Amazon Analysis Using MySQL

Amazon India is analysing customer and sales data from Amazon Brazil to identify key trends that can be leveraged in the Indian market. The main objective of this analysis is to know customer behaviours, product preferences, and payment patterns to improve customer experience and capture new market opportunities in India.

This project uses multiple tables: Customers, Orders, Order Items, Product, Sellers, and Payments. Through SQL queries, various business-critical questions and providing actionable insights are shared.

Analysis1

Que1:

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

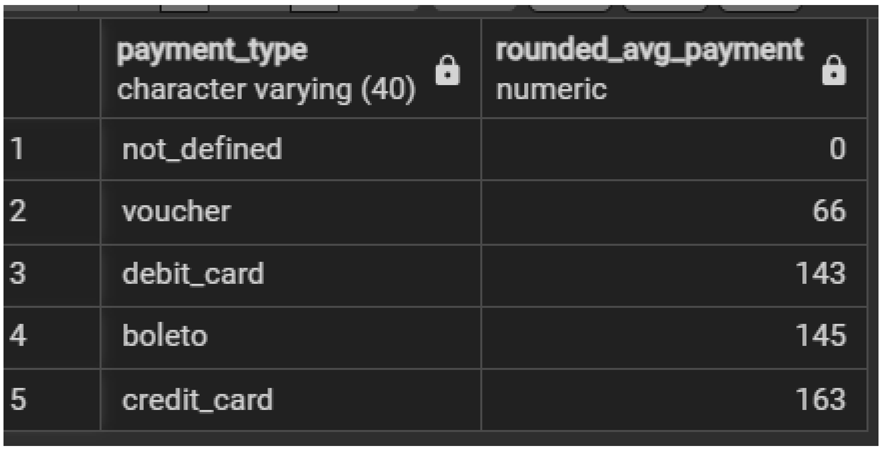
Approach:

1. **Identifying Relevant Tables and Columns:**
    - **Table: payments**
    - **Columns: payment_type, payment_value**
2. **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
3. **Rounding the Averages:**
    - Used **the ROUND()** function to round the average payment values to the nearest integer.
4. **Sort the Results:**
    - Ordered the final results in ascending order based on the rounded average payment.

SQL Query:

```
select payment_type,

Round(avg(payment_value),0) as

rounded_avg_payment from amazon_brazil.payments

group by payment_type

order by rounded_avg_payment asc;
```

Output:



	payment_type character varying (40)	rounded_avg_payment numeric
1	not_defined	0
2	voucher	66
3	debit_card	143
4	boleto	145
5	credit_card	163

Recommendations:

1. **Focus on Popular Payment Methods:**
   - Try making the payment methods like **credit_card** and **boleto** optimised, as their average payment amounts are higher.
   - Introduce targeted promotions for customers using these methods to make higher revenue.
2. **Improve Use of Underperforming Methods:**

   - For methods like **debit_card**, open opportunities for growth by giving promotional discounts or improving transaction processes.

Que2:

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.

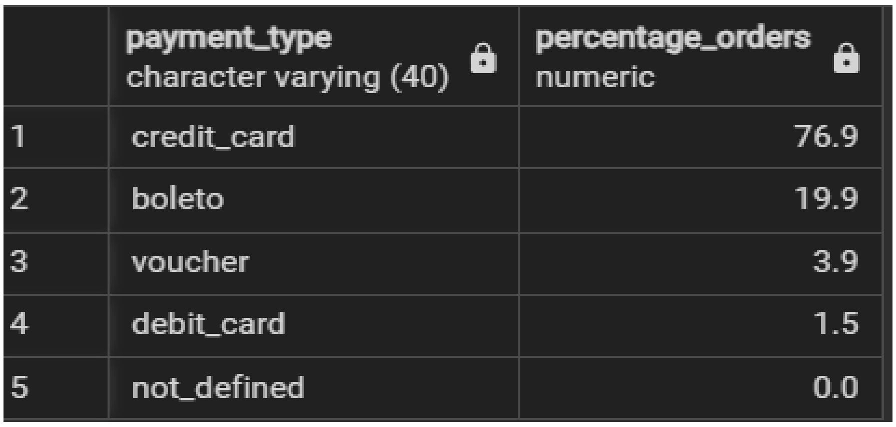
Approach:

1. **Calculating Total Number of Orders:**
   - Determined the total count of all orders across payment types.
2. **Calculating Orders per Payment Type:**
   - Used **COUNT(*)** to get the number of orders for each payment_type
3. **Calculating Percentage of Orders:**
   - Divided the count of orders per payment type by the total number of orders, multiplied by 100, and rounded to one decimal place.
4. **Sorting Results:**
   - Ordered the results in descending order of percentage to highlight the most popular payment methods.

SQL Query:

```
SELECT payment_type,
ROUND(CAST(COUNT(DISTINCT order_id) AS NUMERIC) * 100.0 /
 (SELECT COUNT(DISTINCT order_id) FROM amazon_brazil.payments), 1)
AS percentage_orders FROM amazon_brazil.payments
GROUP BY payment_type
ORDER BY percentage_orders DESC;
```

Output:



	payment_type character varying (40)	percentage_orders numeric
1	credit_card	76.9
2	boleto	19.9
3	voucher	3.9
4	debit_card	1.5
5	not_defined	0.0

Recommendations:

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**, **vouche**r, 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.

Que3:

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.

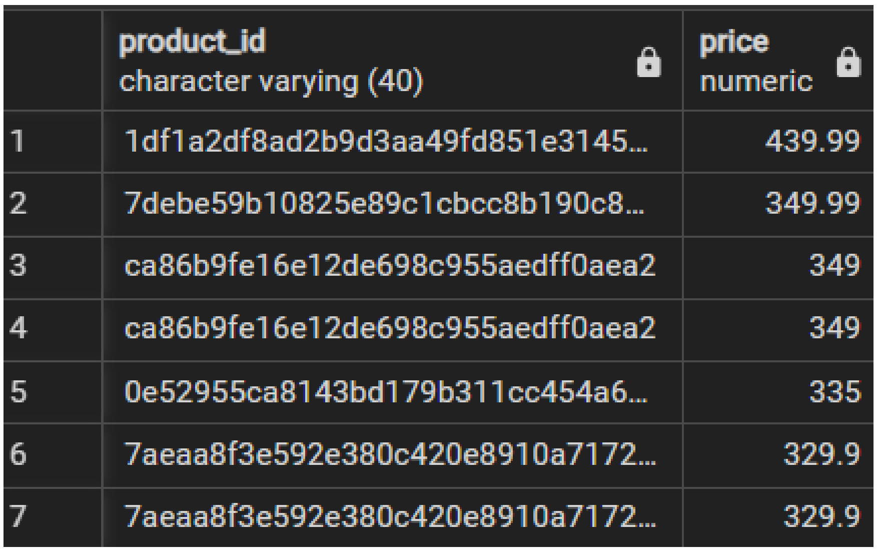
Approach:

1. **Filtering Products by Price Range**:
   - Select products priced between **100 and 500 BRL** using the BETWEEN clause.
2. **Searching for the Word 'Smart'**:
   - Use LIKE '%smart%' to filter product names that contain the word 'Smart', making it case-insensitive.
3. **Join Orders and Products**:
   - Perform an **INNER JOIN** between the Order Items and Product tables on the product_id column to link product details with order prices.
4. **Sort Results**:
   - Sort the filtered products in **descending order** by price to display the most expensive products first.

