

# Title: Data-Driven Insights for Amazon India: Leveraging Amazon Brazil's Market Trends

Amazon, a global e-commerce leader, has successfully connected small and medium businesses with millions of customers in Brazil, leveraging its large population and dynamic consumer base. This project analyzes Amazon Brazil's customer, order, and product data to uncover trends and behaviors that can be applied to the Indian market. Given the similarities between Brazil and India, such as diverse demographics and emerging e-commerce potential, these insights aim to enhance Amazon India's market strategies. The findings will support informed decision-making, improve customer experience, and identify new opportunities for growth in India's competitive e-commerce landscape.

## Overview of the 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.

# Analysis - I



## Question 1

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

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 DESC;
```

Output:

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

Approach:

Identifying Relevant Tables and Columns:

- Table: `payments`
- Columns: `payment_type`, `payment_value`

Calculating Average Payment Value:

- Used the `AVG()` aggregate function on `payment_value`.
- Grouped the results by `payment_type` to get the average for each type.

Rounding the Averages:

- Used the `ROUND()` function to round the average payment values to the nearest integer.

Sort the Results:

- Ordered the final results in ascending order based on the rounded average payment.

### Recommendations:

#### 1. Focus on Popular Payment Methods:

- Optimize popular payment methods like **credit\_card** and **boleto** as their average payment amounts are higher.
- Introduce targeted promotions for customers using these methods to increase revenue.

#### 2. Improve Use of Underperforming Methods:

- For underperforming methods like **debit\_card**, explore growth opportunities by offering promotional discounts or enhancing the transaction process to improve user adoption.

## Question 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

### Query:

```
SELECT
    payment_type,
    ROUND((COUNT(*) * 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:

	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

## Approach:

### 1. Identifying Relevant Tables and Columns:

- **Table:** `payments`
- **Columns:** `payment_type`

### 2. Calculating Percentage of Orders:

- Calculate the total number of orders by using `COUNT(*)` within a subquery.
- For each payment type, count the orders using `COUNT(*)` and calculate the percentage by dividing the count of orders for the payment type by the total number of orders.
- Multiply by 100 to convert the ratio to a percentage and use the `ROUND()` function to round the result to one decimal place.

### 3. Sorting the Results:

- The results are ordered in **descending** order to display the payment types with the highest percentage first.

## Recommendations:

### 1. Focus on Popular Payment Methods:

- Since `credit_card` accounts for 73.9% of orders, Amazon India should optimize this payment method and offer rewards or promotions to retain and attract users.
- 

### 2. Boost Other Payment Methods:

- With `boleto` (19.0%) and `voucher` (5.6%) contributing significantly, Amazon India should increase adoption by offering incentives or targeted promotions.
- 

### 3. Investigate "not\_defined" Payments:

- The `not_defined` category has 0.0% orders, suggesting incomplete data. Amazon India should review and categorize undefined payment methods for better data accuracy.
- 

### 4. Optimize Underutilized Payment Methods:

For `debit_card` (1.5%), Amazon India could enhance integration with banks and offer incentives to encourage higher usage.

### Question 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.

Query:

```
SELECT O.product_id, O.price
FROM amazon_brazil.order_items AS O
INNER JOIN amazon_brazil.product AS P
ON O.product_id = P.product_id
WHERE O.price BETWEEN 100 AND 500
AND P.product_category_name LIKE '%smart%'
ORDER BY O.price DESC;
```

Output:

Output contains 34 rows

	product_id character varying	price real
1	1df1a2df8ad2b9d3aa49fd851e3145...	439.99
2	7debe59b10825e89c1cbcc8b190c8...	349.99
3	ca86b9fe16e12de698c955aedff0aea2	349
4	ca86b9fe16e12de698c955aedff0aea2	349
5	0e52955ca8143bd179b311cc454a6...	335
6	7aeaa8f3e592e380c420e8910a7172...	329.9
7	7aeaa8f3e592e380c420e8910a7172...	329.9
8	7aeaa8f3e592e380c420e8910a7172...	329.9
9	7aeaa8f3e592e380c420e8910a7172...	329.9
10	7aeaa8f3e592e380c420e8910a7172...	329.9
11	7aeaa8f3e592e380c420e8910a7172...	329.9
12	d1b571cd58267d8cac8b2afd6e288b...	299.9
13	d1b571cd58267d8cac8b2afd6e288b...	299.9
14	66ffe28d0fd53808d0535eee4b90a157	254
15	f06796447de379a26dde5fcac6a1a2f7	239.9

## Approach:

### 1. Identify Relevant Tables and Columns:

- Use the `order_items` table to access `price` (to filter by price range) and `product_id` (to link with the `product` table).
- Use the `product` table to access `product_id` (to join with `order_items`) and `product_name` (to filter by the keyword "Smart").

### 2. Filter Data:

- Use `WHERE O.price BETWEEN 100 AND 500` to ensure only products within the specified price range are included.
- Use `P.product_category_name LIKE '%smart%'` to find products with the word "Smart" in their names, ignoring case sensitivity.

### 3. Join Tables:

- Perform an `INNER JOIN` between `order_items` and `product` on the `product_id` column to combine price and product information.

### 4. Sort Results:

- Use `ORDER BY O.price DESC` to display the most expensive products first, aligning with the descending price order requirement.

## Recommendations:

**1. Promote High-Value Products:** Focus on products near 500 BRL, like those priced at 439.99 BRL, for maximum revenue.

**2. Boost Mid-Range Sales:** Offer discounts or bundles for products priced between 200–400 BRL to attract more customers.

**4. Attract Budget Buyers:** Include low-priced products (100–150 BRL) in flash sales to appeal to cost-sensitive shoppers.

## Question 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.

Query:

```
SELECT
    EXTRACT(MONTH FROM o.order_purchase_timestamp) AS month,
    ROUND(SUM(oi.price)::NUMERIC) AS total_sales
FROM
    amazon_brazil.orders o
JOIN
    amazon_brazil.order_items oi ON o.order_id = oi.order_id
WHERE
    o.order_status = 'delivered'
GROUP BY
    EXTRACT(MONTH FROM o.order_purchase_timestamp)
ORDER BY
    total_sales DESC
LIMIT 3;
```

Output:

	month numeric	total_sales numeric
1	5	1466880
2	8	1393270
3	7	1349560

Approach:

### 1. Extract and Transform:

- Use `EXTRACT(MONTH FROM order_purchase_timestamp)` to extract the month for grouping sales by month.
- Calculate total monthly sales using `SUM(oi.price)` and format the result with `ROUND(SUM(oi.price)::NUMERIC)` for clarity in output.

