Advanced Data Analysis using SQL

Amazon Brazil's Data Analysis using SQL

Insights & Recommendations for Indian Market

amazon

By: ***Bhushan Borse***

NextLeap Data Analyst Fellowship C3

Table of Contents

1 Introduction

1.1 Leveraging Data-Driven Insights from Amazon Brazil's Data

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.

Hence, Amazon Brazil's data need to be analysed to identify trends and customer behaviours that can be leveraged in the Indian market. This analysis will help Amazon India make informed decisions, enhance customer experience, and seize new opportunities.

1.2 Objective

The objective of this analysis is to understand and analyse:

- Customer demographics and behaviours using the Amazon Brazil's customers table to understand purchase patterns and preferences.
- Evaluate regional trends and customer density through the Geolocation table.
- Track order lifecycles, product preferences, and seller performance using the Orders, Order Items, Product, and Seller tables.
- Additionally, analyse payment preferences and transaction details via the Payments table.

This comprehensive analysis will help Amazon India enhance customer experience and seize new market opportunities.

1.3 Data Source

The schema consists of 7 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 behaviour.
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.

1.4 Data Preparation

Data is imported into PostgreSQL using CSV files and organized into a dedicated schema. Schema of the provided data, including relationship and primary keys for each table is shown in below figure Fig 1.31. Data has been taken from publicly available forums.

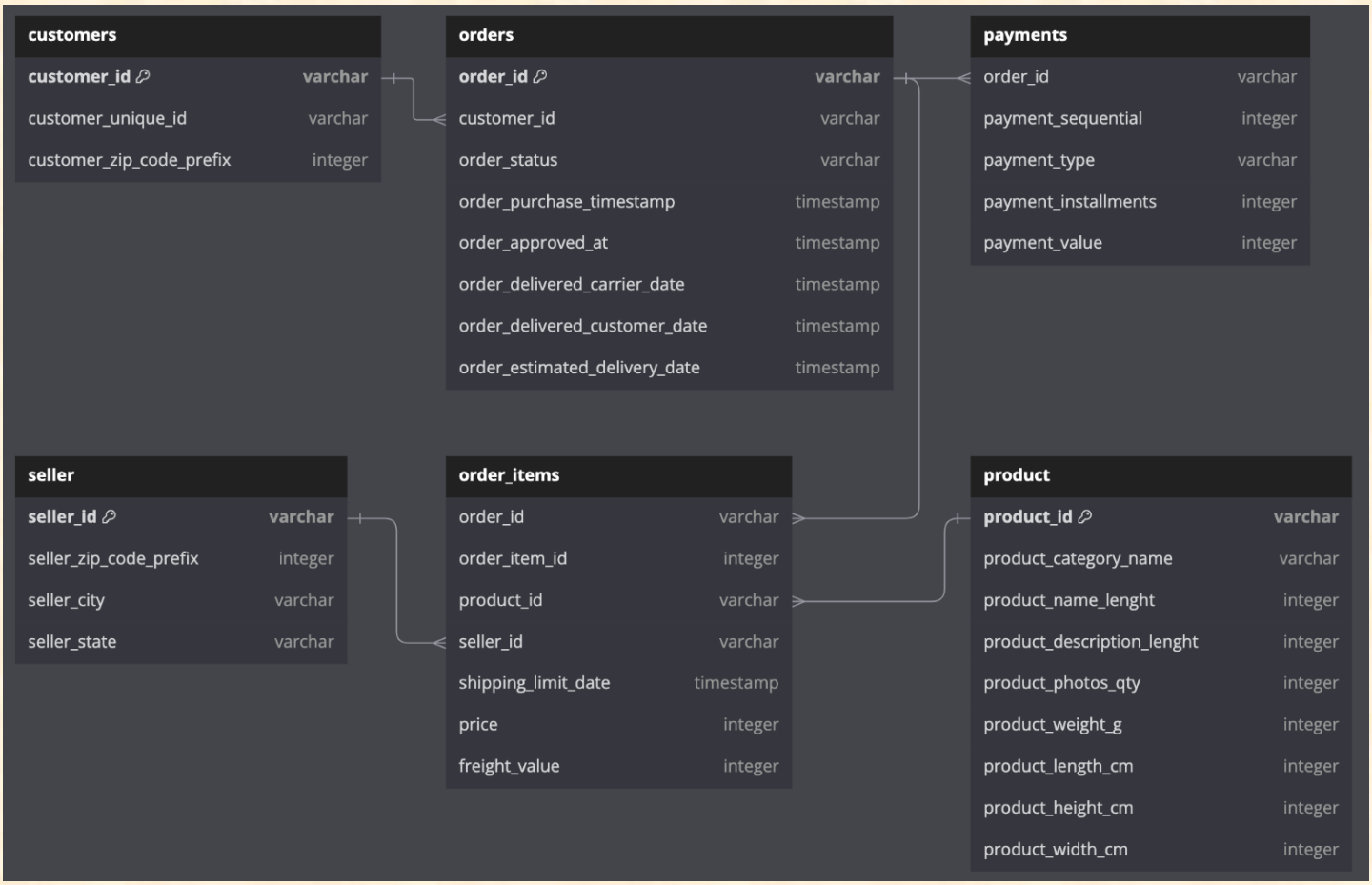


Fig 1.31 Schema

The analysis employs fundamental SQL clause, Aggregate functions, Joins, Window functions and CTEs to extract insights from the given data.

The analysis is divided into **3 parts covering different topics with a total of 19 practical business questions**, each related to a real scenario. Finding answers to this business questions by analysing the customer data can help Amazon India make informed decisions. From identifying high-potential products categories and high value customers to understanding the effective payment types and seasonal sales, each query is crafted to deliver strategic insights.

In summary, this analysis helps in transforming raw relational data into actionable insight using SQL, helping Amazon India enhance customer experience and seize new market opportunities.

