

INTRODUCTION

[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.

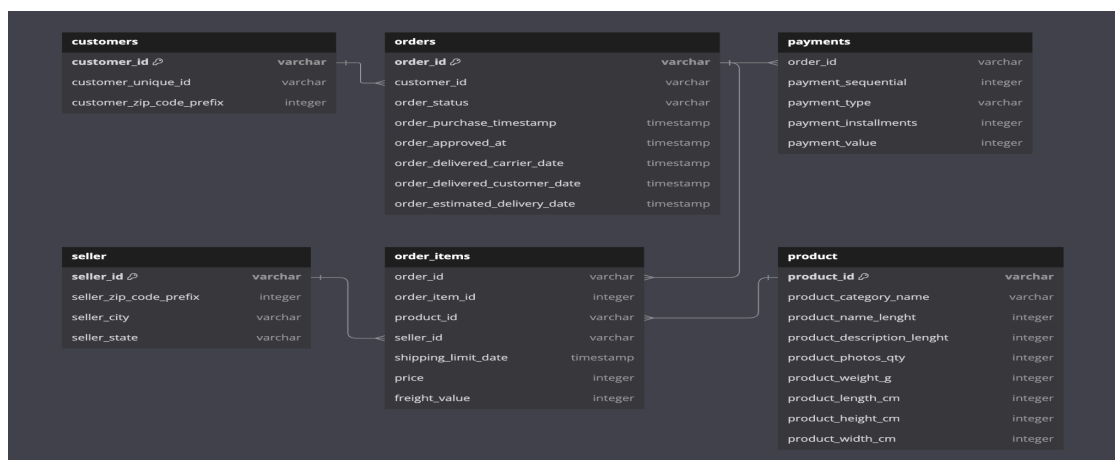
The task is to analyse Amazon Brazil's data to identify trends, customer behaviours, and preferences that could be leveraged in the Indian market. This analysis will help Amazon India make informed decisions, enhance customer experience, and seize new opportunities.

OVERVIEW OF SCHEMA

The schema consists of seven interconnected tables that provide insights into the operations of Amazon Brazil:

- **Customers:** Holds customer information, including unique identifiers and locations, to study customer demographics and behavior.
- **Orders:** Captures details about each order, such as order status and timestamps, essential for tracking the order lifecycle.
- **Order Items:** Lists each item in an order with details like price, seller, and shipping information.
- **Product:** Contains product specifications, such as category, size, and weight, for product-level analysis.
- **Seller:** Provides data about sellers, including their location and unique identifiers, to evaluate seller performance.
- **Payments:** Records transaction details, such as payment type and value, to understand payment preferences and financial performance.

Here is the schema of the provided data, including relationships and primary keys for each table.



ANALYSIS - I

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`

```
-- ANALYSIS1 1
--To simplify its financial reports, Amazon India needs to standardize payment values.
SELECT
    payment_type,
    ROUND(AVG(payment_value)) AS rounded_avg_payment
FROM
    amazon_brazil.payments
GROUP BY
    payment_type
ORDER BY
    rounded_avg_payment ASC;
```

APPROACH:

- Relevant Tables and columns:
 - Tables: `amazon_brazil.payment`
 - Columns: `payment_type`, `payment_value`
- Calculating Average and Round:
 - The `AVG()` function is used to find the average `payment_value`s
 - The `ROUND` function is used to round up the average `payment_value`
- Grouping and Sorting:
 - Grouping the result of average `payment_value` by each `payment_type` using `GROUP BY`
 - Displaying the result of `avg_payment` in sorted ascending order by using `ORDER BY`

OUTPUT:

	payment_type character varying 	rounded_avg_payment numeric 
1	not_defined	0
2	voucher	66
3	debit_card	143
4	boleto	145
5	credit_card	163

RECOMMENDATION:

- **Prioritize High-Value Payment Types**
 - High Average Payments: `credit_card` (163) and `boleto` (145) are the top contributors.
 - Focus financial tracking and promotions around these methods for potential revenue optimization.
- **Improve Use of Underperforming Methods:**
 - Introduce exclusive deals or cashback offers for voucher users.
 - Partner with popular brands to expand voucher acceptance and value.
 - Moderate usage compared to credit cards, may have an untapped user base.
 - Offer limited-time discounts or reward points for debit card transactions.
 - Highlight zero-interest or EMI options (if available) to match credit card appeal.

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`



```
--To refine its payment strategy, Amazon India wants to know the distribution of orders
--by payment type.
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;
```

APPROACH:

- Relevant Tables and columns:
 - Tables: `amazon_brazil.payment`
 - Columns: `payment_type`, `order_id`
- Calculating the Percentage:
 - For each payment type, divide the number of orders placed using that payment method by the total number of orders, then multiply by 100 to get the percentage.

- `(SELECT COUNT(*) FROM Orders)`: This subquery calculates the total number of orders across all payment types.
- `(COUNT(order_id) * 100.0) / (SELECT COUNT(*) FROM Orders)`: This formula calculates the percentage of orders for each payment type.
- Grouping and Ordering:
 - Groups the results by payment type using `GROUP BY`.
 - Orders the results in descending order based on the percentage using `ORDER BY`.

OUTPUT:

	payment_type character varying 	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

RECOMMENDATION:

- **Expand Credit Card & Boleto Adoption**
 - Provide better reward points or cashback on credit card transactions.
 - Increase promotional campaigns to encourage more users to use boleto payments.
 - **Improve Use of Underperforming Methods:**
 - Collaborate with banks to provide special offers or EMI options on debit card payments.
 - Reduce transaction failures and improve processing speed for debit card and voucher payments so that customers can purchase items using debit_card and voucher.
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`

```



--Amazon India seeks to create targeted promotions for products within specific price ranges
SELECT
    A.product_id, B.price
FROM
    amazon_brazil.order_items AS B
INNER JOIN
    amazon_brazil.product AS A
ON B.product_id=A.product_id
WHERE
    B.price BETWEEN 100 AND 500
AND A.product_category_name LIKE '%smart%'
ORDER BY
    B.price DESC;

```

APPROACH:

- Relevant Tables and columns:
 - Tables: `amazon_brazil.product`, `amazon_brazil.order_items`
 - Columns: `price`, `product_category_name`
- JOIN (joining two tables):
 - The `INNER JOIN` ensures that only rows with matching `product_id` in both `order_items` and `product` tables are included in the result set.
- WHERE condition:
 - This condition filters the results to include only products whose price falls between 100 and 500.
 - `LIKE %smart%` This filters the products to include only those whose `product_category_name` contains the word "smart". The `%` symbols are wildcards in SQL, meaning any characters can appear before or after the word "smart".
- Ordering:
 - `ORDER BY B.price DESC` This sorts the results in descending order based on the `price` column from the `order_items` table B.

