashtra	Pune	16	4539	329
2	Madhya Pradesh	Indore	63	4159	1084
3	Uttar Pradesh	Allahabad	9	3081	138
4	Delhi	Delhi	21	2987	277
5	West Bengal	Kolkata	16	2500	216
6	Rajasthan	Udaipur	12	2010	115
7	Kerala	Thiruvananthapuram	11	1871	157
8	Maharashtra	Mumbai	61	1637	727
9	Gujarat	Surat	10	1345	93
10	Haryana	Chandigarh	10	1325	111

Result of query 4

The most profitable cities are Pune, followed by Indore, Allahabad, and Delhi. This may be because these areas are more developed (e.g. having a better internet connection and better logistics). This information may give some recommendations for the sellers who want to expand the market so that they know how to allocate their resources to fulfil the demands of their customers, leading to better customer engagement and higher profit.

Query 5: **Display the details (in terms of *"order_date"*, *"orderid"*, *state"*, and *"customer_name"*) for the first order in each state. Order the result by *"orderid"*.**

```
--Ques5)
--first order in each state
SELECT order_date, orderid, state, customer_name
FROM (SELECT *,
          ROW_NUMBER() OVER (PARTITION BY state ORDER BY state, orderid) AS rownumberperstate
          FROM all_orders_1) firstorder
WHERE rownumberperstate = 1
ORDER BY orderid;
```

SQL code for query 5

In this case, I used **ROW_NUMBER()** function to assign the number for each order based on states and finally set the filter to only show the result for the first order for each state.

Data Output | Messages | Notifications

	order_date date	orderid text	state text	customer_name text
1	2022-04-01	B-25602	Maharashtra	Pearl
2	2022-04-03	B-25603	Madhya Pradesh	Jahan
3	2022-04-03	B-25604	Rajasthan	Divsha
4	2022-04-05	B-25605	West Bengal	Kasheen
5	2022-04-06	B-25606	Karnataka	Hazel
6	2022-04-06	B-25607	Jammu and Kashmir	Sonakshi
7	2022-04-08	B-25608	Tamil Nadu	Aarushi
8	2022-04-09	B-25609	Uttar Pradesh	Jitesh
9	2022-04-09	B-25610	Bihar	Yogesh
10	2022-04-11	B-25611	Kerala	Anita
11	2022-04-12	B-25612	Punjab	Shrichand
12	2022-04-12	B-25613	Haryana	Mukesh
13	2022-04-13	B-25614	Himachal Pradesh	Vandana
14	2022-04-15	B-25615	Sikkim	Bhavna
15	2022-04-15	B-25616	Goa	Kanak
16	2022-04-17	B-25617	Nagaland	Sagar
17	2022-04-18	B-25618	Andhra Pradesh	Manju
18	2022-04-18	B-25619	Gujarat	Ramesh
19	2022-12-10	B-25904	Delhi	Swapnil

Result of query 5

According to the result, Delhi is the last state where this Indian e-commerce website established its footprint. However, the profit generated from Delhi state

is much higher than that from Gujarat state. Hence, we may conclude that the customers from Delhi truly have higher purchasing power.

Query 6: **Determine the number of orders (in the form of a histogram) and sales for different days of the week.**

```
--Ques6)
--sales in different days
SELECT day_of_order,
       LPAD('*', num_of_orders::int, '*') AS num_of_orders_visualization,
       sales
FROM (
    SELECT
        TO_CHAR(order_date, 'Day') AS day_of_order,
        COUNT(DISTINCT orderid) AS num_of_orders,
        SUM(quantity) AS quantity,
        SUM(amount) AS sales
    FROM all_orders_1
    GROUP BY TO_CHAR(order_date, 'Day')
) sales_per_day
ORDER BY sales DESC;
```

SQL code for query 6

To return the weekday name for the *"order_date"*, we may use the **TO_CHAR** function. The next step is to pass the value returned by the **COUNT DISTINCT** function as an argument to the **LPAD()** function to control the number of * characters to return for each day.

