amazon

Brazil's Data Analysis

To Optimise Amazon India's Strategy Using SQL

GUDDU SINGH, Nextleap Milestone Project

Table of Contents

This document provides an in-depth SQL analysis for Amazon India based on Amazon Brazil's data. The following sections will guide you through various data insights, queries, & recommendations.

Background

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.

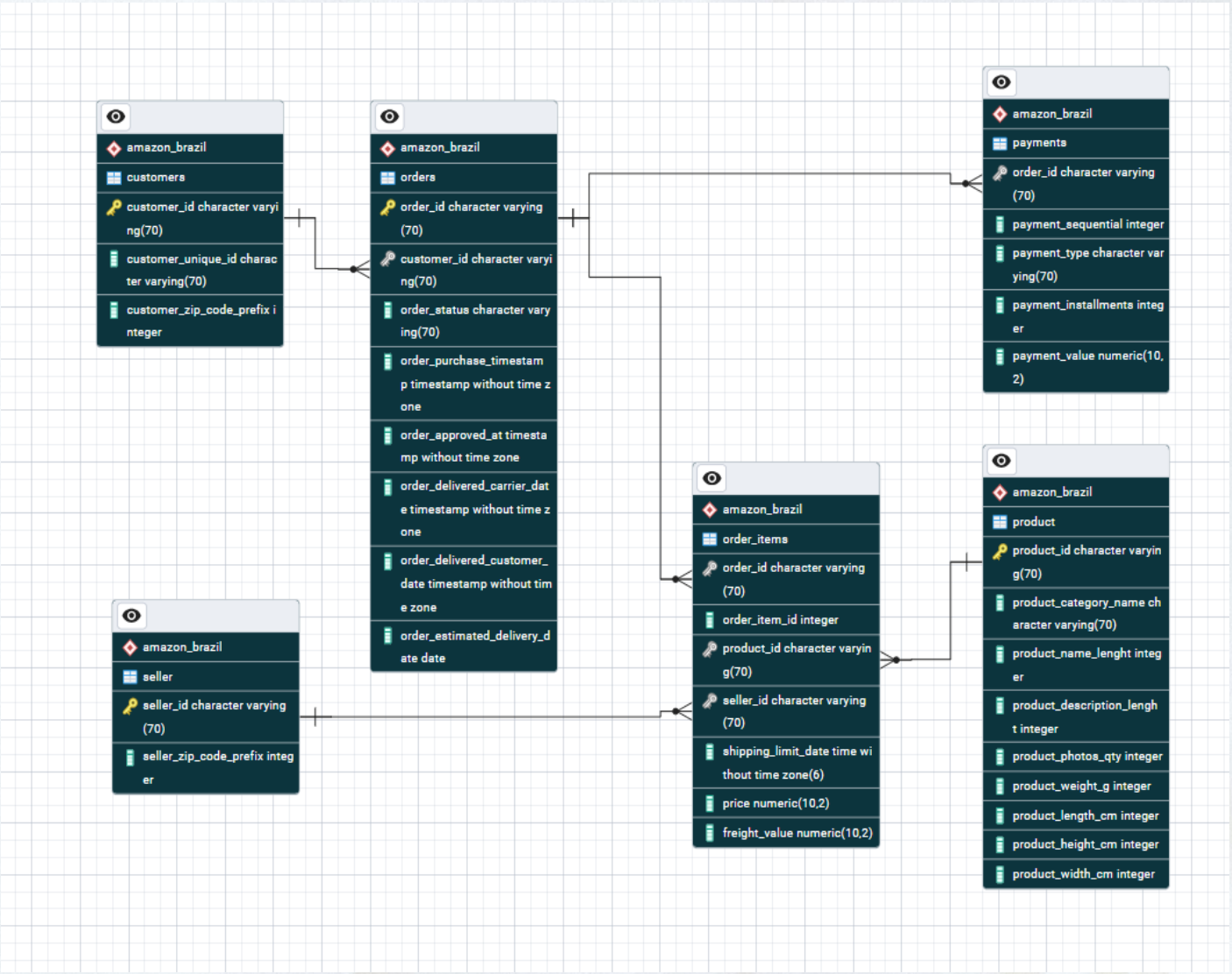
As a data analyst for Amazon India, your task is to analyse Amazon Brazil's data to identify trends, customer behaviours, and preferences that could be leveraged in the Indian market. This analysis will help Amazon India make informed decisions, enhance customer experience, and seize new opportunities.

Overview of 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 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.

Entity Relationship Diagram (ERD)



This ERD (Entity-Relationship Diagram) represents an e-commerce database schema with six interconnected tables:

- **Customers**: Stores customer details like customer_id and customer_unique_id. Each customer can place multiple orders, linking the customers table to the orders table via customer_id.
- **Orders**: Contains order-related data like order_status, timestamps, and estimated delivery dates. It connects to the customers table through customer_id and links to the order_items and payments tables via order_id.
- **Order Items**: Tracks individual items in an order, linking orders to products through product_id and to sellers through seller_id. It also captures details like price and freight value.
- **Payments**: Records payment information for each order, such as payment_type, installments, and payment value, linking to the orders table via order_id.
- **Products**: Holds product details, including product_id, name, category, and dimensions. It connects to order_items via product_id.
- **Sellers**: Stores seller-specific information like seller_id and seller_zip_code_prefix. It links to order_items via seller_id.

Analysis I

1.1. Standardizing and Rounding Average Payment Values

Problem Statement:

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:

- **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.

SQL Query:

```
-- Selecting payment type and rounding the average payment value for each type
SELECT
    payment_type,
    ROUND(AVG(payment_value)) AS rounded_avg_payment
FROM
    amazon_brazil.payments
-- Grouping by payment type
GROUP BY
    payment_type
-- Ordering results in ascending order by rounded average payment
ORDER BY
    rounded_avg_payment ASC;
```

Output:

payment_type	rounded_avg_payment
not_defined	0
voucher	66
debit_card	143
boleto	145
credit_card	163

Recommendation:

1. **Maintain efficiency of Popular Payment Methods:**
    - Since credit_card is very popular, ensure 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 Less 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.

1.2. Distribution by Type Analysis

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.

Approach:

- **Identifying Relevant Tables and Columns:**
  - **Tables:** `payments`
    Columns: `order_id, payment_type`

- **Counting Total Orders by Payment Type:**
  - Used `COUNT(order_id)` to calculate the number of orders for each payment type.

- **Calculating Percentage of Total Orders:**
  - Used `SUM(COUNT(order_id))OVER()` to get the total order count.
  - Divided each payment type's order count by the total orders and multiplied by 100 to compute the percentage.
  - Applied `ROUND()` to round the result to `one decimal place`.

- **Sorting the Results:**
  - Ordered the results in `descending order` based on the `percentage of total orders`.

SQL Query:

```
-- Calculating the percentage of total orders for each payment type
SELECT
    p.payment_type,
    ROUND(COUNT(p.order_id) * 100.0 / SUM(COUNT(p.order_id)) OVER (), 1) AS
percentage_orders
FROM
    amazon_brazil.payments p
GROUP BY
    p.payment_type
ORDER BY
    percentage_orders DESC;
```

Output:

payment_type	percentage_orders
credit_card	73.9
boleto	19
voucher	5.6
debit_card	1.5
not_defined	0

Recommendation:

- **Credit Cards Dominate**: 73.9% of transactions are made using credit cards, indicating a strong preference for digital payments with deferred payment options.
- **Boleto (Bank Slip) is Significant**: 19% of transactions rely on boleto, a popular alternative payment method in Brazil that allows bank transfers or cash payments.
- **Limited Use of Vouchers and Debit Cards**: Vouchers (5.6%) and debit cards (1.5%) are used much less, showing a preference for credit-based transactions.
- **Negligible Undefined Payments**: 0% under "not_defined" suggests a well-structured payment ecosystem with clear payment tracking.