OUTPUT:

	product_id character varying 	price numeric 
1	1df1a2df8ad2b9d3aa49fd851e3145...	439.99
2	7debe59b10825e89c1cbcc8b190c8...	349.99
3	ca86b9fe16e12de698c955aedff0ae...	349
4	ca86b9fe16e12de698c955aedff0ae...	349
5	0e52955ca8143bd179b311cc454a6...	335
6	7aeaa8f3e592e380c420e8910a717...	329.9
7	7aeaa8f3e592e380c420e8910a717...	329.9
8	7aeaa8f3e592e380c420e8910a717...	329.9

RECOMMENDATION:

- **Promote High-Value Products:**
 - Offer premium product bundles with exclusive discounts.
 - Highlight features & value through premium advertising.
 - Target high-spending customers using loyalty programs.
- **Mid-range Products:**
 - Offer discounts or bundles for products priced between 200–400 BRL to attract more customers.
 - Promote through cashback offers and EMI options.
- **Low-Cost Products:**
 - Offer as add-ons to larger orders

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



```
--To identify seasonal sales patterns, Amazon India needs to focus on the most successful months.
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
EXTRACT(MONTH FROM o.order_purchase_timestamp)
ORDER BY
total_sales
LIMIT 3;
```

APPROACH:

- Relevant Tables and columns:
 - Tables: amazon_brazil.orders, amazon_brazil.order_items
 - Columns: price, order_purchase_timestamp
- JOIN (joining two tables):
 - JOIN amazon_brazil.order_items oi ON o.order_id = oi.order_id This join ensures that for each order in the orders table, the corresponding items in the order_items table are included, allowing to calculate the sales.
- EXTRACT months:

- `EXTRACT(MONTH FROM o.order_purchase_timestamp)` This function extracts the month (1 through 12) from the `order_purchase_timestamp`. This makes it possible to group the results by the month of each order's purchase timestamp.
- Grouping, Ordering and limiting:
 - `GROUP BY EXTRACT(MONTH FROM o.order_purchase_timestamp)` This ensures that the sales are grouped by each month. Each month will have a total sales value.
 - `ORDER BY total_sales` orders the result in ascending order based on the total sales, meaning the months with the lowest total sales appear first.
 - `LIMIT 3` ensures that only the first 3 results (the 3 months with the lowest sales) are returned.

OUTPUT:

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

RECOMMENDATION:

- **Prioritize Promotions & Stocking:**
 - May, August, and July have high sales, indicating strong demand.
 - Increase inventory levels and optimize supply chains to avoid stockouts.
 - Consider special discounts and promotions to maximize profits.
 - **Targeted Marketing Campaigns:**
 - Launch aggressive marketing campaigns in April to prepare for peak demand in May.
 - Utilize Prime Day (usually in July) as a key event to boost sales further.
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`

```

--Amazon India is interested in product categories with significant price variations.
SELECT
    A.product_category_name,
    MAX(B.price)- MIN(B.price) AS price_difference
FROM
    amazon_brazil.order_items AS B
JOIN
    amazon_brazil.product AS A
ON B.product_id=A.product_id
GROUP BY
    A.product_category_name
HAVING
    MAX(B.price)- MIN(B.price) >500
ORDER BY
    price_difference DESC;

```

APPROACH:

- Relevant Tables and columns:
 - Tables: `amazon_brazil.product`, `amazon_brazil.order_items`
 - Columns: `price`, `product_category_name`
- JOIN (joining two tables):
 - This `INNER JOIN` links each product in the `order_items` table (B) with its corresponding details in the `product` table (A), based on the `product_id`. This way, we can access both the `price` from `order_items` and the `product_category_name` from each product.
- Calculate the Price Difference:
 - `MAX(B.price) - MIN(B.price) AS price_difference`: The `MAX(B.price)` gives the highest price for each product category, and `MIN(B.price)` gives the lowest price. Subtracting the two gives the price difference for each category.
- Filter the Results:
 - `HAVING MAX(B.price) - MIN(B.price) > 500`: The `HAVING` clause filters the results to only include product categories where the price difference is greater than 500.
- Grouping, Ordering:
 - `GROUP BY A.product_category_name`: The query groups the results by product category name to calculate the price difference within each category.
 - `ORDER BY price_difference DESC` This sorts the categories by the `price_difference` in descending order, ensuring that the product categories with the largest price differences appear first

OUTPUT:

	product_category_name character varying	price_difference numeric
1	utilidades_domesticas	6731.94
2	pcs	6694.5
3	artes	6495.5
4	eletroportateis	4792.5
5	instrumentos_musicais	4394.97
6	consoles_games	4094.81
7	esporte_lazer	4054.5
8	relogios_presentes	3990.91
9	[null]	3977
10	ferramentas_jardim	3923.65
11	bebes	3895.46

RECOMMENDATION:

- **Targeted Marketing & Promotions:**
 - Advertise premium products with personalized campaigns for high-value customers.
 - Highlight budget options for price-sensitive buyers.
- **Improve Category Segmentation:**
 - Consider sub-categorizing items based on price range (e.g., entry-level vs. high-end gaming consoles).
 - Provide better filtering options on the platform to help customers navigate price differences.