2.1 Analysis I –

Question 2.11 –

Problem Statement:

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

Desired Output: payment_type, rounded_avg_payment

Approach:

1. Identifying Relevant Tables and Columns:
   - Table: payments
   - Columns: payment_type, payment_value
2. Calculating Average Payment Value:
   - Use the AVG() aggregate function on payment_value to compute average payment
   - Use GROUP BY to group the results by payment_type to get the average payment for each payment type
3. Rounding the Averages:
   - Use ROUND() function to round the average payment values to the nearest integer.
4. Sorting the Results:
   - Order the final results in ascending order based on the rounded average payment.

SQL Query:

```
select
    payment_type,
    -- average payment value rounded to closest integer
    round(avg(payment_value)) as rounded_avg_payment
from
    amazon_brazil.payments
group by
    payment_type
order by
    rounded_avg_payment desc;
```

Query 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

Recommendations:

1. Focus on Popular Payment Methods:
   - Make payment methods like credit_card and boleto efficient, as their average payment amounts are higher.
   - Introduce targeted promotions for customers using these methods to make higher revenue.
2. Improve the use of Underperforming Methods:
   - For methods like debit_card, open opportunities for growth by giving promotional discounts or improving transaction processes.

Question 2.12 –

Problem Statement:

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

Desired Output: payment_type, percentage_orders

Approach:

1. Calculating Total Number of Orders for different payment types:
   - Determine the total count of all orders across payment types.
   - Use COUNT() to get the number of orders for each payment_type.
2. Calculating Percentage of Orders:
   - Divide the count of orders per payment type by the total number of orders
   - Multiply the ratio by 100 to get the percentage
   - Round the percentage output to one decimal place.
3. Sorting Results:
   - Order the results in descending order of percentage to highlight the most popular payment methods.

SQL Query:

```
select
    payment_type,
    -- computing count of order_id for each payment_type,
    -- dividing with total no. of payments and rounding to find percentage
    round(count(order_id)*100*1.0/
    -- computing total number of payments
    (select count(*) from amazon_brazil.payments),1) as percentage_orders
from
    amazon_brazil.payments
group by
    payment_type
order by
    percentage_orders desc;
```

Query 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

Recommendations:

1. Maintain efficiency of Popular Payment Methods:
   - Since credit_card is highly popular payment method, ensure that this payment method is highly reliable, secure, and easy to use.
   - Consider cashback or reward or loyalty programs for credit card users to incentivize more purchases.
2. Improve and encourage use of Lesser Popular Payment Methods:
   - Explore ways to make boleto, voucher, and debit_card more attractive by offering discounts or simplifying the payment process.
   - Also, investigate reasons why there is low usage of certain methods and address any barriers.

Question 2.13 –

Problem Statement:

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

Desired Output: product_id, price

Approach:

1. Identifying Relevant Tables and Columns:
   - Table: product, order_items
   - Columns: product_id, price
2. Performing joins to connect both the tables on basis of common column
   - Join the tables on basis of a common column product_id
3. Applying filter using condition
   - Use BETWEEN to get products priced between 100 and 500
   - Use LIKE operator to find product_category_name containing word 'smart' in it
4. Sorting the result by price in descending order using ORDER BY.

SQL Query:

```sql
select
    p.product_id,
    oi.price
from
    amazon_brazil.product p
join  -- joining product and order_items tables
    amazon_brazil.order_items oi
on
    p.product_id = oi.product_id
where
    -- for products priced between 100 and 500
    oi.price between 100 and 500
and
    -- for products containing word 'Smart' in their name
    p.product_category_name like '%smart%'
order by
    oi.price desc;
```

Query 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
9	7aeaa8f3e592e380c420e8910a717...	329.9
10	7aeaa8f3e592e380c420e8910a717...	329.9
11	7aeaa8f3e592e380c420e8910a717...	329.9
12	d1b571cd58267d8cac8b2afd6e288...	299.9
13	d1b571cd58267d8cac8b2afd6e288...	299.9
14	66ffe28d0fd53808d0535eee4b90a1...	254
15	f06796447de379a26dde5fcac6a1a2...	239.9
16	d3d5a1d52abe9a7d234908d873fc3...	229.9
17	06ae026e430189633c2fbd0288c86...	217.36
18	49ef750dc5bf23e3788d4f614bc6db...	198
19	33bb7da523efcdef6cd2996cbf72d0...	148
20	33bb7da523efcdef6cd2996cbf72d0...	148
21	6f5795735ab2c629b22669fe889b7...	129.9
22	3626035966a7aaee90d68108caebd...	124.9
23	aeaba104830f91586dae1bff90f54a8a	123.9
24	630c84b1ce83ae0e9ddc05a141039...	110
25	3168b2696b15ca440b92afa9e011a...	109.9
26	dbd55362ec13c706503b1c71a5068...	102
27	dbd55362ec13c706503b1c71a5068...	102
28	dbd55362ec13c706503b1c71a5068...	102
29	dbd55362ec13c706503b1c71a5068...	102
30	dbd55362ec13c706503b1c71a5068...	102
31	dbd55362ec13c706503b1c71a5068...	102
32	dbd55362ec13c706503b1c71a5068...	102
33	dbd55362ec13c706503b1c71a5068...	102
34	aeaba104830f91586dae1bff90f54a8a	100

Recommendations:

1. Segment & Personalize Ads
   - Use customer purchase history to target the customers which are interested in smart products
2. Optimize Keyword-Based Ads
   - Enhance Ads on internet and social media sites by targeting searches related to smart products.

