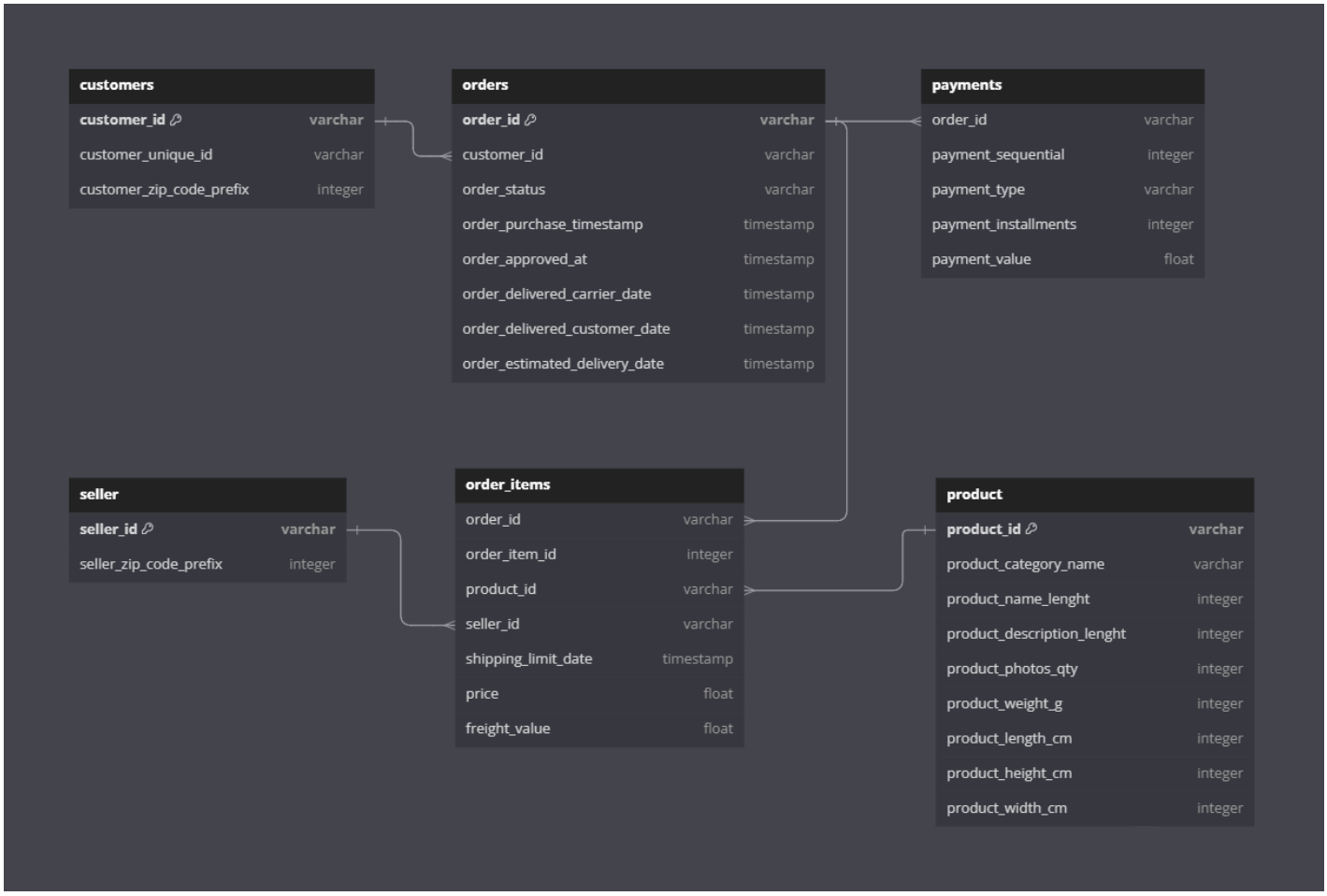
Project Context:

Amazon India is analysing customer and sales data from Amazon Brazil to uncover key trends that can enhance its services in India. The primary goal is to understand customer behaviours, product preferences, and payment methods. This analysis aims to improve the shopping experience and identify new opportunities in the Indian market.

The project utilizes several tables, including **Customers, Orders, Order Items, Product, Sellers, and Payments**, to address important questions through SQL queries and deliver valuable insights.

Overview of Schema:

The schema consists of seven interconnected tables that provide insights into the operations of Amazon Brazil, that includes relationships and primary keys for each table.



ANALYSIS I

Question 1

Problem Statement:

To simplify its financial reports, Amazon India needs to standardize payment values. Round the average payment values to integer (no decimal) for each payment type and display the results sorted in ascending order.

- *Output: payment_type, rounded_avg_payment*

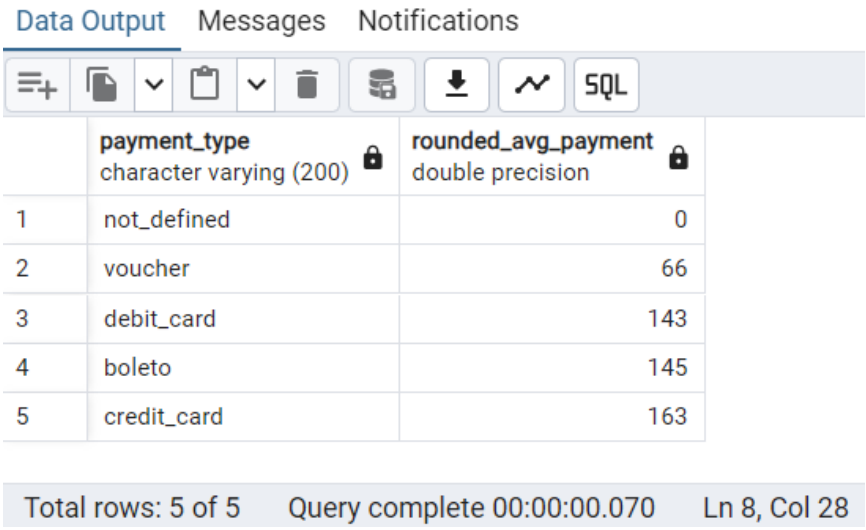
Approach:

1. **Identifying Relevant Tables and Columns:**
   - **Table:** *Payments*
   - **Columns:** payment_type, payment_value
2. **Calculating Average Payment Value:**
   - We used the **AVG()** function to compute the average payment value for each method.
   - The results are grouped by ***payment_type*** to ensure we obtain averages specific to each payment method.
3. **Rounding the Averages:**
   - we applied the **ROUND()** function to round these average values to the nearest whole number.
4. **Sorting the Results:**
   - Finally, we ordered the results in ascending order based on the *rounded average payment values.*

SQL Query:

```
select payment_type,
round(avg(payment_value)) as rounded_avg_payment
from amazon_brazil.payments
group by payment_type
order by rounded_avg_payment;
```

Output:



Data Output | Messages | Notifications

	payment_type character varying (200)	rounded_avg_payment double precision
1	not_defined	0
2	voucher	66
3	debit_card	143
4	boleto	145
5	credit_card	163

Total rows: 5 of 5 Query complete 00:00:00.070 Ln 8, Col 28

Recommendations:

1. **Enhance High-Value Methods:**
   - Focus on improving the user experience for *credit_card* and *boleto*, as they have higher average payment amounts.
2. **Targeted Promotions:**
   - Start promotions specifically for customers using high-value payment methods to encourage loyalty and increase sales.
3. **Increase Engagement with Different Payment Types:**
   - Offer discounts or incentives to promote the use of *debit_card*, *voucher* which currently has a lower average payment value.
4. **Address Undefined Payment Types:**
   - Look into the *not_defined* category to find ways to make improvements.

Question 2

Problem Statement:

To refine its payment strategy, Amazon India wants to know the distribution of orders by payment type. Calculate the percentage of total orders for each payment type, rounded to one decimal place, and display them in descending order.

- ***Output**: payment_type, percentage_orders*

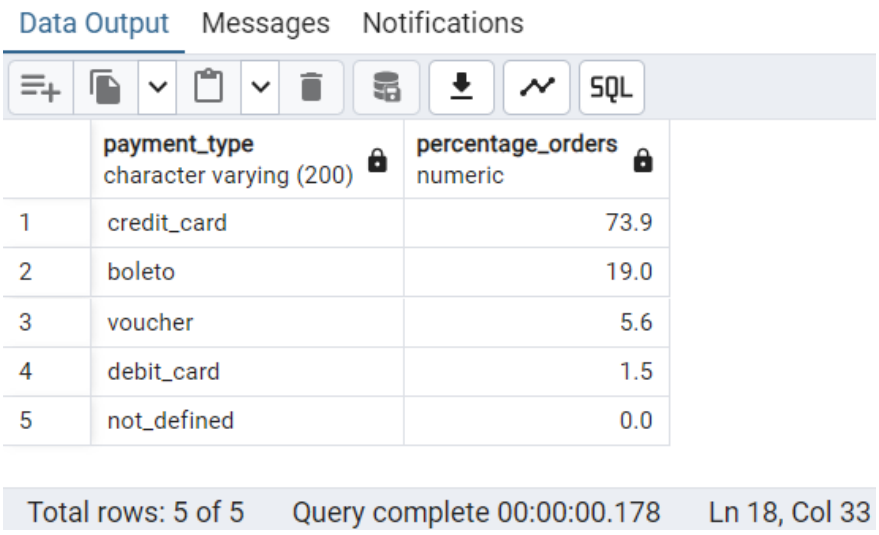
Approach:

1. **Identifying Relevant Tables and Columns:**
   - **Table:** *Payments*
   - **Columns:** payment_type, order_id
2. **Calculating Total Orders:**
   - Used the **COUNT()** function to count the number of orders for each payment type.
3. **Calculating percentage of Orders:**
   - Divided the count of each payment type's orders by the total number of orders and multiplied by 100 to calculate the percentage.
4. **Rounding the results:**
   Used the **ROUND()** function to round the percentages to one decimal place.
5. **Grouping and Sorting:**
   - Grouped the results by ***payment_type***.
   - Sorted the result by ***percentage of orders*** in descending order.