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`

```
--To enhance the customer experience, Amazon India wants to find which
SELECT payment_type, STDDEV(payment_value) AS std_deviation
FROM amazon_brazil.payments
GROUP BY payment_type
ORDER BY std_deviation DESC;
```

APPROACH:

- Relevant Tables and columns:
 - Tables: `amazon_brazil.product`, `amazon_brazil.order_items`
 - Columns: `price`, `product_category_name`
- Standard Deviation Calculation:
 - This selects the `payment_type` column and calculates the standard deviation of the `payment_value` for each group (payment type).
 - `STDDEV(payment_value)` computes the standard deviation of the `payment_value` column for each payment type. The result is aliased as `std_deviation` for clarity.
- Calculate the Price Difference:
 - `GROUP BY payment_type` This groups the rows by the `payment_type`. The standard deviation will be calculated separately for each distinct payment type.
 - `ORDER BY std_deviation DESC` After calculating the standard deviation for each payment type, the result is ordered in descending order.

OUTPUT:

	payment_type character varying 	std_deviation numeric 
1	debit_card	245.793401040647
2	credit_card	222.119310741208
3	boleto	213.581061475527
4	voucher	115.519185430458
5	not_defined	0

RECOMMENDATION:

- **Promote Voucher Payments:**
 - Since vouchers show the most stable transaction amounts, Amazon could encourage customers to use them by offering discounts or cash back incentives.
- **Monitor Debit and Credit Card Transactions:**
 - These payment types have the highest standard deviations, indicating significant fluctuations in transaction amounts. Analyzing the underlying reasons (e.g., transaction size variations,

installment payments) could help optimize the payment experience.

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`

```
--Amazon India wants to identify products that may have incomplete name in order to
SELECT product_id, product_category_name
FROM amazon_brazil.product
WHERE product_category_name ISNULL OR LENGTH(product_category_name)=1;
```

APPROACH:

- Relevant Tables and columns:
 - Tables: `amazon_brazil.product`, `amazon_brazil.order_items`
 - Columns: `price`, `product_category_name`
- Standard Deviation Calculation:
 - This selects the `payment_type` column and calculates the standard deviation of the `payment_value` for each group (payment type).
 - `STDDEV(payment_value)` computes the standard deviation of the `payment_value` column for each payment type. The result is aliased as `std_deviation` for clarity.
- Calculate the Price Difference:
 - `GROUP BY payment_type` This groups the rows by the `payment_type`. The standard deviation will be calculated separately for each distinct payment type.
 - `ORDER BY std_deviation DESC` After calculating the standard deviation for each payment type, the result is ordered in descending order.

OUTPUT:

	product_id [PK] character varying	product_category_name character varying
1	a41e356c76fab66334f36de622ecbd3a	[null]
2	d8dee61c2034d6d075997acef1870e...	[null]
3	56139431d72cd51f19eb9f7dae4d16...	[null]
4	46b48281eb6d663ced748f324108c7...	[null]
5	5fb61f482620cb672f5e586bb132eae9	[null]
6	e10758160da97891c2fdbc35f0f031d	[null]
7	39e3b9b12cd0bf8ee681bbc1c130feb5	[null]
8	794de06c32a626a5692ff50e4985d36f	[null]

RECOMMENDATION:

- **Data Quality Check:**

- Verify if the NULL values result from a data ingestion issue, database corruption, or missing mappings.
- Check if the product categories are available in another table or dataset.

ANALYSIS - II

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`

```
--Amazon India wants to understand which payment types are most popular
SELECT
  CASE
    WHEN payment_value < 200 THEN 'low'
    WHEN payment_value BETWEEN 200 AND 1000 THEN 'medium'
    ELSE 'high'
  END AS order_value_segment, payment_type,
  COUNT(*) AS count
FROM amazon_brazil.payments
GROUP BY
  order_value_segment, payment_type
ORDER BY count DESC;
```

USING CTE (OR)

```
WITH CTE AS(
  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'
    END AS segment
  FROM amazon_brazil.payments
)
SELECT segment , payment_type , COUNT(*)
FROM CTE
GROUP BY payment_type, segment
ORDER BY COUNT(*) DESC;
```

APPROACH:

- Relevant Tables and columns:
 - Tables: `amazon_brazil.payments`
 - Columns: `payment_value`, `payment_type`
- Common Table Expression (CTE):
 - The CTE (Common Table Expression) is used to create a temporary result set that simplifies the main query.
 - It extracts `payment_type` and assigns a `segment` based on `payment_value` using a `CASE` statement.
- Categorize Payment Values Using a CASE Statement:
 - The `CASE` statement is applied to classify the `payment_value` into three categories:
 - Low: `payment_value < 200`
 - Medium: `payment_value BETWEEN 200 AND 1000`
 - High: `payment_value > 1000`
- Aggregate and Count Transactions per Payment Type and Segment:
 - The outer query retrieves `segment`, `payment_type`, and counts the number of transactions in each category.
 - The data is grouped by `payment_type` and `segment` using `GROUP BY`.

OUTPUT:

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

RECOMMENDATION:

- **Encourage Credit Card Usage for High-Value Orders:**
 - Credit cards are already the preferred payment method, but Amazon could offer installment plans or cashback to encourage their use for high-value transactions.
 - **Promote Boleto for Medium-Value Orders:**
 - Since boleto is popular for low/medium-value transactions, offering incentives (like discounts on processing fees) could further drive its adoption in the medium range.
 - **Evaluate Voucher Effectiveness:**
 - Vouchers are mostly used for low-value orders and almost nonexistent in high-value transactions.
 - If Amazon wants to increase voucher usage, they could offer higher-value vouchers or apply them to more products.
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`

```
--Calculate the minimum, maximum, and average price for each category,
SELECT P.product_category_name,
MAX(oi.price) AS max_price,
MIN(oi.price) AS min_price,
AVG(oi.price) AS avg_price
FROM amazon_brazil.product AS P
JOIN amazon_brazil.order_items AS oi
ON P.product_id=oi.product_id
GROUP BY
product_category_name
ORDER BY avg_price DESC;
```

APPROACH:

- Relevant Tables and columns:
 - Tables: `amazon_brazil.order_items`, `amazon_brazil.product`
 - Columns: `price`, `product_category_name`
- MAX, MIN, AVG:
 - `P.product_category_name`: Retrieves the product category name.

