Leveraging Amazon Brazil's Data to Drive Market Insights for Amazon India

Company Overview:

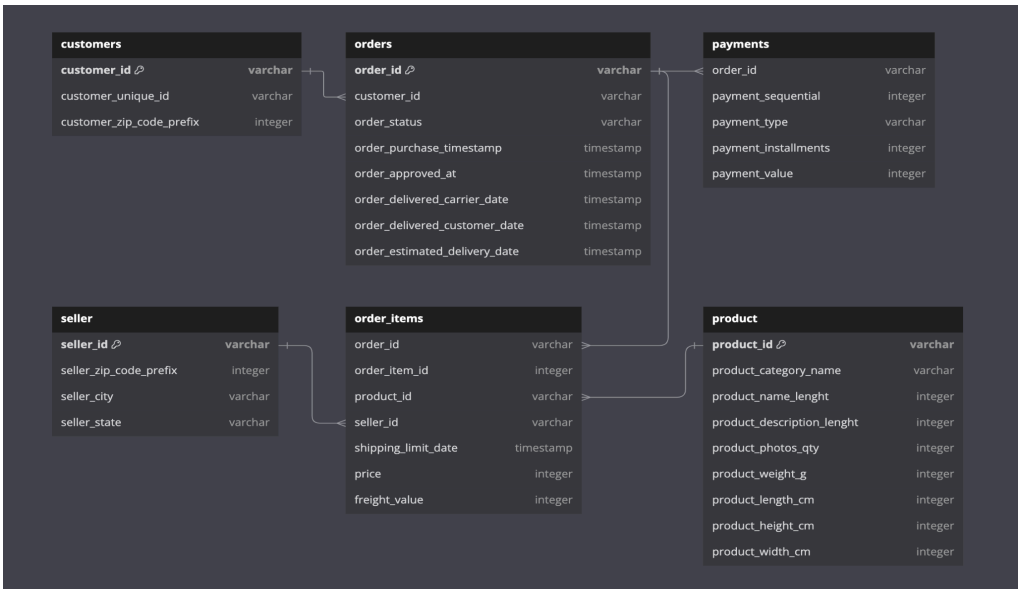
Amazon, a global leader in e-commerce, has achieved significant success in markets like the U.S., Europe, and Asia. In Brazil, Amazon connects small and medium businesses with millions of customers, becoming a key player. Given the similarities between Brazil and India—such as large populations and diverse consumer bases—there's an opportunity to replicate success in India.

Business Problem Statement:

Analyze Amazon Brazil's data to identify trends and customer behaviors that could be leveraged in the Indian market. Focus on customer demographics and behavior using the Customers table to understand purchase patterns and preferences. Evaluate regional trends and customer density through the Geolocation table. Track order lifecycles, product preferences, and seller performance using the Orders, Order Items, Product, and Seller tables. Additionally, analyze payment preferences and transaction details via the Payments table. This comprehensive analysis will help Amazon India enhance customer experience and seize new market opportunities.

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

1. **Problem Statement 1:**

   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:

- Used AVG() function to compute for each 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.

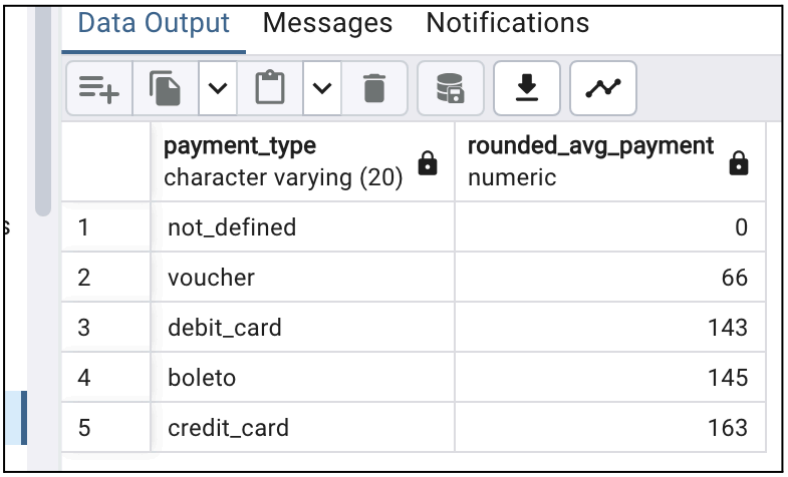
4. Sorting the Results:

- 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 (20)	rounded_avg_payment numeric
1	not_defined	0
2	voucher	66
3	debit_card	143
4	boleto	145
5	credit_card	163

Recommendations:

Payment Improvement Strategies

1. **Enhance High-Value Payment Methods** – Improve user experience for **credit_card** and **boleto** to maximize transaction value.
2. **Leverage Targeted Promotions** – Offer exclusive deals for high-value payment users to boost loyalty and sales.
3. **Encourage Diverse Payment Adoption** – Provide incentives for **debit_card** and **voucher** to increase usage.
4. **Resolve Undefined Payment Issues** – Investigate the **not_defined** category for potential improvements.