### 2. Join Tables:

- Perform an `INNER JOIN` on `orders` and `order_items` using `order_id` to combine order and item-level data.

### 3. Group and Aggregate:

- Group by the extracted month to calculate monthly total sales.

#### 4. Sort and Limit Results:

- Use `ORDER BY total_sales DESC` to rank months by sales, and `LIMIT 3` to focus on the top 3 months.

### Recommendations:

#### 1. Target Seasonal Campaigns:

- Focus marketing efforts during the top-performing months (May, August, and July).
- Offer discounts, exclusive deals, or promotions to capitalize on high customer activity during these periods.

#### 2. Optimize Inventory Management:

- Ensure adequate stock availability for popular products in these months to avoid lost sales due to stockouts.
- Analyze product trends from these months to prioritize high-demand items.

#### 3. Enhance Customer Experience:

- Improve logistics to ensure timely deliveries during peak months.
- Strengthen customer support to handle the increased demand.

## Question 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.

### Query:

```
SELECT
    p.product_category_name,
    MAX(oi.price) - MIN(oi.price) AS price_difference
FROM
    amazon_brazil.order_items oi
JOIN
    amazon_brazil.product p ON oi.product_id = p.product_id
GROUP BY
    p.product_category_name
HAVING
    MAX(oi.price) - MIN(oi.price) > 500
ORDER BY
    price_difference DESC;
```



## Output:

	product_category_name character varying	price_difference real
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	4094.8103
7	esporte_lazer	4054.5
8	relogios_presentes	3990.91
9	[null]	3977
10	ferramentas_jardim	3923.65
11	bebes	3895.46
12	informatica_acessorios	3696.09
13	beleza_saude	3122.8
14	cool_stuff	3102.99
15	construcao_ferramentas_seguranca	3091

## Approach:

### 1. Identify Relevant Tables and Columns:

- Table: `orders_items` provides the `price` column for price calculations.
- Table: `product` provides the `product_category_name` column to group products by category.

### 2. Join Tables:

- Performed an INNER JOIN between `orders_items` and `product` using the `product_id` column to link product information with price details.

### 3. Calculate Price Difference:

- Used the `MAX(oi.price)` and `MIN(oi.price)` aggregate functions to calculate the difference between the highest and lowest prices for each category.

#### 4.Filter Significant Variations:

- Used the **HAVING** clause to filter categories where the price difference exceeds 500 BRL.

#### 5.Group by Product Categories:

- Grouped the results by **p.product\_category\_name** to calculate the price difference for each category.

#### 6.Sort Results:

- Used **ORDER BY price\_difference DESC** to display categories in descending order of price difference.

#### Recommendations:

##### 1.Focus on High-Variation Categories:

- Prioritize categories like "utilidades\_domesticas," "pcs," and "artes" for promotions as they show the highest price differences.
- Highlight the premium and budget options within these categories to attract a wide audience.

##### 2.Enhance Product Offerings:

- Expand the range of products in categories like "eletroportateis" and "instrumentos\_musicais" to cover diverse price segments.
- Introduce bundles to combine high and low-priced items for better value perception.

##### 3.Targeted Promotions:

- Launch tailored discounts for categories with high demand but wide price gaps, such as "esporte\_lazer" and "beleza\_saude."
- Offer personalized recommendations based on browsing history to appeal to both budget-conscious and premium buyers.



## Question 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.

### Query:

```
SELECT
    p.payment_type,
    ROUND(STDDEV(p.payment_value), 2) AS payment_stddev
FROM
    amazon_brazil.payments p
GROUP BY
    p.payment_type
ORDER BY
    payment_stddev ASC;
```

### Output:

	payment_type character varying 	payment_stddev numeric 
1	not_defined	0.00
2	voucher	115.52
3	boleto	213.58
4	credit_card	222.12
5	debit_card	245.79

### Approach:

#### 1. Identify Relevant Table and Columns:

- Table: `payments`
- Columns: `payment_type`, `payment_value`

#### 2. Calculate Standard Deviation for Payment Values:

- Use the `STDDEV()` function to compute the standard deviation of `payment_value` for each `payment_type`.

#### 3. Group Data by Payment Type:

- Apply the `GROUP BY` clause on `payment_type` to calculate the standard deviation for each payment method.

#### 4. Round the Results:

- Use the `ROUND()` function to round the calculated standard deviation (`STDDEV(payment_value)`) to two decimal places for better readability.

## 5.Sort the Results:

- Apply `ORDER BY payment_stddev ASC` to sort the results in ascending order of the rounded standard deviations.

## Recommendations:

### 1.Focus on Consistent Payment Methods:

- Promote voucher and boleto payment types, as they have low standard deviations, ensuring stable transaction values.
- Offer exclusive deals for customers using voucher to encourage usage of this highly consistent method.

### 2.Optimize Popular Payment Types:

- Enhance the checkout experience for credit\_card users, as it balances consistency and popularity.
- Use targeted campaigns to drive more customers toward using boleto, leveraging its reliability and moderate variance.

## Question 7

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

### Query:

```
SELECT
    product_id,
    product_category_name
FROM
    amazon_brazil.product
WHERE
    product_category_name IS NULL
    OR LENGTH(product_category_name) = 1;
```

### Output:

Output contains 614 rows

	product_id [PK] character varying	product_category_name character varying
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	bc815bba008d89458e428078c0b92...	[null]
13	6b82874c6b51b92913dcdb364eaaae0f	[null]
14	c68b419d9c6038271b85bac98adb0fc9	[null]
15	1dcd65bb5dd967d7b4c6b0223cefb838	[null]

### Approach:

#### 1. Identify Relevant Table and Columns:

- Table: `product`
- Columns: `product_id`, `product_category_name`

#### 2. Filter Rows with Incomplete or Missing Data:

- Use the `WHERE` clause to filter rows where `product_category_name` is either:
  - `NULL`: indicating a missing value.
  - A single character: determined by using the `LENGTH()` function to check if the length equals 1.