- MAX(oi.price) AS max_price: Calculates the highest price within each category.
 - MIN(oi.price) AS min_price: Finds the lowest price in each category.
 - AVG(oi.price) AS avg_price: Computes the average price for products in each category.
- Join Tables for Data Extraction:
 - The amazon_brazil.product AS P table contains product details, including the category name.
 - The amazon_brazil.order_items AS oi table contains order-related details, including product prices.
 - The JOIN condition ON P.product_id = oi.product_id ensures that the query links product details with the respective order items.
- Group by Product Category & Sort by Average Price in Descending Order:
 - The GROUP BY product_category_name statement ensures that calculations are performed separately for each category.
 - ORDER BY avg_price DESC ranks categories based on their average price, listing the most expensive categories first.

OUTPUT:

	product_category_name character varying	max_price numeric	min_price numeric	avg_price numeric
1	pcs	6729	34.5	1098.3405418719211823
2	portateis_casa_forno_e_cafe	2899	10.19	624.2856578947368421
3	eletrodomesticos_2	2350	13.9	476.1249579831932773
4	agro_industria_e_comercio	2990	12.99	341.6610426540284360
5	instrumentos_musicais	4399.87	4.9	281.6160000000000000
6	eletroportateis	4799	6.5	280.7784683357879234
7	portateis_cozinha_e_preparadores_de_alimentos	1099	17.42	264.5686666666666667
8	telefonias_fixas	1790	6	225.6931818181818182
9	construcao_ferramentas_seguranca	3099.9	8.9	208.9923711340206186
10	relogios_presentes	3999.9	8.99	200.9118770875083500
11	climatizacao	1599	10.9	185.2692255892255892
12	moveis_quarto	650	6.9	183.7502752293577982
13	pc_gamer	239	129.99	171.7722222222222222

RECOMMENDATION:

- High-Value Categories:**
 - Promote installment payment plans or EMI options to encourage more sales.

- Run seasonal discounts on high-priced items to attract more buyers.
 - **Mid-Value Categories:**
 - Highlight customer reviews and testimonials to build trust.
 - Use flash sales or limited-time discounts to encourage impulse purchases.
 - **Low-Value Categories:**
 - Target price-sensitive customers through social media ads & promotions.
 - Provide affordable shipping options to make purchases more attractive.
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`

```
--Find all customers with more than one order, and display their
SELECT C.customer_unique_id, COUNT(O.order_id) AS total_orders
FROM amazon_brazil.customers AS C
JOIN amazon_brazil.orders AS O
ON C.customer_id=O.customer_id
GROUP BY C.customer_unique_id
HAVING COUNT(O.order_id)>1
order by total_orders DESC;
```

APPROACH:

- Relevant Tables and columns:
 - Tables: `amazon_brazil.customers, amazon_brazil.orders`
 - Columns: `order_id, customer_unique_id`
- Joining the Tables:
 - Since order details are in the `orders` table and customer details are in `customers`, JOIN them using:
`ON C.customer_id = O.customer_id`
- Filtering Customers with More Than One Order:
 - To find only those who placed multiple orders `HAVING COUNT(O.order_id) > 1`
 - This filters out customers who placed only a single order.
- Grouping by Unique Customers & Sorting by Order Count:
 - To count the number of orders per unique customer `GROUP BY C.customer_unique_id`. This ensures that each row represents a unique customer with their total order count.

- To prioritize customers with the highest number of orders, sort in descending order.

OUTPUT:

	customer_unique_id character varying	total_orders bigint
17	f9c4e8531c2fe4159beb562fd7c2bd59	16
18	3d364a7768fae99678635c4370295d...	16
19	6af40347f5dd7bdd65437a35e1b2fa7b	16
20	f300b00a19af4d4f7bdf9f4524c4587a	16
21	75f15790b1852b42b1dbf645d98ffa1c	16
22	8d50f5eadf50201ccdcedfb9e2ac8455	15
23	7c396fd4830fd04220f754e42b4e5bff	14
24	ce2e0ace655301bc4a8cae4abbd8c0...	11
25	b11b7871c2b8be2d11fab954f58542f2	11
26	2ee0d7587a1a04de482c6211ea2988...	9
27	1d8f1ca82cf6c04de2391eb61b9ae364	9
28	3d18156a1dff6d57e215431e455743...	9

RECOMMENDATION:

- **Potential for Subscription or Membership Plans:**
 - Amazon could introduce a membership program offering exclusive benefits (free shipping, early access to deals) to encourage continued spending.
 - **Cross-Selling & Upselling:**
 - If a customer buys frequently from a specific category (e.g., electronics), Amazon can recommend related products to increase basket size.
 - **Targeted Marketing Campaigns:**
 - High-frequency customers (10+ orders) → Offer personalized promotions or exclusive discounts.
 - Moderate repeat customers (2-9 orders) → Use targeted emails, reminders, and promotions to convert them into high-frequency buyers
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_unique_id, customer_type`

```
--Amazon India wants to categorize customers into different types
WITH customer_orders AS (
    SELECT
        customer_id,
        COUNT(DISTINCT order_id) AS orders
    FROM
        amazon_brazil.orders
    GROUP BY
        customer_id
)
SELECT
    c.customer_id,
    CASE
        WHEN co.orders = 1 THEN 'New'
        WHEN co.orders BETWEEN 2 AND 4 THEN 'Returning'
        WHEN co.orders > 4 THEN 'Loyal'
        ELSE 'New'
    END AS customer_type
FROM
    amazon_brazil.customers c
LEFT JOIN
    customer_orders co
ON
    c.customer_id = co.customer_id
ORDER BY customer_type DESC;
```

APPROACH:

- Relevant Tables and columns:
 - Tables: `amazon_brazil.customers, amazon_brazil.orders`
 - Columns: `order_id, customer_id`
- Joining the Tables:
 - A CTE (`customer_orders`) is created to count the number of distinct orders for each customer.
 - It retrieves: `customer_id, COUNT(DISTINCT order_id) AS orders`.
 - The data is grouped by `customer_id` to ensure each customer has a single entry with their total order count.
- Categorize Customers Based on Order Count:
 - The CASE statement is used to classify customers:
 - New: Customers with exactly 1 order.
 - Returning: Customers with 2 to 4 orders.
 - Loyal: Customers with more than 4 orders.