SQL Query:

```
select o.product_id, o.price

from amazon_brazil.order_items as o

join amazon_brazil.product  as p

on o.product_id=p.product_id

where o.price  between 100 and 500

and p.product_category_name like '%smart%'

order by o.price Desc;
```

Output:



	product_id character varying (40)	price numeric
1	1df1a2df8ad2b9d3aa49fd851e3145...	439.99
2	7debe59b10825e89c1cbcc8b190c8...	349.99
3	ca86b9fe16e12de698c955aedff0aea2	349
4	ca86b9fe16e12de698c955aedff0aea2	349
5	0e52955ca8143bd179b311cc454a6...	335
6	7aeaa8f3e592e380c420e8910a7172...	329.9
7	7aeaa8f3e592e380c420e8910a7172...	329.9

Recommendations:

1. **Promotional Campaign**:

   - Focus promotions on **"Smart"** products in the specified price range to attract cost-conscious customers seeking smart features.

2. **Highlighting Mid-Range Products**:

   - Promote these products as affordable yet feature-rich options for customers.

Que4:

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:

1. **Extracting Month from Purchase Date**:
   - Using **EXTRACT(MONTH FROM order_purchase_timestamp)** to retrieve the month for each order.

2. **Calculating Total Sales per Month**:
   - Aggregating **SUM(price)** to compute total sales for each month.

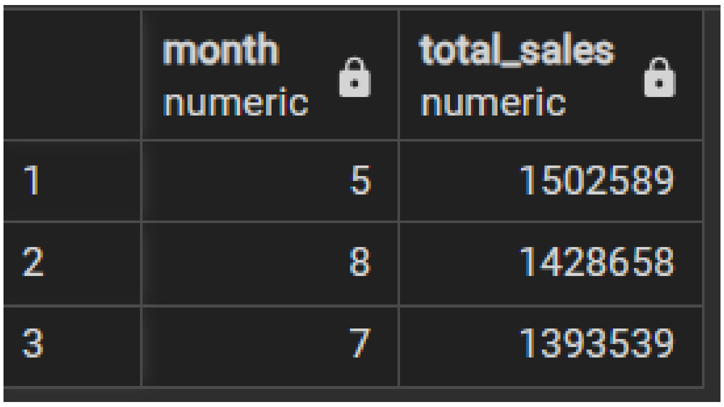
   2. **Rounding Sales Value:**
   - Applying **ROUND()** to round total sales to the nearest integer.

4. **Sorting and Filtering Top 3**:
   - Ordering results in descending order of sales and using **LIMIT 3** to display the top 3 months.

SQL Query:

```
select extract (month  from  order_purchase_timestamp) as month,

round(sum(oi.price)) as total_sales

from amazon_brazil.orders as o

join amazon_brazil.order_items as oi

on o.order_id=oi.order_id

group by extract (month  from  order_purchase_timestamp)

order by total_sales desc limit 3;
```

Output:



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

Recommendations:

1. **Target Promotions for Peak Months**:

   - Allocate more **marketing** and promotion budgets for the top-performing months to maximize revenue.

3. **Analyze Trends**:

   - Investigate **customer behavior** in these months to understand what drives high sales (e.g., festivals, holidays, or sales events)

Que5:

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:

1. **Joining Products with Order Items**:
   - Joining Product and Order Items tables on product_id to associate product categories with prices.
2. **Calculating Price Difference**:
   - Using **MAX(oi.price) - MIN(oi.price)** to compute the price variation for each category.
3. **Filtering Categories**:
   - Applying **HAVING** clause to select categories with a price difference greater than **500 BRL**.
4. **Sorting Results**:
   - Ordering categories by **price_difference** in descending order

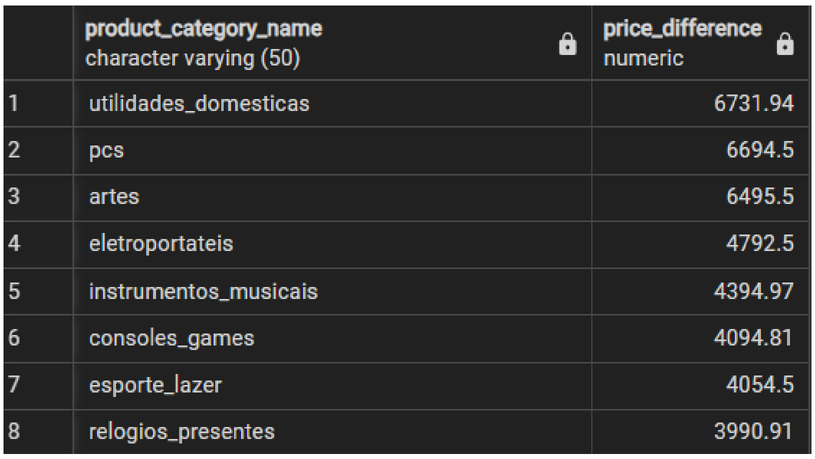
SQL Query:

```
select product_category_name ,
 max(oi.price)- min(oi.price) as price_difference from
amazon_brazil.product p
join amazon_brazil.order_items oi
on p.product_id=oi.product_id
group by p.product_category_name
having max(oi.price) - min(oi.price)> 500
order by price_difference desc;
```

Output:

The analysis revealed significant price variations across product categories, with the top categories showing a difference greater than 500 BRL. Key highlights:
Top Categories:

- Utilidades Domésticas: 6731.94 BRL
- PCS: 6694.5 BRL
- Artes: 6495.5 BRL



	product_category_name character varying (50)	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

Recommendations:

1.Focus on High-Variation Categories:

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

2.Enhance Product Offerings:

- Expand the range of products in categories like "eletroportateis" and "instrumentos_musicais" to cover diverse price segments.

Que6:

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.

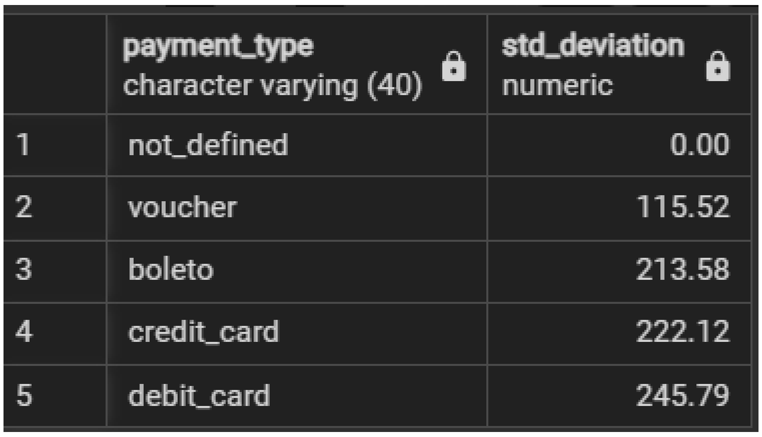
Approach:

1. **Calculating Standard Deviation**:
   - Using **STDDEV**(payment_value) to calculate the standard deviation of transaction amounts for each payment_type.
2. **Grouping by Payment Type**:
   - Grouping the results by payment_type to calculate the standard deviation for each type separately.
3. **Sorting Results**:
   - Ordering the results in **ascending order** of standard deviation to identify the most consistent payment types first.
4. **Rounding the Output**:
   - Applying **ROUND()** to round the standard deviation to **2 decimal places** for clarity.

SQL Query:

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

Output:



	payment_type character varying (40)	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 Stable Payment Methods:**
   - Highlight payment methods with low variance, such as **voucher**, as reliable and consistent for transactions.
2. **Address Issues with High-Variance Methods:**
   - Investigate reasons for high variance in methods like **debit_card** and ensure better user guidance during transactions.

Que7:

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.