Problem Statement 2:

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:

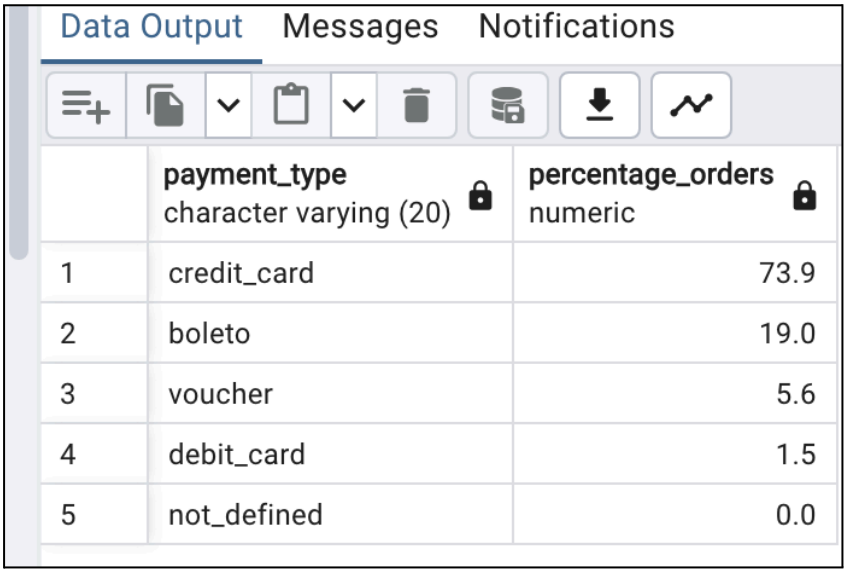
• Grouped the results by **payment_type**.

• 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

	payment_type character varying (20)	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

Recommendations:

Payment Optimization Recommendations

1. **Enhance Popular Payment Methods** – Optimize the user experience for **high-usage** methods like **credit_card.**
2. **Boost Low-Usage Payment Adoption** – Identify barriers for **debit_card and voucher,** offering incentives or simplifying the process to encourage use.
3. **Resolve Undefined Payment Issues** – Investigate the **not_defined category** for potential improvements.

Problem Statement 3:

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.

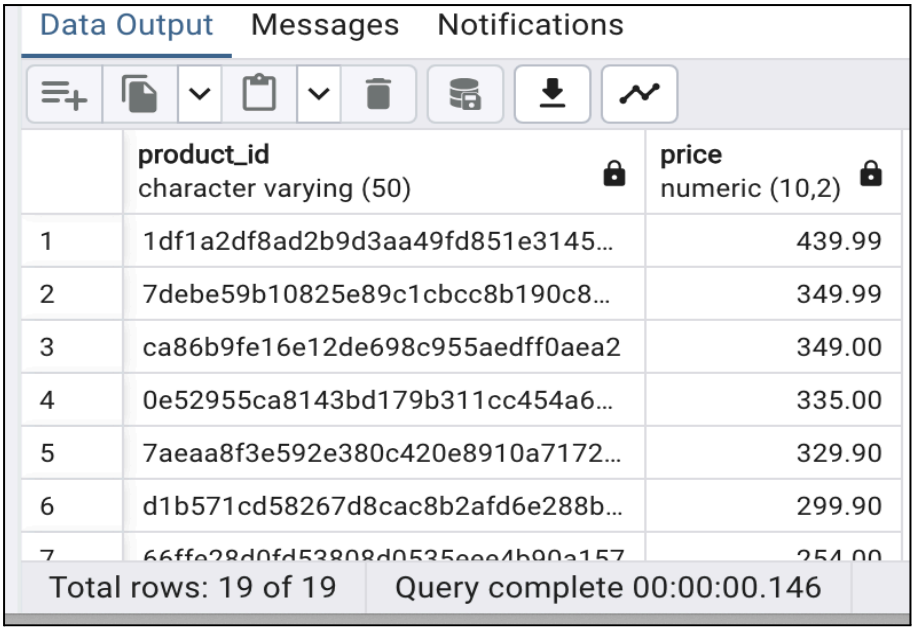
5. Grouping and Sorting:

- 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.products 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:



Recommendations:

Smart Product Promotion Strategies

1. **Highlight Smart Products** – Promote items in the **"smart"** category to attract more customers.
2. **Boost Mid-Range Smart Sales** – Focus on products priced between **100 and 500** to drive higher conversions.
3. **Expand Smart Product Selection** – Introduce more offerings within this price range to enhance variety and appeal.

Problem Statement 4:

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:

- Combine 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.

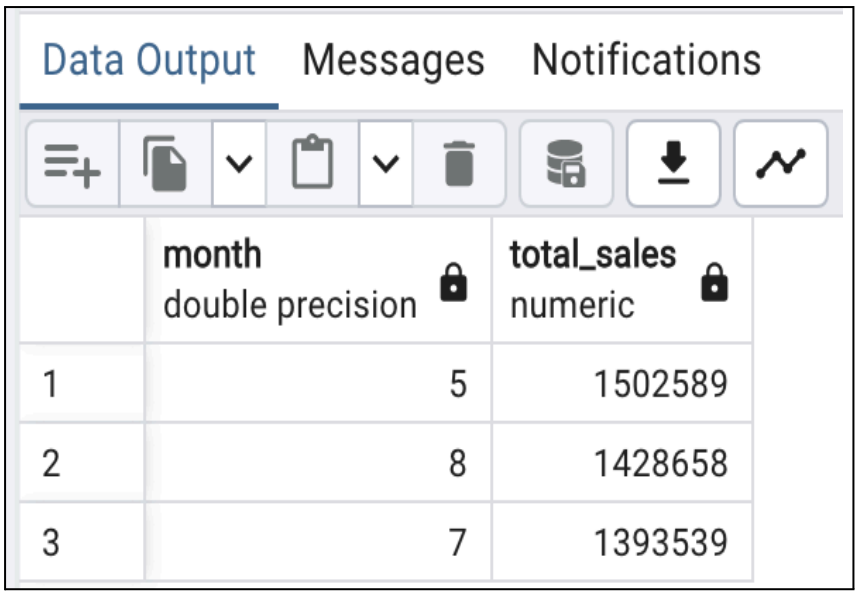
6. Limiting the Results:

- Used LIMIT to display only the top 3 months with the highest sales.