#### 3. Select Required Columns:

- Include `product_id` to uniquely identify the products with issues.

- Include `product_category_name` to verify and correct the incomplete names.

#### **4.Retrieve Results:**

- Execute the query to get the list of products that meet the conditions for incomplete or missing category names.

#### **Recommendations:**

##### **1. Address Product Category Data Quality:**

- Identify and resolve the data quality issues by ensuring that all products have valid, complete product category names.
- Implement validation checks during the product listing process to prevent missing or incomplete category names from being added in the future.

##### **2. Data Clean-Up and Correction:**

- Perform a data clean-up of the 614 products with missing or incomplete category names. Ensure that they are accurately categorized, which will improve the overall accuracy of product searches and filtering on the platform.

## Analysis - II

**Question 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

Query:

```
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;
```

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

## Approach:

### 1. Identify Relevant Table and Columns:

- Table: `amazon_brazil.payments`
- Columns: `payment_type`, `payment_value`

### 2. Categorize Payment Values:

- Use a `CASE` statement inside a CTE (`WITH CTE AS`) to categorize `payment_value` into segments:
  - `< 200` → `low`
  - `200-1000` → `medium`
  - `> 1000` → `high`

### 3. Group and Aggregate Data:

- Use the CTE to group data by `segment` and `payment_type`.
- Calculate the count of each `payment_type` within each segment using `COUNT(*)`.

### 4. Sort Results:

- Sort the grouped results in descending order of count using `ORDER BY COUNT(*) DESC`.

### 5. Retrieve Results:

- Output the columns: `segment`, `payment_type`, `count`.
- This helps identify the most popular payment types across different order value segments.

## Recommendations:

### 1. Understand Popular Payment Methods in Each Segment:

- The results indicate that `credit_card` is the most frequently used payment method across all segments (low, medium, high).
- Focus on optimizing the credit card payment process (e.g., faster processing, secure transactions).

### Improve Low-Value Segment Payment Experience:

- Since the majority of transactions fall into the low segment (e.g., `credit_card`: 60,548 and `boleto`: 16,444), streamline low-value payment methods by



offering options like digital wallets or UPI-based payments to attract more customers.

### 3.Boost Medium-Value Transactions:

- The medium segment (e.g., credit\_card: 15,303) is underutilized compared to the low-value segment. Introduce promotions or discounts for medium-value purchases to increase transaction volumes in this category.

## Question 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.

Query:

```
SELECT
    p.product_category_name,
    MIN(oi.price) AS min_price,
    MAX(oi.price) AS max_price,
    ROUND(AVG(oi.price)::NUMERIC, 2) AS avg_price
FROM
    amazon_brazil.product p
JOIN
    amazon_brazil.order_items oi
    ON p.product_id = oi.product_id
GROUP BY
    p.product_category_name
ORDER BY
    avg_price DESC;
```

Output:

The output contains 79 rows, with each row representing a unique product category.

The data is sorted in **descending order of average price**, meaning the product category with the highest average price appears first.

	product_category_name character varying	min_price real	max_price real	avg_price numeric
1	pcs	34.5	6729	1098.34
2	portateis_casa_forno_e_cafe	10.19	2899	624.29
3	eletrodomesticos_2	13.9	2350	476.12
4	agro_industria_e_comercio	12.99	2990	341.66
5	instrumentos_musicais	4.9	4399.87	281.62
6	eletroportateis	6.5	4799	280.78
7	portateis_cozinha_e_preparadores_de_alimentos	17.42	1099	264.57
8	telefonias_fixas	6	1790	225.69
9	construcao_ferramentas_seguranca	8.9	3099.9	208.99
10	relogios_presentes	8.99	3999.9	200.91
11	climatizacao	10.9	1599	185.27
12	moveis_quarto	6.9	650	183.75
13	pc_gamer	129.99	239	171.77
14	cool_stuff	7	3109.99	167.36
15	moveis_cozinha_area_de_servico_jantar_e_jardim	9.6	1320	164.87
16	moveis_escritorio	25	1189.9	162.01
17	musica	3.85	1165.97	158.80
18	smart	15.5	1460	157.93
19	construcao_ferramentas_construcao	0.85	2300	156.13
20	construcao_ferramentas_ferramentas	6.8	1899	154.41

## Approach:

### 1. Identify Relevant Tables and Columns:

Tables: `amazon_brazil.product`, `amazon_brazil.order_items`

Columns: `product_category_name`, `price`

### 2. Join Tables:

Join `product` and `order_items` tables on `product_id` to associate product categories with their prices.

### 3. Aggregate Data:

Use aggregate functions (`MIN`, `MAX`, `AVG`) to calculate the minimum, maximum, and average prices for each product category.

### 3.Group and Sort Data:

Group by `product_category_name` and order the results in descending order of `avg_price` using **ORDER BY**.

### 4.Retrieve Results:

Output the columns: `product_category_name`, `min_price`, `max_price`, and `avg_price` for insights into price ranges and average prices across categories.

### Recommendations:

#### 1.Capitalize on High-Value Categories:

Product categories like "pcs" and "portateis\_casa\_forno\_e\_cafe," which have the highest average prices, represent premium segments. Focus on enhancing visibility through premium ads, exclusive deals, and loyalty programs to attract high-value customers.

#### 2.Boost Mid-Value Categories:

Categories like "eletrodomesticos\_2" and "agro\_industria\_e\_comercio," which lie in the medium price range, show potential for growth. Introduce targeted promotions, bundle offers, or seasonal discounts to incentivize customers to purchase from these categories.