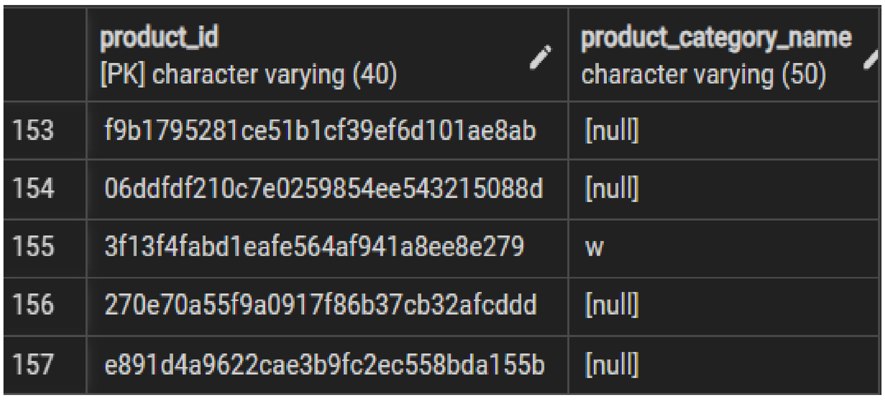
Approach:

1. **Filtering Missing Names**:
   - Using **IS NULL** to identify products where the product_category_name is missing.
2. **Filtering Single-Character Names**:
   - Using **LENGTH(product_category_name) = 1** to find product names with only a single character.
3. **Combining Filters**:
   - Using **OR** to combine both conditions and retrieve products matching either case.
4. **Retrieving Relevant Columns**:
   - Selecting product_id and product_category_name for better identification and resolution.

SQL Query:

```
SELECT product_id, product_category_name

FROM amazon_brazil.product

WHERE

product_category_name IS NULL

OR LENGTH(product_category_name) = 1;
```

Output:



	product_id [PK] character varying (40)	product_category_name character varying (50)
153	f9b1795281ce51b1cf39ef6d101ae8ab	[null]
154	06ddfdf210c7e0259854ee543215088d	[null]
155	3f13f4fabd1eafe564af941a8ee8e279	w
156	270e70a55f9a0917f86b37cb32afcddd	[null]
157	e891d4a9622cae3b9fc2ec558bda155b	[null]

Recommendations:

1. **Fix Missing Category Names:**
   - Update the **NULL** entries with appropriate category names to maintain data completeness.
2. **Resolve Single-Character Names:**
   - Review products with **single-character** names and replace them with full category descriptions.
3. **Improve Data Validation:**
   - Implement **data validation rules** to prevent incomplete or incorrect entries during data entry or import processes.

Analysis2

Que1:

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.

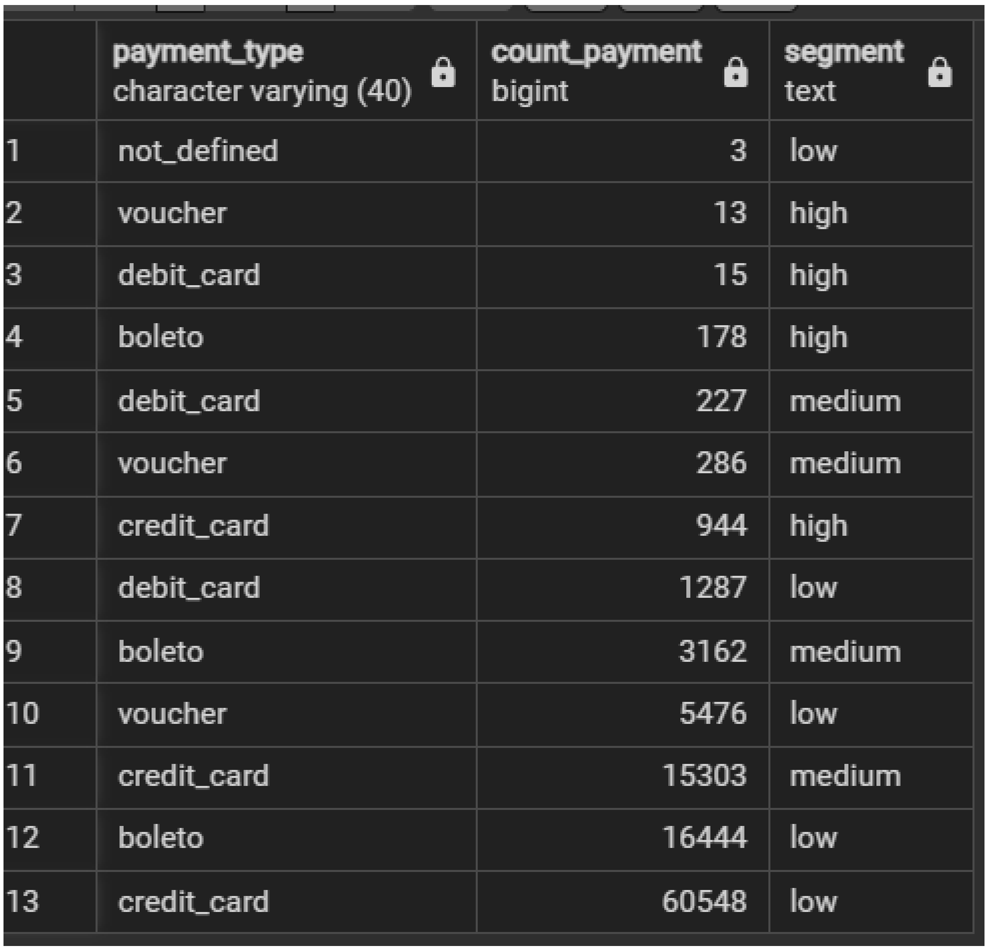
Approach:

1. **Segmenting Order Values**:
   - Used a **CASE** statement to categorize **payment_value** into three segments: low, medium, and high.
2. **Counting Payment Types**:
   - Used **COUNT(*)** to calculate the total number of transactions for each payment_type within each segment.
3. **Grouping by Payment Type and Segment**:
   - Grouped data by **payment_type** and the segment created in the CASE statement.
4. **Sorting Results**:
   - Ordered results by the count of payment types **(count_payment)** in descending order.

SQL Query:

```
select payment_type, count(*) as count_payment,
CASE
 when payment_value < 200 THEN 'low'
 when payment_value Between 200 AND 1000 THEN 'medium'
 when payment_value > 100 THEN 'high'
 END AS segment from amazon_brazil.payments
 group by payment_type, CASE
 when payment_value < 200 THEN 'low'
 when payment_value Between 200 AND 1000 THEN 'medium'
 when payment_value > 100 THEN 'high'
 END
 order by count_payment;
```

Output:



	payment_type character varying (40)	count_payment bigint	segment text
1	not_defined	3	low
2	voucher	13	high
3	debit_card	15	high
4	boleto	178	high
5	debit_card	227	medium
6	voucher	286	medium
7	credit_card	944	high
8	debit_card	1287	low
9	boleto	3162	medium
10	voucher	5476	low
11	credit_card	15303	medium
12	boleto	16444	low
13	credit_card	60548	low

Recommendations:

1. **Boost High-Segment Payments**:
   - Develop strategies to increase high-value transactions by offering premium services, financing options, or loyalty programs.
2. **Uniform Payment Experience**:
   - Ensure all payment types are smooth and reliable across value segments to cater to diverse customer needs.

Que2:

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.

