

AMAZON BRAZIL'S DATA ANALYSIS FOR INDIAN MARKET

Padmini Purandare
October 06, 2024

Table of Contents

PROBLEM STATEMENT	2
ANALYSIS I	2
1. Standardizing Payment Values	2
2. Distribution of orders by payment type	3
3. Identifying Specific Price Range Products with "smart" in Their Name	4
4. Most Successful Months by Sales	5
5. Categories with Significant Price Variations	6
6. Most Consistent Transaction Amounts By Payment Types	7
7. Products with Incomplete or Missing Category Names	8
ANALYSIS II	10
1. Most Popular Payment Type By Order Value Segment	10
2. Price Range and Average Price for Each Product Category	12
3. Identifying Customers with Multiple Orders	13
4. Categorizing Customers Based on Purchase History	14
5. Product Categories that Generate the Most Revenue	15
ANALYSIS III	16
1. Comparing Total Sales Between Different Seasons	16
2. Identifying Products with Sales Volumes Above the Average	18
3. Analyzing Monthly Revenue Trends in 2018	20
4. Customer Segmentation Based on Purchase Frequency	22
5. Ranking High-Value Customers for Rewards Program	24
6. Calculating Monthly Cumulative Sales for Each Product	26
7. Analyzing Monthly Sales Growth by Payment Method	28

PROBLEM STATEMENT

Amazon India wants to analyze Amazon Brazil's data to identify trends, customer behaviors, and preferences to make informed decisions, enhance customer experience, and seize new opportunities in the Indian market.

ANALYSIS I

1. Standardizing Payment Values

Task:

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.

Approach:

- Group by payment type column
- Take the rounded average of payment values from the 'payments' table
- Finally, order the results in ascending order of the rounded values

SQL Query:

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

Result:

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

Recommendation:

- Focus on improving payment methods with higher average values, like credit card payments, by offering more cashback or rewards programs in the Indian market.

2. Distribution of orders by payment type

Task:

Calculate the percentage of total orders for each payment type, rounded to one decimal place, and display them in descending order.



Approach:

- Group by payment type column to calculate total orders by payment type
- Use the count() function to calculate the total orders for each payment type from the payments table.
- Calculate the percentage by dividing each payment type's count by the total number of orders and multiplying by 100.
- Round the result to one decimal place using the round() function and display the result in descending order.

SQL Query:

```
select
    payment_type,
    round(count(payment_type)*100/sum(count(payment_type))over(),1)as percentage_orders
from payments
group by payment_type
order by percentage_orders desc;
```

Result:

	payment_type 	percentage_orders 
	character varying	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:

Since credit card usage is dominant in India, similar trends can be promoted by offering credit card users exclusive discounts or EMI options.

3. Identifying Specific Price Range Products with "smart" in Their Name

Task:

Identify all products priced between 100 and 500 BRL with the word 'smart' in their name. Display these products, sorted by price in descending order.

Approach:

- Join the 'order_items' and 'product' tables using the join function on a common column product_id
- Use a where clause to filter products within the price range of 100 to 500 BRL with the word 'smart'
- Use wildcard character '%' to search for product names containing the word 'Smart'.
- Order the results by price in descending order

SQL Query:

```
select
    distinct oi.product_id,
    oi.price
from order_items oi
join product p
on oi.product_id=p.product_id
where
    p.product_category_name like '%smart%'
    and
    oi.price between 100 and 500
order by oi.price desc;
```

Result:

	product_id character varying	price numeric
1	1df1a2df8ad2b9d3aa49fd851e3145ad	440
2	7debe59b10825e89c1cbcc8b190c85e2	350
3	ca86b9fe16e12de698c955aedff0aea2	349
4	0e52955ca8143bd179b311cc454a6caa	335
5	7aeaa8f3e592e380c420e8910a717255	329.9

Recommendation:

Promote mid-range smart devices in India by highlighting their affordability and features.

4. Most Successful Months by Sales

Task:

Determine the top 3 months with the highest total sales value, rounded to the nearest integer.

Approach:

- Use the `to_char()` function to extract the month from the `order_purchase_timestamp` in the `orders` table.
- Join the `orders` and `order_items` tables on `order_id` to calculate monthly sales.
- Group by the month and order the results by total sales in descending order. Limit the output to the top 3 months.