1.3. Identifying Products for Targeted Promotions within Specific Price Ranges

Problem Statement:

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

Approach:

- **Identifying Relevant Tables and Columns:**
  - **Tables:** `order_items, product`
  - **Columns:**order_items: `product_id, price` & product: `product_category_name`
- **Joining Tables:**
  - Joined `order_items with product` on product_id to access product_category_name.
- **Filtering by product_category_name and price range:**
  - Filtered products where the `price is between 100 and 500 BRL` and the product_category_name contains the word `"Smart"`.
- **Ensuring Unique Results:**
  - Used `DISTINCT` to ensure unique combinations of product_id and price.
- **Sorting Results:**
  - Sorted the results by price in `descending order`.
  -

SQL Query:

```
-- Select distinct product_id and price for products in categories containing 'Smart' and priced between 100 and 500 BRL
SELECT DISTINCT
    oi.product_id,
    oi.price
FROM
    amazon_brazil.order_items oi
-- Join with product table to filter based on product category names
```

```
JOIN
    amazon_brazil.product p
 ON
     oi.product_id = p.product_id
-- Filter for products with price between 100 and 500 BRL and in categories containing 'Smart'
WHERE
    oi.price BETWEEN 100 AND 500
    AND p.product_category_name ILIKE '%Smart%'
-- Sort results by price in descending order
ORDER BY
   oi.price DESC;
```

Output:

product_id	price
1df1a2df8ad2b9d3aa49fd851e3145ad	439.99
7debe59b10825e89c1cbcc8b190c85e2	349.99
ca86b9fe16e12de698c955aedff0aea2	349
0e52955ca8143bd179b311cc454a6caa	335
7aeaa8f3e592e380c420e8910a717255	329.9
d1b571cd58267d8cac8b2afd6e288bbc	299.9
66ffe28d0fd53808d0535eee4b90a157	254
f06796447de379a26dde5fcac6a1a2f7	239.9
d3d5a1d52abe9a7d234908d873fc377b	229.9
06ae026e430189633c2fbd0288c86257	217.36
49ef750dc5bf23e3788d4f614bc6dbe9	198
33bb7da523efcdef6cd2996cbf72d081	148
6f5795735ab2c629b22669fe889b7903	129.9
...	...

Recommendation:

Maintain the efficiency of Popular Payment Methods:
Since credit_card is very popular, ensure this payment method is highly reliable, secure, and easy to use. Consider cash back or reward or loyalty programs for credit card users to incentivise more purchases.

Improve Less 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

1.4. Identifying the Top 3 Months with the Highest Sales

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.

Approach:

- **Identifying Relevant Tables and Columns:**
  - **Table:** `orders, order_items`
  - **Columns:**
    orders: `order_purchase_timestamp`
    order_items: `price`

- **Extracting Month and Year from the Purchase Date:**
  - Extracted the `month` and `year` from the order_purchase_timestamp column to group sales by month.

- **Joining Tables:**
  - Joined `orders with order_items` on the common order_id to get both order and sales data.

- **Calculating Total Sales:**
  - Used `SUM(price)` to calculate the total sales per month.

- **Rounding Total Sales:**
  - Applied the `ROUND()` function to round the total sales to the nearest integer.

- **Sorting and Limiting Results:**
  - Sorted the months in `descending order` of total sales and limited the output to the `top 3 months`.

SQL Query:

```
-- Extracting month and year to calculate total sales per month and rounding the total sales,
SELECT
   TO_CHAR(o.order_purchase_timestamp, 'YYYY-MM') AS month,
   ROUND(SUM(oi.price), 0) AS total_sales
FROM
   amazon_brazil.orders o
JOIN
   amazon_brazil.order_items oi
ON
   o.order_id = oi.order_id
-- Grouping by year and month to aggregate sales
GROUP BY
   month
-- Sorting in descending order to get the highest sales
ORDER BY
   total_sales DESC
-- Limiting results to top 3 months
LIMIT 3;
```

Output:

month	total_sales
2017-11	1010271
2018-04	996648
2018-05	996518

Recommendations:

- **Focus on November Sales Peak**

  November 2017 shows the highest sales value, indicating strong demand, possibly due to holiday shopping or major sales events. Amazon India should prioritize inventory management and marketing efforts during November to maximize revenue.

- **Strengthen April and May Campaigns**

  April and May 2018 also show significant sales figures, suggesting these months may benefit from targeted promotional campaigns. Consider offering seasonal promotions or discounts during these months to further boost sales.

1.5. Analysis of Product Categories with Significant Price Variations

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.

Approach:

- **Identifying Relevant Tables and Columns:**
  - Tables: `product, order_items`
  - Columns:
    product: `product_category_name`
    order_items: `price`

- **Joining Tables:**
  - Joined `product with order_items` on the product_id to access price data for each category.

- **Calculating Price Difference:**
  - Used `MAX(price)` and `MIN(price)` to compute the price difference for each product category.

- **Filtering Categories:**
  - Used the `HAVING clause` to filter for categories with a price difference greater than 500 BRL.

- **Reason for Using HAVING:**
  - The HAVING clause is `necessary with aggregate functions` like MAX and MIN, while the WHERE clause cannot be used in this context.

- **Grouping Results:**
  - Grouped the results `by product_category_name` to aggregate price data for each category.

- **Sorting Results:**
  - Sorted the results by price difference in `descending order` to prioritize categories with the largest variations.

SQL Query:

```
-- Selecting the name of the product category and calculating the price difference
SELECT
    p.product_category_name,
    MAX(oi.price) - MIN(oi.price) AS price_difference
FROM
   amazon_brazil.product p
-- Joining the product tables with order_items table on product_id
JOIN
   amazon_brazil.order_items oi
ON
   p.product_id = oi.product_id
-- Grouping results by product category name
GROUP BY
   p.product_category_name
-- Filter to include only categories with a price difference greater than 500 BRL
HAVING
   MAX(oi.price) - MIN(oi.price) > 500
-- Order the results by price difference in descending order
ORDER BY
   price_difference DESC;
```

Output:

product_category_name	price_difference
utilidades_domesticas	6731.94
pcs	6694.5
artes	6495.5
eletroportateis	4792.5
instrumentos_musicais	4394.97
consoles_games	4094.81
esporte_lazer	4054.5
relogios_presentes	3990.91
NULL	3977
ferramentas_jardim	3923.65
bebes	3895.46
informatica_acessorios	3696.09
beleza_saude	3122.8
cool_stuff	3102.99
construcao_ferramentas_seguranca	3091

industria_comercio_e_negocios	3061.1
agro_industria_e_comercio	2977.01
portateis_casa_forno_e_cafe	2888.81
pet_shop	2495.1
eletronicos	2466.51
telefonia	2423
eletrodomesticos_2	2336.1
construcao_ferramentas_construcao	2299.15
automotivo	2254.51
eletrodomesticos	2083.81
cama_mesa_banho	1992.99
market_place	1954.01
moveis_decoracao	1894.1
construcao_ferramentas_ferramentas	1892.2
telefonia_fixa	1784
brinquedos	1695.09
fashion_bolsas_e_acessorios	1693.99
…	…

Recommendations:

- **Target Different Customer Groups:**

  Amazon can create various price options for product categories like utilidades_domesticas and pcs, which have large price differences. This way, they can attract customers looking for both budget-friendly and high-end products.