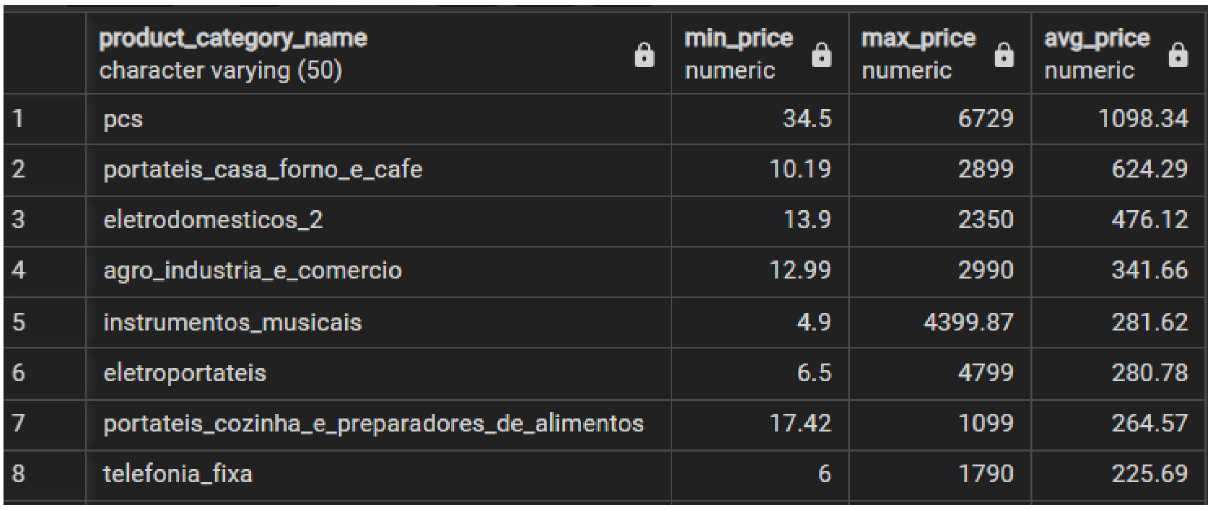
Approach:

1. **Joining Products with Order Items**:
   - Using a JOIN between the **Product** and **Order Items** tables on **product_id** to link categories with prices.
2. **Calculating Price Statistics**:
   - Applying aggregate functions:
     - **MIN**(oi.price) for the lowest price.
     - **MAX**(oi.price) for the highest price.
     - **AVG**(oi.price) for the average price of products in each category.
3. **Rounding Average Price**:
   - Using **ROUND()** to format the average price to **2 decimal places** for clarity.
4. **Grouping and Sorting**:
   - Grouping by **product_category_name** to calculate statistics per category.
   - Sorting the results in descending order of **avg_price.**

SQL Query:

```
SELECT 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.product p

      join amazon_brazil.order_items oi on p.product_id=oi.product_id

GROUP BY product_category_name

ORDER BY avg_price DESC;
```

Output:



	product_category_name character varying (50)	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_preparadores_de_alimentos	17.42	1099	264.57
8	telefonia_fixa	6	1790	225.69

Recommendations:

1. Focus on categories with high average prices to maximize revenue opportunities.
2. Highlight budget-friendly categories for customers seeking affordable options.
3. Maintain a balanced product mix to cater to different customer segments.

Que3:

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.

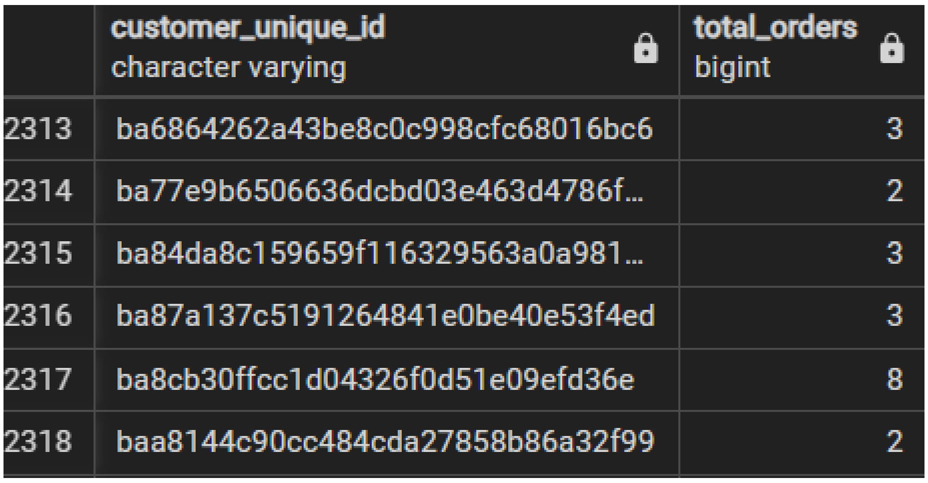
Approach:

1. **Joining Orders with Customers**:
    - Using a **JOIN** between the **Orders** and **Customer** tables on customer_id to associate orders with unique customer IDs.
2. **Counting Total Orders**:
    - Using **COUNT**(order_id) to calculate the total number of orders placed by each customer.
3. **Filtering Customers with Multiple Orders**:
    - Applying **HAVING COUNT**(order_id) > 1 to include only customers who have placed more than one order.
4. **Grouping by Unique ID**:
    - Grouping by **customer_unique_id** to calculate the total orders per customer.

SQL Query:

```
SELECT customer_unique_id,

COUNT(order_id) AS total_orders

FROM amazon_brazil.orders o

JOIN amazon_brazil.customer c

ON o.customer_id = c.customer_id

GROUP BY customer_unique_id

HAVING COUNT(order_id) > 1;
```

Output:



	customer_unique_id character varying	total_orders bigint
2313	ba6864262a43be8c0c998cfc68016bc6	3
2314	ba77e9b6506636dcbd03e463d4786f…	2
2315	ba84da8c159659f116329563a0a981…	3
2316	ba87a137c5191264841e0be40e53f4ed	3
2317	ba8cb30ffcc1d04326f0d51e09efd36e	8
2318	baa8144c90cc484cda27858b86a32f99	2

Recommendations:

1. **Identify Loyal Customers**:
   - Use the list of repeat customers to offer loyalty programs and personalized incentives.
2. **Target High-Frequency Buyers**:
   - Focus marketing efforts on customers with higher order counts for cross-selling and upselling opportunities.
3. **Segment Repeat Buyers**:
   - Group customers based on their purchase frequency for tailored promotions.
4. **Encourage Customer Retention**:
   - Provide discounts or rewards to retain customers who make frequent purchases.

Que4:

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.

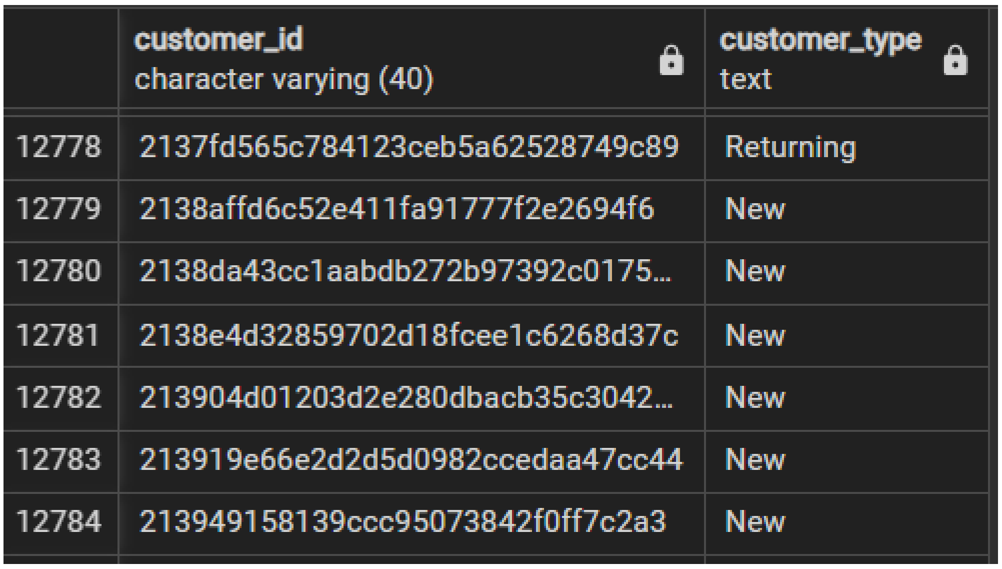
Approach:

1. **Calculating Total Orders**:
   - Use a **temporary table** (CustomerOrderCounts) to calculate the total number of orders for each customer_id.
2. **Defining Categories**:
   - Use a CASE statement to categorize customers as **New**, **Returning**, or **Loyal** based on their order count.
3. **Joining with Customers**:
   - Join the categorized data with the customers table for seamless integration.
4. **Sorting Results**:
   - Order the output by **customer_id** for better readability.

