Amazon brazil data analysis for Indian market insights

Introduction:

Amazon India aims to leverage insights from Amazon Brazil's customer and sales data to identify trends, behaviors, and preferences that can drive informed decision-making in the Indian market. This analysis focuses on evaluating customer demographics, regional trends, payment preferences, and product behaviors. By analyzing key tables such as **Customers**, **Orders**, **Order Items**, **Products**, **Sellers**, and **Payments**, Amazon India can enhance customer experiences, optimize strategies, and seize market opportunities.

Schema:

The image provided painted a picture on how the relationship works between all of the tables and based on that the definitions, primary key and foreign key are set for the tables.

1. Customers Table

Column	Data Type	Constraints
customer_id	VARCHAR(150)	PRIMARY KEY
customer_unique_id	VARCHAR(150)	
customer_zip_code_prefix	INTEGER	

Query:

```
CREATE TABLE customers (
  customer_id VARCHAR(150) PRIMARY KEY,
  customer_unique_id VARCHAR(150),
  customer_zip_code_prefix INTEGER );
```

2. Orders Table

Column	Data Type	Constraints
order_id	VARCHAR(150)	PRIMARY KEY
customer_id	VARCHAR(150)	FOREIGN KEY REFERENCES customers(customer_id)
order_status	VARCHAR(150)	
order_purchase_timestamp	TIMESTAMP	
order_approved_at	TIMESTAMP	
order_delivered_carrier_date	TIMESTAMP	
order_delivered_customer_date	TIMESTAMP	
order_estimated_delivery_date	TIMESTAMP	

Query:

```
CREATE TABLE orders (
    order_id VARCHAR(150) PRIMARY KEY,
    customer_id VARCHAR(150),
    order_status VARCHAR(150),
    order_purchase_timestamp TIMESTAMP,
    order_approved_at TIMESTAMP,
    order_delivered_carrier_date TIMESTAMP,
    order_delivered_customer_date TIMESTAMP,
    order_estimated_delivery_date TIMESTAMP,
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```

3. Payments Table

Column	Data Type	Constraints
order_id	VARCHAR(150)	FOREIGN KEY REFERENCES orders(order_id)
payment_sequential	INTEGER	
payment_type	VARCHAR(150)	
payment_installments	INTEGER	
payment_value	INTEGER	
Primary Key		(order_id, payment_sequential)

Query:

```
CREATE TABLE payments (

   order_id VARCHAR(150),

   payment_sequential INTEGER,

   payment_type VARCHAR(150),

   payment_installments INTEGER,

   payment_value INTEGER,

   PRIMARY KEY (order_id, payment_sequential),

   FOREIGN KEY (order_id) REFERENCES orders(order_id)

);
```

4. Seller Table

Column	Data Type	Constraints
seller_id	VARCHAR(150)	PRIMARY KEY
seller_zip_code_prefix	INTEGER	
seller_city	VARCHAR(150)	
seller_state	VARCHAR(150)	

Query:

```
CREATE TABLE seller (
    seller_id VARCHAR(150) PRIMARY KEY,
    seller_zip_code_prefix INTEGER,
    seller_city VARCHAR(150),
    seller_state VARCHAR(150)
);
```

5. Order_Items Table

Column	Data Type	Constraints
order_id	VARCHAR(150)	FOREIGN KEY REFERENCES orders(order_id)
order_item_id	INTEGER	
product_id	VARCHAR(150)	FOREIGN KEY REFERENCES product(product_id)
seller_id	VARCHAR(150)	FOREIGN KEY REFERENCES seller(seller_id)
shipping_limit_date	TIMESTAMP	
price	INTEGER	
freight_value	INTEGER	
Primary Key		(order_id, order_item_id)

Query:

```
CREATE TABLE order_items (
   order_id VARCHAR(150),
   order_item_id INTEGER,
   product_id VARCHAR(150),
   seller_id VARCHAR(150),
   shipping_limit_date TIMESTAMP,
   price INTEGER,
   freight_value INTEGER,
   PRIMARY KEY (order_id, order_item_id),
   FOREIGN KEY (order_id) REFERENCES orders(order_id),
   FOREIGN KEY (product_id) REFERENCES product(product_id),
   FOREIGN KEY (seller_id) REFERENCES seller(seller_id) );
```

6. Product Table

Column	Data Type	Constraints
product_id	VARCHAR(150)	PRIMARY KEY
product_category_name	VARCHAR(150)	
product_name_length	INTEGER	
product_photos_qty	INTEGER	
product_weight_g	INTEGER	
product_length_cm	INTEGER	
product_height_cm	INTEGER	
product_width_cm	INTEGER	

Query:

```
CREATE TABLE product (
  product_id VARCHAR(150) PRIMARY KEY,
  product_category_name VARCHAR(150),
  product_name_lenght INTEGER,
  product_description_lenght INTEGER,
  product_photos_qty INTEGER,
  product_weight_g INTEGER,
  product_length_cm INTEGER,
  product_height_cm INTEGER,
  product_width_cm INTEGER
);
```

Relationships and Cardinality:

Below are the identified relationships based on the schema:

1. **Customers and Orders**
   - **Relationship:** One-to-Many
   - **Description:** A single customer can place multiple orders, but each order is associated with only one customer.
   - **Implementation:** orders.customer_id is a foreign key referencing customers.customer_id.
2. **Orders and Payments**
   - **Relationship:** One-to-Many
   - **Description:** An order can have multiple payment records (e.g., installments), but each payment record is associated with only one order.
   - **Implementation:** payments.order_id is a foreign key referencing orders.order_id.
   - **Primary Key:** Composite key (order_id, payment_sequential) ensures each payment record is uniquely identifiable per order.
3. **Orders and Order_Items**
   - **Relationship:** One-to-Many
   - **Description:** An order can contain multiple order items, but each order item is associated with only one order.
   - **Implementation:** order_items.order_id is a foreign key referencing orders.order_id.
   - **Primary Key:** Composite key (order_id, order_item_id) ensures each item within an order is uniquely identifiable.
4. **Order_Items and Product**
   - **Relationship:** Many-to-One
   - **Description:** Multiple order items can reference the same product, but each order item references only one product.
   - **Implementation:** order_items.product_id is a foreign key referencing product.product_id.
5. **Order_Items and Seller**
   - **Relationship:** Many-to-One
   - **Description:** Multiple order items can be sold by the same seller, but each order item is associated with only one seller.
   - **Implementation:** order_items.seller_id is a foreign key referencing seller.seller_id.

6. **Product and Category (Implicit)**
   - **Relationship:** Many-to-One (Assumed)
   - **Description:** Although not explicitly detailed in the schema, typically, multiple products can belong to a single category.
   - **Implementation:** product.product_category_name likely references a category table (not defined here).

Summary of Relationships

From Table	To Table	Relationship Type	Cardinality
Customers	Orders	One-to-Many	1 : N
Orders	Payments	One-to-Many	1 : N
Orders	Order_Items	One-to-Many	1 : N
Order_Items	Product	Many-to-One	N : 1
Order_Items	Seller	Many-to-One	N : 1

Relationships:

- **customers** to **orders**: One-to-many (one customer can have multiple orders).
- **orders** to **payments**: One-to-many (one order can have multiple payment entries).
- **orders** to **order_items**: One-to-many (one order can have multiple items).
- **seller** to **order_items**: One-to-many (one seller can sell multiple items).
- **product** to **order_items**: One-to-many (one product can appear in multiple order items).

Analysis 1:

1. **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.

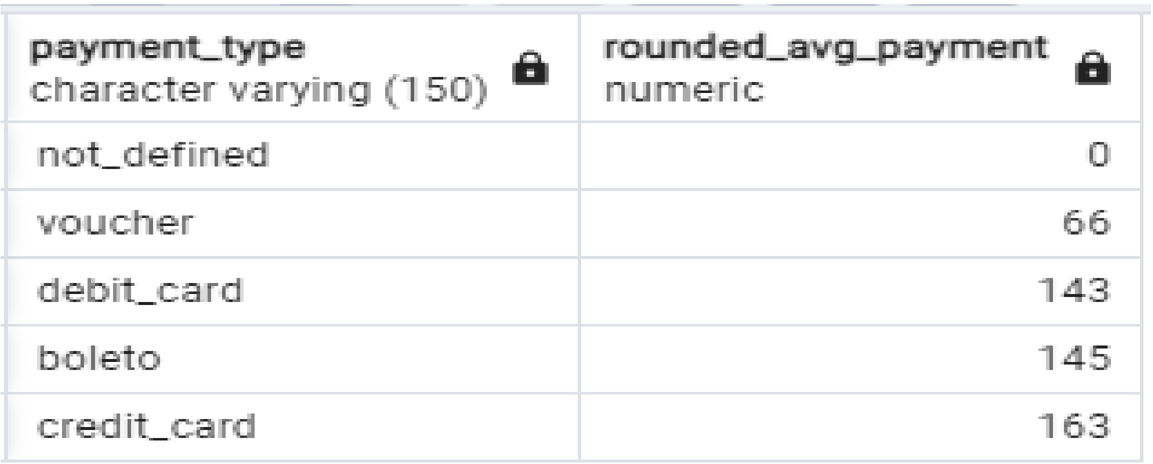
Approach:

- Identified the relevant table (**Payments**) and columns (**payment_type**, **payment_value**).
- Calculated average payment values using the AVG function.
- Rounded values using the ROUND function and sorted results in ascending order.

SQL Query:

SELECT payment_type, ROUND(AVG(payment_value)) AS rounded_avg_payment FROM amazon_brazil.payments GROUP BY payment_type ORDER BY rounded_avg_payment ASC;

Output:



payment_type character varying (150) 🔒	rounded_avg_payment numeric 🔒
not_defined	0
voucher	66
debit_card	143
boleto	145
credit_card	163

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

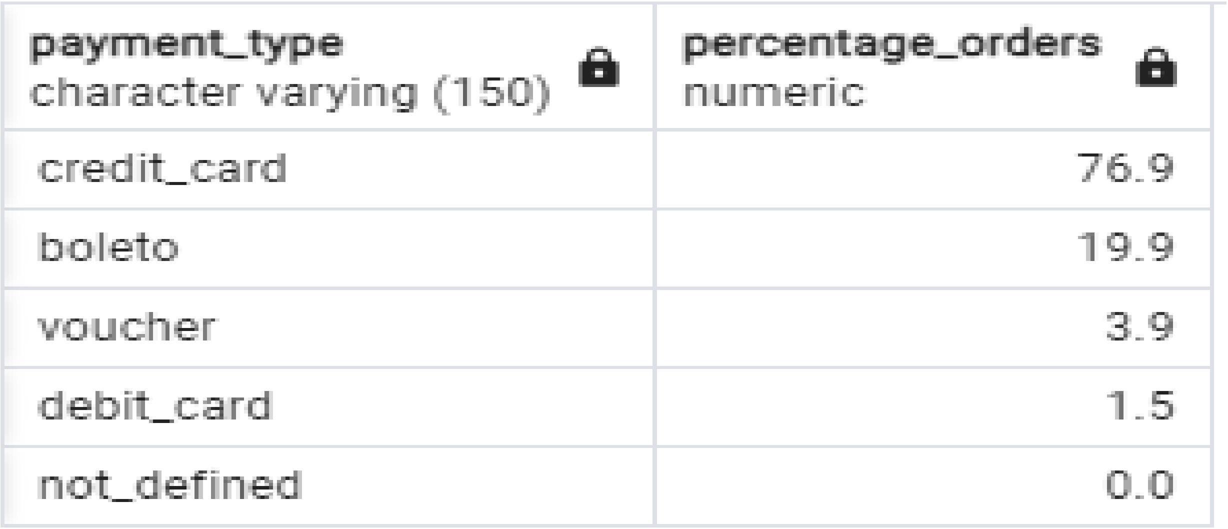
Approach:

- Used **COUNT** to calculate the number of orders per payment type.
- Calculated percentage distribution and rounded results to one decimal place.
- Sorted results in descending order to identify popular payment methods.

SQL Query:

```
SELECT payment_type,ROUND(COUNT(DISTINCT order_id) * 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 (150) 🔒	percentage_orders numeric 🔒
credit_card	76.9
boleto	19.9
voucher	3.9
debit_card	1.5
not_defined	0.0

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

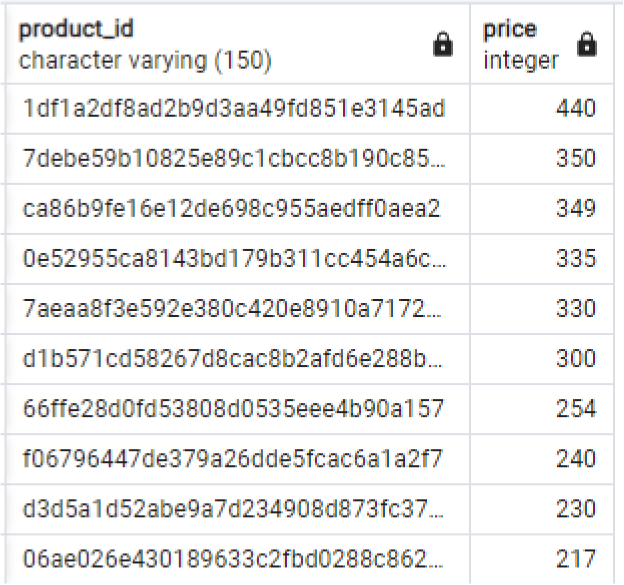
Approach:

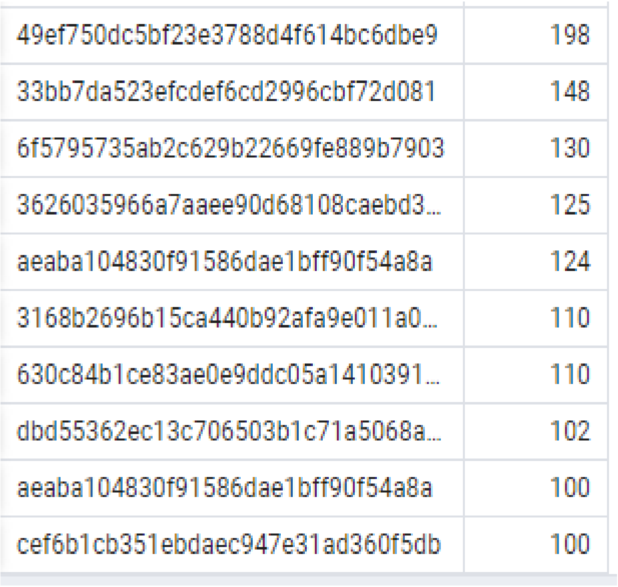
- Filtered products within a price range of 100 to 500 BRL using BETWEEN.
- Searched for products containing the keyword "Smart" using LIKE.
- Sorted results by price in descending order.