- Join with the Customers Table:
 - A **LEFT JOIN** is performed between the **customers** table and the **customer_orders** CTE.
 - This ensures that all customers are included, even if they haven't placed any orders.
 - Grouping by Unique Customers & Sorting by Order Count:
- Order the Results:
 - The results are sorted in descending order of customer type to prioritize.

OUTPUT:

	customer_id [PK] character varying	customer_type text
106	c7f920a7b4aed73af416b5fa6f5c5...	Returning
107	e4636bdf9d42e9f4b6c0b5bf7ec7d...	Returning
108	a0d14322c2c081915c5b59c56418...	Returning
109	f19622365d4c40c686c8ca3bec55...	Returning
110	da0d6b89a4173638a2b43ea9c099...	Returning
111	d1c6e90ab127d42b75ef2e548b44...	Returning
112	a625f7cf467b00ae83700201dce9...	Returning
113	bab0c43a2325bbc9ea398b00f8db...	Returning
114	a00009bf8489ae779a18179b4716...	Returning
115	9ec73448134f871bc7bdfdfd44481...	Returning
116	afe72ed049ec9d0b2197c3982869...	Returning
117	e95ac603dd00da603acf38d05e55...	Returning
118	88e9130b2f120b245d2bf74f8be17...	New
119	88ed0ffd90c7e0c8ab95752a16f01...	New
120	88ed670f1c8478bb99d01d6a8d9c...	New
121	88eddac47472e08eb17d96a25969...	New

RECOMMENDATION:

- **Strategies for "New" Customers:**
 - Offer limited-time deals or suggest related products based on their first order.
 - Send a survey to understand why they haven't placed a second order.
- **Strategies for "Returning" Customers:**
 - Offer exclusive discounts or points on their next purchase to encourage repeat business.
 - Engage them with special sales during key shopping seasons.

- **Strategies for "Loyal" Customers:**

- Provide early access to sales, free shipping, or premium services.
- Encourage them to refer friends with bonus rewards or credits.

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`



```
-- Use joins between the tables to calculate the total revenue
SELECT
p.product_category_name,
SUM(oi.price) AS total_revenue
FROM
amazon_brazil.order_items oi
JOIN
amazon_brazil.product p
ON
oi.product_id = p.product_id
JOIN
amazon_brazil.orders o
ON
oi.order_id = o.order_id
GROUP BY
p.product_category_name
ORDER BY
total_revenue DESC
LIMIT 5;
```

APPROACH:

- Relevant Tables and columns:
 - Tables: `amazon_brazil.product`, `amazon_brazil.order_items`
 - Columns: `price`, `product_category_name`
- Use Joins to Combine Data:
 - Join `order_items` (oi) with product (p) using the `product_id` column to fetch product category names.
 - Join `order_items` (oi) with orders (o) using `order_id` to ensure only valid orders are considered.

- **Aggregate Data:**
 - **SUM(oι.price) AS total_revenue:** Computes the total revenue for each category.
 - **GROUP BY p.product_category_name:** Groups data by product category to calculate the sum for each category.
- **Order & Limit Results:**
 - **ORDER BY total_revenue DESC:** Sorts categories from highest to lowest revenue.
 - **LIMIT 5:** Retrieves the top 5 categories with the highest revenue.

OUTPUT:

	product_category_name  character varying	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

RECOMMENDATION:

- **Increase Marketing & Promotions:**
 - Invest in influencer collaborations, special discounts, and bundle offers to boost sales further.
- **Enhance Home & Lifestyle Product Offerings:**
 - Consider introducing premium, eco-friendly, or luxury collections to cater to different customer segments.

ANALYSIS - III

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`

```
--Use a subquery to calculate total sales for each season (Spring, Summer, Autumn,
SELECT
season,
SUM(total_sales) AS total_sales
FROM (
SELECT
CASE
WHEN EXTRACT(MONTH FROM o.order_purchase_timestamp) IN (3, 4, 5) THEN 'Spring'
WHEN EXTRACT(MONTH FROM o.order_purchase_timestamp) IN (6, 7, 8) THEN 'Summer'
WHEN EXTRACT(MONTH FROM o.order_purchase_timestamp) IN (9, 10, 11) THEN 'Autumn'
ELSE 'Winter'
END AS season,
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
season
ORDER BY
total_sales DESC;
```

APPROACH:

- Relevant Tables and columns:
 - Tables: `amazon_brazil.order`, `amazon_brazil.order_items`
 - Columns: `price`, `order_purchase_timestamp`
- Extract the Month from Order Timestamps:
 - The `EXTRACT(MONTH FROM o.order_purchase_timestamp)` function retrieves the month from the order date.
 - Spring: March (3), April (4), May (5)

- Summer: June (6), July (7), August (8)
 - Autumn: September (9), October (10), November (11)
 - Winter: December (12), January (1), February (2)
- CASE Statement to Categorize Orders into Seasons & Calculate Total Sales per Season:
 - Orders are assigned to a season based on the extracted month.
 - The **SUM(oi.price)** function aggregates the total revenue from the **order_items** table for each season.
- Perform Joins to Get Sales Data:
 - The **orders** table (contains timestamps) is joined with the **order_items** table (contains prices).
 - The condition **o.order_id = oi.order_id** ensures that only relevant items are summed.
 - The subquery creates a **temporary dataset** that includes the season and sales amount.
- Group & Sort Results:
 - **GROUP BY season** aggregates the sales data by season.
 - **ORDER BY total_sales DESC** sorts the seasons in descending order of revenue.