Question 2.14 –

Problem Statement:

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

Desired Output: month, total_sales

Approach:

1. Identifying Relevant Tables and Columns:
   - Table: orders, order_items
   - Columns: order_delivered_customer_date, price
2. Performing join between identified tables
   - Join the tables on basis of a common column order_id
3. Extract month from date
   - Use TO_CHAR to extract month from date
4. Do SUM aggregation to compute total_sales
   - Use SUM() aggregate function to calculate total sales

5. Using WHERE clause to filter out orders with order_status as ‘delivered’
6. Performing grouping
   - Use GROUP BY to group the result by month
7. Find top 3 sales month
   - Use ORDER BY and LIMIT to obtain top 3 months with the highest total sales value

SQL Query:

```sql
select
    -- extracting month from date
    TO_CHAR(o.order_delivered_customer_date, 'Month') as month,
    -- caculating sum of price to get total sales
    round(SUM(oi.price)) as total_sales
from
    amazon_brazil.order_items oi
join -- joining order_items and orders tables
    amazon_brazil.orders o
on
    oi.order_id = o.order_id
where
    -- filtering orders which have been delivered
    o.order_status = 'delivered'
group by
    month
order by
    total_sales desc
limit 3;
```

Output:

	month text	total_sales numeric
1	August	1677321
2	May	1528743
3	June	1434012

Recommendations:

1. Ensure optimal services during peak months
   - Focus on reducing delivery timeline to improve customer satisfaction
   - Ensure high-demand products are well-stocked for the peak months.
2. Introduce discounts/offers during non-peak months
   - Provide discounts/offers during non-performing months to boost sales

---

Question 2.15 –

Problem Statement:

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

Desired Output: product_category_name, price_difference

Approach:

1. Identifying Relevant Tables and Columns:
   - Table: product, order_items
   - Columns: product_category_name, price
2. Perform join between tables
   - Join the tables on basis of a common column ie. product_id
3. Compute price_difference
   - Use MAX and MIN functions to calculate maximum and minimum price and take difference
4. Perform grouping
   - User GROUP BY to group the result by product_category_name
5. Applying filter after grouping
   - Use HAVING to apply required filter after grouping is done

SQL Query:

```
select
    p.product_category_name,
    -- computing difference between max and min price
    max(oi.price) - min(oi.price) as price_difference
from
    amazon_brazil.order_items oi
join -- joining product and order_items tables
    amazon_brazil.product p
on
    oi.product_id = p.product_id
group by
    p.product_category_name
having
    -- filtering only the product category with difference > 500
    max(oi.price) - min(oi.price) > 500
order by
    price_difference desc;
```

Query 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
12	informatica_acessorios	3696.09
13	beleza_saude	3122.8
14	cool_stuff	3102.99
15	construcao_ferramentas_segur...	3091.0
16	industria_comercio_e_negocios	3061.1
17	agro_industria_e_comercio	2977.01
18	portateis_casa_forno_e_cafe	2888.81
19	pet_shop	2495.1
20	eletronicos	2466.51

	product_category_name character varying	price_difference numeric
21	telefonia	2423
22	eletrodomesticos_2	2336.1
23	construcao_ferramentas_constr...	2299.15
24	automotivo	2254.51
25	eletrodomesticos	2083.81
26	cama_mesa_banho	1992.99
27	market_place	1954.01
28	moveis_decoracao	1894.1
29	construcao_ferramentas_ferra...	1892.2
30	telefonia_fixa	1784
31	brinquedos	1695.09
32	fashion_bolsas_e_acessorios	1693.99
33	papelaria	1690.71
34	climatizacao	1588.1
35	smart	1444.5
36	dvds_blu_ray	1411.1
37	construcao_ferramentas_jardim	1341.08
38	moveis_cozinha_area_de_servic...	1310.4
39	construcao_ferramentas_ilumin...	1277.49
40	malas_acessorios	1184.57

	product_category_name character varying	price_difference numeric
38	moveis_cozinha_area_de_servic...	1310.4
39	construcao_ferramentas_ilumin...	1277.49
40	malas_acessorios	1184.57
41	moveis_escritorio	1164.9
42	musica	1162.12
43	casa_construcao	1089.02
44	portateis_cozinha_e_preparado...	1081.58
45	livros_interesse_geral	893.9
46	tablets_impressao_imagem	875.09
47	cine_foto	867.19
48	moveis_sala	826.09
49	casa_conforto	792.01
50	sinalizacao_e_seguranca	735.5
51	livros_importados	730.01
52	alimentos_bebidas	693.4
53	perfumaria	684.91
54	moveis_quarto	643.1
55	bebidas	617
56	audio	584.09
57	artigos_de_festas	563.21

Recommendations:

1. Price Segmentation
   - Categorize products into budget, mid-range, and premium to target right set of audience.
   - Highlight affordability for lower-priced items to attract price-sensitive buyers.
2. Advertising & Targeting
   - Offer seasonal discounts on expensive items while maintaining profitability.

Question 2.16 –

Problem Statement:

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

Desired Output: payment_type, std_deviation

Approach:

1. Identifying Relevant Tables and Columns:
   - Table: payments
   - Columns: payment_type, payment_value
2. Computing standard deviation for payment value for each payment type
   - Use STDDEV() function to calculate std_deviation for payment_value
3. Performing grouping
   - Use GROUP BY to group the result by payment_type
4. Ordering the result by std_deviation
   - Use ORDER BY to order the result in ascending order of std_deviation values

SQL Query:

```sql
select
    payment_type,
    -- computing standard deviation for payment value
    round(stddev(payment_value),2) as std_deviation
from
    amazon_brazil.payments
group by
    -- grouping by payment type
    payment_type
order by
    std_deviation;
```

Output:

	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

Recommendations:

1. Promote Reliable and Consistent Payment Methods
   - Highlight the most consistent payment options like voucher, boleto, credit card in this case during the checkout process.
   - Offer exclusive discounts for using these methods to encourage adoption.
2. Enhance Customer Support for Payment related Issues
   - Enable 24/7 support for the most-used payment types.