- **Use Special Discounts or Dynamic Pricing:**

  In categories like artes and eletroportateis, Amazon can offer special sales or change prices based on customer demand. This approach will help them sell both expensive and cheaper products more effectively.

- **Focus on Stock and Availability:**

  For categories with large price differences, such as instrumentos_musicais and eletroportateis, it's important for Amazon to keep products in stock at different price levels. This will ensure they meet the needs of a wider range of customers.

1.6. Payment Types with Least Variance in Transaction Amounts

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.

Approach:

- **Identifying Relevant Table and Columns:**
  - Table: `payments`
  - Columns:

    `payment_type`: This helps us group the transactions based on different payment methods.

    `payment_value`: We will use this to calculate how much the transaction amounts vary.

- **Calculating Standard Deviation:**
  - We use the `STDDEV()` function to find the `standard deviation of payment_value` for each payment_type. This shows us how much the amounts differ from the average for each payment method.

- **Grouping by Payment Type:**
  - We `group the results by payment_type`. This means we will calculate the standard deviation for each payment method, like credit or debit cards.

- **Sorting by Consistency:**
  - Finally, we sort the results `by std_deviation in ascending order`. This helps us find out which payment types have the most consistent transaction amounts, meaning they don't vary much.

SQL Query:

```
-- Find payment types with the least variance in transaction amounts
-- Calculating the standard deviation of payment amounts
SELECT
   payment_type,
   ROUND(STDDEV(payment_value),2) AS std_deviation
FROM
   amazon_brazil.payments
-- Exclude any undefined payment types from the results
WHERE
   payment_type != 'not_defined'
-- Grouping by payment type to calculate std deviation for each type
GROUP BY
   payment_type
-- Sorting by the smallest standard deviation to find the most consistent payment types
ORDER BY
   std_deviation ASC;
```

Output:

payment_type	std_deviation
voucher	115.52
boleto	213.58
credit_card	222.12
debit_card	245.79

Recommendation:

- **Focus on Vouchers:**

  The voucher payment type has the lowest standard deviation (115.52), indicating that transaction amounts are quite consistent. Amazon could consider promoting this payment method more heavily, as it appears to be reliable for customers.

- **Review Credit and Debit Cards:**

  Both credit_card (222.12) and debit_card (245.79) payment types show higher variances in transaction amounts. Amazon could investigate the factors leading to this variability, such as promotional offers or transaction limits, and consider ways to make these payment methods more predictable for customers.

1.7. Products with Missing or Incomplete Category Names

Problem Statement:

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

Approach:

- **Identifying Relevant Tables and Columns**:
  - **Table**: `products`
  - **Columns**: `product_id, product_category_name`

- **Filtering Results**:
  - Used a `WHERE clause` to check for missing category names or those with a length of 1 character.

SQL Query:

```
-- Selecting product ID and category name for products with missing or incomplete category names
SELECT
   product_id,
   product_category_name
FROM
   amazon_brazil.product
-- Filtering for missing category names or those with a single character
WHERE
   product_category_name IS NULL OR
   LENGTH(product_category_name) = 1;
```

Output:

product_id	product_category_name
a41e356c76fab66334f36de622ecbd3a	NULL
d8dee61c2034d6d075997acef1870e9b	NULL
56139431d72cd51f19eb9f7dae4d1617	NULL
46b48281eb6d663ced748f324108c733	NULL
5fb61f482620cb672f5e586bb132eae9	NULL
…….	…….

Recommendation

- **Improve Data Entry**:

  Train staff on proper data entry practices to reduce the number of incomplete category names. This can help ensure that all product categories are filled out correctly when products are added.

- **Implement Validation Rules:**

  Set up automatic checks in the system that prevent saving products with missing or very short category names. This will help maintain data quality and make it easier to find products in the future.

Analysis 2

2.1. Assessing Popular Payment Types in Low, Medium, and High Order Values

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.

Approach:

- **Segmenting Order Values:**
  - The `CASE` statement is used to create three categories based on the `payment_value`:
    "Low" for payments less than 200 BRL,
    "Medium" for payments between 200 and 1000 BRL,
    "High" for payments over 1000 BRL.

- **Selecting Payment Type:**
  - The query selects the `payment_type` from the payments table.

- **Counting Transactions:**
  - `COUNT(o.order_id)` is used to count the number of orders associated with each payment type within each defined order value segment.

- **Joining Tables:**
  - The query joins the payments table with the orders table on the `order_id` to link each payment with its corresponding order.

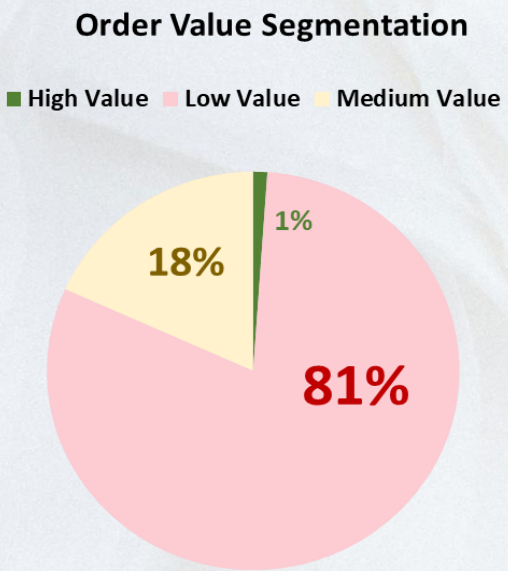
- **Grouping and Sorting:**
  - The results are grouped by both the `order_value_segment` and the `payment_type`, and sorted in descending order by the count of transactions, allowing Amazon to see which payment types are most popular for each order value range.

SQL Query:

```sql
-- Segmenting orders by value and counting payment types for each segment
SELECT
   CASE
      WHEN p.payment_value < 200 THEN 'Low'
      WHEN p.payment_value BETWEEN 200 AND 1000 THEN 'Medium'
       ELSE 'High'
   AS
   order_value_segment,
   p.payment_type,
   COUNT(o.order_id) AS count
FROM
   amazon_brazil.orders o
-- Joining the orders table with the payments table
JOIN
   amazon_brazil.payments p
ON
   o.order_id = p.order_id
-- Grouping by order value segment and payment type
GROUP BY
   order_value_segment, p.payment_type
-- Sorting results by count in descending order
ORDER BY
   count DESC;
```

Output:

order_value_segment	payment_type	count
Low	credit_card	60548
Low	boleto	16444
Medium	credit_card	15303
Low	voucher	5476
Medium	boleto	3162
Low	debit_card	1287
High	credit_card	944
Medium	voucher	286
Medium	debit_card	227
High	boleto	178
High	debit_card	15
High	voucher	13
Low	not_defined	3



Recommendation:

Promote Credit Cards for Higher Segments: Since credit cards are widely used in all segments, Amazon should consider offering cash back or reward points to customers using credit cards for high-value transactions.