SQL Query:

```
WITH CustomerOrderCounts AS (
   SELECT customer_id,
COUNT(order_id) AS total_orders
   FROM amazon_brazil.orders
   GROUP BY customer_id )
SELECT customer_id,
     CASE
         WHEN total_orders = 1 THEN 'New'
         WHEN total_orders BETWEEN 2 AND 4 THEN 'Returning'
 ELSE 'Loyal'
     END AS customer_type
FROM CustomerOrderCounts
ORDER BY customer_id;
```

Output:



	customer_id character varying (40)	customer_type text
12778	2137fd565c784123ceb5a62528749c89	Returning
12779	2138affd6c52e411fa91777f2e2694f6	New
12780	2138da43cc1aabdb272b97392c0175...	New
12781	2138e4d32859702d18fcee1c6268d37c	New
12782	213904d01203d2e280dbacb35c3042...	New
12783	213919e66e2d2d5d0982ccedaa47cc44	New
12784	213949158139ccc95073842f0ff7c2a3	New

Recommendations:

1. **Target New Customers**:
   - Offer onboarding incentives like discounts or free delivery to encourage repeat purchases.
2. **Retain Returning Customers**:
   - Provide personalized offers or rewards for returning customers to boost their engagement.
3. **Reward Loyal Customers**:
   - Implement loyalty programs to reward frequent buyers and encourage continued spending.

Que5:

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.

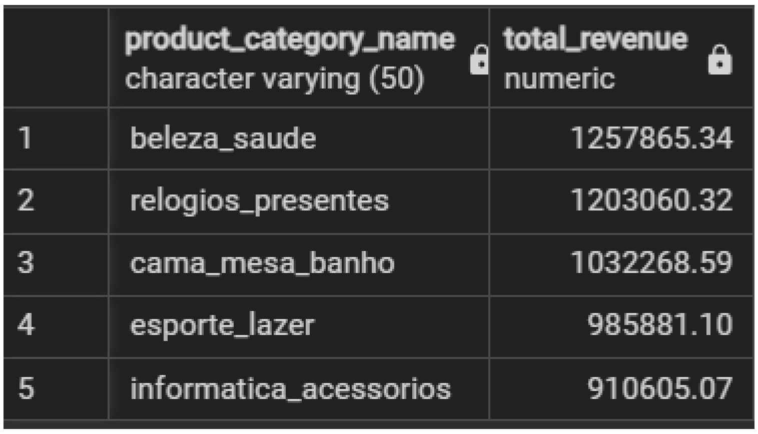
Approach:

1. **Join Products with Order Items**:
    - Use a JOIN between the **Product** and **Order Items** tables on product_id to link product categories with their revenue.
2. **Calculate Total Revenue**:
    - Use **SUM**(oi.price) to calculate the total revenue for each product category.
3. **Group by Product Category**:
    - Group by product_category_name to aggregate the revenue for each category.
4. **Sort and Limit Results**:
    - Order the results in descending order of total revenue and use **LIMIT 5** to display the top 5 categories.

SQL Query:

```
select p.product_category_name, sum(oi.price)

as total_revenue from amazon_brazil.product p

join amazon_brazil.order_items oi

on p.product_id=oi.product_id

group by p.product_category_name

order by total_revenue desc

limit 5;
```

Output:



	product_category_name character varying (50)	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. **Focus on High-Revenue Categories**:
   - Allocate marketing budgets and resources to the top-performing categories.
2. **Optimize Inventory**:
   - Ensure sufficient stock of high-revenue categories to prevent potential revenue loss.
3. **Cross-Sell Opportunities**:
   - Bundle products from top categories with complementary products to boost sales further.

.

Analysis 3

Que1:

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.

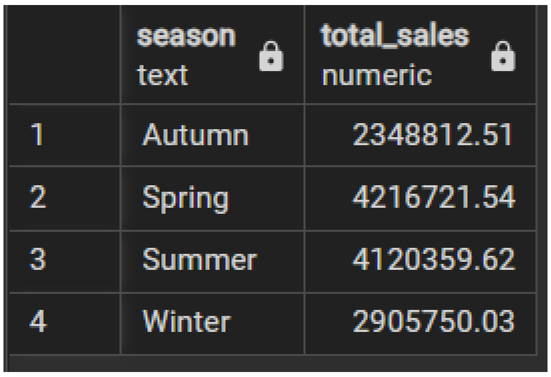
Approach:

1. **Identify Seasons**:
    - Use a CASE statement to classify orders into Spring, Summer, Autumn, and Winter based on the month extracted from **order_purchase_timestamp**.
2. **Join Orders and Order Items**:
    - Use an INNER JOIN to combine data from **Orders** and **Order Items** tables using order_id.
3. **Calculate Total Sales**:
    - Use **SUM** (oi.price) to calculate total sales for each season.
4. **Group and Sort**:
    - Group data by **season** and sort the output by the seasonal order.

SQL Query:

```
SELECT

   CASE

      WHEN EXTRACT(MONTH FROM o.order_purchase_timestamp) IN (3, 4, 5) THEN
'Spring'

      WHEN EXTRACT (MONTH FROM o.order_purchase_timestamp) IN (6, 7, 8) THEN
'Summer'

      WHEN EXTRACT(MONTH FROM o.order_purchase_timestamp) IN (9, 10, 11)
THEN 'Autumn'

      ELSE 'Winter'

   END AS season,

   SUM(oi.price) AS total_sales

FROM   amazon_brazil.orders o

INNER JOIN amazon_brazil.order_items oi

ON o.order_id = oi.order_id

GROUP BY season ORDER BY  season;
```

Output:



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

Recommendations:

1.Allocate Seasonal Marketing Resources:

- Use the sales data to plan promotions during high-revenue seasons.

2. Boost Low-Season Sales:

- Introduce discounts or exclusive deals to drive sales during low-performing seasons.

3. Targeted Advertising Campaigns:

- Focus advertising efforts on customer preferences for specific seasons.

Que2:

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.

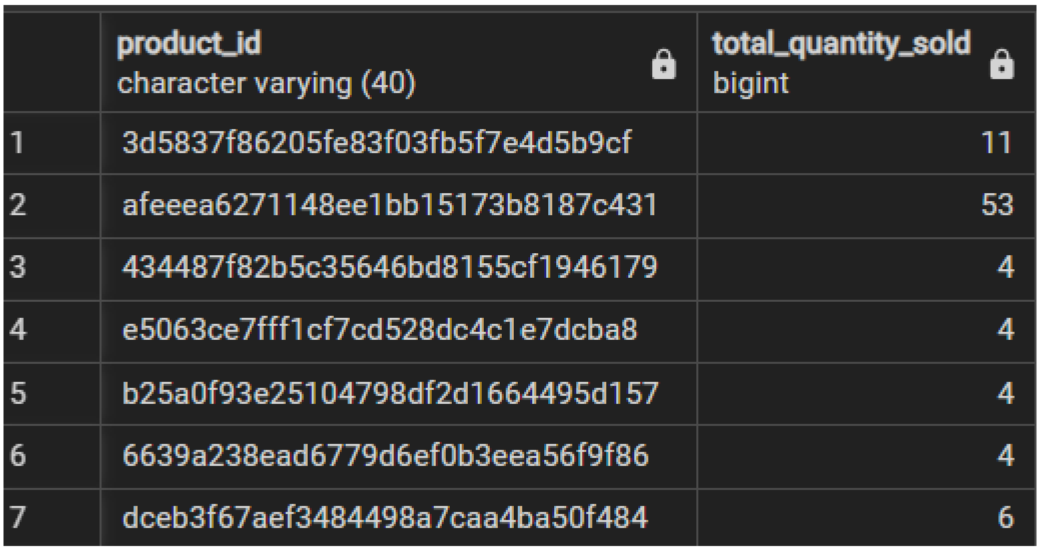
Approach:

1. **Calculating Total Quantity Sold Per Product**:
   - Using **COUNT** (order_item_id) grouped by product_id to calculate the total quantity sold for each product.
2. **Calculating Overall Average Quantity**:
   - Using a **subquery** to calculate the **average total quantity sold** across all products.
3. **Filtering Above-Average Products**:
   - Using a **WHERE** clause to include only products where the total quantity sold is greater than the overall average.
4. **Organizing Results**:
   - Selecting and displaying product_id along with total_quantity_sold for **above-average products**

SQL Query:

```
SELECT product_id, total_quantity_sold

FROM (

   SELECT product_id, COUNT(order_item_id) AS total_quantity_sold

   FROM amazon_brazil.order_items

   GROUP BY product_id)

WHERE total_quantity_sold > (

 SELECT AVG(total_quantity_sold)

FROM ( SELECT COUNT(order_item_id) AS total_quantity_sold

FROM amazon_brazil.order_items

GROUP BY product_id ));
```

Output:



	product_id character varying (40)	total_quantity_sold bigint
1	3d5837f86205fe83f03fb5f7e4d5b9cf	11
2	afeeea6271148ee1bb15173b8187c431	53
3	434487f82b5c35646bd8155cf1946179	4
4	e5063ce7fff1cf7cd528dc4c1e7dcba8	4
5	b25a0f93e25104798df2d1664495d157	4
6	6639a238ead6779d6ef0b3eea56f9f86	4
7	dceb3f67aef3484498a7caa4ba50f484	6

Recommendations:

1. Prioritize High-Selling Products:

- Focus on replenishing stock for above-average products to meet demand.

2. Analyze Trends for Top Products:

- Investigate why these products are performing better to replicate success for other products.

3. Allocate Marketing Resources:

- Direct marketing campaigns toward high-performing products to maximize sales.

Que3:

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:

1. **Filtering Data for 2018**:

- Used **EXTRACT** (YEAR FROM o.order_purchase_timestamp) to filter orders from the year 2018.

2. Extracting Month:

- Applied EXTRACT (MONTH FROM o.order_purchase_timestamp) to identify the month for each order.

3. Calculating Total Revenue:

- Used SUM (oi.price) to calculate total revenue for each month.

4. Grouping and Sorting Data:

- Grouped by month to aggregate revenue and ordered the results by month for chronological representation.

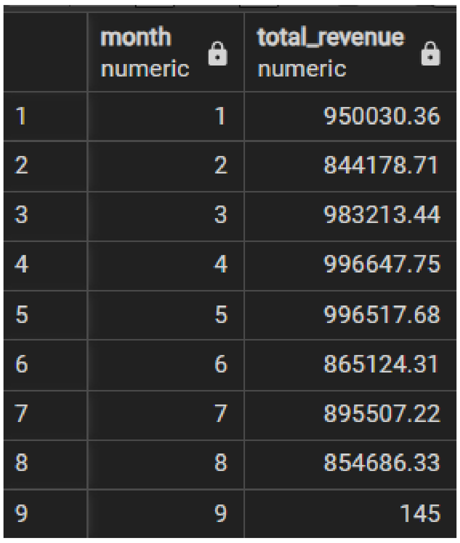
5. Exporting and Visualization:

- Exported the results to Excel and created a graph to represent monthly revenue changes.

SQL Query:

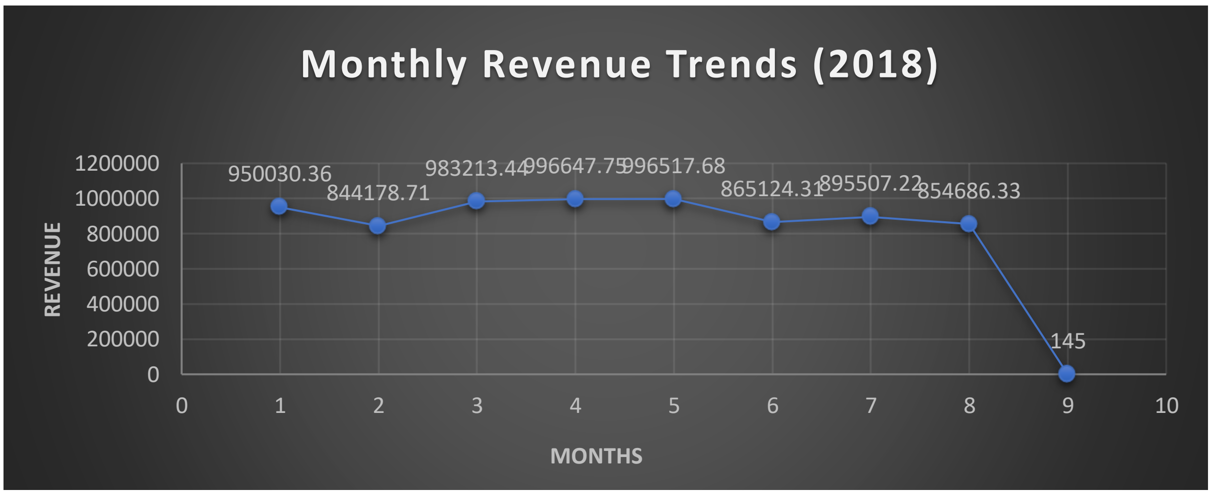
```
SELECT EXTRACT(MONTH FROM o.order_purchase_timestamp) AS month,
 SUM(oi.price) AS total_revenue
FROm amazon_brazil.orders o
JOIN amazon_brazil.order_items oi
ON o.order_id = oi.order_id
WHERE EXTRACT(YEAR FROM o.order_purchase_timestamp) = 2018
GROUP BY month
ORDER BY month;
```

Output:



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

Graphical Visualization:



Recommendations:

1. Seasonal Planning:

- Use the revenue data to plan inventory and promotions ahead of seasonal trends.

2.Analyze Customer Behaviour:

- Study customer preferences during peak months to tailor future strategies.

Que4:

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:

1. Calculating Order Frequency:

- Group by customer_id and count the number of orders for each customer.

2. Classifying Customers:

- Use a CASE statement in SQL or equivalent logic to classify customers as **Occasional**, **Regular**, or **Loyal** based on the number of orders.

3. Aggregating Segment Counts:

- Count the number of customers in each segment for proportion analysis.

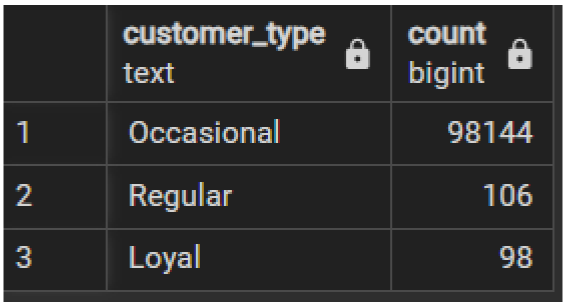
4. **Export and Visualize**:

- Export the results to Excel and create a chart to visually represent the proportion of customer segments.