SQL Query:

```
SELECT DISTINCT oi.product_id, price FROM amazon_brazil.product p JOIN
amazon_brazil.order_items oi ON p.product_id = oi.product_id WHERE
p.product_category_name LIKE '%smart%' AND oi.price BETWEEN 100 AND 500 ORDER BY
oi.price DESC;
```

Output:





product_id character varying (150)	price integer
1df1a2df8ad2b9d3aa49fd851e3145ad	440
7debe59b10825e89c1cbcc8b190c85...	350
ca86b9fe16e12de698c955aedff0aea2	349
0e52955ca8143bd179b311cc454a6c...	335
7aeaa8f3e592e380c420e8910a7172...	330
d1b571cd58267d8cac8b2afd6e288b...	300
66ffe28d0fd53808d0535eee4b90a157	254
f06796447de379a26dde5fcac6a1a2f7	240
d3d5a1d52abe9a7d234908d873fc37...	230
06ae026e430189633c2fbd0288c862...	217
49ef750dc5bf23e3788d4f614bc6dbe9	198
33bb7da523efcdef6cd2996cbf72d081	148
6f5795735ab2c629b22669fe889b7903	130
3626035966a7aaee90d68108caebd3...	125
aeaba104830f91586dae1bff90f54a8a	124
3168b2696b15ca440b92afa9e011a0...	110
630c84b1ce83ae0e9ddc05a1410391...	110
dbd55362ec13c706503b1c71a5068a...	102
aeaba104830f91586dae1bff90f54a8a	100
cef6b1cb351ebdaec947e31ad360f5db	100

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

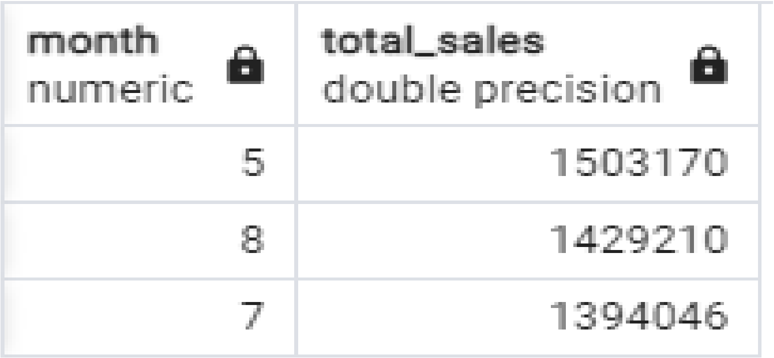
Approach:

- Extracted months from timestamps using EXTRACT.
- Calculated total sales per month and sorted the top three months by total sales.

SQL Query:

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

Output:



month numeric 🔒	total_sales double precision 🔒
5	1503170
8	1429210
7	1394046

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

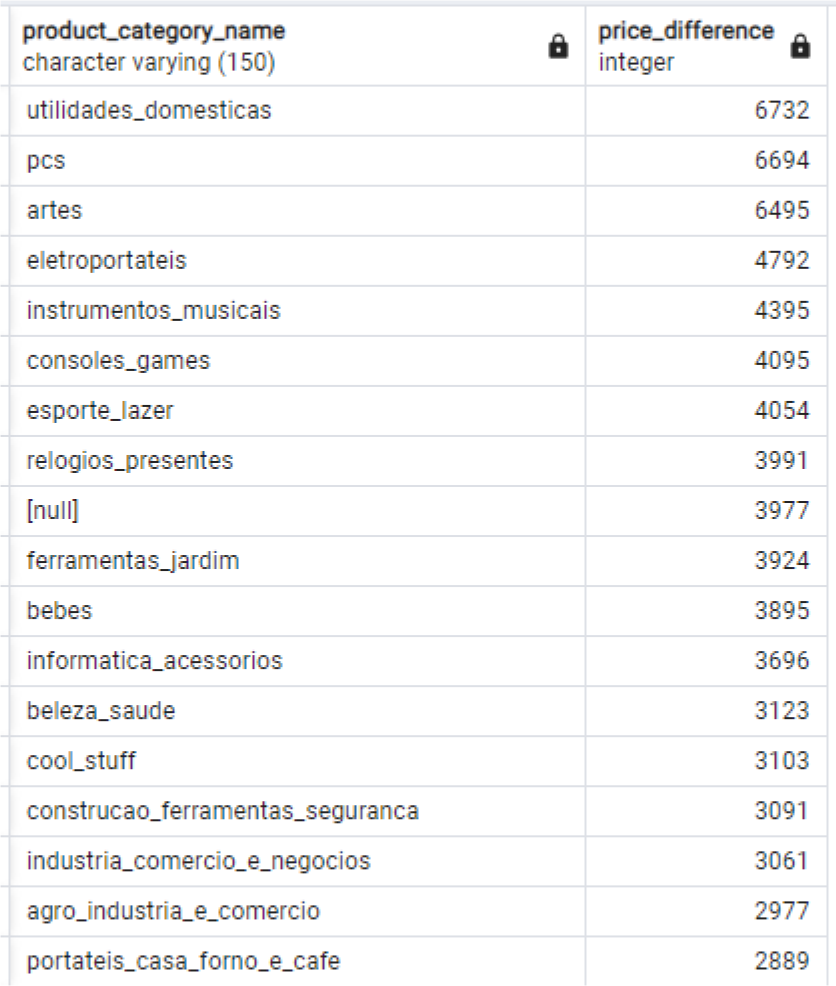
Approach:

- Identified categories with price differences exceeding 500 BRL.
- Used MAX and MIN functions to calculate price variations.