Expand Boleto Usage: Boleto is popular in the low segment, but its presence decreases in higher segments. Amazon can introduce promotional campaigns encouraging Boleto usage for medium and high-value orders.

Explore Voucher Promotions: Vouchers are underutilized, especially in high-value segments. Consider targeted campaigns to increase voucher adoption for medium and high-value orders.
This comprehensive approach helps Amazon India optimize payment method offerings for each order value segment, thereby enhancing the overall customer experience.

2.2. Analyzing Price Range and Average Price for Each Product Category

Problem Statement:

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.

Approach:

- **Selecting Price and Category:**
  - The query selects the product prices (`price`) from the `order_items` table and product categories (`product_category_name`) from the `product` table.

- **Calculating Price Statistics:**
  - `MIN(price)` finds the minimum price.
  - `MAX(price)` finds the maximum price.
  - `Round(AVG(price), 2)` calculates the average price, rounded to two decimal places.

- **Joining Tables:**
  - The `order_items` table is joined with the `product` table using `product_id` to associate each item with its respective product category.

- **Grouping and Sorting:**
  - The results are grouped by product category and sorted in descending order by the average price.

SQL Query:

```
-- Getting minimum, maximum, and average price for each product category name
SELECT
   p.product_category_name,
   MIN(oi.price) AS min_price,
   MAX(oi.price) AS max_price,
   ROUND(AVG(oi.price),2) AS avg_price
FROM
   amazon_brazil.order_items oi
-- Joining the order items table with the product table to get category names
JOIN
   amazon_brazil.product p
ON
   oi.product_id = p.product_id
-- Grouping results by product category
```

```
GROUP BY
    p.product_category_name
-- Order results by average price in descending order
ORDER BY
    avg_price DESC;
```

Output:

product_category_name	min_price	max_price	avg_price
pcs	34.5	6729	1098.34
portateis_casa_forno_e_cafe	10.19	2899	624.29
eletrodomesticos_2	13.9	2350	476.12
agro_industria_e_comercio	12.99	2990	341.66
instrumentos_musicais	4.9	4399.87	281.62
eletroportateis	6.5	4799	280.78
portateis_cozinha_e_preparadores_de_alimentos	17.42	1099	264.57
telefonia_fixa	6	1790	225.69
construcao_ferramentas_seguranca	8.9	3099.9	208.99
relogios_presentes	8.99	3999.9	200.91
…	…	…	…

Recommendation:

Price Optimisation for High-Value Categories: Consider bundling offers for high-value categories like pcs to attract more purchases.

Focus on Mid-Range Categories: Strengthen marketing and promotional efforts around mid-range categories, as they balance affordability and product variety.

Expand in Low-Value Categories: For low-value categories, explore cross-selling strategies to increase overall cart value.

2.3. Identifying Customers with Multiple Orders

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.

Approach:

- **Selecting Customer IDs:**
  - The query selects `customer_unique_id` from the `customers` table to identify each unique customer.
- **Counting Orders:**
  - The query counts the number of orders each customer has placed using `COUNT(o.order_id)`.
- **Joining Tables:**
  - The `customers` table is joined with the `orders` table on `customer_id` to link each customer with their orders.
- **Filtering for Multiple Orders:**
  - A `HAVING` clause filters for customers who have placed more than one order.
- **Sorting:**
  - The results are sorted by the total number of orders in `descending order` to display the most frequent customers first.

SQL Query:

```
-- Getting customer unique IDs and counting total orders for each customer
SELECT
    c.customer_unique_id,
    COUNT(o.order_id) AS total_orders
FROM
    amazon_brazil.customers c
-- Joining the customers table with the orders table
JOIN
    amazon_brazil.orders o
ON
    c.customer_id = o.customer_id
-- Grouping by customer unique ID
GROUP BY
    c.customer_unique_id
-- Filtering for customers with more than one order
HAVING
```

```
    COUNT(o.order_id) > 1
-- Sorting results by total orders in descending order
ORDER BY
    total_orders DESC;
```

Output:

customer_unique_id	total_orders
a91e80fbe80ddc07de66a5cf9270293c	16
a6168cd79131e64acef92e3c74d6cc43	16
363f980585bf04c1a88fdb986011c52e	16
cbd0350d4ccba9772e8e768d4a4a5cbf	16
417b909c0962b2610f1cfeb1c1478986	16
5f94af52aef02c968a2e0f01f430864e	16
1b6d29725255a77667a8c639eeb4ccc0	16
e4bbcc533fdf3917c56dea2c43bf2084	16
930c4390af58f67334447c3a1cf2ba36	16
5bf4ea2d98005b960eea0dbf652ef4e7	16
9159c04b88895d995741dd5b9b7a5f1d	16
4034aa08d48695a538b7030910aae5d5	16
c024307523462166b42112cfb6c8e900	16
0fdc0d21e1983e8af4d399e17671f76d	16
96fd69e8b0df76a9a807b01dc82bef5b	16
7f4f709af2fd8fea44aacd30bca46264	16
f9c4e8531c2fe4159beb562fd7c2bd59	16
3d364a7768fae99678635c4370295d20	16
6af40347f5dd7bdd65437a35e1b2fa7b	16
f300b00a19af4d4f7bdf9f4524c4587a	16
75f15790b1852b42b1dbf645d98ffa1c	16
8d50f5eadf50201ccdcedfb9e2ac8455	15
7c396fd4830fd04220f754e42b4e5bff	14
...	...

Recommendation:

Loyalty Programs:
Implement a loyalty program to reward customers who place multiple orders, encouraging repeat business.

Personalised Marketing:
Send personalised offers or discounts to customers with multiple orders, based on their purchase history.

2.4. Categorizing Customers into New, Returning, and Loyal Based on Order Quantity

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 customer's table to update and display the customer types.

Approach:

- **Identifying Relevant Tables and Columns:**
  - **Tables**: `customers, orders`
  - **Columns**:
    customers: `customer_id`
    orders: `order_id, customer_id`

- **Counting Total Orders for Each Customer:**
  - Used `COUNT(order_id)` from the orders table to calculate the total number of orders each customer has placed.

- **Categorizing Customers Based on Order Count:**
  - Applied a `CASE statement` to classify customers into:
    `New`: If they have exactly 1 order.
    `Returning`: If they have between 2 and 4 orders.
    `Loyal`: If they have more than 4 orders.

- **Creating a Temporary Table:**
  - Created a temporary table (customer_order_count) to store the total order count for each customer.

- **Joining Tables:**
  - Joined the customer_order_count temporary table with the customer's table on customer_id to display customer types.