SQL Query :

```
select payment_type,
round(count(order_id) * 100.0/(select count(*) from amazon_brazil.payments),1)
as percentage_orders
from amazon_brazil.payments
group by payment_type
order by percentage_orders desc;
```

Output:


Data Output Messages Notifications

SQL

	payment_type character varying (200)	percentage_orders numeric
1	credit_card	73.9
2	boleto	19.0
3	voucher	5.6
4	debit_card	1.5
5	not_defined	0.0

Total rows: 5 of 5 Query complete 00:00:00.178 Ln 18, Col 33

Recommendations:

1. **Optimize High-Usage Methods**:
    - Improve user experience for popular payment type like *credit_card*.
2. **Analyse Low-Usage Methods**:
    - Identify barriers for less popular payment types like *debit_card* and *voucher*, enhance them by offering incentives or simplify the process to encourage customers.
3. **Address Undefined Payment Types:**
    - Look into the *not_defined* category to find ways to make improvements.

Question 3

Problem Statement:

Amazon India seeks to create targeted promotions for products within specific price ranges. Identify all products priced between 100 and 500 BRL that contain the word 'Smart' in their name. Display these products, sorted by price in descending order.

- ***Output**: product_id, price*

Approach:

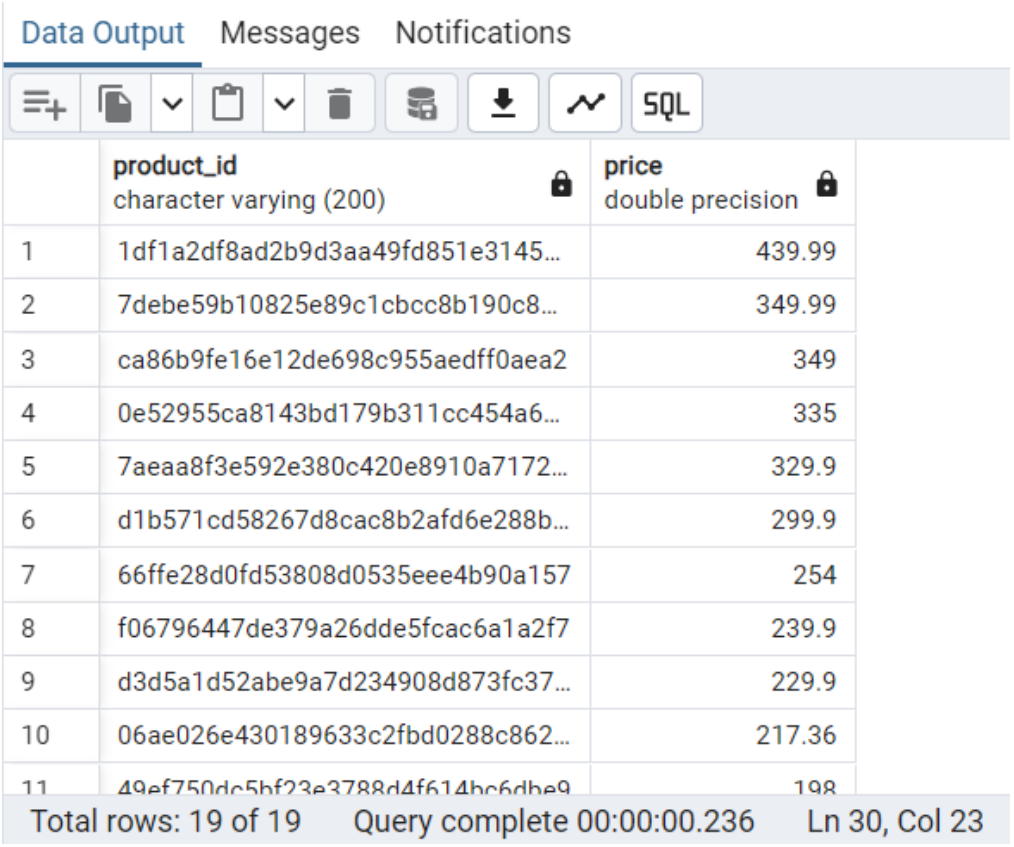
1. **Identifying Relevant Tables and Columns:**
   - **Table:** *Product* and *Order_Items*
   - **Columns:** *product_id, price, product_category_name*
2. **Joining Tables:**
   - Performed an inner join between the *product* and *order_items* tables using *product_id*.
3. **Filtering Results:**
   - Used the **WHERE** clause to filter products with prices between 100 and 500.
4. **Filtering Product Category:**
   - Applied the **LIKE** operator with the *lower()* function to select products containing *"smart"* in their category names.
5. **Grouping and Sorting:**
   - Grouped the results by *product_id, price*.
   - Sorted the result by *price* in descending order.

SQL Query:

```
select p.product_id,o.price
from amazon_brazil.product as p
join amazon_brazil.order_items as o
on p.product_id=o.product_id
where o.price between 100 and 500
and lower(p.product_category_name)like( '%smart%' )
group by p.product_id,o.price
order by o.price desc;
```

Output:

We got a total of 19 product IDs as output from the query that match the given conditions.



Data Output | Messages | Notifications

	product_id character varying (200)	price double precision
1	1df1a2df8ad2b9d3aa49fd851e3145...	439.99
2	7debe59b10825e89c1cbcc8b190c8...	349.99
3	ca86b9fe16e12de698c955aedff0aea2	349
4	0e52955ca8143bd179b311cc454a6...	335
5	7aeaa8f3e592e380c420e8910a7172...	329.9
6	d1b571cd58267d8cac8b2afd6e288b...	299.9
7	66ffe28d0fd53808d0535eee4b90a157	254
8	f06796447de379a26dde5fcac6a1a2f7	239.9
9	d3d5a1d52abe9a7d234908d873fc37...	229.9
10	06ae026e430189633c2fbd0288c862...	217.36
11	49ef750dc5bf23e3788d4f614bc6dbe9	198

Total rows: 19 of 19 Query complete 00:00:00.236 Ln 30, Col 23

Recommendations:

1. **Promote Smart Products:**
   - Highlight products in the "*smart*" category to attract more customers, especially those within the price range.
2. **Focus on Popular Mid-Range Smart Products:**
   - These products are priced between 100 and 500, which is a good range. You could promote them more to increase sales.
3. **Expand Product Range:**
   - Consider adding more products in the "smart" category that fall within this price range to enhance selection.

Question 4

Problem Statement:

To identify seasonal sales patterns, Amazon India needs to focus on the most successful months. Determine the top 3 months with the highest total sales value, rounded to the nearest integer.

- ***Output**: month, total_sales*

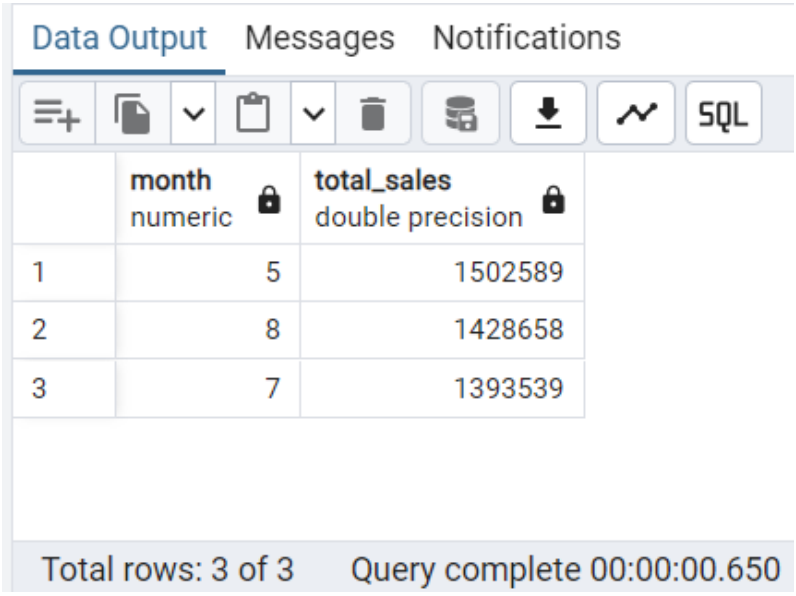
Approach:

1. **Identifying Relevant Tables and Columns:**
   - **Table:** *Orders* and *Order_Items*
   - **Columns:** order_purchased_timestamp, order_id, price
2. **Joining Tables:**
   - Combined the *orders* and *order_items* tables using the *order_id* .
3. **Extract Month:**
   - Used the **EXTRACT()** function to retrieve the month from the *order_purchased_timestamp* in the orders table.
4. **Calculating Total Sales:**
   - Summed the *price* for each month from *order_ items* table using the ***SUM()*** function to calculate *total_sales* and rounded the result to the nearest integer.
5. **Grouping and Sorting:**
   - Grouped the results by *month*.
   - Sorted the results in descending order based on *total sales*.
6. **Limiting the Results:**
   - Used **LIMIT** to display only the top 3 months with the highest sales.