OUTPUT:

	season text	total_sales numeric
1	Spring	4216721.54
2	Summer	4120359.62
3	Winter	2905750.03
4	Autumn	2348812.51

RECOMMENDATION:

- **Seasonal Marketing Campaigns:**
 - Run Spring & Summer promotions focusing on high-demand products.
 - Use Winter holidays and festive shopping seasons to attract more buyers.
- **Inventory & Supply Chain Optimization:**
 - Stock more products before Spring and Summer to meet demand.
 - Reduce excess inventory for low-performing seasons like Autumn.
- **Personalized Discounts & Retargeting:**

- Offer exclusive discounts for returning customers in Winter & Autumn.
- Use targeted advertising & remarketing for customers who purchased in peak seasons.

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`

```
--The inventory team is interested in identifying products that have sales
SELECT product_id,
       COUNT(order_item_id) AS total_quantity_sold
FROM amazon_brazil.order_items
GROUP BY product_id
HAVING COUNT(order_item_id) > (
    SELECT AVG(total_quantity)
    FROM (
        SELECT product_id, COUNT(order_item_id) AS total_quantity
        FROM amazon_brazil.order_items
        GROUP BY product_id
    ) AS avg_table
);
```

APPROACH:

- Relevant Tables and columns:
 - Tables: `amazon_brazil.order_items`
 - Columns: `product_id, order_item_id`
- Calculate Total Quantity Sold per Product:
 - From the `order_items` table, group by `product_id` and sum the `order_item_id` to get the total quantity sold for each product.
- Compute the Overall Average Quantity Sold:
 - Use an aggregation function to determine the average total quantity sold across all products.
 - `SELECT AVG(total_quantity) FROM (subquery)` This calculates the average quantity sold across all products.
- Filter Products Above the Average:
 - Use a subquery to compare each product's total quantity sold with the computed average.
 - `HAVING SUM(order_item_id) > (subquery)`

- Select only those products where the total quantity sold is greater than the overall average.

OUTPUT:

	product_id character varying	total_quantity_sold bigint
1	3d5837f86205fe83f03fb5f7e4d5b9cf	11
2	afeeea6271148ee1bb15173b8187c431	53
3	434487f82b5c35646bd8155cf1946179	4
4	e5063ce7fff1cf7cd528dc4c1e7dcba8	4
5	b25a0f93e25104798df2d1664495d157	4
6	6639a238ead6779d6ef0b3eea56f9f86	4
7	dceb3f67aef3484498a7caa4ba50f484	6
8	3c4e3782469a0f1ac459dc6c47ebef31	4
Total rows: 6366		Query complete 00:00:00.410

RECOMMENDATION:

- **Focus on High-Performing Products:**
 - Products with significantly higher-than-average sales should be prioritized for restocking and promotions.
 - **Reevaluate Low-Selling Products:**
 - Many products have only sold 4 units
 - **Analyze Inventory Trends:**
 - If the top 20% of products contribute to a majority of sales, focus inventory management on these items.
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](#)

```
--To understand seasonal sales patterns, the finance team is analysing the monthly revenue trends over

SELECT EXTRACT(YEAR FROM o.order_purchase_timestamp) AS year, SUM(oi.price) AS total_revenue,
EXTRACT(MONTH FROM o.order_purchase_timestamp) AS month
FROM amazon_brazil.orders AS o
JOIN amazon_brazil.order_items AS oi
ON o.order_id=oi.order_id
GROUP BY EXTRACT(MONTH FROM o.order_purchase_timestamp), EXTRACT(YEAR FROM o.order_purchase_timestamp)
HAVING EXTRACT(YEAR FROM order_purchase_timestamp)='2018'
ORDER BY month;
```

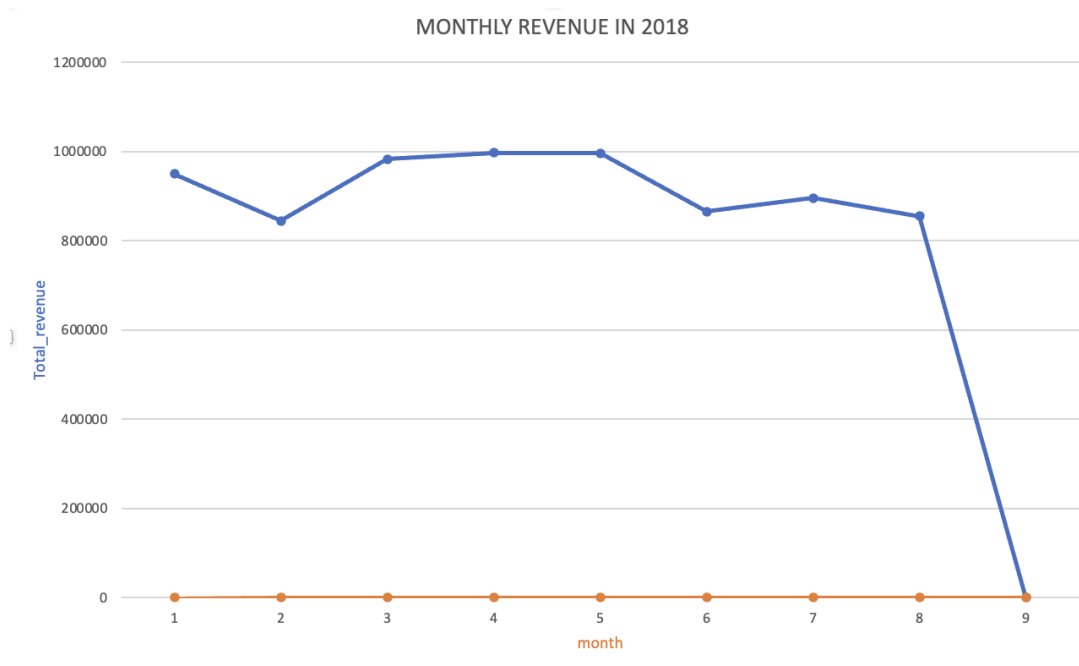
APPROACH:

- Relevant Tables and columns:
 - Tables: `amazon_brazil.order_items`, `amazon_brazil.orders`
 - Columns: `price`, `order_purchase_timestamp`
- Filter Data for the Year 2018 & Extract the Month:
 - Use the `order_purchase_timestamp` column from the `orders` table to extract only records where the year is 2018.
 - Convert the `order_purchase_timestamp` into a month-wise format to group revenue by each month.
- Calculate Total Revenue per Month:
 - Use the `price` column from the `order_items` table to sum up the total revenue for each month.
 - Consider joining `orders` and `order_items` on `order_id` to associate purchases with their respective months.
- Sort the Results Chronologically:
 - Order the final output by month (January to December 2018).