SQL Query:

```
SELECT EXTRACT(MONTH FROM o.order_purchase_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 double precision	total_sales numeric
1	5	1502589
2	8	1428658
3	7	1393539

Recommendations:

Sales Optimization Strategies

1. **Maximize May's Success** – Replicate effective strategies, such as promotions or product launches, to sustain high sales.
2. **Boost Sales in Low-Performing Months** – Analyze variations in July and August and implement targeted promotions or new launches to drive sales.
3. **Identify Key Success Factors** – Determine what contributed to May's peak sales (e.g., promotions, product launches, seasonal trends) and apply similar strategies to other months.

Problem Statement 5:

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.

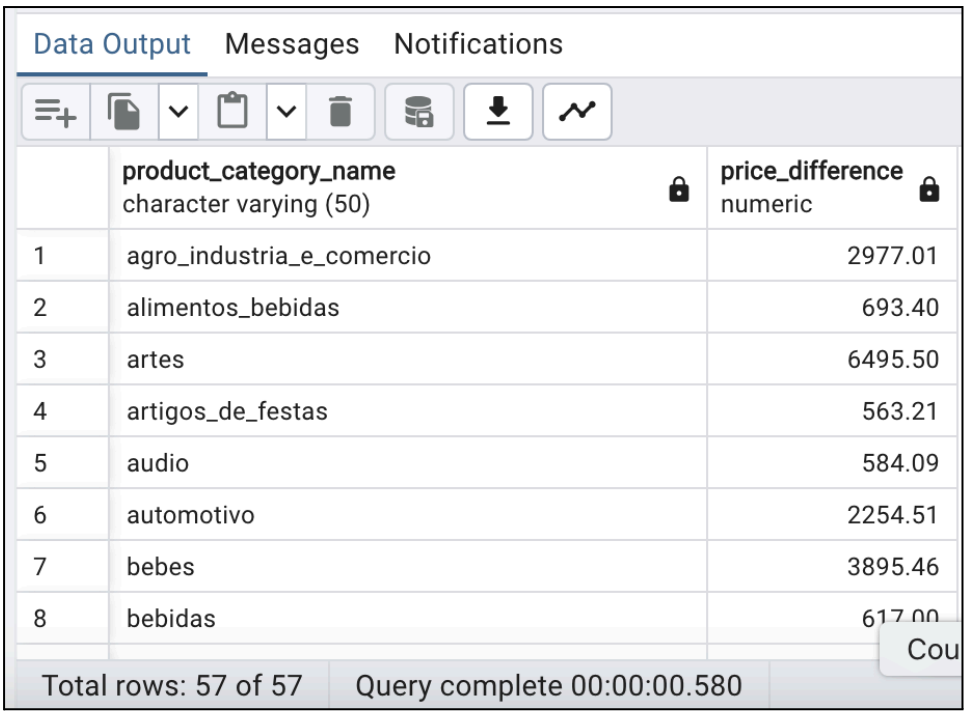
5. Filtering Results:

- Used the HAVING clause to include only those categories where the price The difference is greater than 500.

SQL Query:

```
SELECT product_category_name,
        Max(oi.price)-Min(oi.price) As price_difference
FROM products p
JOIN order_items oi
ON p.product_id = oi.product_id
GROUP BY product_category_name
HAVING Max(oi.price)-MIN(oi.price)>500
```

Output:



Data Output | Messages | Notifications

	product_category_name character varying (50)	price_difference numeric
1	agro_industria_e_comercio	2977.01
2	alimentos_bebidas	693.40
3	artes	6495.50
4	artigos_de_festas	563.21
5	audio	584.09
6	automotivo	2254.51
7	bebes	3895.46
8	bebidas	617.00

Total rows: 57 of 57 | Query complete 00:00:00.580

Recommendations:

Pricing Strategy Recommendations

1. **Refine Pricing Strategy** – Review pricing for "**artes**" and "**bebes**", introducing mid-range options or ensuring better consistency.
2. **Promote Mid-Range Products** – Highlight mid-range options in categories with high price differences to attract hesitant buyers.
3. **Track Competitor Pricing** – Regularly monitor market prices to stay competitive and adjust pricing strategies accordingly.
4. **Educate Customers** – Explain price variations by showcasing product features, quality, and brand value to justify premium pricing.

Problem Statement 6:

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 (20)	std_deviation numeric
1	not_defined	0
2	voucher	115.5141840586437894
3	boleto	213.578362478749
4	credit_card	222.118587643615
5	debit_card	245.753196018534

Recommendations:

Payment Strategy Enhancements

1. **Standardize Payment Categories** – Clearly define all payment types to eliminate confusion, especially for **"not_defined"** transactions.
2. **Promote Voucher Usage** – Leverage vouchers as a stable and reliable payment option to encourage adoption.
3. **Analyze High-Variance Payments** – Investigate variability in **debit and credit card** transactions to understand customer behavior and refine pricing strategies.
4. **Ensure Pricing Consistency** – Maintain uniform pricing across payment methods to enhance customer trust and satisfaction.

Problem Statement 7:

Amazon India wants to identify products that may have incomplete names 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.

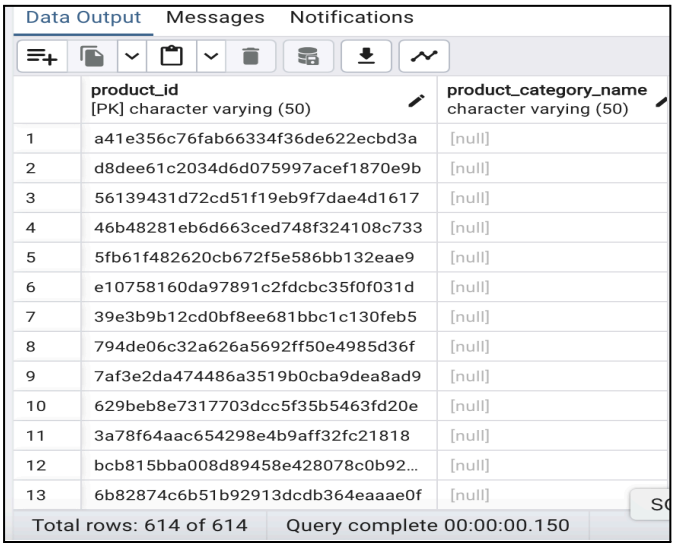
3.Filter for Null or Short Categories:

- 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.products
WHERE product_category_name IS NULL
OR LENGTH (product_category_name)=1;
```

Output:



Data Output | Messages | Notifications

	product_id [PK] character varying (50)	product_category_name character varying (50)
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]
13	6b82874c6b51b92913dcdb364eaaae0f	[null]

Total rows: 614 of 614 Query complete 00:00:00.150

Recommendations:

Data Quality Improvement

1. **Resolve Missing Data** – Investigate and update products with missing or incomplete category names to enhance data accuracy.
2. **Review and Clean Categories** – Ensure all product categories are properly defined to improve analysis and reporting.
3. **Enhance Product Information** – Maintain accurate and complete category names to optimize search, sorting, and customer experience.

ANALYSIS II

Problem Statement 1:

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'.

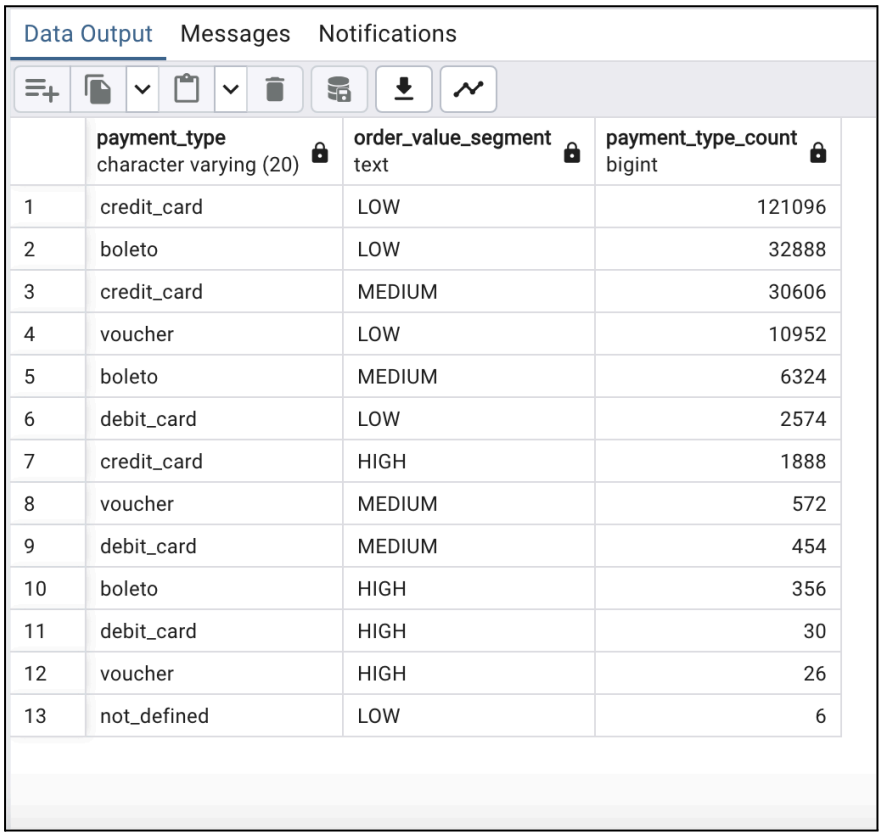
3. Counting and Sorting:

- 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

	payment_type character varying (20)	order_value_segment text	payment_type_count bigint
1	credit_card	LOW	121096
2	boleto	LOW	32888
3	credit_card	MEDIUM	30606
4	voucher	LOW	10952
5	boleto	MEDIUM	6324
6	debit_card	LOW	2574
7	credit_card	HIGH	1888
8	voucher	MEDIUM	572
9	debit_card	MEDIUM	454
10	boleto	HIGH	356
11	debit_card	HIGH	30
12	voucher	HIGH	26
13	not_defined	LOW	6

Recommendations:

Payment Value Optimization Strategies

1. **Boost Low-Value Payments** – Identify reasons for lower spending and implement strategies to encourage higher transactions.
2. **Enhance Medium-Value Spending** – Offer incentives or rewards to convert medium-value customers into high-value spenders.
3. **Drive High-Value Payments** – Analyze key factors behind high-value transactions and apply those insights to increase overall sales.