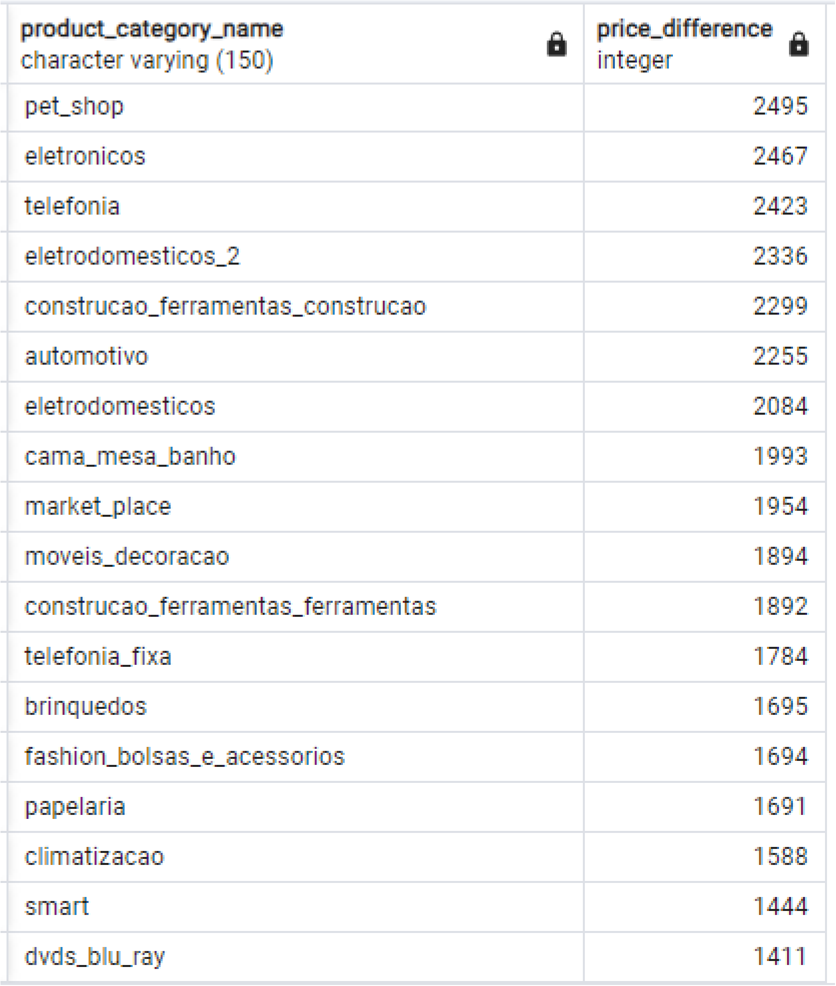
SQL Query:

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

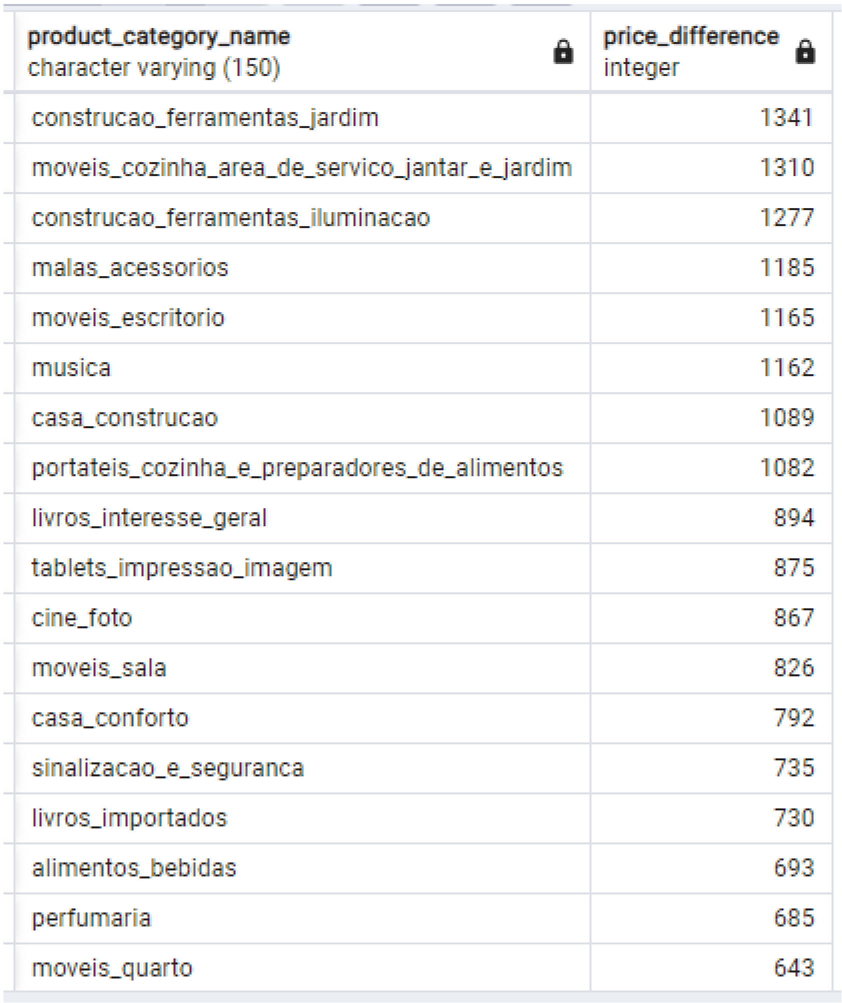
Output:



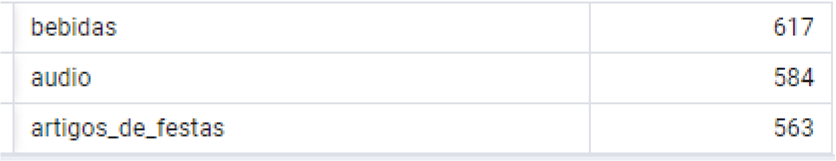
product_category_name character varying (150)	price_difference integer
utilidades_domesticas	6732
pcs	6694
artes	6495
eletroportateis	4792
instrumentos_musicais	4395
consoles_games	4095
esporte_lazer	4054
relogios_presentes	3991
[null]	3977
ferramentas_jardim	3924
bebes	3895
informatica_acessorios	3696
beleza_saude	3123
cool_stuff	3103
construcao_ferramentas_seguranca	3091
industria_comercio_e_negocios	3061
agro_industria_e_comercio	2977
portateis_casa_forno_e_cafe	2889



product_category_name character varying (150)	price_difference integer
pet_shop	2495
eletronicos	2467
telefonia	2423
eletrodomesticos_2	2336
construcao_ferramentas_construcao	2299
automotivo	2255
eletrodomesticos	2084
cama_mesa_banho	1993
market_place	1954
moveis_decoracao	1894
construcao_ferramentas_ferramentas	1892
telefonia_fixa	1784
brinquedos	1695
fashion_bolsas_e_acessorios	1694
papelaria	1691
climatizacao	1588
smart	1444
dvds_blu_ray	1411



product_category_name character varying (150)	price_difference integer
construcao_ferramentas_jardim	1341
moveis_cozinha_area_de_servico_jantar_e_jardim	1310
construcao_ferramentas_iluminacao	1277
malas_acessorios	1185
moveis_escritorio	1165
musica	1162
casa_construcao	1089
portateis_cozinha_e_preparadores_de_alimentos	1082
livros_interesse_geral	894
tablets_impressao_imagem	875
cine_foto	867
moveis_sala	826
casa_conforto	792
sinalizacao_e_seguranca	735
livros_importados	730
alimentos_bebidas	693
perfumaria	685
moveis_quarto	643