Question 2.17 –

Problem Statement:

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

Desired Output: product_id, product_category_name

Approach:

1. Identifying Relevant Tables and Columns:
   - Table: product
   - Columns: product_id, product_category_name
2. Fetching required columns
   - Fetch product_id and product_category_name columns as required
3. Applying required filtering using WHERE clause
   - Use LIKE operator to filter out product_category_name that has only single character
   - Use IS NULL to filter out product_category_name that are null
4. Combining result of both queries
   - Use UNION ALL to combine the output from both the queries

SQL Query:

```
select
    product_id,
    product_category_name
from
    amazon_brazil.product
where
    -- product category name with only single character
    product_category_name like '_'

union all -- combining result of both the queries

select
    product_id,
    product_category_name
from
    amazon_brazil.product
where
    -- product category name which is null
    product_category_name is null;
```

Output: (only first 10 displayed here)

	product_id character varying	product_category_name character varying
1	c7fce98e1aa3d8a6cbaaebc7ab99cbe6	f
2	3f13f4fabd1eafe564af941a8ee8e279	w
3	afbe1e973aefbf72a330e3bc72d4b476	t
4	ce6f74096c84567f22728c84f3d6e7fc	c
5	a41e356c76fab66334f36de622ecbd3a	[null]
6	d8dee61c2034d6d075997acef1870e...	[null]
7	56139431d72cd51f19eb9f7dae4d1617	[null]
8	46b48281eb6d663ced748f324108c7...	[null]
9	5fb61f482620cb672f5e586bb132eae9	[null]
10	e10758160da97891c2fdcbc35f0f031d	[null]

Recommendations:

1. Identify product_id and corresponding product_category_name that needs to be fixed.
   - Apply data validation rules to prevent incomplete entries during product listing.

2.2 Analysis II –

Question 2.21 –

Problem Statement:

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

Desired Output: order_value_segment, payment_type, count

Approach:

1. Identifying Relevant Tables and Columns:
   - Table: payments
   - Columns: payment_type, payment_value
2. Segmenting Payment Value:
   - Use CASE to segment payment values into 3 segments (low, medium, high) on the basis of provided condition.
3. Computing the count for each payment type:
   - Use COUNT() function to calculate the count of each payment type within each order value segment (low, medium, high).
4. Sorting the Results:
   - Order the final result by payment type and descending order of count.

SQL Query:

```sql
SELECT
    CASE
        -- segmenting order value as low, medium, high
        WHEN payment_value < 200 THEN 'low'
        WHEN payment_value BETWEEN 200 AND 1000 THEN 'medium'
        WHEN payment_value > 1000 THEN 'high'
    END AS order_value_segment,
    payment_type,
    --  computing count of each payment type within each order value segment
    COUNT(*) as count
FROM
    amazon_brazil.payments
GROUP BY
    order_value_segment, payment_type
order by
    payment_type, count(*) desc ;
```

Query Output: (only first 10 displayed here)

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

Recommendations:

1. Make Popular Payment Methods secure and more reliable:
   - Make popular payment methods like credit card and boleto secure and reliable, as their count for low and medium order value segments is significantly greater compared to other payment types.
   - Created targeted ads for customers using these payment types to generate higher revenue.
2. Improve the use of Underperforming Methods:
   - For payment type like debit card and voucher, open opportunities for growth by giving promotional discounts or improving transaction processes.

Question 2.22 –

Problem Statement:

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

Desired Output: product_category_name, min_price, max_price, avg_price

Approach:

1. Identifying Relevant Tables and Columns:
   - Table: order_items, product
   - Columns: product_category_name, price
2. Computing minimum, maxium and average values for price:
   - Use MIN(), MAX() and AVG() aggregate function to do the required aggregation over price values.
3. Joining tables using common column:
   - Join order_items and product tables on a common column product_id.
4. Grouping the result:
   - Use GROUP BY to group the results by product_category_name to get the minimun, maximum and average price values for each product_category.
5. Sorting the Results:
   - Order the final result by descending order of computed average prices.

SQL Query:

```sql
select
    p.product_category_name,
    -- computing minimum price for product category
    min(oi.price) as min_price,
    -- computing maximum price for product category
    max(oi.price) as max_price,
    -- computing average price for product category
    round(avg(oi.price),2) as avg_price
from
    amazon_brazil.order_items oi
join -- joining order_items and product tables
    amazon_brazil.product p
on
    oi.product_id = p.product_id
group by
    p.product_category_name
order by
    avg_price desc;
```

Query Output: (only first 20 displayed here)

	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	10.19	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_prepara...	17.42	1099	264.57
8	telefonia_fixa	6	1790	225.69
9	construcao_ferramentas_seg...	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_ser...	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_co...	0.85	2300	156.13
20	construcao_ferramentas_ferr...	6.8	1899	154.41

Recommendations:

1. Plan Price based Product Segmentation:
   - Identify premium categories and target high-spending customer with personalized offers.
   - Promote affordable categories with discounts for budget-conscious customers.
2. Inventory Management on basis of high average pricing and demand
   - Ensure adequate stock of popular high-average-price categories.
   - Plan seasonal restocking based on demand fluctuations.

Question 2.23 –

Problem Statement:

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

Desired Output: customer_unique_id, total_orders

Approach:

1. Identifying Relevant Tables and Columns:
   - Table: orders, customers
   - Columns: customer_unique_id, customer_id
2. Calculating count of order per customer
   - Use COUNT() function to count the total number of orders for each customer.

3. Joining tables using common column:
   - Join orders and customers tables on a common column customer_id.
4. Grouping the result:
   - Use GROUP BY to group result by customer_id to get the count of total orders for each customer.
5. Filtering the result
   - Use HAVING clause to filter the customer ids having more than 1 order.
6. Sorting the Results:
   - Order the final result by descending order of total orders.