Problem Statement 2:

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.

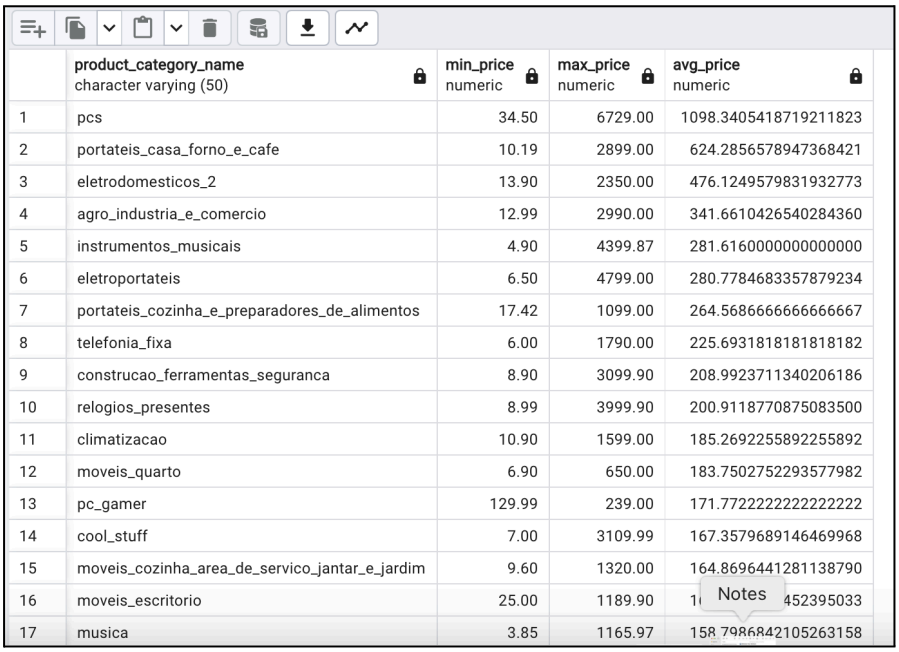
4. Grouping and Sorting:

- 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.products 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:



	product_category_name character varying (50)	min_price numeric	max_price numeric	avg_price numeric
1	pcs	34.50	6729.00	1098.3405418719211823
2	portateis_casa_forno_e_cafe	10.19	2899.00	624.2856578947368421
3	eletrodomesticos_2	13.90	2350.00	476.1249579831932773
4	agro_industria_e_comercio	12.99	2990.00	341.6610426540284360
5	instrumentos_musicais	4.90	4399.87	281.6160000000000000
6	eletroportateis	6.50	4799.00	280.7784683357879234
7	portateis_cozinha_e_preparadores_de_alimentos	17.42	1099.00	264.5686666666666667
8	telefonia_fixa	6.00	1790.00	225.6931818181818182
9	construcao_ferramentas_seguranca	8.90	3099.90	208.9923711340206186
10	relogios_presentes	8.99	3999.90	200.9118770875083500
11	climatizacao	10.90	1599.00	185.2692255892255892
12	moveis_quarto	6.90	650.00	183.7502752293577982
13	pc_gamer	129.99	239.00	171.7722222222222222
14	cool_stuff	7.00	3109.99	167.3579689146469968
15	moveis_cozinha_area_de_servico_jantar_e_jardim	9.60	1320.00	164.8696441281138790
16	moveis_escritorio	25.00	1189.90	1[illegible]452395033
17	musica	3.85	1165.97	158.7986842105263158

Notes

Recommendations:

Pricing Optimization Strategies

1. **Promote High-Priced Categories** – Identify and highlight categories with higher average prices to maximize sales potential.
2. **Review Low-Priced Categories** – Assess lower-priced categories for possible improvements or pricing adjustments.
3. **Analyze Price Variations** – Examine large price gaps within categories to determine if they stem from product versions or features.

Problem Statement 3.

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

4. Grouping by Customer:

- Used the COUNT() function to count the number of orders for each customer.
- Group the results by customer_unique_id to get a total for each customer.

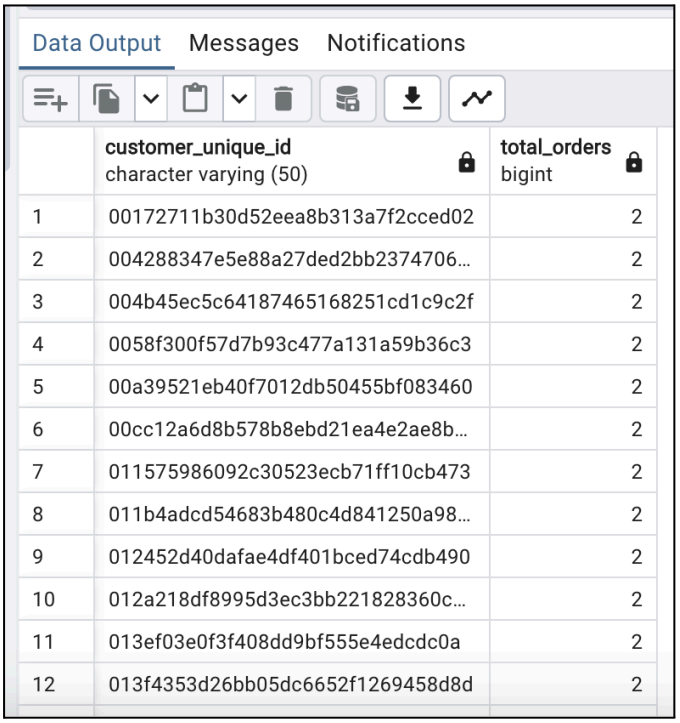
5. Filtering Results:

- 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

	customer_unique_id character varying (50)	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
11	013ef03e0f3f408dd9bf555e4edcdc0a	2
12	013f4353d26bb05dc6652f1269458d8d	2

Recommendations:

Customer Retention and Growth Strategies

1. **Engage Repeat Customers** – Offer special deals or loyalty rewards to encourage continued purchases.
2. **Analyze Buying Patterns** – Study repeat customers' preferences to refine marketing and product strategies.
3. **Enhance Customer Experience** – Optimize the ordering process for a seamless and satisfying experience.
4. **Attract New Customers** – Leverage insights from repeat buyers to target and acquire new customers with similar interests.

Problem Statement 4.

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

Problem Statement 5.

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

4. Grouping by Category:

Used the SUM() function to calculate the total revenue for each product category. Grouped the results by product_category_name to aggregate revenue data for each category.

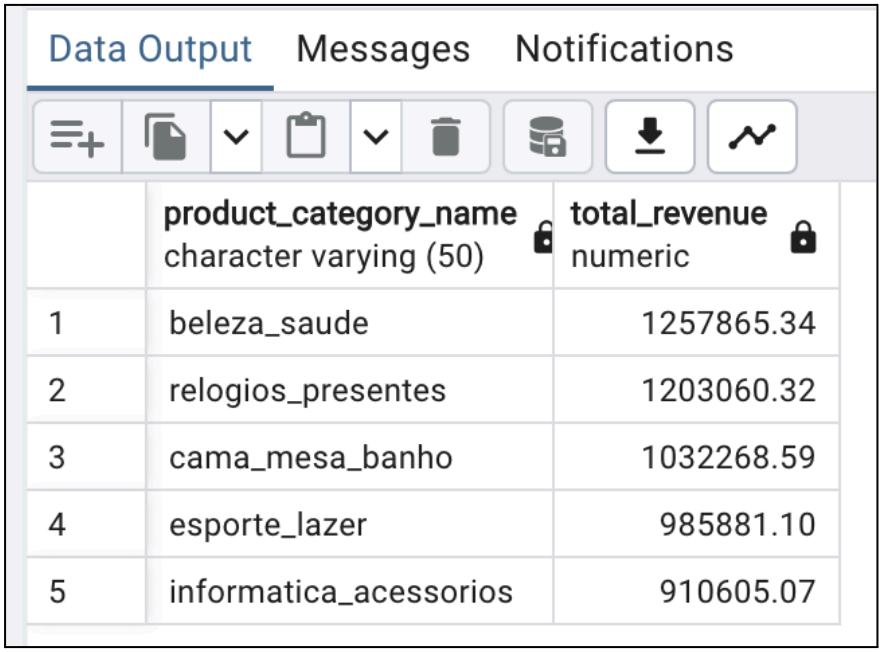
5. Sorting and Limiting Results:

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.products 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 (50)	total_revenue numeric
1	beleza_saude	1257865.34
2	relogios_presentes	1203060.32
3	cama_mesa_banho	1032268.59
4	esporte_lazer	985881.10
5	informatica_acessorios	910605.07

Recommendations:

ANALYSIS III

Problem Statement 1.

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

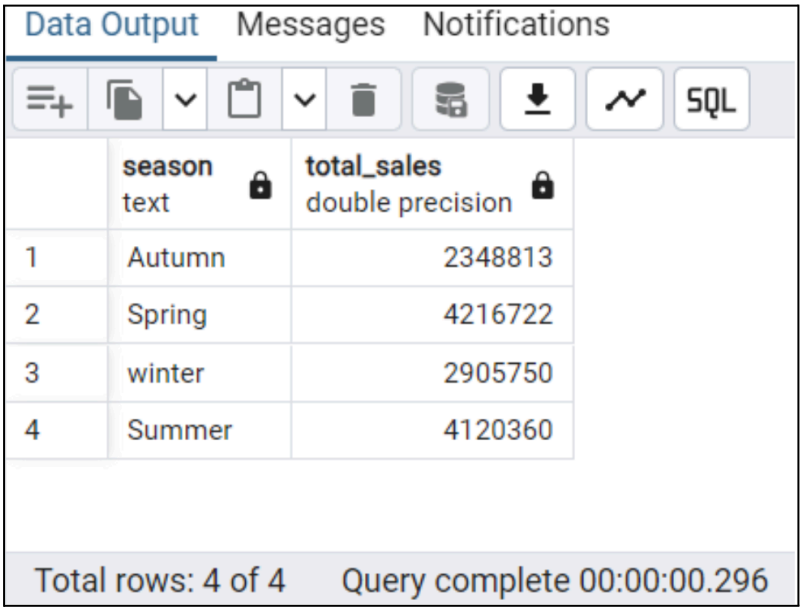
     Autumn: September, October, November

     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:



Recommendations:

Seasonal Sales Optimization Strategies

1. **Maximize High-Sales Seasons** – Run promotions and special offers during **Spring and Summer** to further boost revenue.
2. **Boost Winter & Autumn Sales** – Introduce discounts or holiday deals to drive purchases during lower-sales seasons.
3. **Plan for Seasonal Demand** – Stock up and promote key products ahead of peak seasons to meet increased demand effectively.