SQL Query:

```
select extract(month from o.order_purchased_timestamp) as month,
round(sum(oi.price )) as total_sales
from amazon_brazil.orders o
join amazon_brazil.order_items oi
on o.order_id = oi.order_id
group by month
order by total_sales desc
limit 3;
```

Output:



	month numeric	total_sales double precision
1	5	1502589
2	8	1428658
3	7	1393539

Total rows: 3 of 3 Query complete 00:00:00.650

Recommendations:

1. **Focus Marketing Efforts on May:**
   - Since May had the highest sales, focus on repeating whatever worked well. Either it was promotions or new product launches in order to sustain high performance.
2. **Boost Sales in Other Months:**
   - Analyse why there is a slight variation in July and August compared to May. Consider targeted promotions or new product launches during these months to encourage more purchases.
3. **Analyze What Worked in May:**
   - Look what factors contributed to the highest sales in May, such as promotions, product launches, or seasonal trends, and try to apply these strategies in other months.

Question 5

Problem Statement

Amazon India is interested in product categories with significant price variations. Find categories where the difference between the maximum and minimum product prices is greater than 500 BRL.

- ***Output**: product_category_name, price_difference*

Approach:

1. **Identifying Relevant Tables and Columns:**
   - **Table:** *Product* and *Order_Items*
   - **Columns:** *product_category_name, product_id, price*
2. **Joining Tables:**
   - Combined the *product* and *order_items* tables using the *product_id* to access product categories and their prices.
3. **Calculate Price Difference:**
   - Used the **MAX()** function to find the highest price and the **MIN()** function for the lowest price in each product category.
   - Calculated the price difference by subtracting the maximum price from the minimum price.
4. **Grouping and Sorting:**
   - Grouped the results by *product_category_name* to calculate prices within each category.
5. **Filtering Results:**
   - Used the **HAVING** clause to include only those categories where the price difference is greater than 500.

SQL Query:

```
select p.product_category_name, max(o.price) - min(o.price)
as price_difference
from amazon_brazil.product p
join amazon_brazil.order_items o
on p.product_id = o.product_id
group by p.product_category_name
having max(o.price) - min(o.price) > 500;
```

Output:

We got a total of 57 product_category_names as output from the query that match the given conditions.

Data Output | Messages | Notifications

	product_category_name character varying (200)	price_difference double precision
1	climatizacao	1588.1
2	livros_importados	730.01
3	[null]	3977
4	ferramentas_jardim	3923.65
5	dvds_blu_ray	1411.1
6	cine_foto	867.19
7	beleza_saude	3122.8
8	livros_interesse_geral	893.9
9	tablets_impressao_imagem	875.09
10	papelaria	1690.71

Total rows: 57 of 57 Query complete 00:00:00.095 Ln 55, Col 42

Recommendations:

1. **Adjust Pricing Strategy:**
    - For categories like "ferramentas_jardim" and "beleza_saude" the price differences are large, consider reviewing your pricing strategy. There might be opportunities to introduce mid-range products or offer better pricing consistency.
2. **Promote Mid-Range Products:**
    - For categories with high price differences, consider promoting mid-range products to attract customers who may be hesitant to buy the highest-priced items.
3. **Monitor Competitors:**
    - Monitor competitors by regularly tracking how they price similar products in the market. This practice helps to stay competitive and allows to adjust pricing strategy as needed to attract more customers.
4. **Customer Education:**
    - Provide information about why there are significant price differences in certain categories. Educating customers on features, quality, or brand value can help justify the higher prices for premium products.

Question 6

Problem Statement:

To enhance the customer experience, Amazon India wants to find which payment types have the most consistent transaction amounts. Identify the payment types with the least variance in transaction amounts, sorting by the smallest standard deviation first.

- ***Output**: payment_type, std_deviation*

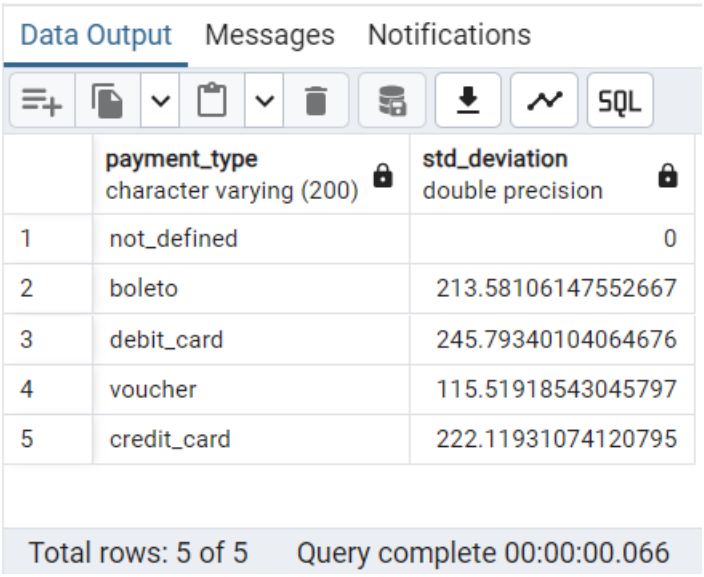
Approach:

1. **Identifying Relevant Tables and Columns:**
   - **Table:** *Payments*
   - **Columns:** payment_type
2. **Calculating Standard Deviation:**
   - Used the **STDDEV()** function to compute the standard deviation of *payment_value* for each *payment_type*.
3. **Grouping and Sorting:**
   - Grouped the results by *payment_type.*
   - Ordered the results in ascending order based on the standard deviation to see which payment types have the least to most variability in payment amounts.

SQL Query:

```
select payment_type, stddev(payment_value) as std_deviation
from amazon_brazil.payments
group by payment_type
order by std_deviation asc ;
```

Output:



Data Output | Messages | Notifications

	payment_type character varying (200)	std_deviation double precision
1	not_defined	0
2	boleto	213.58106147552667
3	debit_card	245.79340104064676
4	voucher	115.51918543045797
5	credit_card	222.11931074120795

Total rows: 5 of 5 Query complete 00:00:00.066

Recommendations:

1. **Standardize Payment Options:**
   - Since "not_defined" has no variation, ensure all payment types are clearly defined and categorized to avoid confusion for customers.
2. **Focus on Voucher Promotions:**
   - With the lowest standard deviation, vouchers have more consistent values. Consider promoting vouchers as a reliable payment option to encourage customer use.
3. **Analyze Higher Variability Payments:**
   - For payment types like debit and credit cards, which have higher standard deviations, identify the reasons. This can help to understand customer behaviour and improve pricing strategies.
4. **Focus on Consistency**:
   - Aim to provide more consistent pricing across all payment methods. This can enhance customer trust and satisfaction.

Question 7

Problem Statement:

Amazon India wants to identify products that may have incomplete name in order to fix it from their end. Retrieve the list of products where the product category name is missing or contains only a single character.

- ***Output**: product_id, product_category_name*

Approach:

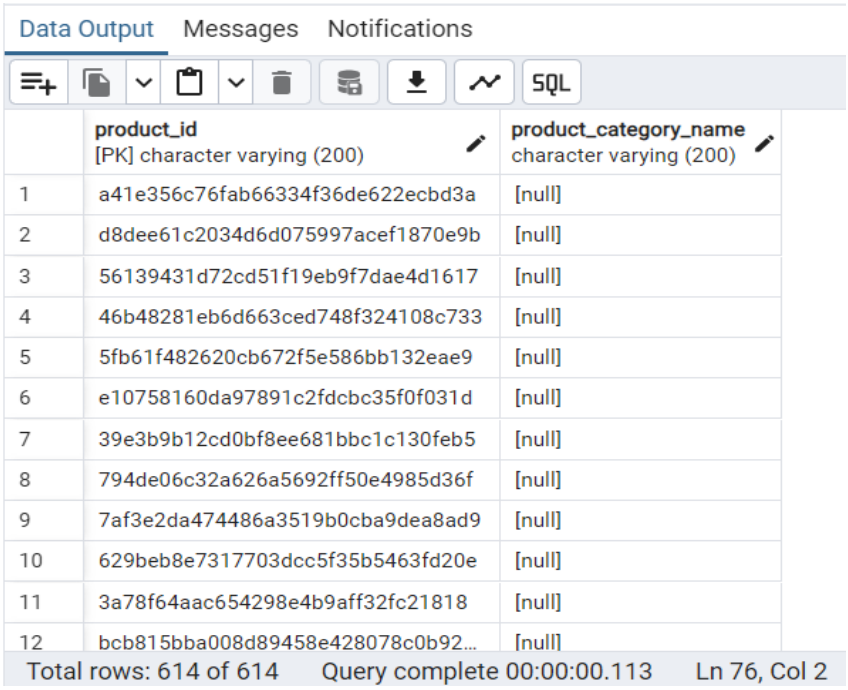
1. **Identifying Relevant Tables and Columns:**
   - **Table:** *Product*
   - **Columns:** product_id and product_category_name
2. **Select the relevant data:**
   - Retrieve product_id and product_category_name from the product table.
3. **Filter for Null or Short Categories:**
   - Use a **WHERE** clause to find products where *product_category_name* is either null or has a length of 1 character.