OUTPUT:

	year numeric 🔒	total_revenue numeric 🔒	month numeric 🔒
1	2018	950030.36	1
2	2018	844178.71	2
3	2018	983213.44	3
4	2018	996647.75	4
5	2018	996517.68	5
6	2018	865124.31	6
7	2018	895507.22	7
8	2018	854686.33	8
9	2018	145	9

EXCEL LINE CHART:



RECOMMENDATION:

- **Potential Data Issue in September (Month 9):**
 - The total revenue for September (Month 9) is only 145, which is extremely low compared to other months.
 - This suggests data inconsistency, missing transactions, or reporting errors.
- **Peak Revenue Periods:**
 - The highest revenue months appear to be March, April, May.
 - This could be driven by seasonal demand, promotions, or product launches.
- **Low Revenue Periods:**
 - February (844,178.71) and August (854,686.33) are among the lower-performing months.
 - This could indicate off-season demand dips.

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`

```
--A loyalty program is being designed for Amazon India
WITH customer_order AS (
  SELECT customer_id,
  COUNT(order_id) AS total_orders
  FROM amazon_brazil.orders
  GROUP BY customer_id
)

SELECT customer_type,
  COUNT(*) AS customer_count
FROM
  (SELECT customer_id, total_orders,
    CASE
      WHEN total_orders BETWEEN 1 AND 2 THEN 'Occasional'
      WHEN total_orders BETWEEN 3 AND 5 THEN 'Regular'
      ELSE 'Loyal'
    END AS customer_type
  FROM customer_order
)customer
GROUP BY customer_type;
```

APPROACH:

- Relevant Tables and columns:
 - Tables: `amazon_brazil.order_orders`
 - Columns: `customer_id, order_id`
- Count the Number of Orders per Customer:
 - The `orders` table contains both `customer_id` and `order_id`, which allows us to track how many times each customer has placed an order.
 - Group the `orders` table by `customer_id`.
 - Count the number of `order_ids` per `customer_id` to determine the total orders each customer has placed.
- Categorize Customers Based on Purchase Frequency:
 - **"Occasional"** → Customers who placed **1-2 orders**.
 - **"Regular"** → Customers who placed **3-5 orders**.
 - **"Loyal"** → Customers who placed **more than 5 orders**.

- Aggregate the Count of Customers in Each Segment:
 - After classifying customers into segments, count how many customers fall into each segment.
 - This step helps understand the distribution of customer types.

OUTPUT:

	customer_type text	customer_count bigint
1	Occasional	98144
2	Regular	106
3	Loyal	98

EXCEL BAR CHART:



RECOMMENDATION:

- **Challenges & Opportunities:**
 - Majority (99%) are Occasional buyers, indicating a low customer retention rate.
 - Loyal customers (98) are high-value users and should be nurtured to become brand advocates.
 - Regular customers (106) are in a transition phase and could be encouraged to make more purchases.
 - **Retain & Engage Regular Customers:**
 - Offer exclusive coupons for their next purchase.
 - Provide bonus points for referring friends.
 - **Reward Loyal Customers to Sustain Engagement:**
 - Launch a "VIP Tier" with early access to deals, premium support, and surprise gifts.
 - Provide anniversary/birthday rewards.
 - Offer priority shipping or free delivery as an exclusive benefit.
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

```
--Amazon wants to identify high-value customers to target for an
WITH customer_revenue AS(
SELECT o.customer_id,  AVG(oi.price) AS avg_order_value
      FROM amazon_brazil.orders AS o
 JOIN amazon_brazil.order_items AS oi
 ON o.order_id=oi.order_id
 GROUP BY o.customer_id),
ranked_customers AS(
SELECT
  customer_id,
  avg_order_value,
  RANK() OVER(ORDER BY avg_order_value DESC) AS customer_rank
  FROM customer_revenue
)
SELECT customer_id, avg_order_value, customer_rank
FROM ranked_customers
WHERE customer_rank <= 20;
```

APPROACH:

- Relevant Tables and columns:
 - Tables: `amazon_brazil.order_orders`, `amazon_brazil.order_order_items`
 - Columns: `customer_id`, `price`
- Calculate Total Spending per Customer:
 - Use the `orders` table to get `customer_id` and join it with `order_items` to calculate the total revenue per customer.
 - Average the `price` column in `order_items` for each `customer_id`.
- Rank Customers Based on AOV:
 - Use a ranking function (like `RANK()`) to assign ranks based on descending `avg_order_value`.
- Filter the Top 20 Customers:
 - Select only the top 20 customers based on their rank.

OUTPUT:

	customer_id character varying	avg_order_value numeric	customer_rank bigint
1	c6e2731c5b391845f6800c97401a43...	6735.0000000000000000	1
2	f48d464a0baaea338cb25f816991ab1f	6729.0000000000000000	2
3	3fd6777bbce08a352fddd04e4a7cc8f6	6499.0000000000000000	3
4	df55c14d1476a9a3467f131269c2477f	4799.0000000000000000	4
5	24bbf5fd2f2e1b359ee7de94defc4a15	4690.0000000000000000	5
6	3d979689f636322c62418b6346b1c6...	4590.0000000000000000	6
7	1afc82cd60e303ef09b4ef9837c9505c	4399.8700000000000000	7
8	35a413c7ca3c69756cb75867d6311c...	4099.9900000000000000	8
9	e9b0d0eb3015ef1c9ce6cf5b9dcbee9f	4059.0000000000000000	9
10	c6695e3b1e48680db36b487419fb03...	3999.9000000000000000	10
11	926b6a6fb8b6081e00b335edaf578d...	3999.0000000000000000	11
12	3be2c536886b2ea4668eced3a80dd0...	3980.0000000000000000	12
Total rows: 20		Query complete 00:00:00.233	

RECOMMENDATION:

- **Top Customers Identified:**
 - The top 20 customers have been ranked based on average order value (`avg_order_value`).
 - The highest avg order value is \$6,735, and the lowest in the top 20 is around \$3,980.