Problem Statement 2.

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 (50)	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
11	2b4609f8948be18874494203496bc3...	260

Recommendations:

Product Sales Optimization Strategies

1. **Market Best-Sellers** – Promote high-selling products to capitalize on their popularity.
2. **Ensure Stock Availability** – Keep top-selling items in stock to prevent lost sales.
3. **Boost Low-Sellers** – Adjust pricing or run promotions to improve sales of underperforming products.

Problem Statement 3.

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*.

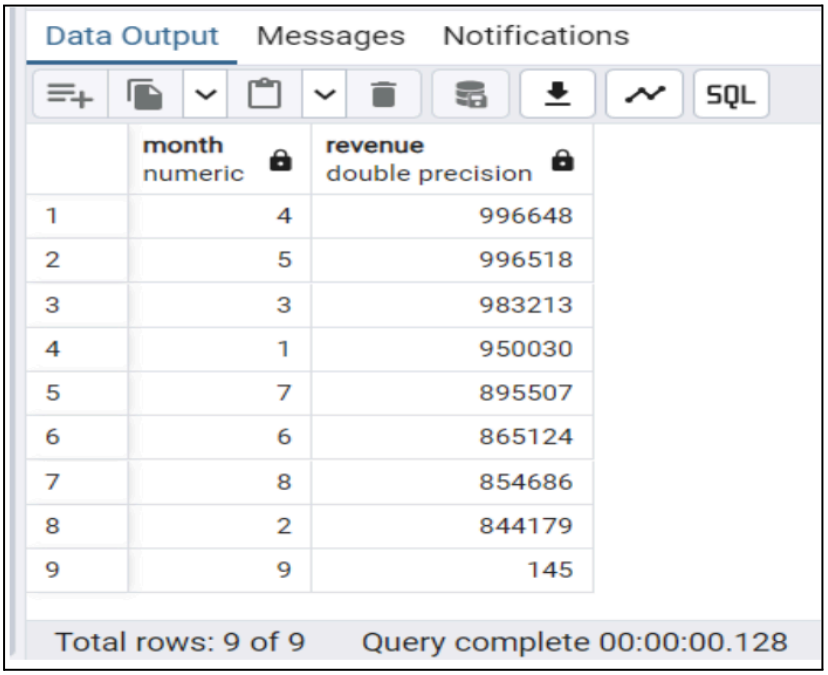
6. Grouping and Sorting Results:

- 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

Recommendations:

Revenue Optimization Strategies

1. **Leverage High-Revenue Months** – Maximize sales in **April and May** with targeted marketing campaigns and special promotions.
2. **Boost Low-Revenue Months** – Investigate **September's** revenue dip and introduce promotions or discounts to drive sales.
3. **Implement Seasonal Promotions** – Develop sales strategies aligned with peak months to encourage repeat purchases and attract new customers.

Problem Statement 4.

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.

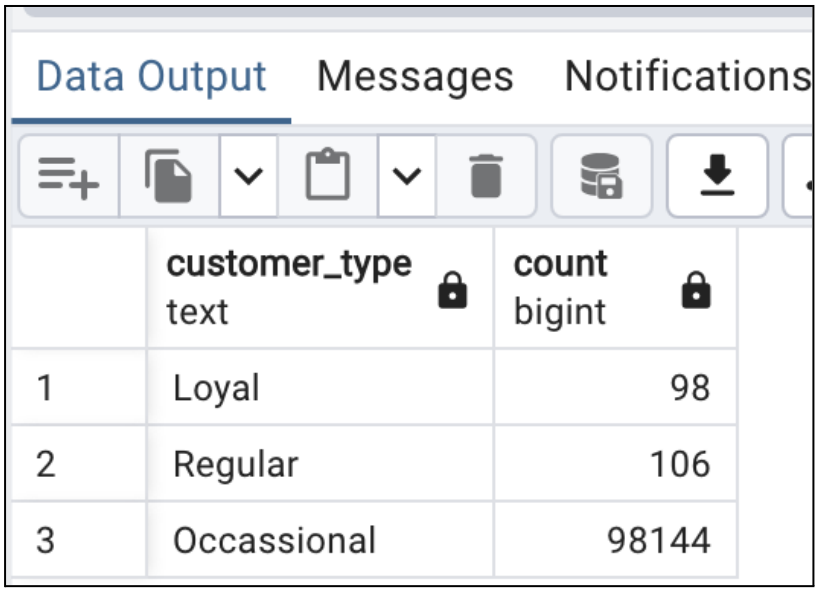
5. Grouping and Sorting Results:

- 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

Recommendations:

Customer Engagement & Retention Strategies

1. **Convert Occasional Shoppers** – Offer special promotions to encourage repeat purchases and build long-term engagement.
2. **Reward Regular Customers** – Implement a loyalty program to strengthen customer relationships and increase retention.
3. **Retain Loyal Customers** – Provide exclusive deals and personalized services to enhance customer satisfaction and long-term loyalty.