SQL Query:

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

Result:

	month text	total_sales numeric
1	May	1502629
2	August	1428696
3	July	1393578

Recommendation:

Amazon should focus on major sales campaigns during these months. Offering time-limited deals or exclusive discounts during these months can help replicate the sales peaks observed in Brazil.

5. Categories with Significant Price Variations

Task:

Find categories where the difference between the maximum and minimum product prices exceeds 500 BRL.

Approach:

- Join the 'order_items' and 'product' tables using the join function on a common column product_id
- Use max() and min() to calculate the price range for each product category from the product table
- Subtract the minimum price from the maximum price and filter for categories where the price difference exceeds 500 BR
- Group by product category and order by price difference

SQL Query:

```
select
    p.product_category_name,
    max(oi.price)-min(oi.price) as price_difference
from order_items oi
join 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;
```

Result:

	product_category_name character varying	price_difference numeric
1	utilidades_domesticas	6731.9
2	pcs	6694.5
3	artes	6495.5
4	eletroportateis	4792.5
5	instrumentos_musicais	4395.0
6	consoles_games	4094.8
7	esporte_lazer	4054.5

Recommendation:

Offer more detailed filtering and comparison tools in the Indian market to help customers find products within their price range in high-variation categories.

6. Most Consistent Transaction Amounts By Payment Types

Task:

Identify the payment types with the least variance in transaction amounts, sorting by the smallest standard deviation first.



Approach:

- Use the payments table to calculate the standard deviation of the payment_value for each payment_type.
- Group the results by payment_type and apply the stddev() function to compute the standard deviation for each type.
- Order the results by the standard deviation in ascending order to find the most consistent payment methods.

SQL Query:

```
select
    payment_type,
    round(stddev(payment_value),2) as std_deviation
from payments
group by payment_type
order by std_deviation asc;
```

Result:

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

Recommendation:

Payment types with low variance could be promoted more heavily in India to enhance user trust and convenience.

7. Products with Incomplete or Missing Category Names

Task:

Retrieve the list of products where the product category name is missing or contains only a single character.

Approach:

- Use the product table to locate records where the product_category_name is either NULL or contains a single character.
- Use the where clause and wildcard character '_' to check that category_name contains a single character and the 'is null' condition to identify missing entries.

SQL Query:

```
select
    product_id,
    product_category_name
from product
where
    product_category_name is null or
    product_category_name like '_'
order by product_category_name, product_id;
```


Result:

	product_id character varying	product_category_name character varying
1	ce6f74096c84567f22728c84f3d6e7fc	c
2	c7fce98e1aa3d8a6cbaaebc7ab99cb...	f
3	afbe1e973aefbf72a330e3bc72d4b4...	t
4	3f13f4fabd1eafe564af941a8ee8e279	w
5	0082684bb4a60a862baaf7a60a584...	[null]
6	00ab8a8b9fe219511dc3f178c6d796...	[null]

Recommendation:

Amazon India should prioritize cleaning and updating these product categories to improve searchability and user experience.

ANALYSIS II

1. Most Popular Payment Type By Order Value Segment

Task:

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

Approach:

- Use the 'payment' table to create order value segments using the CASE statement to categorize orders into "low", "medium", and "high" based on the total order value.
- Group the results by order_value_segment and payment_type, and count the number of occurrences for each payment type in each segment.
- Order the results by count in descending order.

SQL Query:

```
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(payment_type) as count
from payments
group by
    order_value_segment,
    payment_type
order by count desc;
```

Result:

	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

Recommendation:

- Amazon India can tailor its payment methods to customer preferences within each segment.
- Credit card payments are more popular for high-value orders, Amazon can prioritize this method for premium services.
- Different payment methods are dominant for low-value orders, so they could be used for promotions or smaller purchases.

2. Price Range and Average Price for Each Product Category

Task:

Calculate the minimum, maximum, and average price for each product category and list them in descending order by average price.

Approach:

- Use the 'product' and 'order_item' table by joining them on product_id.
- Calculate the minimum (MIN()), maximum (MAX()), and average (AVG()) price for each category using the window function and partition them by product_category
- Order the results by the average price in descending order.