- **Elite Customer Segment:**
 - Customers with avg order values above \$6,500 should be categorized as elite spenders and prioritized for premium loyalty programs.
 - They are the most profitable and likely to respond well to exclusive rewards or VIP services.
 - **Growth Opportunity:**
 - Customers ranked between 10-20 (avg order value ~\$3,980 - \$4,100) can be nurtured to increase spending.
 - Introduce targeted promotions, limited-time offers, and upselling strategies.
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

```
-- Amazon wants to analyze sales growth trends for its key products over
WITH monthly_sales AS (
    SELECT
        oi.product_id,
        TO_CHAR(o.order_purchase_timestamp, 'YYYY-MM') 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 oi.product_id, sale_month
)
SELECT
    product_id,
    sale_month,
    SUM(monthly_sales) OVER (
        PARTITION BY product_id
        ORDER BY sale_month ASC
    ) AS total_sales
FROM monthly_sales
ORDER BY product_id, sale_month;
```


APPROACH:

- Relevant Tables and columns:
 - Tables: `amazon_brazil.order_orders`, `amazon_brazil.order_order_items`
 - Columns: `order_purchase_timestamp`, `price`
- Extract Monthly Sales Data:
 - Convert the `order_purchase_timestamp` (from the `orders` table) into year-month format (`YYYY-MM`).
 - Sum the `price` (from the `order_items` table) grouped by product and month.
- Recursive CTE:
 - The base case gets the first month's sales for each product.
 - The recursive case adds the current month's sales to the cumulative total from the previous month.
- Compute Cumulative Sales:
 - Using a self-referencing join within the recursive CTE.
 - Ensuring that the current month includes all previous months' sales.

OUTPUT:

	product_id character varying	sale_month text	total_sales numeric
1	00066f42aeeb9f3007548bb9d3f33c38	2018-05	101.65
2	00088930e925c41fd95ebfe695fd2655	2017-12	129.9
3	0009406fd7479715e4bef61dd91f2462	2017-12	229
4	000b8f95fcb9e0096488278317764d19	2018-08	117.8
5	000d9be29b5207b54e86aa1b1ac548...	2018-04	199
6	0011c512eb256aa0dbbb544d8dffcf6e	2017-12	52
7	00126f27c813603687e6ce486d909d01	2017-09	498
8	001795ec6f1b187d37335e1c4704762e	2017-10	38.9
9	001795ec6f1b187d37335e1c4704762e	2017-11	116.7
Total rows: 62176		Query complete 00:00:00.219	

RECOMMENDATION:

- **Identify High-Growth Products:**
 - Products with consistently increasing sales over months should be prioritized for marketing and stock replenishment.

- **Monitor Seasonal Sales Trends:**
 - Plan inventory and marketing campaigns around peak sales months.
- **Optimize Product Lifecycle Strategies:**
 - Products with consistent monthly growth should have extended promotion and better supply chain planning.
 - Products with sporadic sales may need seasonal promotions or bundling strategies.

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.

```
--To understand how different payment methods affect monthly sales growth, Amazon
WITH monthly_sales AS (
    SELECT
        p.payment_type,
        TO_CHAR(o.order_purchase_timestamp, 'YYYY-MM') AS sale_month,
        SUM(oi.price) AS monthly_total
    FROM amazon_brazil.orders o
    JOIN amazon_brazil.order_items oi
        ON o.order_id = oi.order_id
    JOIN amazon_brazil.payments p
        ON o.order_id = p.order_id
    WHERE EXTRACT(YEAR FROM o.order_purchase_timestamp) = 2018 -- Only 2018 sales
    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
        )) * 100.0 / NULLIF(LAG(monthly_total) OVER (
            PARTITION BY payment_type
            ORDER BY sale_month
        ), 0), 2
    ) AS monthly_change
FROM monthly_sales
ORDER BY payment_type, sale_month;
```

APPROACH:

- Relevant Tables and columns:
 - Tables: `amazon_brazil.order_orders`, `amazon_brazil.order_order_items`
 - Columns: `order_purchase_timestamp`, `price`
- Calculate Monthly Sales for Each Payment Method:
 - Extract year and month (YYYY-MM) from the `order_purchase_timestamp` field in the `orders` table.
 - Join with the `order_payments` table to get `payment_method` (`payment_type`).
 - Compute total sales per month per payment type by summing `price` from `order_items`.
 - Filter for only the year 2018 (`WHERE year = 2018`).
- Compute Month-over-Month Growth (%):
 - Use `LAG()` window function to get the previous month's sales for each `payment_type`.
- Return the Final Output:
 - Order the results by payment method and month (YYYY-MM).

OUTPUT:

	payment_type character varying	sale_month text	monthly_total numeric	monthly_change numeric
1	boleto	2018-01	170651.40	[null]
2	boleto	2018-02	153165.77	-10.25
3	boleto	2018-03	157807.38	3.03
4	boleto	2018-04	162940.61	3.25
5	boleto	2018-05	166571.60	2.23
6	boleto	2018-06	126379.90	-24.13
7	boleto	2018-07	162938.42	28.93
8	boleto	2018-08	118214.12	-27.45
9	credit_card	2018-01	760252.76	[null]
Total rows: 33		Query complete 00:00:00.171		

RECOMMENDATION:

- **"Boleto" Shows Instability in Sales Trends:**

- There is no consistent growth in sales for "boleto"; it fluctuates significantly.
 - "boleto" usage declined sharply in June and August. Consider targeted promotions or incentives to stabilize growth.
- **Credit Card Transactions Are Likely More Stable:**
 - credit card data is missing or that it dominates sales volume in a way that wasn't included in further months.
- **Evaluate Seasonal Patterns for Payment Preferences:**
 - Since "boleto" saw sharp drops in June and August, this could indicate a seasonal trend where users switch to other payment methods during these months.