Problem Statement 5.

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.

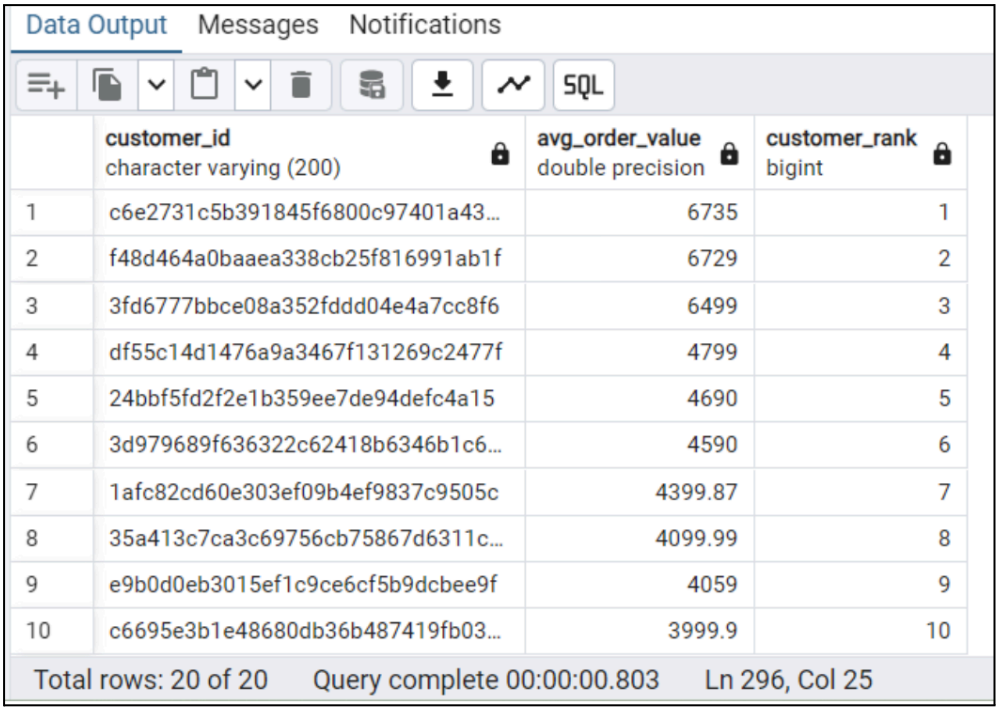
6. Sorting and Limiting Results:

• 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 oion 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:

Customer Spending Optimization Strategies

1. **Engage High-Spending Customers** – Offer exclusive deals and personalized offers to maintain their loyalty.
2. **Analyze Spending Trends** – Study high-value customers' purchasing habits to refine marketing strategies.
3. **Increase Average Order Value** – Use promotions or bundle deals to encourage mid-tier customers to spend more.

Problem Statement 6.

Amazon wants to analyze 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*.

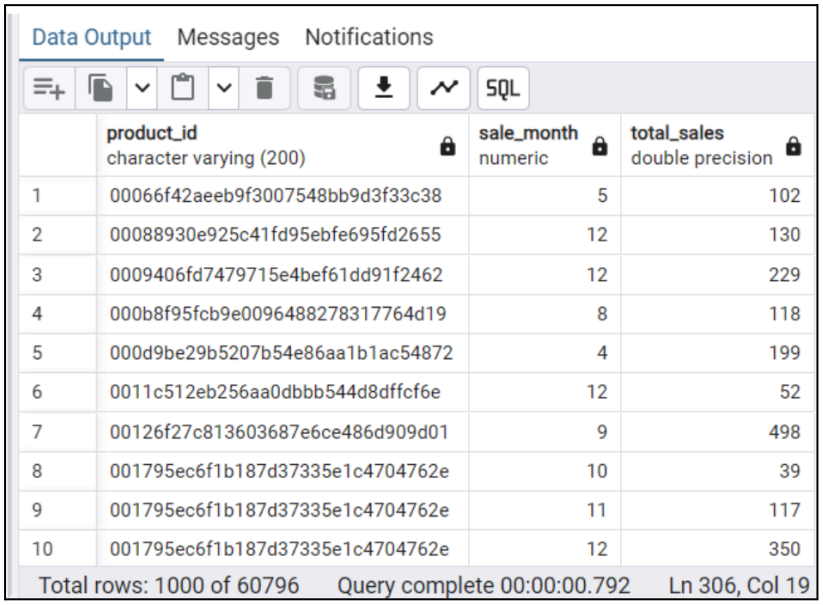
4. Sorting Results:

• 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 oion 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

SQL

	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:

Product Performance Strategies

1. **Track Best-Sellers** – Monitor top-selling products monthly to ensure adequate stock availability.
2. **Leverage Seasonal Demand** – Plan promotions for high-demand products during peak sales months.
3. **Improve Low-Selling Products** – Analyze underperforming items, adjusting pricing or addressing negative reviews to boost sales.

Problem Statement 7.

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 a 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 tre*nds 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:

Payment Trend Optimization Strategies

1. **Track Payment Trends** – Monitor shifts in payment method popularity and investigate declines (e.g., drops in "boleto" usage).
2. **Encourage Preferred Methods** – Promote consistently growing payment options like **credit cards** through targeted marketing.
3. **Address Declines Proactively** – Analyze and resolve sudden drops in payment usage, considering customer preferences, technical issues, or competition.