SQL Query:

```
select product_id, product_category_name
from amazon_brazil.product
where product_category_name is null
or length(product_category_name) = 1;
```

Output:

The output consists of total 614 Product_IDs which meet the given conditions.



Data Output | Messages | Notifications

SQL

	product_id [PK] character varying (200)	product_category_name character varying (200)
1	a41e356c76fab66334f36de622ecbd3a	[null]
2	d8dee61c2034d6d075997acef1870e9b	[null]
3	56139431d72cd51f19eb9f7dae4d1617	[null]
4	46b48281eb6d663ced748f324108c733	[null]
5	5fb61f482620cb672f5e586bb132eae9	[null]
6	e10758160da97891c2fdcbc35f0f031d	[null]
7	39e3b9b12cd0bf8ee681bbc1c130feb5	[null]
8	794de06c32a626a5692ff50e4985d36f	[null]
9	7af3e2da474486a3519b0cba9dea8ad9	[null]
10	629beb8e7317703dcc5f35b5463fd20e	[null]
11	3a78f64aac654298e4b9aff32fc21818	[null]
12	bcb815bba008d89458e428078c0b92...	[null]

Total rows: 614 of 614 Query complete 00:00:00.113 Ln 76, Col 2

Recommendations:

1. **Fix Missing Data:**
    - Look into why some products have no category or very short names. Consider updating or correcting these entries to improve data quality.
2. **Category Review:**
    - Short or missing categories could lead to problems in analysis or reporting. It's important to clean up these fields for better insights.
3. **Improve Product Information:**
    - Make sure all products have accurate and complete category names to help with search, sorting , and customer understanding.

ANALYSIS II

Question 1

Problem Statement:

Amazon India wants to understand which payment types are most popular across different order value segments (e.g., low, medium, high). Segment order values into three ranges: orders less than 200 BRL, between 200 and 1000 BRL, and over 1000 BRL. Calculate the count of each payment type within these ranges and display the results in descending order of count

- *Output: order_value_segment, payment_type, count*

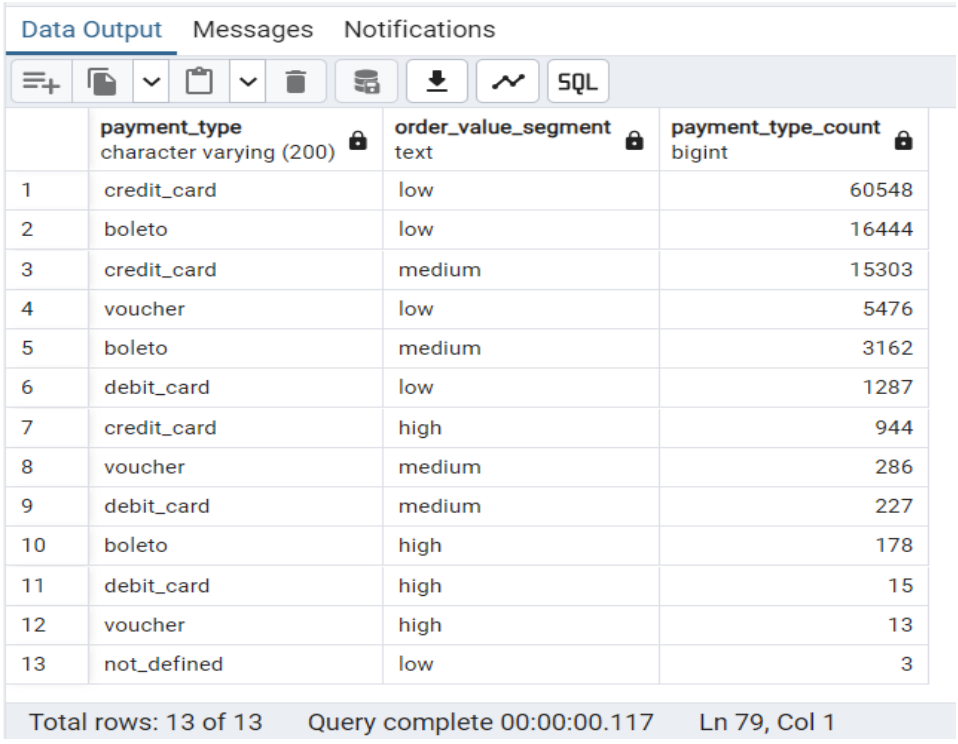
Approach:

1. **Identifying Relevant Tables and Columns:**
   - **Table:** *Payments*
   - **Columns:** payment_type and payment_value
2. **Classifying Payment Values:**
   - Used the **CASE** statement to classify *payment_value* into segments: 'low', 'medium', and 'high'.
3. **Counting and Sorting:**
   - *Count()* is used to count how many payments fall into each category (low, medium, high) for every payment method, and then sorted the results in descending order as per *payment_type_count*.

SQL Query:

```
select payment_type,
case
when payment_value < 200 then 'low'
when payment_value between 200 and 1000 then 'medium'
when payment_value>1000 then 'high'
else 'NA'
end as order_value_segment,
count (*) as payment_type_count
from amazon_brazil.payments
group by payment_type,order_value_segment
order by payment_type_count desc;
```

Output:



Data Output | Messages | Notifications

SQL

	payment_type character varying (200)	order_value_segment text	payment_type_count bigint
1	credit_card	low	60548
2	boleto	low	16444
3	credit_card	medium	15303
4	voucher	low	5476
5	boleto	medium	3162
6	debit_card	low	1287
7	credit_card	high	944
8	voucher	medium	286
9	debit_card	medium	227
10	boleto	high	178
11	debit_card	high	15
12	voucher	high	13
13	not_defined	low	3

Total rows: 13 of 13 Query complete 00:00:00.117 Ln 79, Col 1

Recommendations:

1. **Focus on Low-Value Payments:**
    - Focus on the 'low' segment to understand why these payments are lower. Find ways to encourage customers to spend more.
2. **Enhance Medium-Value Offers:**
    - For the 'medium' segment, explore ways to convert these customers into high-value spenders through offers or rewards.
3. **Encourage High-Value Payments:**
    - Analyse what makes high-value payments happen and try to use those ideas to increase sales in other areas.

Question 2:

Problem Statement:

Amazon India wants to analyse the price range and average price for each product category. Calculate the minimum, maximum, and average price for each category, and list them in descending order by the average price.

- *Output: product_category_name, min_price, max_price, avg_price*

Approach:

1. **Identifying Relevant Tables and Columns:**
   - **Table:** *Product* and *Order_Items*
   - **Columns:** product_category_name, product_id, price
2. **Joining Tables:**
   - Combined the *product* and *order_items* tables using the *product_id* to link products and their prices.
3. **Calculating Price:**
   - Used aggregate functions:
     MIN() to find the lowest price.
     MAX() to find the highest price.
     AVG() to calculate the average price.
4. **Grouping and Sorting:**
   - Grouped the results by *product_category_name* to get the price data for each category.
   - Ordered the results by *average price* in descending order to see which categories have the highest average prices.

SQL Query:

```
select p.product_category_name,
min(o.price) as min_price,
max(o.price) as max_price,
avg(o.price) as avg_price
from amazon_brazil.product p
join amazon_brazil.order_items o
on p.product_id=o.product_id
```

group by p.product_category_name

order by avg_price desc;

Output:

This output consists of total 79 Product_category_names with min, max and avg_prices.

Data Output | Messages | Notifications

	product_category_name character varying (200)	min_price double precision	max_price double precision	avg_price double precision
1	pcs	34.5	6729	1098.3405418719212
2	portateis_casa_forno_e_cafe	10.19	2899	624.2856578947369
3	eletrodomesticos_2	13.9	2350	476.1249579831935
4	agro_industria_e_comercio	12.99	2990	341.6610426540285
5	instrumentos_musicais	4.9	4399.87	281.6159999999996
6	eletroportateis	6.5	4799	280.7784683357877
7	portateis_cozinha_e_preparadores_de_alimentos	17.42	1099	264.5686666666667
8	telefonia_fixa	6	1790	225.69318181818198
9	construcao_ferramentas_seguranca	8.9	3099.9	208.9923711340207
10	relogios_presentes	8.99	3999.9	200.9118770875104
11	climatizacao	10.9	1599	185.2692255892258

Total rows: 79 of 79 Query complete 00:00:00.135 Ln 104, Col 25

Recommendations:

1. **Focus on High-priced Categories:**
    - By using this data, we can try to find which categories have higher average prices and consider promoting these products more.
2. **Check Low-Priced Categories:**
    - Check the categories with lower prices to see if any improvement can be made on them or adjust pricing.
3. **Check Price Differences:**
    - If some categories have a big difference between the highest and lowest prices, check if it's because of different product versions or features.

Question 3:

Problem Statement:

Amazon India wants to identify the customers who have placed multiple orders over time. Find all customers with more than one order, and display their customer unique IDs along with the total number of orders they have placed.