SQL Query:

```
select
    c.customer_unique_id,
    -- counting total number of orders for each customer
    count(o.customer_id) as total_orders
from
    amazon_brazil.orders o
join -- joining orders and customers tables
    amazon_brazil.customers c
on
    o.customer_id  = c.customer_id
group by
    c.customer_unique_id
having
    -- filtering only customers having more than 1 order
    count(o.customer_id) > 1
order by
    total_orders desc;
```

Query Output: (only first 10 displayed here)

	customer_unique_id character varying	total_orders bigint
1	a91e80fbe80ddc07de66a5cf9270293c	16
2	a6168cd79131e64acef92e3c74d6cc43	16
3	363f980585bf04c1a88fdb986011c52e	16
4	cbd0350d4ccba9772e8e768d4a4a5c…	16
5	417b909c0962b2610f1cfeb1c1478986	16
6	5f94af52aef02c968a2e0f01f430864e	16
7	1b6d29725255a77667a8c639eeb4cc…	16
8	e4bbcc533fdf3917c56dea2c43bf2084	16
9	930c4390af58f67334447c3a1cf2ba36	16
10	5bf4ea2d98005b960eea0dbf652ef4e7	16

Recommendations:

1. Implement Loyalty & Reward Programs
   - Offer exclusive discounts or cashback to repeat customers.
   - Introduce membership benefits based on order frequency.
2. Personalized Promotions & Targeted Ads
   - Use past purchase data to create personalized product recommendations.
   - Send customized discount codes to encourage further purchases.

Question 2.24 –

Problem Statement:

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

Desired Output: customer_id, customer_type

Approach:

1. Identifying Relevant Tables and Columns:
   - Table: orders, customers
   - Columns: customer_id
2. Creating a temporary table using CTE
   - Use CTE to generate a temporary table containing customer id and count of total number of orders for each customer.
3. Segregating customers on the basis of order quantity
   - Use CASE to perform segregation of customers on the basis of order quantity
4. Joining tables using common column:
   - Join customers and generated temporary table, temp_table on a common column customer_id.
5. Sorting the Results:
   - Order the final result by descending order of customer type.

SQL Query:

```sql
with -- creating a temporary table using CTE
temp_table as
    (
    select
        customer_id,
        -- counting the total number of orders for each customer id
        count(*) as order_count
    from
        amazon_brazil.orders
    group by
        customer_id
    )
select
    c.customer_id,
    case -- segregation of customer_id on basis of order quantity
        when t.order_count = 1 then 'New'
        when t.order_count between 2 and 4 then 'Returning'
        when t.order_count > 4 then 'Loyal'
        end as customer_type
from
    amazon_brazil.customers c
join --joining customer and temp_table on common column customer_id
    temp_table t
on
    c.customer_id = t.customer_id
order by
    customer_type desc;
```

Query Output: (only first 10 displayed here)

	customer_id [PK] character varying	customer_type text
1	7203eb38fea2b4a15dd6c3957531...	Returning
2	314675b44a38443e4ba09ab0f462...	Returning
3	a6f0724c88ff8742ef397186be22c7...	Returning
4	d7314e9d240737da330e0f83ac4b...	Returning
5	6ee311afc0adb5663ad53868d46ba...	Returning
6	d1c6e90ab127d42b75ef2e548b44...	Returning
7	9ce1401cf82bd5420fe255e02d629...	Returning
8	6c5aeca0152670c75e44c9c67eb7...	Returning
9	690d9e177e7d2a54b8d4ad4c08d2...	Returning
10	08999a20d9741d9513c368676f55...	Returning

Recommendations:

1. Personalized Marketing for Each Customer Type
   - New Customers: Offer first-purchase discounts to encourage repeat orders.
   - Returning Customers: Introduce personalized promotions based on past purchases.
   - Loyal Customers: Provide VIP benefits, such as early access to sales or premium support.
2. Re-engagement Campaigns for Inactive Customers
   - Identify customers who haven't ordered recently and send reactivation offers.
   - Use personalized email reminders with recommendations based on past purchases.

Question 2.25 –

Problem Statement:

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

Desired Output: product_category_name, total_revenue

Approach:

1. Identifying Relevant Tables and Columns:
   - Table: order_items, product
   - Columns: product_category_name, price
2. Calculating total revenue for each product category
   - Use SUM to calculate total revenue for each product category
3. Joining tables using common column:
   - Join order_items and product on a common column product_id.

4. Sorting and Limiting the output:
    - User ORDER BY to order the final result by descending order of total revenue and return only the top 5 products by total revenue by using LIMIT.

SQL Query:

```sql
select
    p.product_category_name,
    -- computing total revenue for each product category
    sum(oi.price) as total_revenue
from
    amazon_brazil.order_items oi
join -- joining order_items and product on common column product_id
    amazon_brazil.product p
on
    oi.product_id = p.product_id
group by
    p.product_category_name
order by
    total_revenue desc
limit 5; -- displaying just first five
```

Query 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

Recommendations:

1. Maximize Marketing & Advertising Efforts
    - Increase ad spend for high-performing categories using targeted promotions.
    - Implement seasonal and event-based campaigns to drive demand.
2. Optimize Inventory & Supply Chain
    - Ensure adequate stock availability for top-selling products to prevent shortages.
3. Improve Pricing & Discount Strategies
    - Offer exclusive discounts on best-selling products to maintain engagement.

2.3 Analysis III –

Question 2.31 –

Problem Statement:

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

Desired Output: season, total_sales

Approach:

1. Identifying Relevant Tables and Columns:
   - Table: orders, order_items
   - Columns: order_id, order_purchase_timestamp, price
2. Creating a temporary table using CTE
   - Use CTE to generate temporary table containing order id and season in which the order was made.
3. Aggregating price for each season
   - Use SUM() to perform aggregation of price to calculate total sales for each season
4. Joining tables using common column:
   - Join order_items and generated temporary table, temp_table on a common column order_id.
5. Grouping the Result:
   - Group the final result by season to compute total sales for each season.