bebidas	617
audio	584
artigos_de_festas	563

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

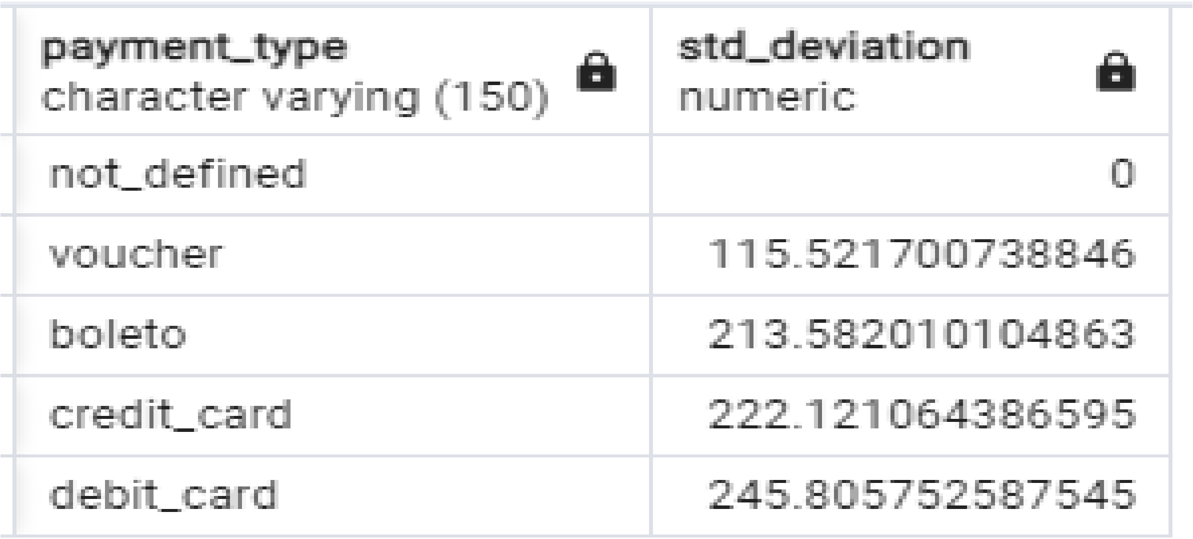
Approach:

- Calculated the standard deviation of transaction amounts for each payment type.
- Sorted payment types by least variance.

SQL Query:

```
SELECT payment_type, STDDEV(payment_value) AS std_deviation FROM
amazon_brazil.payments GROUP BY payment_type ORDER BY std_deviation ASC;
```

Output:



payment_type character varying (150)	std_deviation numeric
not_defined	0
voucher	115.521700738846
boleto	213.582010104863
credit_card	222.121064386595
debit_card	245.805752587545

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

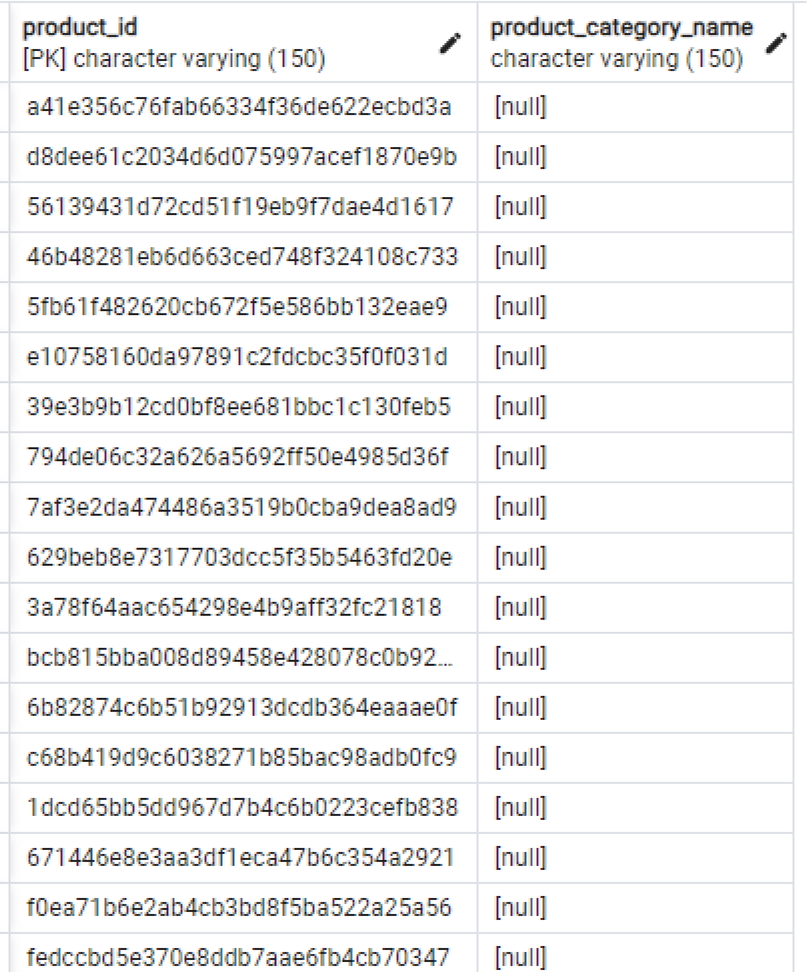
Approach:

- Retrieved products with missing or incomplete category names using IS NULL and LENGTH functions.

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 (150)	product_category_name character varying (150)
a41e356c76fab66334f36de622ecbd3a	[null]
d8dee61c2034d6d075997acef1870e9b	[null]
56139431d72cd51f19eb9f7dae4d1617	[null]
46b48281eb6d663ced748f324108c733	[null]
5fb61f482620cb672f5e586bb132eae9	[null]
e10758160da97891c2fdcbc35f0f031d	[null]
39e3b9b12cd0bf8ee681bbc1c130feb5	[null]
794de06c32a626a5692ff50e4985d36f	[null]
7af3e2da474486a3519b0cba9dea8ad9	[null]
629beb8e7317703dcc5f35b5463fd20e	[null]
3a78f64aac654298e4b9aff32fc21818	[null]
bcb815bba008d89458e428078c0b92...	[null]
6b82874c6b51b92913dcdb364eaaae0f	[null]
c68b419d9c6038271b85bac98adb0fc9	[null]
1dcd65bb5dd967d7b4c6b0223cefb838	[null]
671446e8e3aa3df1eca47b6c354a2921	[null]
f0ea71b6e2ab4cb3bd8f5ba522a25a56	[null]
fedccbd5e370e8ddb7aae6fb4cb70347	[null]

Recommendations:

- Optimize popular payment methods (**credit_card**, **boleto**) with promotions and rewards.
- Address underperforming payment methods (**debit_card**, **voucher**) through targeted campaigns.
- Focus on high-value products in specific price ranges and categories.
- Enhance data completeness for accurate product categorization.
- Leverage seasonal trends to design effective marketing campaigns.