- *Output: customer_unique_id, total_orders*

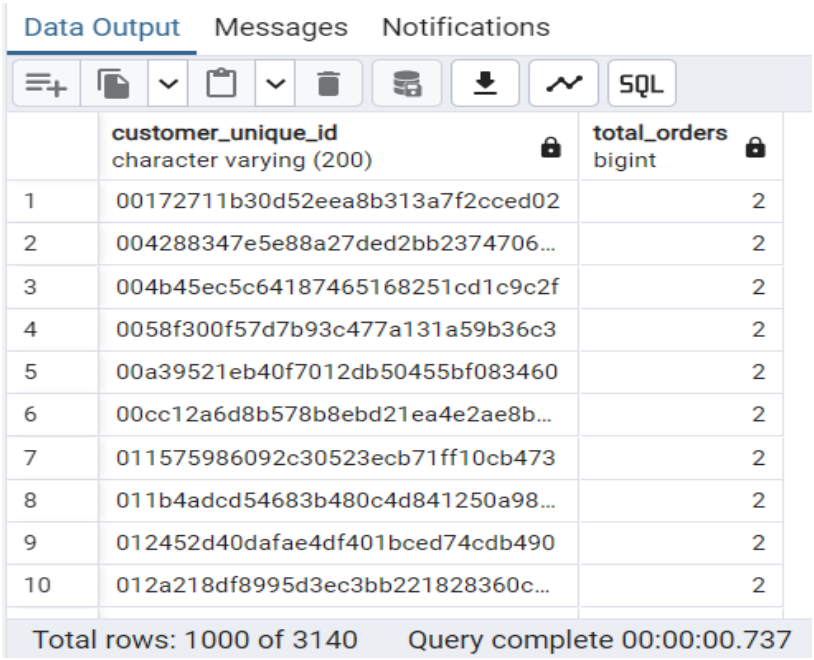
Approach:

1. **Identifying Relevant Tables and Columns:**
   - Table: *Customers* and *Orders*
   - Columns: customer_unique_id, customer_id, order_id.
2. **Joining Tables:**
   - Combined the customers and orders tables using customer_id to connect customers with their orders.
3. **Counting Orders:**
   - Used the **COUNT()** function to count the number of orders for each customer.
4. **Grouping by Customer:**
   - Group the results by *customer_unique_id* to get a total for each customer.
5. **Filtering Results:**
   - Used the **HAVING** clause to include only those customers who have placed more than one order.

SQL Query:

```
select c.customer_unique_id,
count(o.order_id) as total_orders
from amazon_brazil.customers c
join amazon_brazil.orders o
on c.customer_id=o.customer_id
group by c.customer_unique_id
having count(o.order_id)>1
```

Output:



Data Output | Messages | Notifications

SQL

	customer_unique_id character varying (200)	total_orders bigint
1	00172711b30d52eea8b313a7f2cced02	2
2	004288347e5e88a27ded2bb2374706...	2
3	004b45ec5c64187465168251cd1c9c2f	2
4	0058f300f57d7b93c477a131a59b36c3	2
5	00a39521eb40f7012db50455bf083460	2
6	00cc12a6d8b578b8ebd21ea4e2ae8b...	2
7	011575986092c30523ecb71ff10cb473	2
8	011b4adcd54683b480c4d841250a98...	2
9	012452d40dafae4df401bced74cdb490	2
10	012a218df8995d3ec3bb221828360c...	2

Total rows: 1000 of 3140 Query complete 00:00:00.737

Recommendations:

1. **Identify Repeat Customers:**
    - Reach out to customers who place multiple orders. Consider sending them special offers or loyalty rewards to encourage even more purchases.
2. **Analyze Customer Behavior:**
    - Look into what these repeat customers are buying. Understanding their preferences can help tailor marketing strategies and product offerings.
3. **Improve Customer Experience:**
    - Ensure that the ordering process is smooth and efficient, as satisfied customers are more likely to return.
4. **Target New Customers:**
    - Use insights from repeat customers to attract new ones by promoting similar products or services that have proven popular among existing customers.

Question 4:

Problem Statement:

Amazon India wants to categorize customers into different types ('New – order qty. = 1' ; 'Returning' –order qty. 2 to 4; 'Loyal' – order qty. >4) based on their purchase history. Use a temporary table to define these categories and join it with the customers table to update and display the customer types.

- *Output: customer_id, customer_type*

Approach:

1. **Identifying Relevant Tables and Columns:**
   - Table: *Customers* and *Orders*
   - Columns: *customer_type, customer_id, order_id.*
2. **Creating Temporary Table**:
   - Created a temporary table called *customer_categories* to categorize customers based on the number of orders they have placed.
3. **Using Case Statement**:
   - Used a **CASE** statement to define customer types:

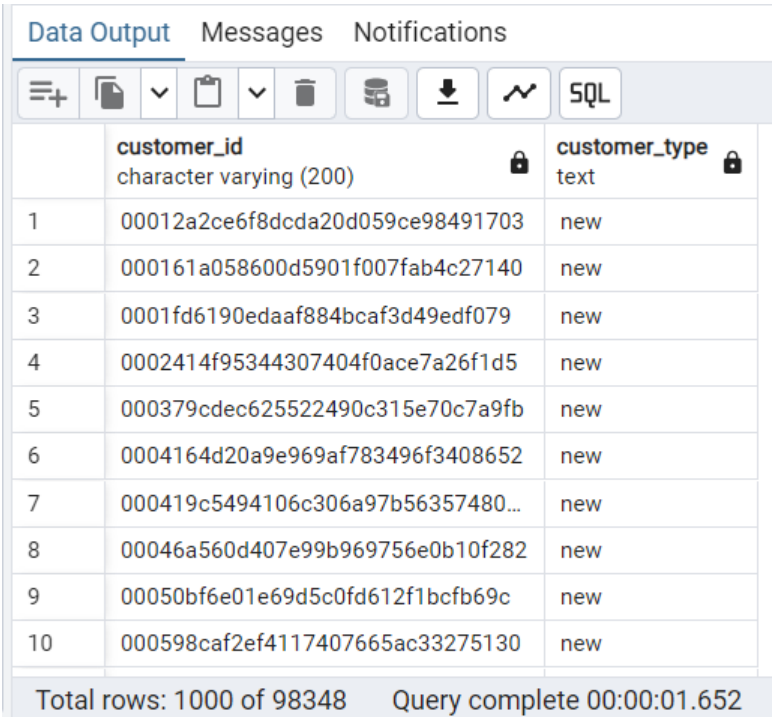
1. 'new' for customers with 1 order.
2. 'returning' for customers with 2 to 4 orders.
3. 'loyal' for customers with more than 4 orders.

4. **Grouping by Customer ID**:
   - Grouped the results by *customer_id* to count the orders for each customer.
5. **Joining with Customers Table**:
   - Joined the *customer_categories* temporary table with the customers table to retrieve customer IDs and their corresponding types.
6. **Grouping and Sorting Results**:
   - Grouped the final results by *customer ID* and customer type, then sorted by *customer ID*.

SQL Query:

```
create temp table customer_categories as
select o.customer_id,
case
when count(order_id) = 1 then 'new'
when count(order_id) between 2 and 4 then ‘returning'
when count(order_id) >4 then 'loyal'
end as customer_type
from amazon_brazil.orders o
group by o.customer_id;
select c.customer_id as customer_id,
cc.customer_type
from amazon_brazil.customers c
join customer_categories cc
on cc.customer_id=c.customer_id
group by c.customer_id, cc.customer_type
order by c.customer_id;
```

Output:



Data Output | Messages | Notifications

SQL

	customer_id character varying (200)	customer_type text
1	00012a2ce6f8dcda20d059ce98491703	new
2	000161a058600d5901f007fab4c27140	new
3	0001fd6190edaaf884bcaf3d49edf079	new
4	0002414f95344307404f0ace7a26f1d5	new
5	000379cdec625522490c315e70c7a9fb	new
6	0004164d20a9e969af783496f3408652	new
7	000419c5494106c306a97b56357480...	new
8	00046a560d407e99b969756e0b10f282	new
9	00050bf6e01e69d5c0fd612f1bcfb69c	new
10	000598caf2ef4117407665ac33275130	new

Total rows: 1000 of 98348 Query complete 00:00:01.652

Recommendations:

1. **Focus on New Customers:**
    - Offer discounts or incentives to encourage new customers to make more purchases.
2. **Engage Returning Customers:**
    - Send personalized offers to returning customers to make them shop more and become loyal.
3. **Reward Loyal Customers:**
    - Create loyalty programs or special offers to keep loyal customers happy and coming back.

Question 5:

Problem Statement:

Amazon India wants to know which product categories generate the most revenue. Use joins between the tables to calculate the total revenue for each product category. Display the top 5 categories.

- *Output: product_category_name, total_revenue*

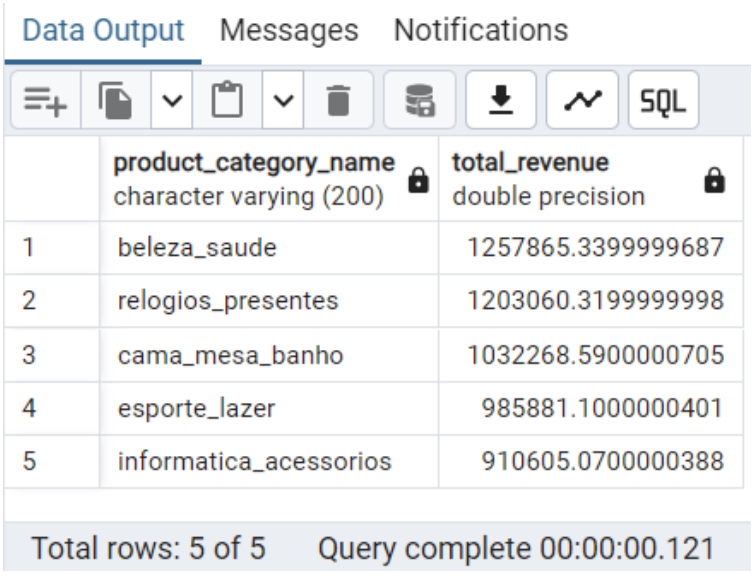
Approach:

1. **Identifying Relevant Tables and Columns:**
   - Table: *Product* and *Order_items*
   - Columns: *product_category_name, price, product_id.*
2. **Joining Tables**:
   - Combined the *product* and *order_items* tables using product_id to link products with their sales data.
3. **Calculating Revenue**:
   - Used the **SUM()** function to calculate the total revenue for each product category.
4. **Grouping by Category**:
   - Grouped the results by *product_category_name* to aggregate revenue data for each category.
5. **Sorting and Limiting Results**:
   - Ordered the results in descending order based on *total revenue* and limited the output to the top five categories.