## Question 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.

### 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
ORDER BY
    total_orders DESC;
```

## Output:

The output contains 1000 rows

	customer_unique_id character varying	total_orders bigint
1	a91e80fbe80ddc07de66a5cf9270293c	16
2	a6168cd79131e64acef92e3c74d6cc43	16
3	363f980585bf04c1a88fdb986011c52e	16
4	cbd0350d4ccba9772e8e768d4a4a5cbf	16
5	417b909c0962b2610f1cfeb1c1478986	16
6	5f94af52aef02c968a2e0f01f430864e	16
7	1b6d29725255a77667a8c639eeb4cc...	16
8	e4bbcc533fdf3917c56dea2c43bf2084	16
9	930c4390af58f67334447c3a1cf2ba36	16
10	5bf4ea2d98005b960eea0dbf652ef4e7	16
11	9159c04b88895d995741dd5b9b7a5f...	16
12	4034aa08d48695a538b7030910aae5...	16
13	c024307523462166b42112cfb6c8e900	16
14	0fdc0d21e1983e8af4d399e17671f76d	16
15	96fd69e8b0df76a9a807b01dc82bef5b	16
16	7f4f709af2fd8fea44aacd30bca46264	16
17	f9c4e8531c2fe4159beb562fd7c2bd59	16
18	3d364a7768fae99678635c4370295d20	16
19	6af40347f5dd7bdd65437a35e1b2fa7b	16
20	f300b00a19af4d4f7bdf9f4524c4587a	16

## Approach:

### 1. Identify Relevant Tables and Columns:

- Tables: `amazon_brazil.customers`, `amazon_brazil.orders`
- Columns: `customer_unique_id` (from `customers`), `customer_id` (from `customers` and `orders`), `order_id` (from `orders`)

## 2.Join Tables:

- Perform a **JOIN** between the **customers** and **orders** tables on the **customer\_id** column to link each customer with their corresponding orders.

## 3.Aggregate Data:

- Use the **COUNT** function to calculate the total number of orders (**total\_orders**) for each customer, grouping the data by **customer\_unique\_id**.

## 4.Filter Data:

- Use the **HAVING** clause to filter customers with more than one order (**COUNT(o.order\_id) > 1**).

## 5.Sort Data:

- Use **ORDER BY total\_orders DESC** to display customers in descending order of the number of orders placed.

## 6.Retrieve Results:

- Output the columns **customer\_unique\_id** and **total\_orders** to provide insights into customers who are most active in placing orders.

## Recommendations:

### 1.Target Loyal Customers:

- Customers with multiple orders are likely loyal. Amazon India can focus on rewarding these customers with personalized discounts, loyalty points, or exclusive offers to enhance customer retention.

### 2.Identify Purchase Trends:

- Analyze the purchase patterns of these customers to understand which categories or products are frequently purchased. Use this insight to optimize inventory or promote similar products.

### 3.Targeted Marketing Campaigns:

- Segment these 1000 customers for special campaigns such as:
  - Early access to sales or new product launches.
  - Festival or seasonal discounts specifically tailored to their purchase history.

## Question 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.

Query:

```
CREATE TEMPORARY TABLE temp_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 t.orders = 1 THEN 'New'
        WHEN t.orders BETWEEN 2 AND 4 THEN 'Returning'
        WHEN t.orders >= 5 THEN 'Loyal'
        ELSE 'New'
    END AS customer_type
FROM
    amazon_brazil.customers c
LEFT JOIN
    temp_customer_orders t
ON c.customer_id = t.customer_id
```

Output: There are 99441 rows as output.

	customer_id [PK] character varying	customer_type text
1	00012a2ce6f8dcda20d059ce98491703	New
2	000161a058600d5901f007fab4c27140	New
3	0001fd6190edaaf884bcaf3d49edf079	New
4	0002414f95344307404f0ace7a26f1d5	New
5	000379cdec625522490c315e70c7a9fb	New
6	0004164d20a9e969af783496f3408652	New

## Approach:

### 1. Temporary Table:

Calculate and store unique order counts (`orders`) for each `customer_id` using `GROUP BY`.

### 2. Aggregation:

Use `COUNT(DISTINCT order_id)` to get the total orders per customer.

### 3. Join Tables:

Perform a `LEFT JOIN` between `customers` and the temporary table to include all customers.

### 4. Categorization:

Use a `CASE` statement to assign customer types (`New`, `Returning`, `Loyal`) based on order count.

### 5. Output:

Display `customer_id` and `customer_type` for customer segmentation.

## Recommendations:

### 1. For New Customers:

- **Welcome Offers:** Provide special discounts or coupons on their next purchase to encourage repeat business.

### 2. For Returning Customers:

- **Loyalty Points:** Introduce a loyalty program where they can earn points on each purchase which can be redeemed for discounts or special products.

### 3. For Loyal Customers:

- **Personalized Services:** Offer personalized services like customized recommendations or special bundle deals based on their shopping habits.



## Question 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.

Query:

```
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
WHERE
    o.order_status = 'delivered'
GROUP BY
    p.product_category_name
ORDER BY
    total_revenue DESC
LIMIT 5;
```

Output:

	product_category_name 	total_revenue 
1	beleza_saude	1.2323242e+06
2	relogios_presentes	1.1642355e+06
3	cama_mesa_banho	1.0187424e+06
4	esporte_lazer	952702.25
5	informatica_acessorios	887388.1

Approach:



## 1.Join Tables:

- Perform joins between `orders_items`, `product`, and `orders` tables using `product_id` and `order_id` to combine relevant data.

## 2.Filter Data:

- Use a `WHERE` clause to include only orders with `order_status = 'delivered'` to ensure that only completed orders are counted.

## 3.Aggregation:

- Use `SUM(oi.price)` to calculate the total revenue for each product category.

## 4.Group By:

- Group the results by `product_category_name` to aggregate the total revenue for each category.

## 5.Order Results:

- Use `ORDER BY total_revenue DESC` to sort the categories by total revenue in descending order.

## 6.Limit Results:

- Use `LIMIT 5` to display the top 5 categories with the highest total revenue.

## Recommendations:

### 1.Boost High Revenue Categories:

Focus on increasing visibility and promotions for top categories like "beleza\_saude" and "relogios\_presentes" to drive more sales.

### 2.Targeted Marketing:

Implement personalized marketing campaigns for each customer segment based on their purchase history in these categories.

### 3.Inventory Management:

Ensure that popular items within these top categories are always in stock to meet customer demand.

## Analysis - III

### Question 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.

Query:

```
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
) seasonal_sales
GROUP BY
    season
ORDER BY
    total_sales DESC;
```

## Output:

	season text	total_sales real
1	Spring	4.216928e+06
2	Summer	4.1205548e+06
3	Winter	2.905667e+06
4	Autumn	2.348754e+06

## Approach:

### 1. Season Classification:

- Use a **CASE** statement to classify orders into seasons (**Spring**, **Summer**, **Autumn**, **Winter**) based on the **order\_purchase\_timestamp** month.

### 2. Join Tables:

- Join the **orders** and **orders\_items** tables on **order\_id** to combine purchase and product-related data.

### 3. Calculate Total Sales:

- Use **oi.price** to extract the sales value for each order item and alias it as **total\_sales**.

### 4. Aggregate Data:

- Within the outer query, sum up the **total\_sales** for each season using **SUM(total\_sales)**.

### 5. Group By Seasons:

- Group the data by **season** to aggregate total sales for each season.

### 6. Order Results:

- Use **ORDER BY total\_sales DESC** to sort the seasons by their total sales in descending order.

## Recommendations:

### 1.Maximize High-Sales Seasons (Spring & Summer):

- Focus on aggressive marketing campaigns and promotions during Spring and Summer.
- Promote season-specific products to leverage high customer demand.

### 2.Boost Low-Sales Seasons (Winter & Autumn):

- Introduce discounts, holiday bundles, and targeted ads to increase sales.
- Engage customers with loyalty programs and pre-launch offers for upcoming seasons.

### 3.Optimize Strategy with Insights:

- Analyze product and demographic trends driving sales in Spring and Summer.
- Align marketing with seasonal events and holidays to enhance customer engagement.

## Question 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.

Query:

```
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
                COUNT(order_item_id) AS total_quantity
            FROM
                amazon_brazil.order_items
```

```

GROUP BY
    product_id
) product_totals
);

```

### Output:

Output contains 6366 rows.

	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
9	ecaaaccb5eb3102553f001d62db6389e	6
10	98ad26989524a790f1d29686025b6fcc	4
11	de4fb1ddae276a3503afed39c8227cff	4
12	9048cbd294fe0c1a3ec8c8248bc2cadd	15

### Approach:

#### 1. Aggregate Product Quantities:

- Use `COUNT(order_item_id)` to calculate the total quantity sold for each product (`product_id`), grouped by `product_id`.

#### 2. Subquery for Average Quantity:

- Use a subquery to calculate the average quantity sold across all products:
  - First, calculate the total quantity sold for each product by grouping on `product_id`.
  - Then, compute the average of these totals using `AVG(total_quantity)`.

### 3.Filter Above-Average Products:

- In the **HAVING** clause, compare each product's total quantity sold (**COUNT(order\_item\_id)**) with the average quantity obtained from the subquery.
- Filter only those products where the total quantity sold is greater than the average.

### 4.Group and Aggregate Data:

- Group the data by **product\_id** to ensure that each product's total sales volume is aggregated.

### 5.Output Results:

- The final output displays **product\_id** and **total\_quantity\_sold** for products whose sales volume exceeds the average.

## Recommendations

### 1.Data-Driven Inventory Management

- Adjust inventory for high-performing products to prevent stockouts.
- Set up dynamic pricing for these items based on demand patterns.

### 2.Focus on Top-Performing Products

- Highlight products with sales above average through targeted marketing campaigns.
- Bundle high-performing products with complementary items to increase the basket size.

## Question 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.

### Query:

```
SELECT  
    EXTRACT(MONTH FROM o.order_purchase_timestamp) AS month,
```

```

    SUM(oi.price) AS total_revenue
FROM
    amazon_brazil.orders AS o
JOIN
    amazon_brazil.order_items AS oi
ON
    o.order_id = oi.order_id
WHERE
    EXTRACT(YEAR FROM o.order_purchase_timestamp) = 2018
GROUP BY
    EXTRACT(MONTH FROM o.order_purchase_timestamp)
ORDER BY
    month;

```

### Output:

	month numeric 🔒	total_revenue real 🔒
1	1	950045.9
2	2	844193
3	3	983225.1
4	4	996660.5
5	5	996529.25
6	6	865134.75
7	7	895518.75
8	8	854697.44
9	9	145

### Approach:

1.Extract Month and Year: Use **EXTRACT(MONTH)** and **EXTRACT(YEAR)** on **order\_purchase\_timestamp** to isolate the relevant time periods.

- Filter for the year 2018 (**WHERE EXTRACT(YEAR) = 2018**).
- Group by month (**GROUP BY EXTRACT(MONTH)**).

### 2.Join Tables:

Perform an inner join between `orders` and `order_items` using `order_id` to combine order-level and item-level data.

### 3. Calculate Revenue:

Use `SUM(oi.price)` to calculate the total revenue for each month.

### 4. Sort the Results:

Order the results by `month` for chronological display.

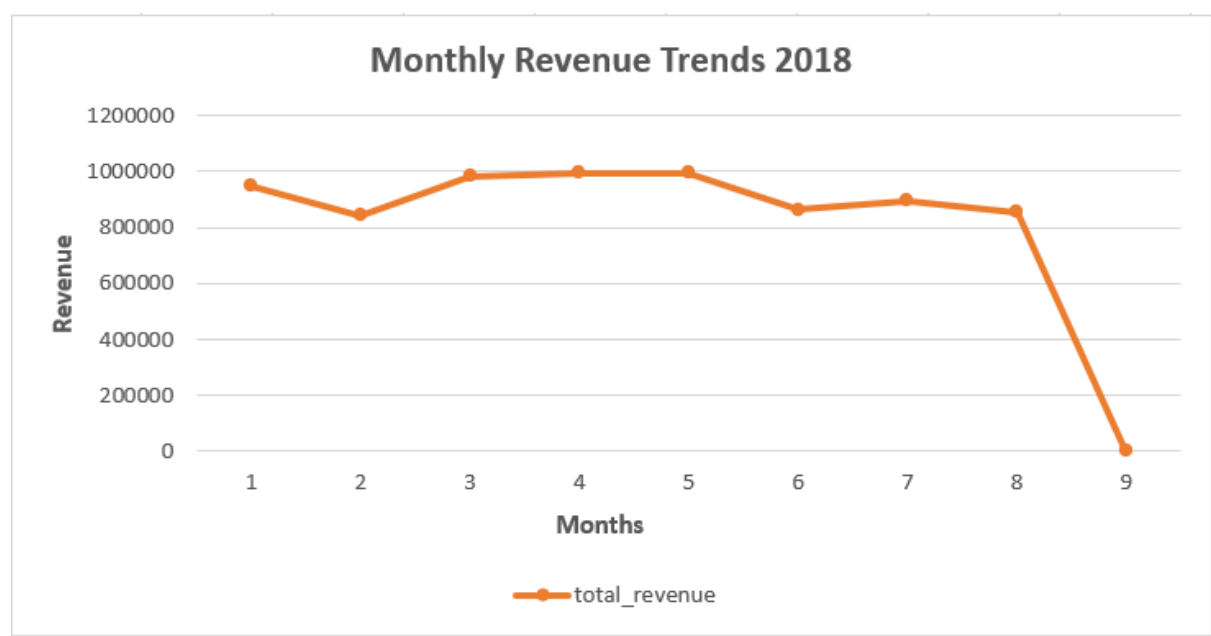
## Recommendations:

### 1. Encourage Transition from Occasional to Regular Customers:

- Since the majority of customers are `Occasional`, create targeted campaigns such as personalized discounts or loyalty points after the second purchase to encourage repeat transactions.
- Provide incentives like free shipping or combo deals for customers completing 3+ orders.

### 2. Retain Loyal Customers and Expand Their Value:

- Offer `Loyal` customers premium perks, such as early access to sales, dedicated customer support, or priority shipping.
- Introduce referral bonuses to encourage them to bring in new customers.





## Question 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.

Query:

```
WITH customer_segments AS (  
    SELECT  
        o.customer_id,  
        COUNT(o.order_id) AS order_count  
    FROM  
        amazon_brazil.orders AS o  
    GROUP BY  
        o.customer_id  
)  
  
customer_classification AS (  
    SELECT  
        CASE  
            WHEN order_count BETWEEN 1 AND 2 THEN 'Occasional'  
            WHEN order_count BETWEEN 3 AND 5 THEN 'Regular'  
            WHEN order_count > 5 THEN 'Loyal'  
        END AS customer_type  
    FROM  
        customer_segments  
)  
  
SELECT  
    customer_type,  
    COUNT(*) AS count  
FROM  
    customer_classification  
GROUP BY  
    customer_type  
ORDER BY  
    count DESC;
```

## Output:

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

## Approach:

### 1. Segment Customers by Order Count:

- Use the `customer_segments` Common Table Expression (CTE) to calculate the total number of orders (`order_count`) for each customer (`customer_id`) by grouping the data from the `orders` table.

### 2. Classify Customers Based on Frequency:

- Use a second CTE (`customer_classification`) to assign each customer to one of three segments (`Occasional`, `Regular`, `Loyal`) based on their `order_count`.
- Use a `CASE` statement to define these classifications:
  - `Occasional`: 1-2 orders.
  - `Regular`: 3-5 orders.
  - `Loyal`: More than 5 orders.

### 3. Aggregate Customer Types:

- In the final `SELECT` statement, group the data by `customer_type` and calculate the number of customers (`COUNT(*)`) in each segment.

### 4. Order Results:

- Sort the output by `count` in descending order to highlight the most dominant customer segments.

### 5. Order Results:

- Sort the output by `count` in descending order to highlight the most dominant customer segments.

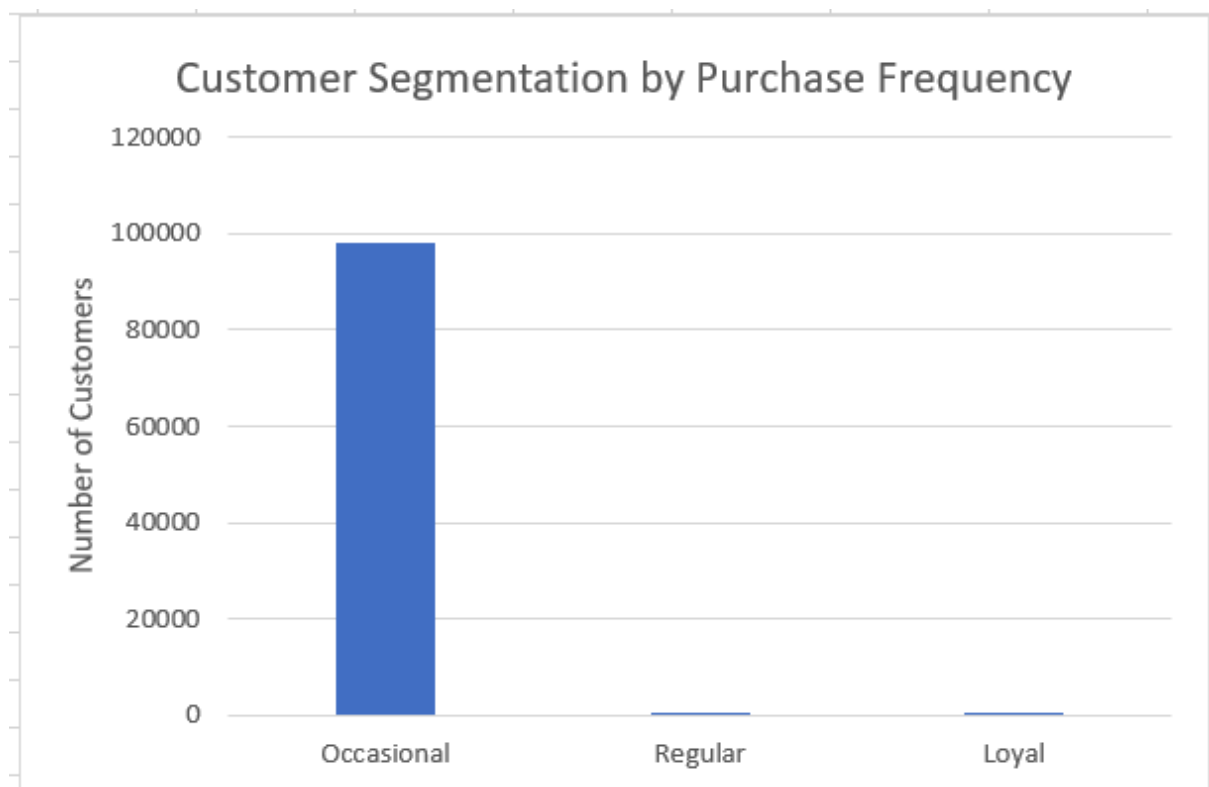
## Recommendations:

### 1.Encourage Transition from Occasional to Regular Customers:

- Since **Occasional** customers form the majority, introduce targeted campaigns to increase engagement:
  - Offer personalized discounts or loyalty points after completing the second purchase.
  - Promote combo deals or free shipping for customers making 3+ orders.

### 2.Retain Loyal Customers and Maximize Their Value:

- Reward **Loyal** customers with premium benefits like early access to sales, priority customer support, and exclusive discounts.
- Launch referral programs to incentivize them to bring in new customers, offering rewards for successful referrals.



## Question 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.

Query:

```
WITH customer_avg_order_value 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  
)  
customer_ranking AS (  
    SELECT  
        customer_id,  
        avg_order_value,  
        RANK() OVER (ORDER BY avg_order_value DESC) AS  
customer_rank  
    FROM  
        customer_avg_order_value  
)  
SELECT  
    customer_id,  
    avg_order_value,  
    customer_rank  
FROM  
    customer_ranking  
WHERE  
    customer_rank <= 20  
ORDER BY  
    customer_rank;
```

## Output:

	customer_id character varying	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	4399.8701171875	7
8	35a413c7ca3c69756cb75867d6311c...	4099.990234375	8
9	e9b0d0eb3015ef1c9ce6cf5b9dcbee9f	4059	9
10	c6695e3b1e48680db36b487419fb03...	3999.89990234375	10
11	926b6a6fb8b6081e00b335edaf578d35	3999	11
12	3be2c536886b2ea4668eced3a80dd0...	3980	12
13	31e83c01fce824d0ff786fcd48dad009	3930	13
14	eb7a157e8da9c488cd4ddc48711f10...	3899	14
15	19b32919fa1198aefc0773ee2e46e693	3700	15
16	66657bf1753d82d0a76f2c4719ab8b85	3699.989990234375	16
17	addc91fdf9c2b3045497b57fc710e820	3690	17
18	39d6658037b1b5a07d0a24d423f0bd...	3549	18
19	e7c905bf4bb13543e8df947af4f3d9e9	3399.989990234375	19
20	46bb3c0b1a65c8399d0363cefbcc4f37	3124	20

## Approach:

### 1. Calculate Average Order Value for Each Customer:

- Use the first CTE (`customer_avg_order_value`) to compute the average value of orders (`AVG(oi.price)` or include freight if needed) for each customer (`customer_id`) by grouping the joined data from the `orders` and `order_items` tables.

### 2. Rank Customers Based on Average Order Value:

- Use the second CTE (`customer_ranking`) to rank customers in descending order of their `avg_order_value` using the `RANK()` function. Handle ties by assigning the same rank to customers with identical values.

### 3.Filter Top 20 Customers:

- In the main query, filter out the top 20 ranked customers using the condition `customer_rank <= 20`.

### 4.Sort the Final Output:

- Use `ORDER BY customer_rank` to display the top 20 customers in ascending rank order.

## Recommendations:

### 1.Target High-Value Customers:

- Focus exclusive rewards and loyalty programs on the identified top 20 customers with the highest average order value to maximize retention and future revenue.

### 2.Incentivize Repeat Purchases:

- Offer additional perks, such as cashback, exclusive discounts, or early access to sales, to encourage repeat purchases among the top 20 customers.

## Question 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.

### Query:

```
WITH RECURSIVE MonthlySales AS (

    SELECT
        product_id,
        DATE_TRUNC('month', MIN(o.order_purchase_timestamp))
    AS sale_month,
```

```

        SUM(oi.price) AS monthly_sales,
        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 product_id

    UNION ALL

    SELECT
        ms.product_id,
        ms.sale_month + INTERVAL '1 month' AS sale_month,
        COALESCE(next_month.monthly_sales, 0) AS
monthly_sales,
        ms.total_sales + COALESCE(next_month.monthly_sales,
0) AS total_sales
    FROM MonthlySales ms
    LEFT JOIN (
        SELECT
            product_id,
            DATE_TRUNC('month', o.order_purchase_timestamp)
AS sale_month,
            SUM(oi.price) AS monthly_sales
        FROM amazon_brazil.Orders o
        JOIN amazon_brazil.Order_Items oi ON o.order_id =
oi.order_id
        GROUP BY product_id, DATE_TRUNC('month',
o.order_purchase_timestamp)
    ) next_month
    ON ms.product_id = next_month.product_id
    AND next_month.sale_month = ms.sale_month + INTERVAL
'1 month'
    WHERE ms.sale_month < (SELECT MAX(DATE_TRUNC('month',
order_purchase_timestamp)) FROM amazon_brazil.Orders)
)
    SELECT
        product_id,
        TO_CHAR(sale_month, 'YYYY-MM') AS sale_month, -- Format

```