SQL Query:

```
select
    distinct p.product_category_name,
    min(oi.price)over(partition by p.product_category_name)as min_price,
    max(oi.price)over(partition by p.product_category_name)as max_price,
    round(avg(oi.price)over(partition by p.product_category_name),2)as avg_price
from order_items oi
join product p
on oi.product_id=p.product_id
order by avg_price desc;
```

Result:

	product_category_name character varying	min_price numeric	max_price numeric	avg_price numeric
1	pcs	34.5	6729	1098.34
2	portateis_casa_forno_e_cafe	10.2	2899	624.29
3	eletrodomesticos_2	13.9	2350	476.13
4	agro_industria_e_comercio	13	2990	341.66
5	instrumentos_musicais	4.9	4399.9	281.62

Recommendation:

Amazon India can use this data to strategize product pricing by focusing on high-value categories for premium promotions. For low-priced categories, Amazon can introduce budget-friendly marketing campaigns.

3. Identifying Customers with Multiple Orders

Task:

Find all customers with more than one order, displaying their unique customer IDs along with the total number of orders.

Approach:

- Join the 'customer' and 'orders' tables together to count how many orders each customer _unique_id has placed.
- Group by customer_unique_id and use the having clause to filter customers with more than one order.

SQL Query:

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

Result:

	customer_unique_id character varying	total_orders bigint
1	a91e80fbe80ddc07de66a5cf9270293c	16
2	a6168cd79131e64acef92e3c74d6cc43	16
3	363f980585bf04c1a88fdb986011c52e	16
4	cbd0350d4ccba9772e8e768d4a4a5cbf	16
5	417b909c0962b2610f1cfef1c1478986	16
6	5f94af52aef02c968a2e0f01f430864e	16

Recommendation:

Amazon India can focus on retaining these customers by offering loyalty rewards, personalized recommendations, or exclusive deals.

4. Categorizing Customers Based on Purchase History

Task:

Categorize customers into different types ('New – order qty. = 1' ; 'Returning' –order qty. 2 to 4; 'Loyal' – order qty. >4) based on their purchase history.

Approach:

- Use the 'orders' table to create customer_type segments using the CASE statement to categorize customers into "New", "Returning", and "Loyal" based on the total orders.
- Group the results by customer_id.

SQL Query:

```
select
    customer_id,
    case
        when count(customer_id)=1 then 'New'
        when count(customer_id) between 2 and 4 then 'Returning'
        else 'loyal'
    end as customer_type
from orders o
group by customer_id;
```

Result:

	customer_id character varying	customer_type text
1	09033cfedb9bab5c54a33f339fd94ad0	New
2	9f184cd2aa748ce963a8d5e075aff28e	New
3	1e11025f8a103ddb95ddaca105f7f4a	New
4	f796cc9e19019673220efa160fe7e610	New
5	68d03ff74911622915ef4ec24e2919a9	New

Recommendation:

Amazon India can use this segmentation to target customers more effectively: new customers with introductory offers, returning customers with loyalty perks, and loyal customers with premium services or rewards programs.

5. Product Categories that Generate the Most Revenue

Task:

Use joins between the tables to calculate the total revenue for each product category. Display the top 5 categories.



Approach:

- Use products, and order_items, tables to calculate total revenue by product category.
- Perform a JOIN between the tables, summing up product price.
- Group by product_category_name and order by total revenue in descending order, limiting the results to the top 5 categories.

SQL Query:

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

Result:

	product_category character varying 	total_revenue numeric 
1	beleza_saude	1257907.6
2	relogios_presentes	1203067.6
3	cama_mesa_banho	1032303.4
4	esporte_lazer	985898.3
5	informatica_acessorios	910625.9

Recommendation:

Amazon India can focus on promoting these high-revenue product categories through targeted marketing campaigns.

ANALYSIS III

1. Comparing Total Sales Between Different Seasons

Task:

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 Autumn is between September and November and the rest months are Winter.



Approach:

- Use the orders and order_items tables to extract the order_purchase_timestamp and compute the total sales for each season.
- Use the extract() function to derive the month from the order date.
- Assign each month to the appropriate season (Spring, Summer, Autumn, Winter) using conditional logic (Case).
- Sum the total sales for each season and group by season.