SQL Query:

```
select p.product_category_name,
sum(o.price) as total_revenue
from amazon_brazil.product p
join amazon_brazil.order_items o
on p.product_id=o.product_id
group by p.product_category_name
order by total_revenue desc
limit 5;
```

Output:



Data Output | Messages | Notifications

	product_category_name character varying (200)	total_revenue double precision
1	beleza_saude	1257865.3399999687
2	relogios_presentes	1203060.3199999998
3	cama_mesa_banho	1032268.5900000705
4	esporte_lazer	985881.1000000401
5	informatica_acessorios	910605.0700000388

Total rows: 5 of 5 Query complete 00:00:00.121

Recommendations:

1. **Expand High-Revenue Categories:**
    - Since "beleza_saúde" makes the most money, consider adding more products in this area to increase sales even further.
2. **Promote Popular Products:**
    - For "relogios_presentes," run marketing campaigns to highlight popular items and make them more visible to customers.
3. **Look for Growth Opportunities:**
    - Check trends in "cama_mesa_banho" and "Esporte_lazer" and “informatica_accessorios” to find new products or promotions that could help boost sales.

ANALYSIS III

Question 1:

Problem Statement:

The marketing team wants to compare the total sales between different seasons. Use a subquery to calculate total sales for each season (Spring, Summer, Autumn, Winter) based on order purchase dates, and display the results. Spring is in the months of March, April and May. Summer is from June to August and Autumn is between September and November and rest months are Winter.

- *Output: season, total_sales*

Approach:

1. **Identifying Relevant Tables and Columns:**
   - Table: *Orders* and *Order_items*
   - Columns: *price, order_id, order_purchased_timestamp.*
2. **Joining Tables:**
   - Combined the *order_items* and *orders* tables using order_id to connect sales data with order dates.
3. **Retrieving Season:**
   - Used a *subquery* and *case when* statement to categorize each order into a season based on the month of the purchase

     Spring: March, April, May

     Summer: June, July, August

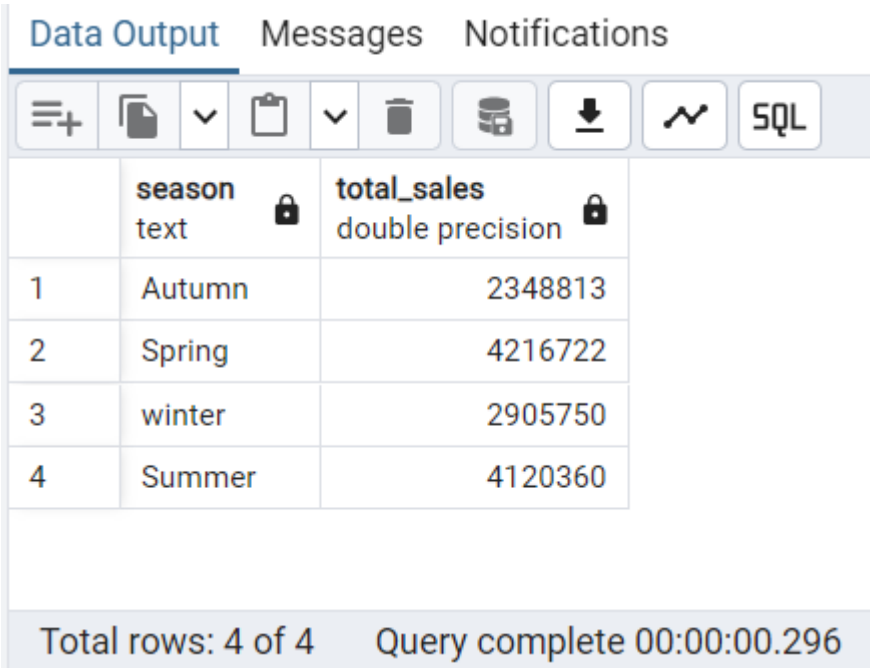
     Autumn: September, October, November

     Winter: December, January, February
4. **Calculating Total Sales:**
   - Used the **SUM()** function to calculate total sales for each season.
5. **Grouping by Season:**
   - Grouped results by *season* to aggregate sales data.

SQL Query:

```
select season,
round(sum(oi.price )) as total_sales
from amazon_brazil.order_items oi
join(
select o.order_id,
case
when extract(month from o.order_purchased_timestamp) in(03, 04, 05)then 'Spring'
when extract(month from o.order_purchased_timestamp) in(06, 07, 08)then 'Summer'
when extract(month from o.order_purchased_timestamp) in(09, 10, 11)then 'Autumn'
else 'winter'
end as season
from amazon_brazil.orders o
)sales
on sales.order_id=oi.order_id
group by season;
```

Output:



	season text	total_sales double precision
1	Autumn	2348813
2	Spring	4216722
3	winter	2905750
4	Summer	4120360

Total rows: 4 of 4 Query complete 00:00:00.296

Recommendations:

1. **Focus on High-Sales Seasons:**
    - Since Spring and Summer have the highest sales, can run promotions or special offers during these seasons to boost revenue even more.
2. **Improve Winter, Autumn Sales:**
    - Since Winter and Autumn are having lower sales, try offering discounts or holiday deals to encourage more purchases.
3. **Plan for Seasonal Trends:**
    - Use this data to stock up and promote products during the seasons when sales are higher, ensuring to be ready for increased demand.

Question 2:

Problem Statement:

The inventory team is interested in identifying products that have sales volumes above the overall average. Write a query that uses a subquery to filter products with a total quantity sold above the average quantity.

- *Output: product_id, total_quantity_sold*

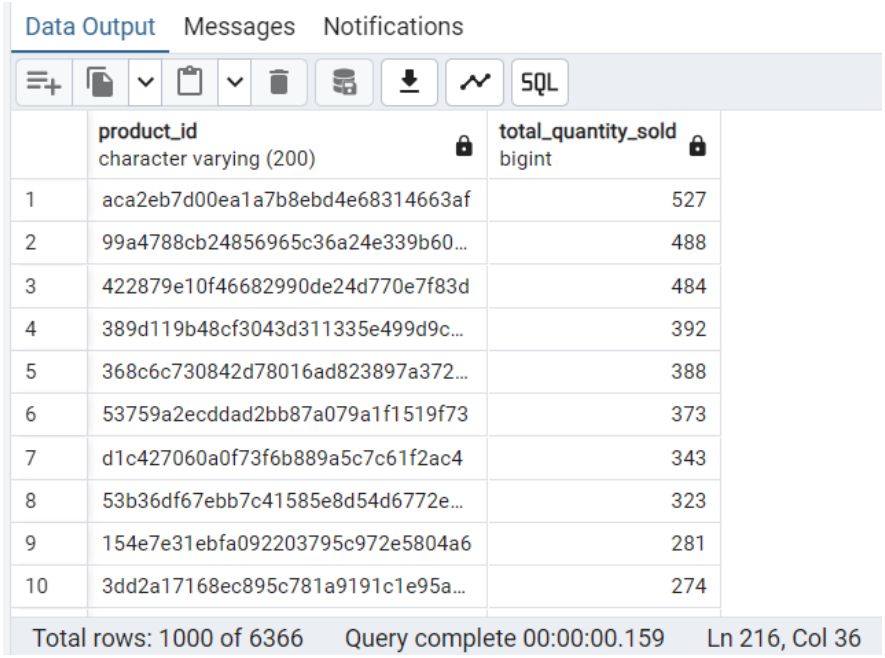
Approach:

1. **Identifying Relevant Tables and Columns:**
   - Table: *Order_items*
   - Columns: *product_id, order_id.*
2. **Counting Total Quantity Sold:**
   - Counted the number of orders for each product using **COUNT().**
3. **Calculating Average Quantity:**
   - Used a subquery to calculate the average quantity sold across all products.
4. **Filtering Products:**
   - Used the **HAVING** clause to filter out products that sold more than the average quantity.
5. **Sorting Results:**
   - Ordered the results by *total quantity sold* in descending order.