SQL Query:

```
-- Create a temporary table to categorize customers based on their order quantity
WITH customer_order_count AS (
    SELECT
        c.customer_id,
        COUNT(o.order_id) AS total_orders
    FROM
        amazon_brazil.customers c
    -- Joining the customers table with the orders table
    JOIN
        amazon_brazil.orders o
    ON
     c.customer_id = o.customer_id
    -- Grouping by customer_id to get total orders per customer
    GROUP BY
        c.customer_id
)


-- Select from the temporary table and categorize customers as New, Returning, or Loyal
SELECT
    c.customer_id,
    CASE
        WHEN coc.total_orders = 1 THEN 'New'
        WHEN coc.total_orders BETWEEN 2 AND 4 THEN 'Returning'
        ELSE 'Loyal'
    END AS customer_type
FROM
    customer_order_count coc
-- Joining the customer_order_count with the customers table
JOIN
    amazon_brazil.customers c
ON
coc.customer_id = c.customer_id;
```

Output:

customer_id	customer_type
00012a2ce6f8dcda20d059ce98491703	New
000161a058600d5901f007fab4c27140	New
0001fd6190edaaf884bcaf3d49edf079	New
0002414f95344307404f0ace7a26f1d5	New
000379cdec625522490c315e70c7a9fb	New
0004164d20a9e969af783496f3408652	New
000419c5494106c306a97b5635748086	New
00046a560d407e99b969756e0b10f282	New
00050bf6e01e69d5c0fd612f1bcfb69c	New
000598caf2ef4117407665ac33275130	New
...	...

Recommendation:

Target Marketing:
Use the customer types to tailor marketing strategies; for example, special offers for 'New' customers to encourage repeat purchases.

Loyalty Programs:
Develop loyalty programs aimed at 'Loyal' customers to enhance retention.

Re-engagement Campaigns:
Create campaigns specifically targeting 'Returning' customers to increase their purchase frequency.
This approach allows Amazon India to better understand and segment their customer base, leading to more effective marketing and sales strategies.

2.5. Identifying Product Categories with the Most Revenue

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.

Approach:

- **Identifying Relevant Tables and Columns:**
  - **Tables**: order_items, product
  - **Columns**:
    order_items: product_id, price
    product: product_id, product_category_name

- **Joining Tables:**
  - Joined `order_items` and `product` tables on product_id to access both product categories and their associated prices.

- **Calculating Total Revenue:**
  - Calculated the total revenue for each product category using `SUM(order_items.price)`

- **Sorting and Limiting Results:**
  - Sorted the results by total revenue in `descending order` and `limited` the output to the top 5 categories.

SQL Query:

```
-- Calculating total revenue using SUM for each product category
SELECT
   p.product_category_name,
   SUM(oi.price) AS total_revenue
FROM
   amazon_brazil.order_items oi
JOIN                    -- Joining the order_items table with the product table to get category names
   amazon_brazil.product p
ON
   oi.product_id = p.product_id
--Grouping by product category and then setting them in descending order by limiting the top 5 results
GROUP BY
   p.product_category_name
ORDER BY
   total_revenue DESC
LIMIT 5;
```

Output:

product_category_name	total_revenue
beleza_saude	1257865
relogios_presentes	1203060
cama_mesa_banho	1032269
esporte_lazer	985881.1
informatica_acessorios	910605.1

Recommendation:

Focus on Top Categories: Prioritize marketing and stock replenishment for the highest revenue-generating categories like "beleza_saude" and "relogios_presentes."

Targeted Promotions: Develop promotional campaigns specifically aimed at these top categories to boost sales further.

Inventory Management: Ensure sufficient inventory levels for the most popular categories to avoid stockout and missed sales opportunities.

Customer Feedback: Gather customer feedback on these top categories to improve product offerings and customer satisfaction.

Analysis 3

3.1 Comparing Total Sales Between Different Seasons

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.

Approach:

- **Identifying Relevant Tables and Columns:**
  - **Table**: `orders, order_items`
  - **Columns**:
    orders: `order_purchase_timestamp`
    order_items: `price`

- **Extracting Month from Order Dates**:
  - Extracted the month from the `order_purchase_timestamp` column to assign each order to a specific season.

- **Defining Seasons**:
  - Used a `CASE statement` to define each season based on the month:
    Spring: March to May (3,4,5)
    Summer: June to August (6,7,8)
    Autumn: September to November (9,10,11)
    Winter: December to February (12,1,2)

- **Joining Tables**:
  - Joined orders with order_items on order_id to get both order date and sales data.

- **Calculating Total Sales**:
  - Summed the prices from order_items to calculate total sales for each season.

SQL Query:

```
-- Calculating total sales for each season based on order dates
-- Defining the season based on the month extracted from order_purchase_timestamp
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,
-- Summing up prices for each season to calculate total sales
   SUM(oi.price) AS total_sales
FROM
   amazon_brazil.orders o
-- Joining the orders table with the order_items table to get order and sales data
JOIN
   amazon_brazil.order_items oi
ON
    o.order_id = oi.order_id
GROUP BY
   season                          -- Grouping results by season
ORDER BY
   total_sales DESC;               -- Sorting results by total sales in descending order
```

Output:

season	total_sales (BRL)
Spring	4216722
Summer	4120360
Winter	2905750
Autumn	2348813

Recommendation:

- **Targeted Seasonal Promotions:**
  Since Spring and Summer show the highest sales, plan special discounts and marketing campaigns during these seasons to boost revenue even further.

- **Seasonal Bundling Offers:**
  Create bundled product offers for high-selling seasons like Spring and Summer to increase average order value.

3.2 Identifying Products with Sales Volumes Above the Overall Average Using CTE

Problem Statement:

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.

Approach:

- **Identifying Relevant Tables and Columns**:
  - **Tables**: `product, order_items`
  - **Columns**:
    product: `product_id`
    order_items: `product_id, order_id`

- **product_totals CTE:**
  - A CTE is used first to calculate the `total quantity sold` for each product by grouping the order_items table based on product_id.

- **avg_quantity CTE**:
  - A second CTE calculates the overall `average quantity sold` across all products by averaging the total quantity sold from the first CTE.

- **Main Query:**
  - The main query selects `products` from `product_totals` where the total quantity sold is greater than the average calculated in `avg_quantity`

- **Ordering Results**:
  - Results are sorted in `descending order` of total quantity sold.

SQL Query:

```
-- CTE to calculate the total quantity sold for each product
WITH product_totals AS (
    SELECT
        p.product_id,
        COUNT(DISTINCT oi.order_id) AS total_quantity_sold -- Calculate total quantity sold for each product
    FROM
        amazon_brazil.product p
    -- Joining product table with order_items table to get sales data
    JOIN
        amazon_brazil.order_items oi
    ON
        p.product_id = oi.product_id
    -- Grouping by product_id to calculate the total quantity sold for each product
    GROUP BY
        p.product_id
),


-- CTE to calculate the average total quantity sold across all products
Avg_quantity AS (
    SELECT
        AVG(total_quantity_sold) AS avg_quantity_sold  -- Calculate overall average total quantity sold
    FROM
        product_totals
),


-- Main query to filter products with total quantity sold above the average
SELECT
    product_id,                                    -- Selecting product_id
    total_quantity_sold                            -- Displaying the total quantity sold for each product
FROM
    product_totals
-- Filtering products whose total quantity sold is greater than the overall average
WHERE
    total_quantity_sold > (SELECT avg_quantity_sold FROM avg_quantity)
-- Sorting results by the total quantity sold in descending order
ORDER BY
    total_quantity_sold DESC;
```

Output:

product_id	total_quantity_sold
99a4788cb24856965c36a24e339b6058	467
aca2eb7d00ea1a7b8ebd4e68314663af	431
422879e10f46682990de24d770e7f83d	352
d1c427060a0f73f6b889a5c7c61f2ac4	323
389d119b48cf3043d311335e499d9c6b	311
53b36df67ebb7c41585e8d54d6772e08	306
368c6c730842d78016ad823897a372db	291
53759a2ecddad2bb87a079a1f1519f73	287
Continue…..	Continue…

Recommendation:

Prioritize Inventory for High-Demand Products: Ensure top-selling items are always in stock to avoid losing potential sales opportunities.