Analysis 2:

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

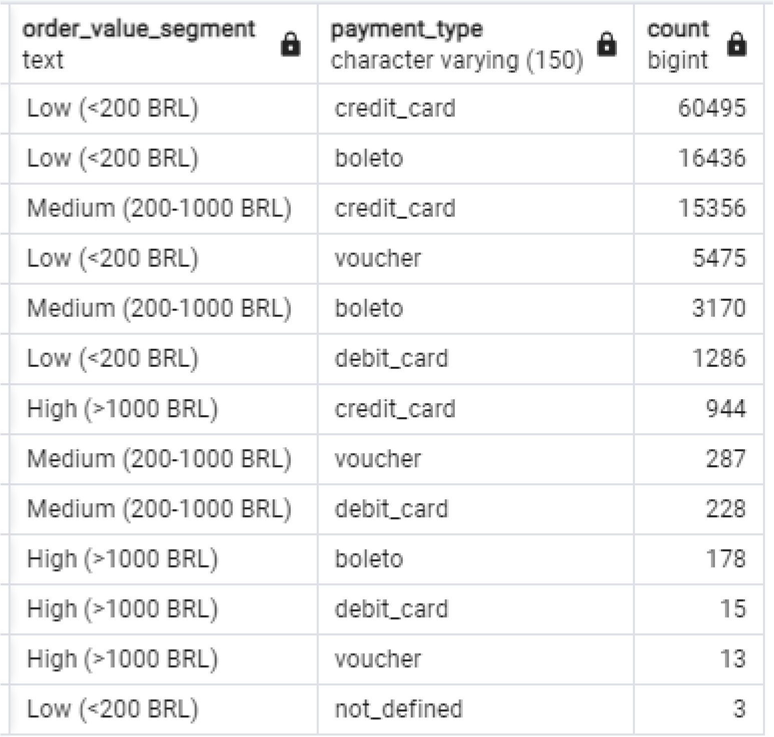
Approach:

- Segmented orders into three ranges (low, medium, high) based on payment value.
- Counted occurrences of each payment type within these ranges and sorted by count.

SQL Query:

```
SELECT CASE WHEN payment_value < 200 THEN 'Low (<200 BRL)' WHEN payment_value
BETWEEN 200 AND 1000 THEN 'Medium (200-1000 BRL)' ELSE 'High (>1000 BRL)' END AS
order_value_segment, payment_type, COUNT(*) AS count FROM amazon_brazil.payments
GROUP BY order_value_segment, payment_type ORDER BY count DESC;
```

Output:



order_value_segment text	payment_type character varying (150)	count bigint
Low (<200 BRL)	credit_card	60495
Low (<200 BRL)	boleto	16436
Medium (200-1000 BRL)	credit_card	15356
Low (<200 BRL)	voucher	5475
Medium (200-1000 BRL)	boleto	3170
Low (<200 BRL)	debit_card	1286
High (>1000 BRL)	credit_card	944
Medium (200-1000 BRL)	voucher	287
Medium (200-1000 BRL)	debit_card	228
High (>1000 BRL)	boleto	178
High (>1000 BRL)	debit_card	15
High (>1000 BRL)	voucher	13
Low (<200 BRL)	not_defined	3

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

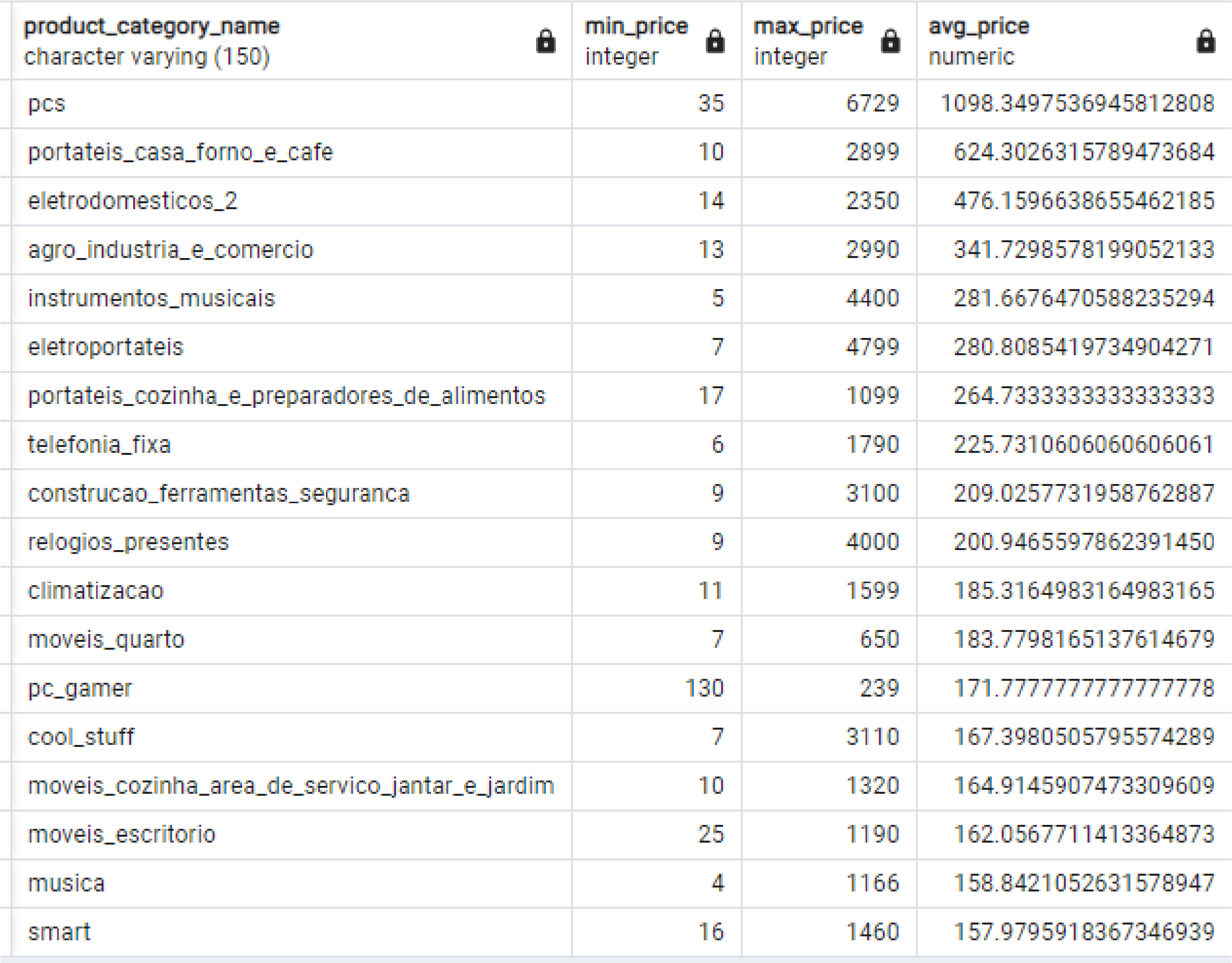
Approach:

- Calculated minimum, maximum, and average prices for each category.
- Sorted results by average price in descending order.