SQL Query:

```
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,
    sum(oi.price) as total_sales
from
    orders o
join
    order_items oi
    on oi.order_id = o.order_id
group by
    season
order by
    total_sales desc;
```


Result:

	season text 	total_sales numeric 
1	Spring	4216841.2
2	Summer	4120471.4
3	Winter	2905839.5
4	Autumn	2348871.6

Recommendation:

Promotions and inventory should focus on products that resonate with spring and summer demands (such as travel, fashion, and cooling products).

2. Identifying Products with Sales Volumes Above the Average

Task:

Write a query that uses a subquery to filter products with a total quantity sold above the average quantity.



Approach:

- Use a CTE (Common Table Expression) to find product quantity by product_id.
- Taking Average of total_quantity_sold using subquery inside where clause.
- Order the result by total_quantity_sold.

SQL Query:

```
----- CTE 1 -----  
with pro_quantity as(  
    select  
        product_id,  
        count(product_id) as total_quantity_sold  
    from order_items  
    group by product_id  
)  
----- MAIN QUERY -----  
select  
    product_id,  
    total_quantity_sold  
from  
    pro_quantity  
where  
    total_quantity_sold >  
        (select  
            round(avg(total_quantity_sold)) as overall_average  
        from pro_quantity)  
order by  
    total_quantity_sold desc;
```

Result:

	product_id character varying 	total_quantity_sold bigint 
1	aca2eb7d00ea1a7b8ebd4e68314663af	527
2	99a4788cb24856965c36a24e339b60...	488
3	422879e10f46682990de24d770e7f83d	484
4	389d119b48cf3043d311335e499d9c...	392
5	368c6c730842d78016ad823897a372...	388

Recommendation:

Focus marketing efforts on these high-performing products. Additionally, optimize stock levels for these products to prevent stockouts and ensure availability during peak sales periods.

3. Analyzing Monthly Revenue Trends in 2018

Task:

Calculate total revenue generated each month in 2018 and visualize revenue trends.

Approach:

- Use the orders and order_items tables to compute monthly revenue.
- Extract the month from order_purchase_timestamp and filter data to include only orders placed in 2018.
- Sum the price for each month.
- Group by month

SQL Query:

```
select
    to_char(order_purchase_timestamp, 'Month') as month,
    sum(price) as total_revenue
from orders
join order_items
    on orders.order_id = order_items.order_id
where
    extract(year from order_purchase_timestamp) = 2018
group by
    month,
    extract(month from order_purchase_timestamp)
order by
    extract(month from order_purchase_timestamp) asc;
```

Result:

	month text	total_revenue numeric
1	January	950059.4
2	February	844204.5
3	March	983244.2
4	April	996674.8
5	May	996542.4
6	June	865147.6
7	July	895533.7
8	August	854710.1
9	September	145

Recommendation:

Amazon India can use these insights to forecast future revenue trends and plan marketing and inventory strategies accordingly.

4. Customer Segmentation Based on Purchase Frequency

Task:

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.

Approach:

- Use a CTE (Common Table Expression) to count no. of orders placed by the customers by grouping customer_id from orders table.
- Make segment of no. of orders by using case-when.
- Count the number of customers in each segment by grouping customer_type.

SQL Query:

```
----- CTE 1 -----
with no_of_orders as (
    select
        customer_id,
        count(customer_id) as no_of_orders
    from
        orders
    group by customer_id)

----- MAIN QUERY -----
select
    case
        when no_of_orders in (1,2) then 'Occasional'
        when no_of_orders in (3,4,5) then 'Regular'
        else 'Loyal'
    end as customer_type,
    count(*) as count
from
    no_of_orders
group by
    customer_type
```

Result:

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

Recommendation:

Amazon India can tailor its loyalty programs to these customer segments, offering special incentives for "Loyal" customers and targeted promotions to convert "Occasional" customers into more frequent buyers.

5. Ranking High-Value Customers for Rewards Program

Task:

Rank customers based on their average order value and identify the top 20.

Approach:

- Calculate the average order value for each customer using the orders and order_items table.
- Rank the customers based on their average order value using rank() window function with order by avg_order_value in the descending order and retrieve the top 20 customers.