Focus Marketing on High Performers: Allocate more marketing and advertising resources to promote these high-demand products further.

Introduce Product Variants: Consider expanding the product line by introducing new models or variants to attract a wider customer base.

Bundle Top Performers: Create bundled offers with top-performing products to increase average order value and customer satisfaction.

Optimise Supply Chain: Strengthen supply chain operations for these products to ensure they are delivered quickly and efficiently to customers.

3.3 Monthly Revenue Calculation for 2018 with Month Names

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.

Approach:

The goal of this query is to calculate the total revenue generated in each month during the year 2018. The month names are displayed as three-letter abbreviations (e.g., Jan, Feb, etc.) instead of numerical month values. The approach involves using SQL functions like `TO_CHAR` to convert the date to the required format and `SUM` to aggregate the total revenue for each month. The results are grouped by month and sorted in the correct calendar order (January to December).

- **Identifying Relevant Tables**:
  - **Tables**: `orders, order_items`
  - **Columns**:
    orders: `timestamp`
    order_items: `price`

- **Filtering Data for 2018**:
  - The query uses the `EXTRACT(YEAR FROM order_purchase_timestamp)` function to ensure only the data from 2018 is analyzed.

- **Converting Month Numbers to Month Names**:
  - The `TO_CHAR(order_purchase_timestamp,'Mon')` function is used to convert the month number into a 3-letter month abbreviation (like Jan for January).

- **Summing Monthly Revenue**:
  - The `SUM(oi.price)` function is used to calculate the total revenue for each month by summing up the prices from the `order_items` table.

- **Sorting by Calendar Month**:
  - The query is grouped by both the `month name and month number`, and the results are ordered by the extracted month number to ensure the results are in calendar order (January to December).

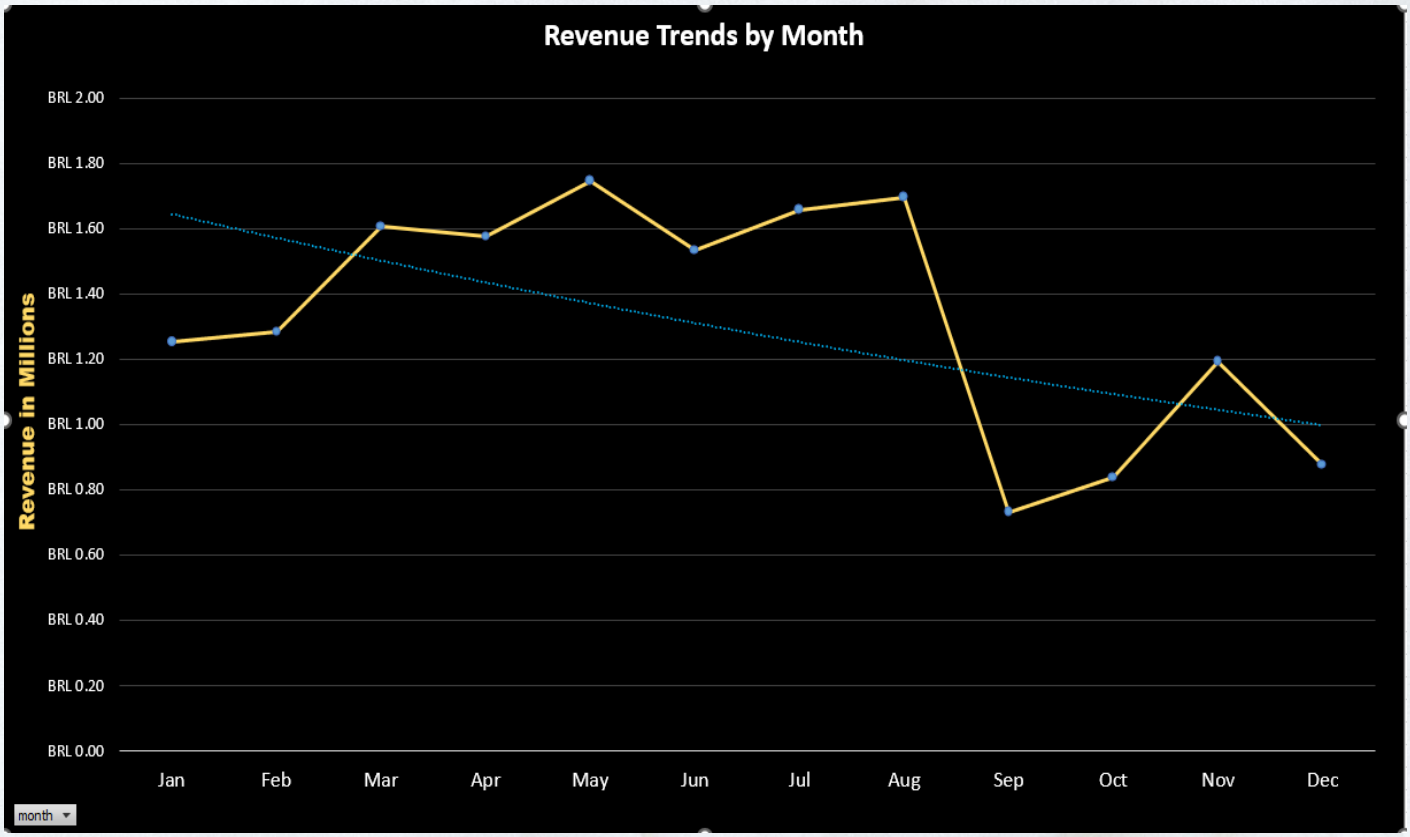
SQL Query:

```
-- Converting the order purchase timestamp to a 3-letter month abbreviation (Jan, Feb)
-- Calculating the total revenue for each month by summing the prices of all ordered items
SELECT
   TO_CHAR(o.order_purchase_timestamp, 'Mon') AS month,
   SUM(oi.price) AS total_revenue
FROM
   amazon_brazil.orders o
JOIN
   amazon_brazil.order_items oi
ON
   o.order_id = oi.order_id
-- Filtering only the orders from the year 2018
WHERE
   EXTRACT(YEAR FROM o.order_purchase_timestamp) = 2018
-- Grouping the results by month to aggregate revenue for each month
-- Grouping by the formatted month name
-- Grouping by the month number to preserve the correct month order
GROUP BY
   TO_CHAR(o.order_purchase_timestamp, 'Mon'),
   EXTRACT(MONTH FROM o.order_purchase_timestamp)
-- Sorting the results in calendar order (January to December)
ORDER BY
   EXTRACT(MONTH FROM o.order_purchase_timestamp);
```

Output:

Month	Total Revenue (BRL)
May	17,46,900.97
Aug	16,96,821.64
Jul	16,58,923.67
Mar	16,09,515.72
Apr	15,78,573.51
Jun	15,35,156.88
Feb	12,84,371.35
Jan	12,53,492.22
Nov	11,94,882.80
Dec	8,78,421.10
Oct	8,39,358.03
Sep	7,32,454.23

Chart:



Recommendation:

- **Focused Marketing Campaigns:**
  Launch targeted marketing campaigns during high-revenue months like May and August to further capitalise on strong sales trends.