Data Output | Messages | Notifications

	day_of_order text	num_of_orders_visualization text	sales bigint
1	Friday	**********	86740
2	Tuesday	**********	71070
3	Saturday	**********	67784
4	Wednesday	**********	62381
5	Sunday	**********	52069
6	Thursday	**********	46280
7	Monday	**********	43749

Result of query 6

The result of this query matches our expectation, whereby we could see that the highest sales happened on Sunday. However, if we are solely looking at the number of orders is the highest on Monday. This may happen because the customers selected the items they want to order on Sunday, and placed their orders on Monday.

Query 7: **Check the monthly profitability and monthly quantity sold to see if there are patterns in the dataset.**

```
--Ques7)
SELECT
    CONCAT(TO_CHAR(order_date, 'Month'),'-',TO_CHAR(order_date, 'YYYY')) AS month_of_year,
    SUM(profit) AS total_profit,
    SUM(quantity) AS total_quantity
FROM all_orders_1
GROUP BY
    TO_CHAR(order_date, 'Month'),
    TO_CHAR(order_date, 'YYYY')
ORDER BY
    (TO_CHAR(order_date, 'Month') = 'April' AND TO_CHAR(order_date, 'YYYY') = '2022') DESC,
    (TO_CHAR(order_date, 'Month') = 'May' AND TO_CHAR(order_date, 'YYYY') = '2022') DESC,
    (TO_CHAR(order_date, 'Month') = 'June' AND TO_CHAR(order_date, 'YYYY') = '2022') DESC,
    (TO_CHAR(order_date, 'Month') = 'July' AND TO_CHAR(order_date, 'YYYY') = '2022') DESC,
    (TO_CHAR(order_date, 'Month') = 'August' AND TO_CHAR(order_date, 'YYYY') = '2022') DESC,
    (TO_CHAR(order_date, 'Month') = 'September' AND TO_CHAR(order_date, 'YYYY') = '2022') DESC,
    (TO_CHAR(order_date, 'Month') = 'October' AND TO_CHAR(order_date, 'YYYY') = '2022') DESC,
    (TO_CHAR(order_date, 'Month') = 'November' AND TO_CHAR(order_date, 'YYYY') = '2022') DESC,
    (TO_CHAR(order_date, 'Month') = 'December' AND TO_CHAR(order_date, 'YYYY') = '2022') DESC,
    (TO_CHAR(order_date, 'Month') = 'January' AND TO_CHAR(order_date, 'YYYY') = '2023') DESC,
    (TO_CHAR(order_date, 'Month') = 'February' AND TO_CHAR(order_date, 'YYYY') = '2023') DESC,
    (TO_CHAR(order_date, 'Month') = 'March' AND TO_CHAR(order_date, 'YYYY') = '2023') DESC;
```

SQL code for query 7

I used the **TO_CHAR** functions to extract the month and year from the *"order_date"* . It can be noticed that I used **CONCAT()** function too. This is because I want to add a dash between the extractions. After that, I grouped the profit and quantity sold by the month using the **GROUP BY** clause and order the result based on the month in ascending order.

Data Output Messages Notifications

	month_of_year text	total_profit bigint	total_quantity bigint
1	September-2022	-4963	331
2	April -2022	-2742	370
3	June -2022	-4970	369
4	November -2022	11619	578
5	July -2022	-2138	240
6	August -2022	-2180	446
7	March -2023	10077	751
8	October -2022	3093	419
9	December -2022	5284	412
10	February -2023	5917	512
11	May -2022	-3584	423
12	January -2023	9760	745

Result of query 7

Losses occurred from April 2022 to September 2022. Luckily, there was a high profit from October 2018 onwards, followed along with an increase in the quantity sold (although it fluctuates). The total profit was able to cover all the losses it suffered previously. Besides, it also indicates that consumers started to shift toward online shopping.
However, a high number of products sold does not guarantee a high profit because the highest loss occurred in June 2022, but the quantity sold was the lowest in July 2022.