SQL Query:

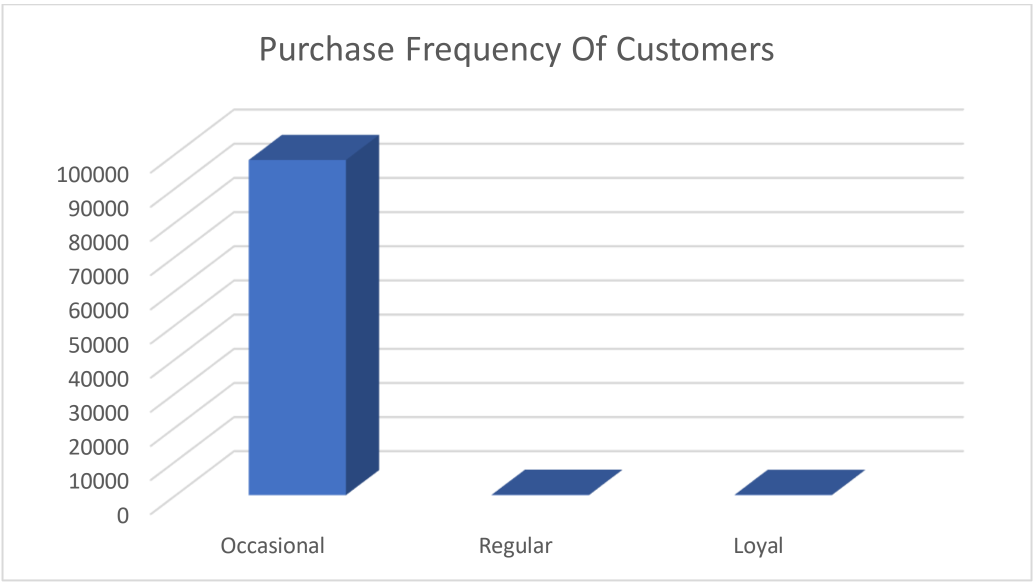
```
WITH purchase_frequency AS (
   SELECT customer_id,
   COUNT(order_id) AS order_count
   FROM amazon_brazil.orders
   GROUP BY customer_id)
SELECT customer_id,
     CASE
        WHEN order_count BETWEEN  1  AND 2 THEN 'Occasional'
        WHEN order_count BETWEEN 3 AND 5 THEN 'Regular'
        ELSE 'Loyal'
     END AS customer_type
FROM purchase_frequency
ORDER BY order_count;
```

Output:



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

Graphical Visualization:



Recommendations:

1. Engage Occasional Customers:

- Run personalized campaigns to convert occasional buyers into regular customers.

2. Reward Loyal Customers:

- Provide exclusive rewards for loyal customers to retain them and encourage long-term engagement.

Que5:

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.

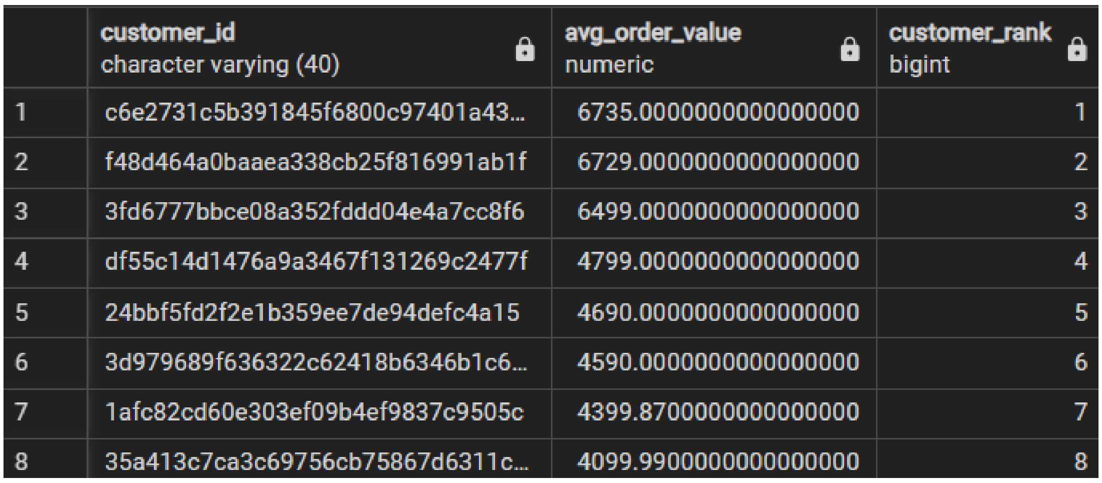
Approach:

1. **Calculating Average Order Value**:
   - Use **AVG** (price) to calculate the average order value for each customer by combining data from the **Orders** and **Order Items** tables.
2. **Ranking Customers**:
   - Use the **RANK ()** window function to assign a rank to customers based on their avg_order_value in descending order.
3. **Limit to Top 20 Customers**:
   - Use **LIMIT 20** to retrieve only the top 20 high-value customers.
4. **Organize Results**:
   - Select and display customer_id, avg_order_value, and customer_rank for the top 20 customers.

SQL Query:

```
WITH CustomerOrderValue AS (
 SELECT customer_id,
 AVG (price) AS avg_order_value
 FROM amazon_brazil.orders o
    JOIN amazon_brazil.order_items oi
ON o.order_id = oi.order_id
 GROUP BY customer_id )
SELECT customer_id, avg_order_value,
 RANK () OVER (ORDER BY avg_order_value DESC)
 AS customer_rank FROM CustomerOrderValue
ORDER BY avg_order_value DESC LIMIT 20;
```

Output:



	customer_id character varying (40)	avg_order_value numeric	customer_rank bigint
1	c6e2731c5b391845f6800c97401a43...	6735.0000000000000000	1
2	f48d464a0baaea338cb25f816991ab1f	6729.0000000000000000	2
3	3fd6777bbce08a352fddd04e4a7cc8f6	6499.0000000000000000	3
4	df55c14d1476a9a3467f131269c2477f	4799.0000000000000000	4
5	24bbf5fd2f2e1b359ee7de94defc4a15	4690.0000000000000000	5
6	3d979689f636322c62418b6346b1c6...	4590.0000000000000000	6
7	1afc82cd60e303ef09b4ef9837c9505c	4399.8700000000000000	7
8	35a413c7ca3c69756cb75867d6311c...	4099.9900000000000000	8

Recommendations:

1. **Reward High-Value Customers**:
   - Create exclusive rewards and loyalty benefits to retain top customers.
2. **Offer Personalized Experiences**:
   - Provide personalized offers or premium support to high-value customers to enhance satisfaction.
3. **Monitor Spending Patterns**:
   - Analyze the purchasing behavior of these customers to tailor marketing strategies.

Que6:

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.