SQL Query:

```
with  -- creating a temporary table using CTE to determine the season for each order
    temp_table as
    (
    select
        order_id,
        case
        when cast(to_char(order_purchase_timestamp, 'mm') as integer) between 3 and 5 then 'Spring'
        when cast(to_char(order_purchase_timestamp, 'mm') as integer) between 6 and 8 then 'Summer'
        when cast(to_char(order_purchase_timestamp, 'mm') as integer) between 9 and 11 then 'Autum'
        else 'Winter' end as season
    from
        amazon_brazil.orders)

select
    t.season,
    -- sum aggregation of price to calculate total sales for each season
    sum(oi.price) as total_sales
from
    amazon_brazil.order_items oi
join -- joining order_items and temp_table on common column order_id
    temp_table t
on
    oi.order_id = t.order_id
group by
    t.season;
```

Query Output:

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

Recommendations:

1. Seasonal Promotions & Discounts
   - Identify the season with highest sales and introduce special promotions for that period.
   - If sales are low in specific seasons, offer limited-time discounts to boost revenue.
   - Use historical purchasing data to target customers likely to buy in specific seasons.
2. Inventory and Demand Management
   - Stock the best-selling seasonal items ahead of peak months to prevent shortages.

Question 2.32 –

Problem Statement:

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

Desired Output: product_id, total_quantity_sold

Approach:

1. Identifying Relevant Tables and Columns:
   - Table: order_items
   - Columns: product_id
2. Calculating quantity sold for each product
   - Use COUNT() to calculate total quantity of each product sold
3. Grouping the Result:
   - Group the result by product_id to compute total quantity for each product.
4. Filtering the output
   - Use HAVING clause to filter only those products which have total quantity above overall average

SQL Query:

```
select
    product_id,
    -- calculating total quantity sold of each product
    count(product_id) as total_quantity_sold
from
    amazon_brazil.order_items
group by
    product_id
having
    -- filtering only products with quantity sold above overall average
    count(product_id) >
    (select
        round(count(*)*1.0/count(distinct product_id),2)
    from
        amazon_brazil.order_items)
order by
    total_quantity_sold desc;
```

Query Output: (only first 7 displayed here)

	product_id character varying	total_quantity_sold bigint
1	aca2eb7d00ea1a7b8ebd4e68314663...	527
2	99a4788cb24856965c36a24e339b60...	488
3	422879e10f46682990de24d770e7f8...	484
4	389d119b48cf3043d311335e499d9c...	392
5	368c6c730842d78016ad823897a37...	388
6	53759a2ecddad2bb87a079a1f1519f...	373
7	d1c427060a0f73f6b889a5c7c61f2ac4	343

Recommendations:

1. Optimize Inventory & Stock Replenishment
   - Ensure fast-moving products remain in stock to prevent shortages.
   - Improve supply chain management for quick restocking.
2. Boost Marketing & Advertising for High Selling Products
   - Increase targeted promotions for high-selling products to increase revenue.
   - Offer limited time discounts on low selling products to improve sales

---

Question 2.33 –

Problem Statement:

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

Desired Output: month, total_revenue

Approach:

1. Identifying Relevant Tables and Columns:
   - Table: orders, order_items
   - Columns: order_delivered_customer_date, price, order_status
2. Extracting month from date column
   - Use TO_CHAR() function to extract month from order_delivered_customer_date
3. Aggregating price for each month
   - Use SUM() to perform aggregation of price to calculate total revenue for each month
4. Joining tables using common column:
   - Join order_items and orders tables on a common column order_id.
5. Filtering records:
   - Consider only those records where order status is delivered and year for delivery date is 2018.
6. Grouping the Result:
   - Group the final result by month to compute total revenue for each month.

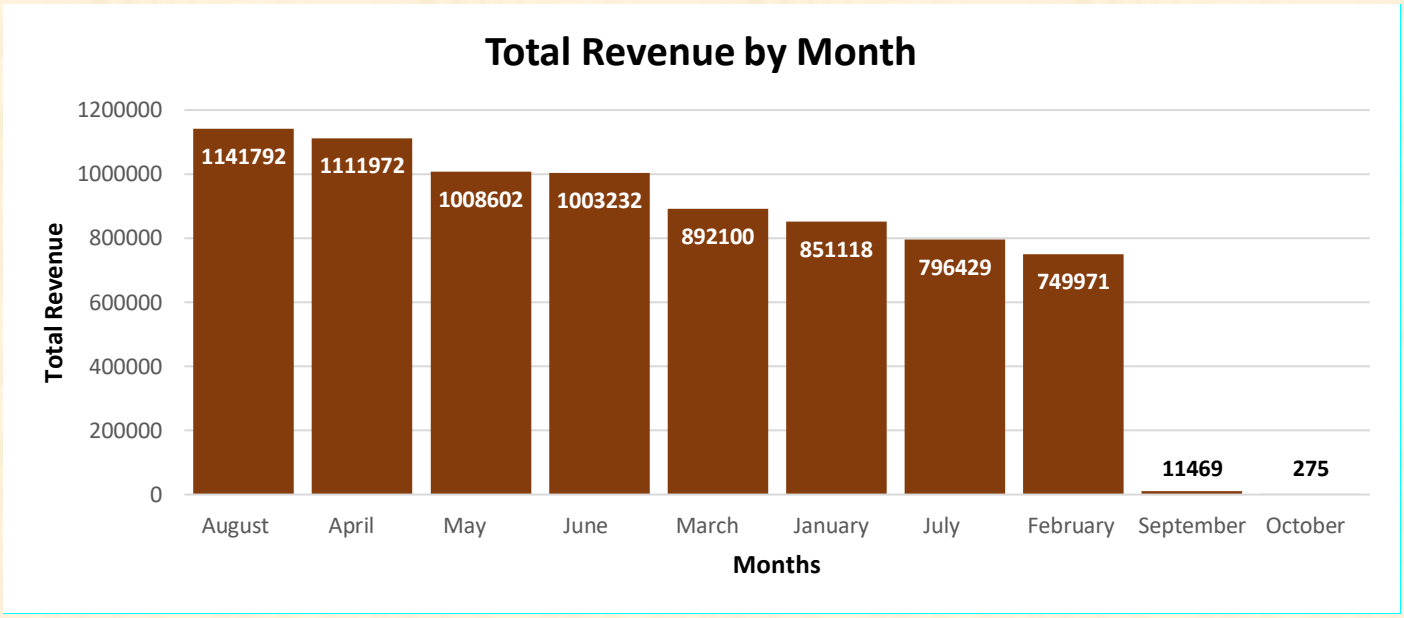
SQL Query:

```
select
    -- extracting month from order delivered customer date
    TO_CHAR(o.order_delivered_customer_date, 'Month') as month,
    -- computing total revenue using SUM () aggregate function
    round(SUM(oi.price)) as total_revenue
from
    amazon_brazil.order_items oi
join -- joining order_items and orders tables on common column order id
    amazon_brazil.orders o
on
    oi.order_id = o.order_id
where --filtering records with delivered status and year as 2018
    o.order_status = 'delivered'
    and
    TO_CHAR(o.order_delivered_customer_date, 'yyyy') = '2018'
group by
    month
order by
    total_revenue desc;
```