- **Promotional Offers During Low-Performing Months:**
  Introduce discounts and bundle offers during traditionally lower-revenue months like September and October to stimulate demand and increase sales.

- **Seasonal Product Strategy:**
  Review the product categories contributing to high sales in top-performing months (e.g., May and August) to refine product placement and promotion strategies.

- **Optimise Inventory Management:**
  Ensure sufficient inventory for top-selling months to meet customer demand, while adjusting stock levels for low-revenue months to minimise holding costs.

3.4 Customer Segmentation Analysis

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 and generate a chart in Excel to show the proportion of each segment.

Approach:

- **Calculate Total Orders per Customer:**
  - Using a `Common Table Expression (CTE)` named `customer_order_count`, calculate the number of orders for each customer by `grouping the orders table` using `customer_id` and applying `COUNT(order_id)`.

- **Categories Customers into Segments:**
  - Create a `second CTE` named customer_segments `to categorise customers` based on their `order_count`
    Occasional: 1-2 orders
    Regular: 3-5 orders
    Loyal: More than 5 orders

- **Count the Number of Customers in Each Segment:**
  - Use `the customer_segments CTE` to `group customers` by their respective `customer_type` and `count the number of customers` in each category.

- **Sorting and Displaying Results:**
  - Sort the results in `descending order` of the count to display the segments with the highest number of customers first.

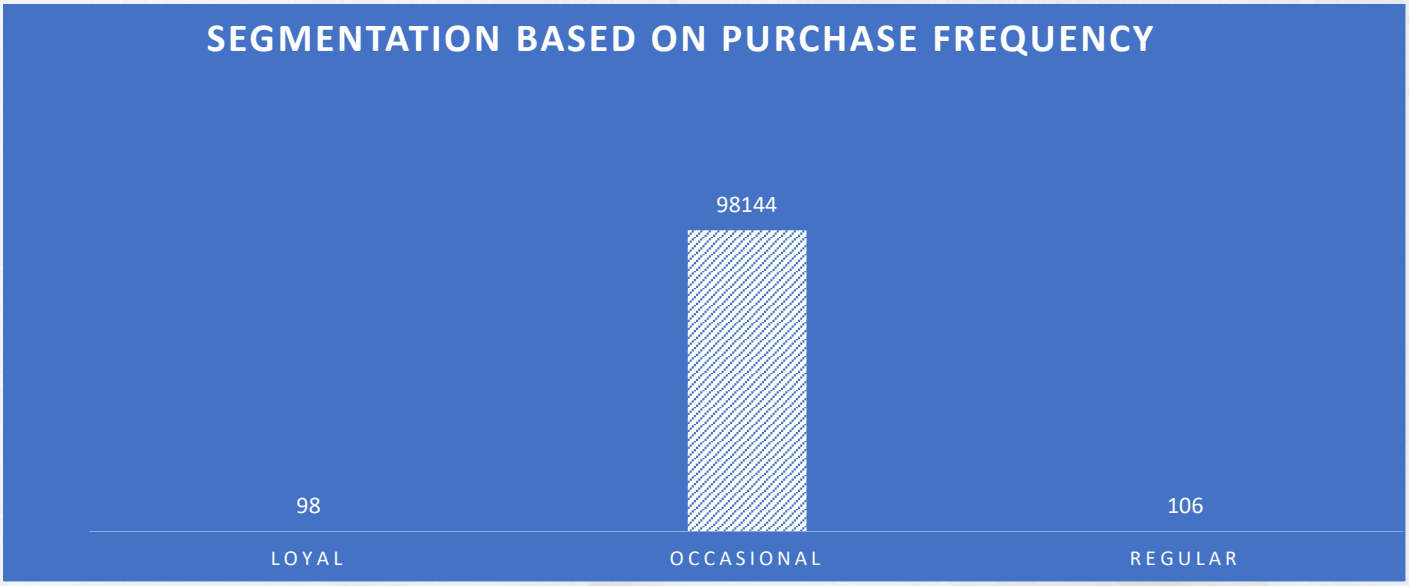
SQL Query:

```
--Setting the search path
SET search_path TO amazon_brazil;

--Using CTE calculate the number of orders for each customer
WITH customer_order_count AS (
    SELECT
        customer_id,
        COUNT(order_id) AS order_count
    FROM orders
    GROUP BY customer_id
),

--2nd CTE to categorize customers based on their order_count
customer_segments AS (
    SELECT
        customer_id,
        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_order_count
)


--Using the customer_segments CTE to group customers by their respective customer_type
SELECT
    customer_type,
    COUNT(customer_id) AS count
FROM customer_segments
GROUP BY customer_type
ORDER BY count DESC;
```

Output:

Customer Type	Customer Count
Occasional	98,144
Regular	106
Loyal	98

Chart:



Recommendation:

- **Targeted Marketing for Regular and Loyal Customers:**

  Implement personalised promotions, such as exclusive discounts or early access to new products, to further engage *Regular* and *Loyal* customers, encouraging them to make more frequent purchases.

- **Increase Engagement with Occasional Customers:**

  Use targeted email campaigns and loyalty programs to incentivise *Occasional* customers to move into the *regular loyal* segments, thereby increasing the overall customer lifetime value.

- **Analyse Purchasing Patterns:**

  Conduct a deeper analysis to identify what products or services are preferred by each segment and adjust product recommendations accordingly to increase satisfaction and sales.

3.5 High-Value Customer Identification

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.

Approach:

- **Calculate the Average Order Value per Customer:**
  - Creating a Common Table Expression (`CTE`) named `Customer_Order_Value` to calculate the average order value for each customer by `joining the Orders` and `Order_Items` tables using the common order_id.
  - Using `AVG(price)` to compute the `average order value` and group by customer_id.
- **Rank Customers Based on Average Order Value:**
  - Using the `RANK()` function to `assign a ranking` to each customer based on their average order value in descending order.
  - The `rank is determined by the highest average order value`, with the customer having the largest average receiving rank 1.
- **Limit the Results to the Top 20 Customers:**
  - Displaying the top 20 customers using `LIMIT 20` to identify high-value customers for the exclusive rewards program

SQL Query:

```
SET search_path TO amazon_brazil;

-- Using CTE to calculate the average order value for each customer
WITH customer_order_value AS (
  SELECT customer_id,
       AVG(price) AS avg_order_value
  FROM Orders o
  JOIN Order_Items oi ON o.order_id = oi.order_id
  GROUP BY customer_id
)

--Using the RANK() function to assign a ranking to each customer based on their average order value
SELECT customer_id,
```

```
    avg_order_value,
    RANK() OVER (ORDER BY avg_order_value DESC) AS customer_rank
FROM customer_order_value
ORDER BY avg_order_value DESC
LIMIT 20;
```

Output:

Customer ID	Average Order Value	Customer Rank
c6e2731c5b391845f6800c97401a43a9	6735.00	1
f48d464a0baaea338cb25f816991ab1f	6729.00	2
3fd6777bbce08a352fddd04e4a7cc8f6	6499.00	3
df55c14d1476a9a3467f131269c2477f	4799.00	4
24bbf5fd2f2e1b359ee7de94defc4a15	4690.00	5
3d979689f636322c62418b6346b1c6d2	4590.00	6
1afc82cd60e303ef09b4ef9837c9505c	4399.87	7
35a413c7ca3c69756cb75867d6311c0d	4099.99	8
e9b0d0eb3015ef1c9ce6cf5b9dcbee9f	4059.00	9
c6695e3b1e48680db36b487419fb0398	3999.90	10
926b6a6fb8b6081e00b335edaf578d35	3999.00	11
3be2c536886b2ea4668eced3a80dd0bb	3980.00	12
31e83c01fce824d0ff786fcd48dad009	3930.00	13
CONTINUE…….	CONTINUE…….	CONTINUE…….