SQL Query:

SELECT product_category_name, MIN(price) AS min_price, MAX(price) AS max_price, AVG(price) 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 (150)	min_price integer	max_price integer	avg_price numeric
pcs	35	6729	1098.3497536945812808
portateis_casa_forno_e_cafe	10	2899	624.3026315789473684
eletrodomesticos_2	14	2350	476.1596638655462185
agro_industria_e_comercio	13	2990	341.7298578199052133
instrumentos_musicais	5	4400	281.6676470588235294
eletroportateis	7	4799	280.8085419734904271
portateis_cozinha_e_preparadores_de_alimentos	17	1099	264.7333333333333333
telefonia_fixa	6	1790	225.7310606060606061
construcao_ferramentas_seguranca	9	3100	209.0257731958762887
relogios_presentes	9	4000	200.9465597862391450
climatizacao	11	1599	185.3164983164983165
moveis_quarto	7	650	183.7798165137614679
pc_gamer	130	239	171.7777777777777778
cool_stuff	7	3110	167.3980505795574289
moveis_cozinha_area_de_servico_jantar_e_jardim	10	1320	164.9145907473309609
moveis_escritorio	25	1190	162.0567711413364873
musica	4	1166	158.8421052631578947
smart	16	1460	157.9795918367346939

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

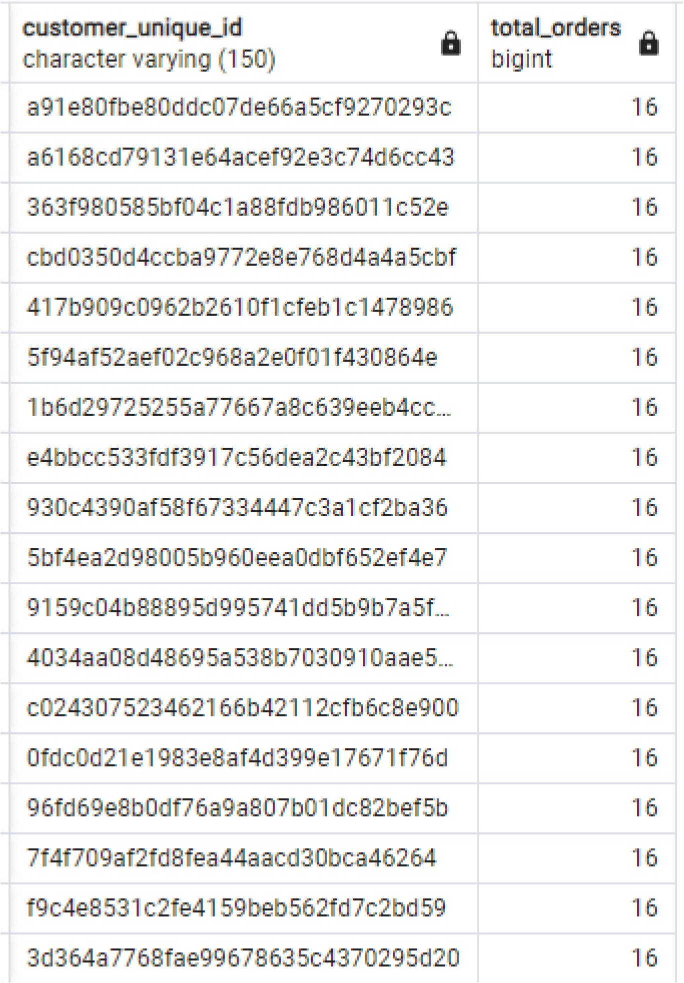
Approach:

- Counted orders for each customer using GROUP BY.
- Filtered customers with more than one order.

SQL Query:

SELECT customer_unique_id,COUNT(order_id) AS total_orders FROM amazon_brazil.orders o JOIN amazon_brazil.customers c ON o.customer_id = c.customer_id GROUP BY customer_unique_id HAVING COUNT(order_id) > 1 ORDER BY total_orders DESC;

Output:



customer_unique_id character varying (150)	total_orders bigint
a91e80fbe80ddc07de66a5cf9270293c	16
a6168cd79131e64acef92e3c74d6cc43	16
363f980585bf04c1a88fdb986011c52e	16
cbd0350d4ccba9772e8e768d4a4a5cbf	16
417b909c0962b2610f1cfeb1c1478986	16
5f94af52aef02c968a2e0f01f430864e	16
1b6d29725255a77667a8c639eeb4cc...	16
e4bbcc533fdf3917c56dea2c43bf2084	16
930c4390af58f67334447c3a1cf2ba36	16
5bf4ea2d98005b960eea0dbf652ef4e7	16
9159c04b88895d995741dd5b9b7a5f...	16
4034aa08d48695a538b7030910aae5...	16
c024307523462166b42112cfb6c8e900	16
0fdc0d21e1983e8af4d399e17671f76d	16
96fd69e8b0df76a9a807b01dc82bef5b	16
7f4f709af2fd8fea44aacd30bca46264	16
f9c4e8531c2fe4159beb562fd7c2bd59	16
3d364a7768fae99678635c4370295d20	16

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

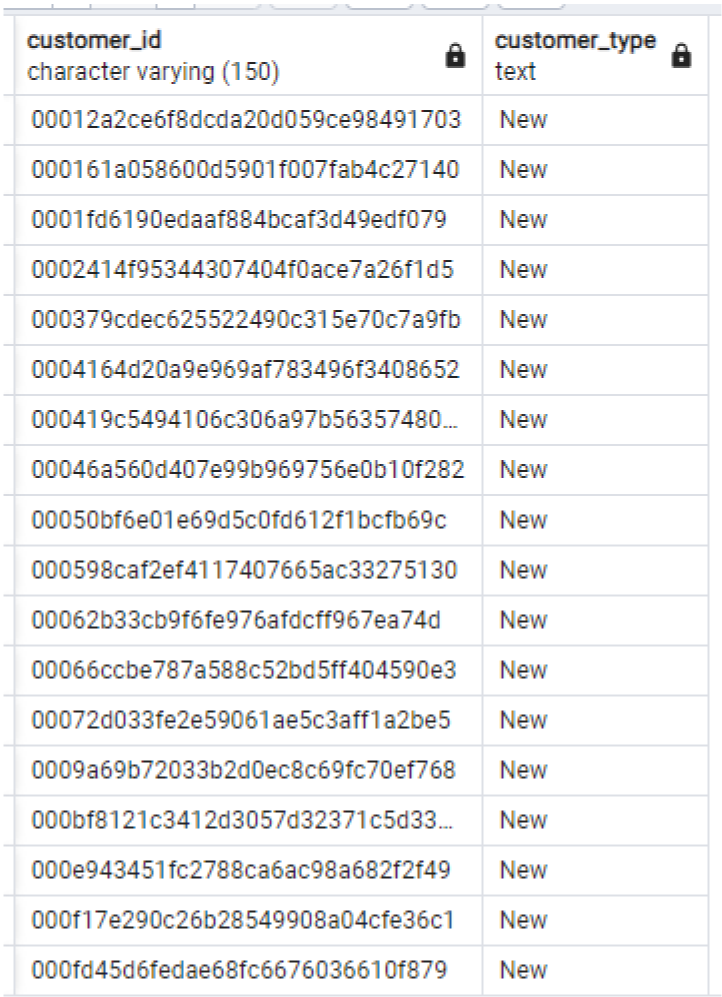
Approach:

- Categorized customers as "New," "Returning," and "Loyal" based on order counts.
- Used a temporary table for categorization and updated customer types.