Query Output:

	month text	total_revenue numeric
1	August	1141792
2	April	1111972
3	May	1008602
4	June	1003232
5	March	892100
6	January	851118
7	July	796429
8	February	749971
9	September	11469
10	October	275

Visual Representation:



Recommendations:

1. Improve Customer Engagement Based on Purchase Trends
   - Use seasonal personalization for e.g. recommend winter essentials during colder months, etc
   - Offer loyalty rewards for frequent buyers in peak shopping periods.
2. Offer Seasonal Discounts & Bundling
   - Introduce price drops or bundle offers during slower months to boost revenue.
   - Example: Winter deals on electronics or summer bundles for travel essentials.

Question 2.34 –

Problem Statement:

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

Desired Output: customer_type, count

Approach:

1. Identifying Relevant Tables and Columns:
   - Table: orders
   - Columns: customer_id
2. Creating a temporary table using CTE
   - Use CTE to generate a temporary table containing customer id and segregation for each customer into respective segment on basis of purchase frequency.
3. Segregating customers

- Use CASE to perform the segregation on the basis of purchase frequency

4. Sorting the Results:
   - Order the final result by descending order of customer count.

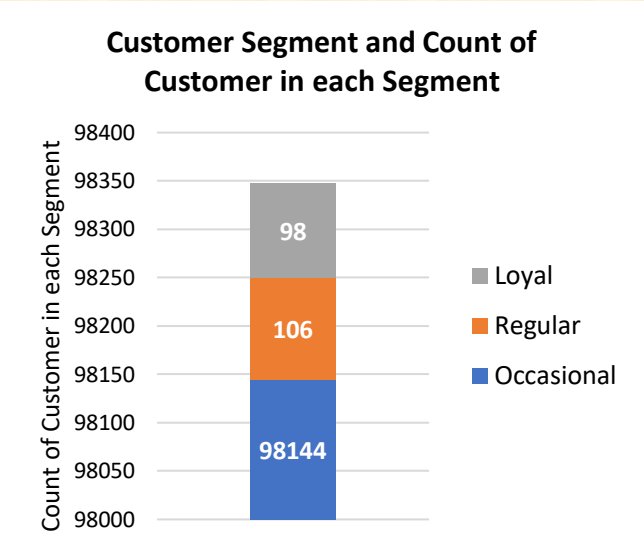
SQL Query:

```sql
with -- creating a temporary table to include customer id and
    -- assigning group to customers on basis of number of orders
    temp_table as
    (select
        customer_id,
        -- assigning groups to customers using CASE
        case
        when count(customer_id) between 1 and 2 then 'Occasional'
        when count(customer_id) between 3 and 5 then 'Regular'
        when count(customer_id) > 5 then 'Loyal' end as customer_type
    from
        amazon_brazil.orders
    group by
        customer_id)
select
    t.customer_type,
    -- computing number of customers present in each customer type
    count(*) as "count"
from
    temp_table t
group by
    t.customer_type
order by
    "count" desc;
```

Query Output:

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

Visual Representation:



Recommendations:

1. Tailored Loyalty Benefits for Each Segment
   Occasional Customers (1-2 orders)
   - Offer first-time purchase discounts to encourage repeat orders.
   - Use personalized email reminders with product recommendations.

   Regular Customers (3-5 orders)
   - Introduce exclusive promotions based on their purchase history.
   - Provide limited-time discounts to boost repeat sales.

   Loyal Customers (>5 orders)
   - Reward with VIP perks like early access to sales, priority shipping, or cashback offers.
2. Improve Customer Support & Engagement
   - Loyal customers get priority service & faster issue resolution.

Question 2.35 –

Problem Statement:

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

Desired Output: customer_id, avg_order_value, and customer_rank

Approach:

1. Identifying Relevant Tables and Columns:
   - Table: orders, order_items
   - Columns: customer_id, price, order_id

2. Aggregating price for each customer
   - Use AVG() to perform aggregation of price to calculate average order value for each customer
3. Ranking customer on average order value
   - Use DENSE_RANK() window function to rank the customers on basis of descending order of their respective average order values
4. Joining tables using common column:
   - Join order_items and orders tables on a common column order_id.
5. Grouping the Result:
   - Group the final result by customer_id to compute average order value for each customer.
6. Fetching Top 20 records:
   - Order the result by descending average order value and then use LIMIT 20 to fetch top 20 records.