```

the output as 'YYYY-MM'
    monthly_sales,
    total_sales
FROM MonthlySales
ORDER BY product_id, sale_month;

```

**Output:**

**Output Contain 374782 rows.**

	product_id character varying	sale_month text	monthly_sales real	total_sales real
1	00066f42aeeb9f3007548bb9d3f33c38	2018-05	101.65	101.65
2	00066f42aeeb9f3007548bb9d3f33c38	2018-06	0	101.65
3	00066f42aeeb9f3007548bb9d3f33c38	2018-07	0	101.65
4	00066f42aeeb9f3007548bb9d3f33c38	2018-08	0	101.65
5	00066f42aeeb9f3007548bb9d3f33c38	2018-09	0	101.65
6	00066f42aeeb9f3007548bb9d3f33c38	2018-10	0	101.65
7	00088930e925c41fd95ebfe695fd2655	2017-12	129.9	129.9
8	00088930e925c41fd95ebfe695fd2655	2018-01	0	129.9
9	00088930e925c41fd95ebfe695fd2655	2018-02	0	129.9
10	00088930e925c41fd95ebfe695fd2655	2018-03	0	129.9
11	00088930e925c41fd95ebfe695fd2655	2018-04	0	129.9
12	00088930e925c41fd95ebfe695fd2655	2018-05	0	129.9
13	00088930e925c41fd95ebfe695fd2655	2018-06	0	129.9
14	00088930e925c41fd95ebfe695fd2655	2018-07	0	129.9
15	00088930e925c41fd95ebfe695fd2655	2018-08	0	129.9
16	00088930e925c41fd95ebfe695fd2655	2018-09	0	129.9
17	00088930e925c41fd95ebfe695fd2655	2018-10	0	129.9
18	0009406fd7479715e4bef61dd91f2462	2017-12	229	229
19	0009406fd7479715e4bef61dd91f2462	2018-01	0	229
20	0009406fd7479715e4bef61dd91f2462	2018-02	0	229
21	0009406fd7479715e4bef61dd91f2462	2018-03	0	229

**Approach:**

**1.Calculate First Month's Sales for Each Product:**



- Use the first part of the recursive CTE (**MonthlySales**) to calculate the first sale month for each product (**MIN(o.order\_purchase\_timestamp)**), and sum the sales for that month to initialize the **total\_sales**.

## **2.Add Monthly Sales Recursively:**

- Use the recursive part of the CTE to increment the **sale\_month** by one month at a time and add the sales for each subsequent month to the cumulative **total\_sales**. Handle months with no sales by setting **monthly\_sales = 0**.

## **3.Ensure Complete Lifecycle:**

- In the main query, ensure that all months from the product's first sale to the last recorded sale month are covered. This includes months where there are no sales.

## **4.Sort and Output Results:**

- Format the output to show **product\_id**, **sale\_month** (formatted as **YYYY-MM**), and cumulative **total\_sales**. Ensure the results are ordered by **product\_id** and **sale\_month**.

## **Recommendations:**

### **1. Focus on High-Performing Products:**

- Prioritize restocking and advertising for products with consistent sales growth.
- Ensure these products are featured prominently in promotions and ads.

### **2. Address Low/Zero Sales Products:**

- Investigate customer feedback to understand why sales are low.
- Consider improving product visibility or running flash sales to generate interest.

### **4. Optimize Inventory:**

- Track inventory levels to avoid both overstock and stockouts.
- Plan stock levels based on seasonal trends and upcoming product launches.

## Question 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 query to first calculate total monthly sales for each payment method, then compute the percentage change from the previous month.

Query:

```
WITH monthly_sales AS (  
  
    SELECT  
        p.payment_type,  
        DATE_TRUNC('month', o.order_purchase_timestamp) AS  
sale_month,  
        SUM(p.payment_value) AS monthly_total  
    FROM  
        amazon_brazil.orders o  
    JOIN  
        amazon_brazil.payments p  
    ON  
        o.order_id = p.order_id  
    WHERE  
        DATE_PART('year', o.order_purchase_timestamp) = 2018  
    GROUP BY  
        p.payment_type, DATE_TRUNC('month',  
o.order_purchase_timestamp)  
),  
monthly_growth AS (  
  
    SELECT  
        payment_type,  
        sale_month,  
        monthly_total,  
  
        LAG(monthly_total) OVER (PARTITION BY payment_type ORDER  
BY sale_month) AS previous_month_total,  
  
        ROUND(  
            (monthly_total - LAG(monthly_total) OVER (PARTITION BY
```

```
payment_type ORDER BY sale_month))  
        / NULLIF(LAG(monthly_total) OVER (PARTITION BY  
payment_type ORDER BY sale_month), 0) * 100,  
        2  
    ) AS monthly_change  
FROM  
    monthly_sales  
)  
  
SELECT  
    payment_type,  
    sale_month,  
    monthly_total,  
    COALESCE(monthly_change, 0) AS monthly_change  
FROM  
    monthly_growth  
ORDER BY  
    payment_type, sale_month;
```

## Output:

	payment_type character varying 🔒	sale_month timestamp without time zone 🔒	monthly_total numeric 🔒	monthly_change numeric 🔒
1	boleto	2018-01-01 00:00:00	204844.66	0
2	boleto	2018-02-01 00:00:00	183112.72	-10.61
3	boleto	2018-03-01 00:00:00	191538.02	4.60
4	boleto	2018-04-01 00:00:00	193547.09	1.05
5	boleto	2018-05-01 00:00:00	195378.93	0.95
6	boleto	2018-06-01 00:00:00	153350.28	-21.51
7	boleto	2018-07-01 00:00:00	198041.24	29.14
8	boleto	2018-08-01 00:00:00	143805.90	-27.39
9	credit_card	2018-01-01 00:00:00	868880.38	0
10	credit_card	2018-02-01 00:00:00	778803.00	-10.37
11	credit_card	2018-03-01 00:00:00	933770.10	19.90
12	credit_card	2018-04-01 00:00:00	934306.00	0.06
13	credit_card	2018-05-01 00:00:00	927556.35	-0.72
14	credit_card	2018-06-01 00:00:00	811508.56	-12.51
15	credit_card	2018-07-01 00:00:00	803674.49	-0.97
16	credit_card	2018-08-01 00:00:00	797648.89	-0.75
17	debit_card	2018-01-01 00:00:00	11543.55	0
18	debit_card	2018-02-01 00:00:00	7469.53	-35.29

## Approach:

### 1. Calculate Monthly Sales:

- Use the first CTE (`monthly_sales`) to compute the total sales (`SUM(p.payment_value)`) for each payment method (`payment_type`) by month (`sale_month`), filtering for the year 2018.

### 2. Compute Monthly Growth:

- Use the second CTE (`monthly_growth`) to calculate the month-over-month growth rate. Use the `LAG` function to get the previous month's total sales and calculate the percentage change.

### 3. Handle Null Values:

- In the final query, use **COALESCE** to handle null values for the **monthly\_change** column, ensuring that the first month has a growth rate of 0%.

#### **4.Output the Results:**

- Select the payment method, sale month, total monthly sales, and the month-over-month percentage change in the final query.

#### **5.Sort the Final Output:**

- Order the results by **payment\_type** and **sale\_month** to present the data chronologically and by payment method.

### **Recommendations:**

#### **1.Optimize Popular Payment Methods:**

- Enhance the checkout experience for **credit\_card** and **boleto** users.
- Offer targeted promotions like cashback and discounts.

#### **2.Boost Debit Card Usage:**

- Simplify the transaction process and provide incentives for debit card users.
- Address any transaction issues causing fluctuations.

#### **3.Promote Vouchers:**

- Run consistent promotions to maintain interest in vouchers.
- Use vouchers in loyalty programs and bundled offers.