Query 8: **Find the total sales, total profit, and total quantity sold for each category and sub-category. Return the maximum cost and maximum price for each sub-category too.**

```sql
--Ques8)
--Find order quantity, profit, amount for each subcategory
--electronic games & tables subcategories resulted in loss
CREATE VIEW order_detail_by_total AS
SELECT category,sub_category,
       SUM(quantity) AS total_order_quantity,
       sum(profit) AS total_profit,
       SUM(amount) AS total_amount
FROM order_details
GROUP BY category,sub_category
ORDER by total_order_quantity DESC;


--maximum cost per unit & maximum price per unit for each subcategory
CREATE VIEW order_detail_by_unit AS
SELECT category, sub_category, MAX(cost_per_unit) AS max_cost, max(price_per_unit) AS max_price
FROM (SELECT *, round((amount-profit)/quantity, 2) AS cost_per_unit, round(Amount/Quantity,2) AS price_per_unit
       FROM order_details)c
GROUP BY category,sub_category
ORDER BY max_cost DESC;


--combine order_details_by_unit table and order_details_by_total table
SELECT t.category, t.sub_category, t.total_order_quantity, t.total_profit, t.total_amount, u.max_cost, u.max_price
FROM order_detail_by_total As t
INNER JOIN order_detail_by_unit AS u
ON t.sub_category=u.sub_category;
```

SQL code for query 8

To find the total for quantity, profit, and amount, I used the **SUM()** function and stored the result in a new view called *"order_detail_by_total"*.

After that, we need to find the cost of the product and the price of the product. The cost of the product is obtained by deducting the profit from the amount and dividing the result by the quantities sold, while the price of the product is obtained by dividing the amount by the quantities sold. To find the maximum cost and price, I used the **MAX()** function and stored the result in a new view called *"order_detail_by_unit"*.

The final result is obtained by simply performing an inner join on the *"order_detail_by_total"* and *"order_detail_by_unit"* views.

Data Output | Messages | Notifications

	category text	sub_category text	total_order_quantity bigint	total_profit bigint	total_amount bigint	max_cost numeric	max_price numeric
1	Furniture	Tables	61	-4011	22614	811.00	872.00
2	Electronics	Phones	304	2207	46119	617.00	654.00
3	Clothing	Trousers	135	2847	30039	538.00	569.00
4	Furniture	Chairs	277	577	34222	459.00	423.00
5	Furniture	Bookcases	297	4888	56861	394.00	438.00
6	Electronics	Printers	291	5964	58252	344.00	378.00
7	Electronics	Electronic Games	297	-1236	39168	309.00	312.00
8	Electronics	Accessories	262	3559	21728	201.00	260.00
9	Clothing	Saree	782	352	53511	197.00	212.00
10	Furniture	Furnishings	310	844	13484	106.00	116.00
11	Clothing	Stole	671	2559	18546	53.00	55.00
12	Clothing	Hankerchief	754	2098	14608	49.00	53.00
13	Clothing	T-shirt	305	1500	7382	46.00	50.00
14	Clothing	Shirt	271	1131	7555	46.00	49.00
15	Clothing	Kurti	164	181	3361	44.00	49.00
16	Clothing	Leggings	186	260	2106	17.00	19.00
17	Clothing	Skirt	248	235	1946	13.00	13.00

Result of query 8

All types of clothing made a profit. The top 3 best-selling sub-categories are under the category of clothing, and they are saree, handkerchief, and stole. The sellers can provide products that are complementary to these top 3 best-selling products to boost sales as there is a high chance that customers buy clothing products in pairs. For example, leggings and insoles.

On the other hand, the sellers should avoid selling electronic games and focus more on selling printers and accessories because electronic games led to losses although the quantity of electronic games is higher than that of printers and accessories.

Conclusion

This SQL project shed light on the sales performance of an Indian e-commerce platform, exposing significant patterns in user behaviour, product performance,

and market trends. Customers were split into distinct groups using the RFM model, with roughly 45% of them being loyal or champions. Strategies for retaining and converting other categories were offered, with a focus on targeted marketing and loyalty programs.

The study also identified Delhi as a high-purchasing power region, emphasizing the opportunity for targeted marketing initiatives. Furthermore, prosperous cities such as Pune and Indore were identified as prime locations for corporate expansion. The project found that Sundays had the most sales and Mondays had the most orders, indicating that buying behaviour patterns are worth investigating.

Furthermore, the analysis revealed that, while high product sales do not always translate into high profitability, the clothing category, specifically sarees, handkerchiefs, and stoles, was extremely profitable. In contrast, electronic games, despite large sales, resulted in losses, indicating the need for strategic modifications. Overall, this study made concrete recommendations for increasing consumer engagement, refining product offers, and broadening market reach.