SQL Query:

```
select  product_id,
count(order_id) as total_quantity_sold
from amazon_brazil.order_items
group by product_id
having count(order_id) > ( select avg(total_quantity)
from (
select count(order_id) AS total_quantity
from amazon_brazil.order_items
group by product_id
) as avg_sales
)order by total_quantity_sold desc;
```

Output:



Data Output | Messages | Notifications

	product_id character varying (200)	total_quantity_sold bigint
1	aca2eb7d00ea1a7b8ebd4e68314663af	527
2	99a4788cb24856965c36a24e339b60...	488
3	422879e10f46682990de24d770e7f83d	484
4	389d119b48cf3043d311335e499d9c...	392
5	368c6c730842d78016ad823897a372...	388
6	53759a2ecddad2bb87a079a1f1519f73	373
7	d1c427060a0f73f6b889a5c7c61f2ac4	343
8	53b36df67ebb7c41585e8d54d6772e...	323
9	154e7e31ebfa092203795c972e5804a6	281
10	3dd2a17168ec895c781a9191c1e95a...	274

Total rows: 1000 of 6366 Query complete 00:00:00.159 Ln 216, Col 36

Recommendations:

1. **Promote Best-Sellers**:
    - Focus on marketing the products with the highest sales since they're already popular with customers.
2. **Keep Stock Ready**:
    - Ensure these top-selling products are always available to avoid losing sales due to stock shortages.
3. **Improve Low-Sellers**:
    - Check products that aren't selling as well and try adjusting prices or running promotions to boost their sales.

Question 3:

Problem Statement:

To understand seasonal sales patterns, the finance team is analysing the monthly revenue trends over the past year (year 2018). Run a query to calculate total revenue generated each month and identify periods of peak and low sales. Export the data to Excel and create a graph to visually represent revenue changes across the months.

- *Output: month, total_revenue*

Approach:

1. **Identifying Relevant Tables and Columns:**
   - Table: *Orders* and *Order_items*
   - Columns: *order_id, order_purchased_timestamp.*
2. **Joining Tables:**
   - Combine the *orders* and *order_items* tables using *order_id* to connect sales data with order dates.
4. **Extracting Month and Revenue**:
   - Used the **EXTRACT()** function to get the month from the order date.
   - Calculated *total revenue* for each month using the **SUM()** function.
5. **Filtering by Year:**
   - Used a *WHERE* clause to focus only on orders from the year *2018*.
6. **Grouping and Sorting Results:**
   - Grouped results by *month* to aggregate revenue data.
   - Ordered the results by *revenue* in descending order to see which months generated the most revenue.

SQL Query:

```
select extract(month from o.order_purchased_timestamp) as month,
round(sum(oi.price)) as revenue
from amazon_brazil.orders o
join amazon_brazil.order_items oi
on o.order_id = oi.order_id
where extract(year from o.order_purchased_timestamp) = 2018
group by month
order by revenue desc;
```

Output:

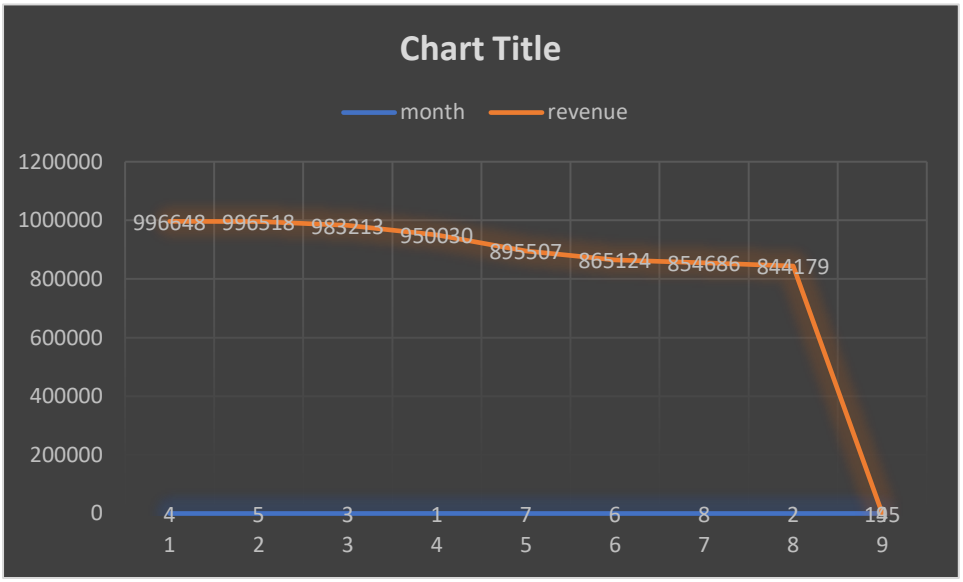
Data Output | Messages | Notifications

SQL

	month numeric	revenue double precision
1	4	996648
2	5	996518
3	3	983213
4	1	950030
5	7	895507
6	6	865124
7	8	854686
8	2	844179
9	9	145

Total rows: 9 of 9 Query complete 00:00:00.128

Graphical Representation of Revenue Changes:



Recommendations:

1. **Focus on High Revenue Months**:
   - April and May have the highest revenues. Plan marketing campaigns or special promotions during these months to maximize sales.
2. **Analyse Low Revenue Months**:
   - The revenue in September is significantly low. Investigate the reasons for this and consider running targeted promotions or discounts to boost sales during this month.
3. **Seasonal Promotions**:
   - Use the data to create seasonal sales strategies, especially in the months with higher sales, to encourage repeat purchases and attract new customers.

Question 4:

Problem Statement:

A loyalty program is being designed for Amazon India. Create a segmentation based on purchase frequency: 'Occasional' for customers with 1-2 orders, 'Regular' for 3-5 orders, and 'Loyal' for more than 5 orders. Use a CTE to classify customers and their count and generate a chart in Excel to show the proportion of each segment.

- *Output: customer_type, count*

Approach:

1. **Identifying Relevant Tables and Columns:**
   - Table: *Orders*
   - Columns: *customer_id order_id, customer_type.*
2. **Creating CTE:**
   - Used a Common Table Expression (CTE) called *order_total* to calculate the total number of orders for each customer by counting *order_id*.
3. **Classifying Customers:**
   - Used a **CASE** statement to categorize customers based on their total orders.
     "Occasional" for customers with 1 to 2 orders.
     "Regular" for customers with 3 to 5 orders.
     "Loyal" for customers with more than 5 orders.
4. **Counting Customers in Each Category:**
   - Counted the number of *distinct customers* in each category.
5. **Grouping and Sorting Results:**
   - Grouped the results by *customer type* and sorted them by the count of customer_id.

SQL Query:

```
with order_total as  (

select distinct(customer_id), count(order_id) as total_orders from
amazon_brazil.orders

group by customer_id  )

select

case  when total_orders between 1 and 2 then 'Occassional'

when total_orders between 3 and 5 then 'Regular'

else 'Loyal'
```

```
end as customer_type,
count(distinct(customer_id)) as count
from order_total
group by customer_type
order by count;
```

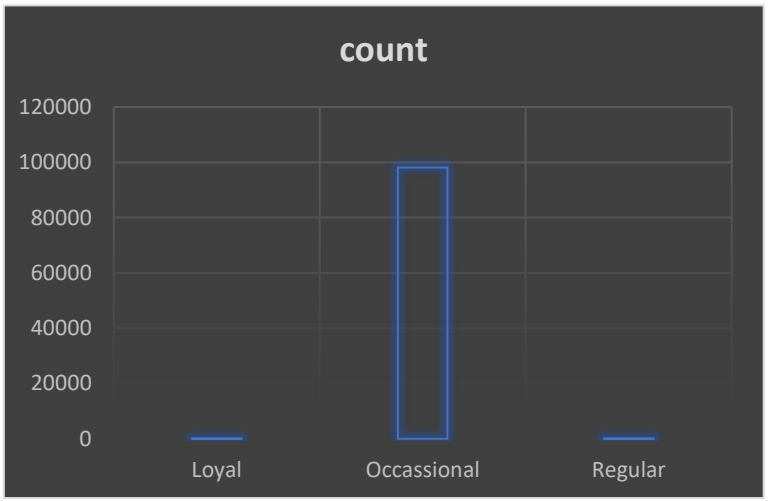
Output:

Data Output | Messages | Notifications

	customer_type text	count bigint
1	Loyal	98
2	Regular	106
3	Occassional	98144

Total rows: 3 of 3 Query complete 00:00:02.333

Chart:



Recommendations:

1. **Engage Occasional Customers:**
   - With a huge number of occasional customers, focus on sending them special offers or promotions to encourage them to make more purchases and become regular customers.
2. **Reward Regular Customers:**
   - Consider implementing a loyalty program or rewards for regular customers to turn them into loyal customers, increasing their likelihood to return.
3. **Maintain Loyal Customers:**
   - Keep your loyal customers happy by offering exclusive deals or personalized services to ensure they continue shopping with you and feel valued.

Question 5:

Problem Statement:

Amazon wants to identify high-value customers to target for an exclusive rewards program. You are required to rank customers based on their average order value (avg_order_value) to find the top 20 customers.

- *Output: customer_id, avg_order_value, and customer_rank*

Approach:

1. **Identifying Relevant Tables and Columns:**
   - Table: *Orders* and *Order_items*
   - Columns: *customer_id, price, order_id*
2. **Joining Tables:**
   - Combined the *orders* and *order_items* tables using *order_id* to connect each order with its items.
3. **Calculating Average Order Value:**
   - Used the **AVG()** function to find the average price of items ordered by each customer.
4. **Ranking Customers:**
   - Used the **RANK()** window function to assign a rank to each customer based on their average order value, with higher values receiving a higher rank.
5. **Grouping by Customer:**
   - Grouped the results by *customer_id* to aggregate data for each customer.
6. **Sorting and Limiting Result**s:
   - Ordered the results by *average order value* in *descending orde*r and limit to the top *20 customers*.