Recommendation:

- **Create a Rewards Program for Top Customers:**

  Design a rewards program offering exclusive benefits like special discounts, early access to sales, or premium support to the top 20 customers to maintain their loyalty and encourage repeat purchases.

- **Identify Purchase Patterns:**

  Analyse the purchase behaviour of these high-value customers to identify any common patterns in the types of products or services they buy. Use this insight to create personalised marketing strategies.

- **Engage with High-Value Customers Regularly:**

  Regularly engage these customers through newsletters, personalised messages, and special offers to keep them informed and incentivised to continue their high spending. Expand the Program to Include More Customers: After a successful pilot with the top 20, consider expanding the program to include more high-value customers, based on adjusted criteria such as top 50 or top 10

3.6 Monthly Cumulative Sales Analysis For Key Products

Problem Statement:

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.

Approach:

To achieve this, we utilise a Common Table Expression (`CTE`) to first `calculate the total monthly sales for each product`. We then apply a `window function` to `compute the cumulative sales for each product month by month`. The SQL query effectively aggregates sales data, allowing us to track the performance of products over time.

Use a Common Table Expression (`CTE) to calculate the monthly sales for each product based on order data`.

Implement a `window function` to compute the `cumulative monthly sales` for each product. Apply the `SUM` window function using `PARTITION BY` product_id and ORDER BY sale_month to maintain a sequential accumulation of sales values.

Output the product ID, sale month, and total cumulative sales in `ascending order` to track performance over time.

SQL Query:

```
-- Setting the search path
SET search_path TO amazon_brazil;

-- Creating a CTE for monthly sales for each product
WITH Monthly_Sales AS (
  SELECT
    ol.product_id,
    EXTRACT(MONTH FROM o.order_purchase_timestamp) AS sale_month,
    SUM(ol.price) AS monthly_sales
  FROM
    orders o
  JOIN
    order_items ol
  ON
  o.order_id = ol.order_id
  GROUP BY
  ol.product_id, sale_month),
```

```
-- Calculating the cumulative monthly sales using a window function
With Cumulative_Sales AS (
   SELECT
      product_id,
      sale_month,
      SUM(monthly_sales) OVER (PARTITION BY product_id ORDER BY sale_month) AS total_sales
   FROM
      Monthly_Sales)

-- Final output of cumulative sales month by month for each product
SELECT
   product_id,
   sale_month,
   total_sales
FROM
   Cumulative_Sales
ORDER BY
   product_id, sale_month;
```

Output:

product_id	sale_month	total_sales
00066f42aeeb9f3007548bb9d3f33c38	5	101.65
00088930e925c41fd95ebfe695fd2655	12	129.90
0009406fd7479715e4bef61dd91f2462	12	229.00
000b8f95fcb9e009648827831776 4d19	8	117.80
000d9be29b5207b54e86aa1b1ac54872	4	199.00
Continue….	Continue….	Continue….

Recommendation:

- **Targeted Marketing Campaigns:**
  Leverage data on high-sales months to create focused marketing efforts, ensuring that promotions align with periods of high consumer interest.

- **Inventory Management:**
  Use cumulative sales insights to optimise inventory levels. Ensure that high-demand products are sufficiently stocked ahead of peak sales months to meet customer demand.

- **Product Performance Review:**
  Conduct regular assessments of cumulative sales data to identify underperforming products. Consider adjustments to marketing, pricing, or product features to enhance their sales potential.

- **Seasonal Strategies:**
  Develop strategies tailored to months with high cumulative sales, such as bundling popular items or offering limited-time discounts during peak sales period.

3.7 Analysing Monthly Sales Growth By Payment Method

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.

Approach:

- **Calculate Monthly Sales:**
  - Use a `CTE` to calculate the total monthly sales for each payment method for the year 2018.
- **Compute Month-over-Month Growth:**
  - Use the `LAG window function` to compare the current month's sales with the previous month for each payment method.
- **Calculate Percentage Change:**
  - Calculate the `percentage change to get the month-over-month growth rate`.
- **Output the Results:**
  - Display payment_type, sale_month, monthly_total, and monthly_change to analyse sales trends for each payment method.

SQL Query:

```
-- Setting the search path to the specified schema
SET search_path TO amazon_brazil;

-- Creating a CTE to calculate monthly sales per payment method for the year 2018
WITH Monthly_Sales AS (
   SELECT
      payment_type,
      DATE_TRUNC('month', order_purchase_timestamp) AS sale_month,
      SUM(payment_value) AS monthly_total
   FROM
      orders o
   JOIN
      payments p
   ON
   o.order_id = p.order_id
   WHERE
      EXTRACT(YEAR FROM order_purchase_timestamp) = 2018
   GROUP BY
     payment_type, sale_month
),
-- Calculating the month-over-month percentage change
```

```
Monthly_Change AS (
    SELECT
        payment_type,
        sale_month,
        monthly_total,
        LAG(monthly_total) OVER (PARTITION BY payment_type ORDER BY sale_month) AS
previous_month_total,
        CASE
            WHEN LAG(monthly_total) OVER (PARTITION BY payment_type ORDER BY sale_month)
IS NOT NULL
            THEN ((monthly_total - LAG(monthly_total) OVER (PARTITION BY payment_type ORDER
BY sale_month)) / LAG(monthly_total) OVER (PARTITION BY payment_type ORDER BY
sale_month)) * 100
            ELSE NULL
        END AS monthly_change
    FROM
        Monthly_Sales)
-- Final output of total sales and month-over-month change for each payment method
SELECT
    payment_type,
    sale_month,
    monthly_total,
    monthly_change
FROM
    Monthly_Change
ORDER BY
    payment_type, sale_month;
```

Output:

payment_type	sale_month	monthly_total	monthly_change
boleto	2018-01-01	170651.40	NULL
boleto	2018-02-01	153165.77	-10.25
boleto	2018-03-01	157807.38	3.03
boleto	2018-04-01	162940.61	3.25
boleto	2018-05-01	166571.60	2.23
boleto	2018-06-01	126379.90	-24.13
boleto	2018-07-01	162938.42	28.93
boleto	2018-08-01	118214.12	-27.45
credit_card	2018-01-01	760252.76	NULL
credit_card	2018-02-01	680198.99	-10.53
credit_card	2018-03-01	813565.25	19.61

Recommendation:

- **Optimise Payment Method Offerings:**
    - Focus on promoting the most stable and high-growth payment methods like credit card, which show consistent performance across months.

- **Develop Targeted Campaigns for Declining Methods:**
    - Implement targeted campaigns or discounts for declining payment methods like voucher, which has shown a sharp drop in sales, to revitalise interest.

- **Monitor and Adapt to Seasonality Trends:**
    - Regularly monitor the sales trends for each payment method to identify seasonal or periodic trends (e.g., debit card sales spike in June). Utilise this information to adapt marketing strategies and payment offerings.

---------- Thank You --------