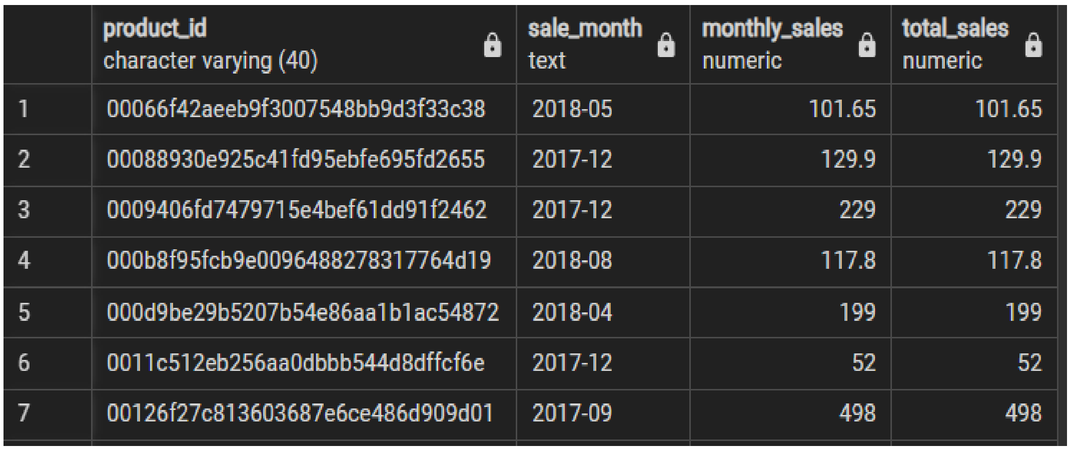
Approach:

1. **Calculate Monthly Sales**:
    - Use a **Common Table Expression (CTE)** to calculate the total sales (monthly_sales) for each product for each month by grouping data by product_id and month.
2. **Extract and Format Month**:
    - Use **DATE_TRUNC**('month', order_purchase_timestamp) to extract the month from the purchase date and format it as YYYY-MM for readability.
3. **Compute Cumulative Sales**:
    - Use a **window function (SUM() OVER)** to calculate cumulative sales (total_sales) for each product, ordered by month.
4. **Organize Results**:
    - Select and display product_id, sale_month, and total_sales for each product month by month.

SQL Query:

```
WITH MonthlySales AS (

   SELECT product_id,

  TO_CHAR(DATE_TRUNC('month', o.order_purchase_timestamp), 'YYYY-MM') AS
sale_month,

  SUM(oi.price) AS monthly_sales

 FROM amazon_brazil.orders o

JOIN amazon_brazil.order_items oi ON o.order_id = oi.order_id

GROUP BY product_id, sale_month )

SELECT product_id,  sale_month, monthly_sales,

 SUM(monthly_sales) OVER (PARTITION BY product_id ORDER BY sale_month) AS
total_sales

FROM MonthlySales  ORDER BY product_id, sale_month;
```

Output:



	product_id character varying (40)	sale_month text	monthly_sales numeric	total_sales numeric
1	00066f42aeeb9f3007548bb9d3f33c38	2018-05	101.65	101.65
2	00088930e925c41fd95ebfe695fd2655	2017-12	129.9	129.9
3	0009406fd7479715e4bef61dd91f2462	2017-12	229	229
4	000b8f95fcb9e0096488278317764d19	2018-08	117.8	117.8
5	000d9be29b5207b54e86aa1b1ac54872	2018-04	199	199
6	0011c512eb256aa0dbbb544d8dffcf6e	2017-12	52	52
7	00126f27c813603687e6ce486d909d01	2017-09	498	498

Recommendations:

1. **Identify Growth Trends**:
   - Use cumulative sales data to track the lifecycle and growth trends of key products.
2. **Focus on High-Growth Products**:
   - Allocate resources and marketing efforts to products showing consistent monthly growth.
3. **Manage Inventory Effectively**:
   - Use sales trends to predict demand and optimize inventory management for key products.

Que7:

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.

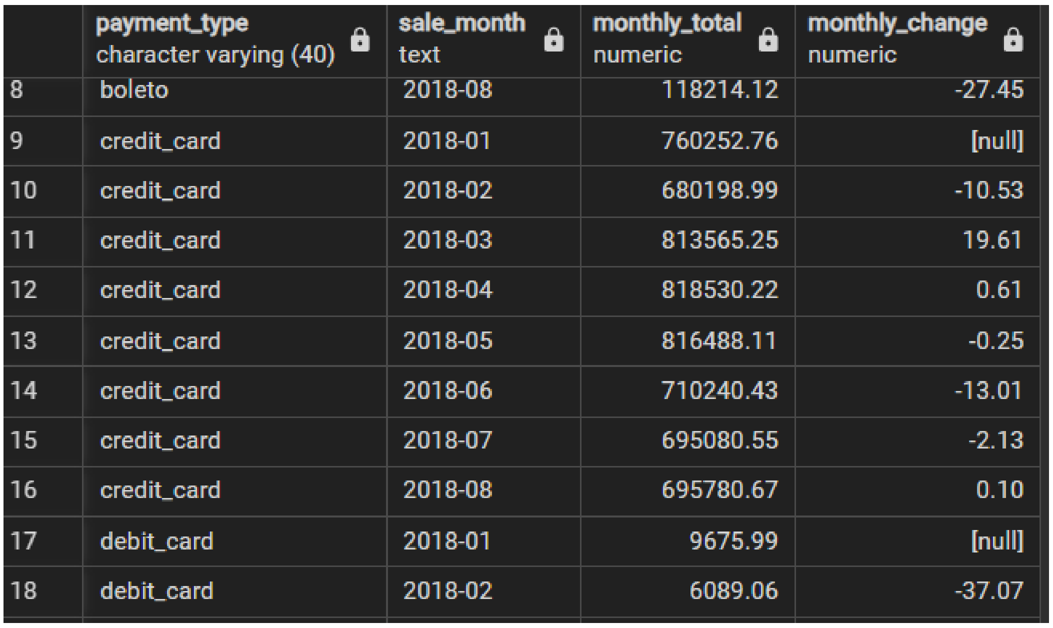
Approach:

1. **Filtering Data for 2018**:
    - Use **EXTRACT** (YEAR FROM order_purchase_timestamp) to select orders from 2018.
2. **Calculating Monthly Sales**:
    - Group data by payment_type and sale_month to calculate monthly_total sales using SUM(oi.price).
3. **Computing Month-over-Month Growth**:
    - Use the **LAG ()** window function to access the sales of the previous month for each payment_type.
    - Calculate the percentage change using the formula: **((current_month_sales - previous_month_sales) / previous_month_sales) * 100.**
4. **Organizing Results**:
    - Display payment_type, sale_month, monthly_total, and monthly_change, ordered by payment_type and sale_month.

SQL Query:

```
WITH MonthlyPaymentSales AS (
    SELECT  p.payment_type,
TO_CHAR(DATE_TRUNC('month', o.order_purchase_timestamp), 'YYYY-MM') AS
sale_month,
  SUM (oi.price) AS monthly_total
FROM amazon_brazil.orders o
JOIN amazon_brazil.order_items oi ON o.order_id = oi.order_id
JOIN amazon_brazil.payments p ON o.order_id = p.order_id
 WHERE EXTRACT(YEAR FROM o.order_purchase_timestamp) = 2018
 GROUP BY p.payment_type, sale_month )
SELECT payment_type,  sale_month,  monthly_total,
ROUND (((monthly_total - LAG(monthly_total) OVER (PARTITION BY payment_type
ORDER BY sale_month)) /
LAG(monthly_total) OVER (PARTITION BY payment_type ORDER BY sale_month)) *
100, 2) AS monthly_change
FROM MonthlyPaymentSales
ORDER BY payment_type, sale_month;
```

Output:



	payment_type character varying (40)	sale_month text	monthly_total numeric	monthly_change numeric
8	boleto	2018-08	118214.12	-27.45
9	credit_card	2018-01	760252.76	[null]
10	credit_card	2018-02	680198.99	-10.53
11	credit_card	2018-03	813565.25	19.61
12	credit_card	2018-04	818530.22	0.61
13	credit_card	2018-05	816488.11	-0.25
14	credit_card	2018-06	710240.43	-13.01
15	credit_card	2018-07	695080.55	-2.13
16	credit_card	2018-08	695780.67	0.10
17	debit_card	2018-01	9675.99	[null]
18	debit_card	2018-02	6089.06	-37.07

Recommendations:

1. **Enhance Customer Experience for Growing Methods**:

   - Focus on optimizing and promoting payment methods with consistent positive growth.

2. **Investigate Declining Trends**:

   - Analyze reasons for decline in monthly sales for certain payment methods and address barriers.