SQL Query:

```
select
    customer_id,
    round(avg(price)) as avg_order_value,
    rank() over (order by avg(price) desc) as customer_rank
from
    orders o
join
    order_items oi on o.order_id = oi.order_id
group by
    customer_id
order by
    avg_order_value desc
limit 20;
```


Result:

	customer_id character varying	avg_order_value numeric	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	4400	7
8	35a413c7ca3c69756cb75867d6311c...	4100	8
9	e9b0d0eb3015ef1c9ce6cf5b9dcbee9f	4059	9
10	c6695e3b1e48680db36b487419fb03...	4000	10

Recommendation:

These top customers should be targeted with exclusive offers and rewards to maintain their loyalty and encourage further high-value purchases.

6. Calculating Monthly Cumulative Sales for Each Product

Task:

Calculate monthly cumulative sales for each product from the date of its first sale.

Approach:

- Use CTE to calculate monthly sales
- In the main query use the sum() window function to calculate cumulative sales by taking the data from CTE.
- Order the result by product_id and sale_month

SQL Query:

```
----- CTE 1 -----  
with m_sales as (  
  select  
    oi.product_id,  
    TO_CHAR(o.order_purchase_timestamp, 'yyyy-mm') as sale_month,  
    sum(oi.price) as monthly_sales  
  from orders o  
  join order_items oi  
  on o.order_id=oi.order_id  
  group by  
    sale_month,  
    product_id  
)  
----- MAIN QUERY -----  
select  
  product_id,  
  sale_month,  
  monthly_sales,  
  sum(monthly_sales) over(partition by product_id order by sale_month) as total_sales  
from m_sales  
order by product_id,sale_month;
```

Result:

	product_id character varying	sale_month text	monthly_sales numeric	total_sales numeric
13	001b72dfd63e9833e8c02742adf472e3	2017-03	35	140
14	001b72dfd63e9833e8c02742adf472e3	2017-07	105	245
15	001b72dfd63e9833e8c02742adf472e3	2017-08	70	315
16	001b72dfd63e9833e8c02742adf472e3	2017-09	70	385
17	001b72dfd63e9833e8c02742adf472e3	2017-11	35	420
18	001b72dfd63e9833e8c02742adf472e3	2017-12	70	490
19	001c5d71ac6ad696d22315953758fa04	2017-01	79.9	79.9
20	00210e41887c2a8ef9f791ebc780cc36	2017-05	33	33
21	00210e41887c2a8ef9f791ebc780cc36	2017-06	201	234

Recommendation:

This information can help Amazon India track the growth and lifecycle of key products, allowing for better forecasting and inventory management.

7. Analyzing Monthly Sales Growth by Payment Method

Task:

Compute total monthly sales for each payment method and calculate the month-over-month growth rate.





Approach:

- Using two CTEs, one will calculate monthly sales using orders, order_item, and payments tables and another one will be used to calculate previous month sales.
- Calculate the growth rate using these two CTEs

SQL Query:

```
----- CTE 1 -----
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 orders o
join order_items oi on o.order_id = oi.order_id
join payments p on o.order_id = p.order_id
where extract(year from order_purchase_timestamp) = 2018
group by
    payment_type,
    sale_month),
----- CTE 2 -----
previous_mon_sales as(
select
    payment_type,
    sale_month,
    monthly_total,
    lag(monthly_total) over (partition by payment_type order by sale_month) as previous_value
from monthly_sales)
----- MAIN QUERY -----
select
    payment_type,
    sale_month,
    monthly_total,
    round((monthly_total-previous_value)* 100/previous_value,2) as monthly_change
from previous_mon_sales;
```

Result:

	payment_type character varying 	sale_month text 	monthly_total numeric 	monthly_change numeric 
1	boleto	2018-01	170658.2	[null]
2	boleto	2018-02	153170.7	-10.25
3	boleto	2018-03	157813.7	3.03
4	boleto	2018-04	162946.8	3.25
5	boleto	2018-05	166576.6	2.23
6	boleto	2018-06	126383.5	-24.13
7	boleto	2018-07	162943.5	28.93
8	boleto	2018-08	118218.3	-27.45
9	credit_card	2018-01	760273.9	[null]

Recommendation:

Amazon India can optimize its payment methods and promotions accordingly.