SQL Query:

```
select o.customer_id,

avg(oi.price) as avg_order_value,

rank() over(order by avg(oi.price) desc ) as customer_rank

from amazon_brazil.orders o

join amazon_brazil.order_items oi
```

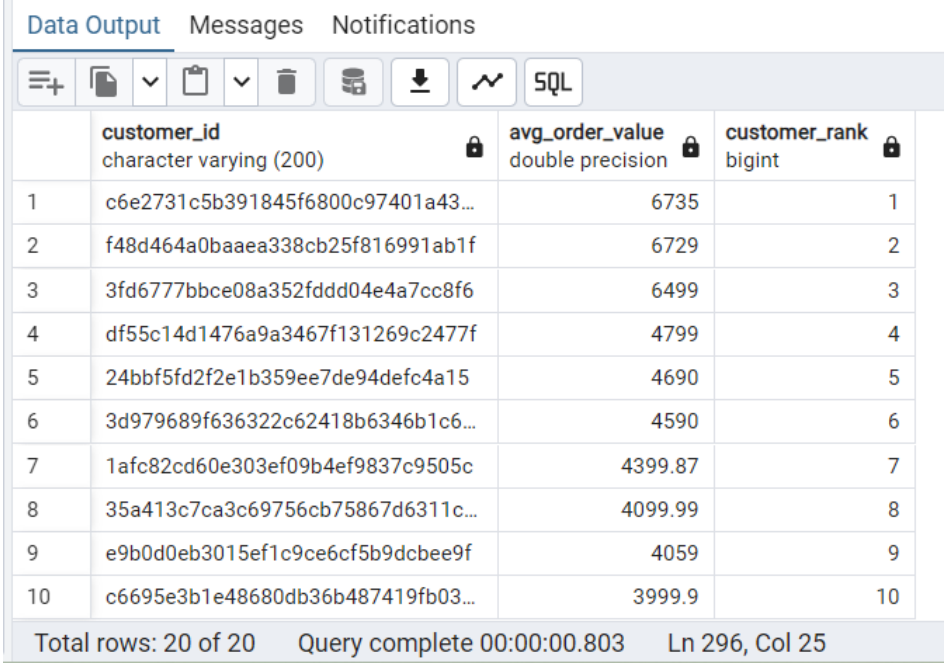
on o.order_id=oi.order_id

group by o.customer_id

order by avg_order_value desc

limit 20;

Output:



Data Output | Messages | Notifications

	customer_id character varying (200)	avg_order_value double precision	customer_rank bigint
1	c6e2731c5b391845f6800c97401a43...	6735	1
2	f48d464a0baaea338cb25f816991ab1f	6729	2
3	3fd6777bbce08a352fddd04e4a7cc8f6	6499	3
4	df55c14d1476a9a3467f131269c2477f	4799	4
5	24bbf5fd2f2e1b359ee7de94defc4a15	4690	5
6	3d979689f636322c62418b6346b1c6...	4590	6
7	1afc82cd60e303ef09b4ef9837c9505c	4399.87	7
8	35a413c7ca3c69756cb75867d6311c...	4099.99	8
9	e9b0d0eb3015ef1c9ce6cf5b9dcbee9f	4059	9
10	c6695e3b1e48680db36b487419fb03...	3999.9	10

Total rows: 20 of 20 Query complete 00:00:00.803 Ln 296, Col 25

Recommendations:

1. **Target High-Spending Customers:**

   - Since the top customers spend a lot on average, consider offering them exclusive deals or personalized offers to encourage them to keep shopping.

2. **Analyse Spending Patterns:**

   - Look into what products these high-spending customers are buying. Understanding their preferences can help you tailor your marketing strategies.

3. **Encourage More Spending from Others**:

   - For customers who are not in the top ranks, think about promotions or bundles that could encourage them to increase their average order value.

Question 6:

Problem Statement:

Amazon wants to analyse sales growth trends for its key products over their lifecycle. Calculate monthly cumulative sales for each product from the date of its first sale. Use a recursive CTE to compute the cumulative sales (total_sales) for each product month by month.

- *Output: product_id, sale_month, and total_sales*

Approach:

1. **Identifying Relevant Tables and Columns:**

   Table: payments, orders

   Columns: payment_type, order_purchased_timestamp, price, order_id.

2. **Creating a CTE:**

   - Used a Common Table Expression (CTE) called sales to calculate monthly sales for each product. This includes:
     - Extracting the month from the order date.
     - Summing the sales prices for each product per month.

3. **Calculating Cumulative Sales:**

   - In the main query, used the **SUM()** function with the **OVER()** clause to calculate cumulative sales for each product, partitioned by product_id and ordered by *sale_month*.

4. **Sorting Results:**

   - Ordered the final results by *product_id* and *sale_month* to see the sales progression for each product over time

SQL Query:

```
with sales as(
select product_id,
extract(month from o.order_purchased_timestamp) as sale_month,
sum(oi.price) as monthly_sales
from amazon_brazil.orders o
join amazon_brazil.order_items oi
```

```
on o.order_id = oi.order_id
group by product_id, sale_month
)
select
product_id,
sale_month,
round(sum(monthly_sales)over(partition by product_id order by sale_month)) as total_sales
from sales
order by product_id,sale_month;
```

Output:

Data Output | Messages | Notifications

	product_id character varying (200)	sale_month numeric	total_sales double precision
1	00066f42aeeb9f3007548bb9d3f33c38	5	102
2	00088930e925c41fd95ebfe695fd2655	12	130
3	0009406fd7479715e4bef61dd91f2462	12	229
4	000b8f95fcb9e0096488278317764d19	8	118
5	000d9be29b5207b54e86aa1b1ac54872	4	199
6	0011c512eb256aa0dbbb544d8dffcf6e	12	52
7	00126f27c813603687e6ce486d909d01	9	498
8	001795ec6f1b187d37335e1c4704762e	10	39
9	001795ec6f1b187d37335e1c4704762e	11	117
10	001795ec6f1b187d37335e1c4704762e	12	350

Total rows: 1000 of 60796 Query complete 00:00:00.792 Ln 306, Col 19

Recommendations:

1. **Watch Popular Products:**
    - Keep track of which products are selling well each month. This helps you know what to stock up on.
2. **Run Sales During Busy Months:**
    - If certain products sell better at specific times of the year, plan promotions during those months to boost sales even more.
3. **Check Low-Selling Products:**
    - Look into products that aren't selling much. Find out why, like if they're priced too high or have bad reviews and make changes to improve sales.

Question 7:

Problem Statement:

To understand how different payment methods affect monthly sales growth, Amazon wants to compute the total sales for each payment method and calculate the month-over-month growth rate for the past year (year 2018). Write query to first calculate total monthly sales for each payment method, then compute the percentage change from the previous month.

- *Output: payment_type, sale_month, monthly_total, monthly_change.*

Approach:

1. **Identifying Relevant Tables and Columns:**
   - **Table:** *payments, orders, order_items*
   - **Columns:** *payment_type, order_purchased_timestamp, price, order_id.*
2. **Creating a CTE**:
   - Used a Common Table Expression (CTE) called total to calculate the monthly total sales for each payment type. This includes:
     i. Extracting the month from the order date.
     ii. Summing the prices of order items associated with each payment type.
3. **Calculating Monthly Totals**:
   - Grouped the results by *payment_type* and *sale_month* to get total sales for each payment method each month.
4. **Calculating Percentage Change**:
   - In the main query, used the **LAG()** function to find the previous month's total for each payment type.
   - Calculated the percentage change in sales from the previous month using a formula that compares the current month's total to the last month's total.
5. **Sorting Results**:
   - Ordered the final results by *payment_type* and *sale_month* to see trends over time.

SQL Query:

```
with total as(
select p.payment_type,
extract(month from o.order_purchased_timestamp) as sale_month,
round(sum(oi.price)) as monthly_total
from amazon_brazil.payments p
join amazon_brazil.orders o
on p.order_id=o.order_id
join amazon_brazil.order_items oi
on o.order_id=oi.order_id
where
extract(year from o.order_purchased_timestamp)= 2018
group by p.payment_type,sale_month
)
select
payment_type, sale_month,monthly_total,
round((monthly_total-lag(monthly_total)over(partition by payment_type order by sale_month))/
lag(monthly_total)over(partition by payment_type order by sale_month)*100.0)
end as monthly_change
from total
order by payment_type, sale_month;
```

Output:

Data Output | Messages | Notifications

	payment_type character varying (200)	sale_month numeric	monthly_total double precision	round double precision
1	boleto	1	170651	[null]
2	boleto	2	153166	-10
3	boleto	3	157807	3
4	boleto	4	162941	3
5	boleto	5	166572	2
6	boleto	6	126380	-24
7	boleto	7	162938	29
8	boleto	8	118214	-27
9	credit_card	1	760253	[null]
10	credit_card	2	680199	-11

Total rows: 33 of 33 Query complete 00:00:00.181 Ln 348, Col 1

Recommendations:

1. **Monitor Payment Trends:**
    - Monitor which payment methods are gaining or losing popularity. For example, if "boleto" sales drop in a certain month, investigate why.
2. **Promote Popular Payment Methods:**
    - If a specific payment type, like "credit card," shows consistent growth, consider promoting it through marketing campaigns to encourage more customers to use it.
3. **Address Declines Quickly:**
    - If you notice significant drops in sales for any payment method, analyze the reasons, whether it’s due to customer preferences, technical issues, or competition and then take related actions.