SQL Query:

```sql
select
    o.customer_id,
    -- calculating average order value for each customer
    round(avg(oi.price),2) as avg_order_value,
    -- ranking each customer on the basis of average order value
    dense_rank() over (order by round(avg(oi.price),2) desc) as customer_rank
from
    amazon_brazil.order_items oi
join -- joining order_items and orders tables on common column order_id
    amazon_brazil.orders o
on
    oi.order_id = o.order_id
group by
    o.customer_id
order by
    avg_order_value desc
    -- fetching only the top 20 customers
limit 20;
```

Query Output:

	customer_id character varying	avg_order_value numeric	customer_rank bigint
1	c6e2731c5b391845f6800c97401a43a9	6735.00	1
2	f48d464a0baaea338cb25f816991ab1f	6729.00	2
3	3fd6777bbce08a352fddd04e4a7cc8f6	6499.00	3
4	df55c14d1476a9a3467f131269c2477f	4799.00	4
5	24bbf5fd2f2e1b359ee7de94defc4a15	4690.00	5
6	3d979689f636322c62418b6346b1c6d2	4590.00	6
7	1afc82cd60e303ef09b4ef9837c9505c	4399.87	7
8	35a413c7ca3c69756cb75867d6311c0d	4099.99	8
9	e9b0d0eb3015ef1c9ce6cf5b9dcbee9f	4059.00	9
10	c6695e3b1e48680db36b487419fb0398	3999.90	10

11	926b6a6fb8b6081e00b335edaf578d35	3999.00	11
12	3be2c536886b2ea4668eced3a80dd0bb	3980.00	12
13	31e83c01fce824d0ff786fcd48dad009	3930.00	13
14	eb7a157e8da9c488cd4ddc48711f1097	3899.00	14
15	19b32919fa1198aefc0773ee2e46e693	3700.00	15
16	66657bf1753d82d0a76f2c4719ab8b85	3699.99	16
17	addc91fdf9c2b3045497b57fc710e820	3690.00	17
18	39d6658037b1b5a07d0a24d423f0bd19	3549.00	18
19	e7c905bf4bb13543e8df947af4f3d9e9	3399.99	19
20	46bb3c0b1a65c8399d0363cefbcc4f37	3124.00	20

Recommendations:

1. Exclusive VIP Rewards Program
   - Offer high spending customers with early access to sales, new product launches and priority customer support.
   - Provide personalized discount codes or cashback for repeat purchases.
2. Enhance Loyalty Benefits and Memberships
   - Introduce a premium offers and perks like free shipping, priority delivery, or dedicated account managers for such customers.
   - Reward consistent high-spenders with bonus points or surprise gifts.

Question 2.36 –

Problem Statement:

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

Desired Output: payment_type, sale_month, monthly_total, monthly_change

Approach:

1. Identifying Relevant Tables and Columns:
   - Table: orders, payments
   - Columns: payment_type, order_purchase_timestamp, payment_value
2. Creating a temporary tables using CTE
   - Use CTE to generate temporary tables containing payment types, sales month, monthly total sales and previous month sales.
3. Extracting month and year from timestamp
   - Use TO_CHAR() function to extract month and year in MM-YYYY format

4. Aggregating payment value for each payment type and sales month

- Use SUM() to perform aggregation of payment value to calculate monthly sales for each payment type and for each month of 2018

5. Joining tables using common column:
   - Join orders and payments tables on a common column order_id.
6. Filtering result for year 2018
   - Use WHERE clause to mention condition to consider records for year 2018 only
7. Grouping the Result:
   - Group the final result by payment type and sale month to compute total sales for each payment type and sales month.
8. Fetching previous month sales
   - Use LAG() windows function to fetch sales for previous month
9. Calculating month-over-month growth rate
   - Calculate percentage difference between current month and previous month sale to understand month-over-month growth rate.

SQL Query:

```
WITH
    -- using CTE to calculate total monthly sales
    -- and percentage change from previous month for each payment method
    monthly_sales AS
    (SELECT
        payment_type,
        -- extracting month and year from order purchase timestamp
        TO_CHAR(o.order_purchase_timestamp, 'MM-YYYY') AS sale_month,
        -- sum aggregation of payment value for each payment type
        SUM(p.payment_value) AS monthly_total
    FROM
        amazon_brazil.orders o
    JOIN -- joining orders and payments tables on common column order_id
        amazon_brazil.payments p
    ON
        o.order_id = p.order_id
    WHERE  -- extracting records only for year 2018
        o.order_purchase_timestamp >= '2018-01-01'
        AND
        o.order_purchase_timestamp < '2018-12-31'
    GROUP BY
        payment_type, sale_month
    ),
```

```sql
    monthly_growth AS
    (SELECT
        payment_type,
        sale_month,
        monthly_total,
        -- fetching total sales for previous month
        LAG(monthly_total) OVER (PARTITION BY payment_type ORDER BY sale_month) AS previous_month_total
    FROM
        monthly_sales
    )
SELECT
    payment_type,
    sale_month,
    monthly_total,
    CASE
        WHEN previous_month_total IS NULL OR previous_month_total = 0 THEN NULL
        ELSE round((monthly_total - previous_month_total)* 100.0 / previous_month_total,3)
    END AS monthly_change
FROM
    monthly_growth
ORDER BY
    payment_type, sale_month;
```

Query Output: (only first 20 displayed here)

	payment_type character varying	sale_month text	monthly_total numeric	monthly_change numeric
1	boleto	01-2018	204844.66	[null]
2	boleto	02-2018	183112.72	-10.609
3	boleto	03-2018	191538.02	4.601
4	boleto	04-2018	193547.09	1.049
5	boleto	05-2018	195378.93	0.946
6	boleto	06-2018	153350.28	-21.511
7	boleto	07-2018	198041.24	29.143
8	boleto	08-2018	143805.90	-27.386
9	credit_card	01-2018	868880.38	[null]
10	credit_card	02-2018	778803.00	-10.367
11	credit_card	03-2018	933770.10	19.898
12	credit_card	04-2018	934306.00	0.057
13	credit_card	05-2018	927556.35	-0.722
14	credit_card	06-2018	811508.56	-12.511
15	credit_card	07-2018	803674.49	-0.965
16	credit_card	08-2018	797648.89	-0.750
17	debit_card	01-2018	11543.55	[null]
18	debit_card	02-2018	7469.53	-35.293
19	debit_card	03-2018	8375.11	12.124
20	debit_card	04-2018	10782.53	28.745

Recommendations:

1. Prioritize High-Growth Payment Methods
   - Highlight fastest-growing payment methods at checkout for seamless user experience.
   - Ensure one-click payments for high-revenue methods to boost repeat purchases.
2. Targeted Promotions Based on Payment Preferences
   - Offer exclusive discounts, cashback, or reward points for frequently used payment methods.