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 (150)	customer_type text
00012a2ce6f8dcda20d059ce98491703	New
000161a058600d5901f007fab4c27140	New
0001fd6190edaaf884bcaf3d49edf079	New
0002414f95344307404f0ace7a26f1d5	New
000379cdec625522490c315e70c7a9fb	New
0004164d20a9e969af783496f3408652	New
000419c5494106c306a97b56357480...	New
00046a560d407e99b969756e0b10f282	New
00050bf6e01e69d5c0fd612f1bcfb69c	New
000598caf2ef4117407665ac33275130	New
00062b33cb9f6fe976afdcff967ea74d	New
00066ccbe787a588c52bd5ff404590e3	New
00072d033fe2e59061ae5c3aff1a2be5	New
0009a69b72033b2d0ec8c69fc70ef768	New
000bf8121c3412d3057d32371c5d33...	New
000e943451fc2788ca6ac98a682f2f49	New
000f17e290c26b28549908a04cfe36c1	New
000fd45d6fedae68fc6676036610f879	New

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

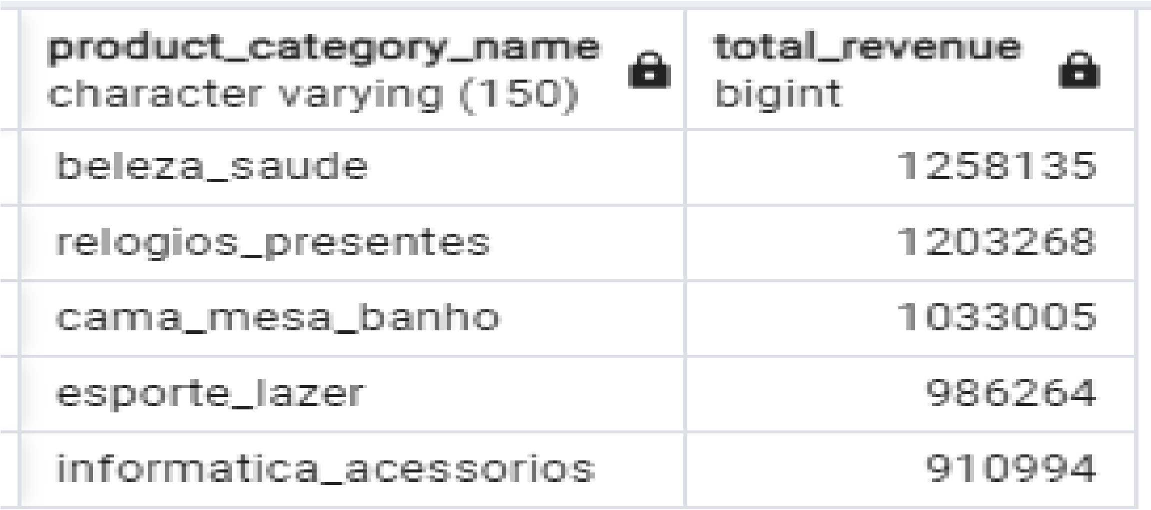
Approach:

- Joined product and order tables to calculate total revenue per category.
- Sorted and displayed the top five 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 (150)	total_revenue bigint
beleza_saude	1258135
relogios_presentes	1203268
cama_mesa_banho	1033005
esporte_lazer	986264
informatica_acessorios	910994

Recommendations

- Promote high-revenue categories through targeted inventory and marketing strategies.
- Use customer segmentation to design loyalty programs and personalized offers.
- Expand focus on mid-range price products for broader market appeal.

Analysis 3:

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

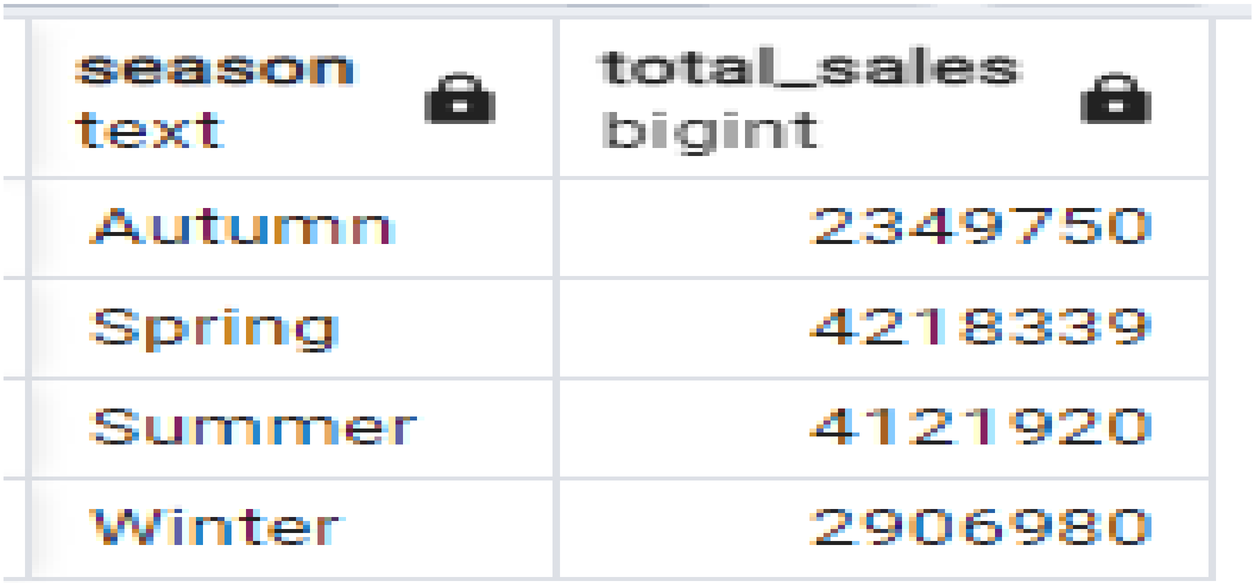
Approach:

- Grouped sales data by season (Spring, Summer, Autumn, Winter).
- Calculated total sales for each season.

SQL Query:

```
SELECT season, SUM(price) AS total_sales FROM (SELECT oi.price,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 FROM
amazon_brazil.orders o JOIN amazon_brazil.order_items oi ON o.order_id = oi.order_id) AS
season_sales GROUP BY season ORDER BY season;
```

Output:



season text	total_sales bigint
Autumn	2349750
Spring	4218339
Summer	4121920
Winter	2906980

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

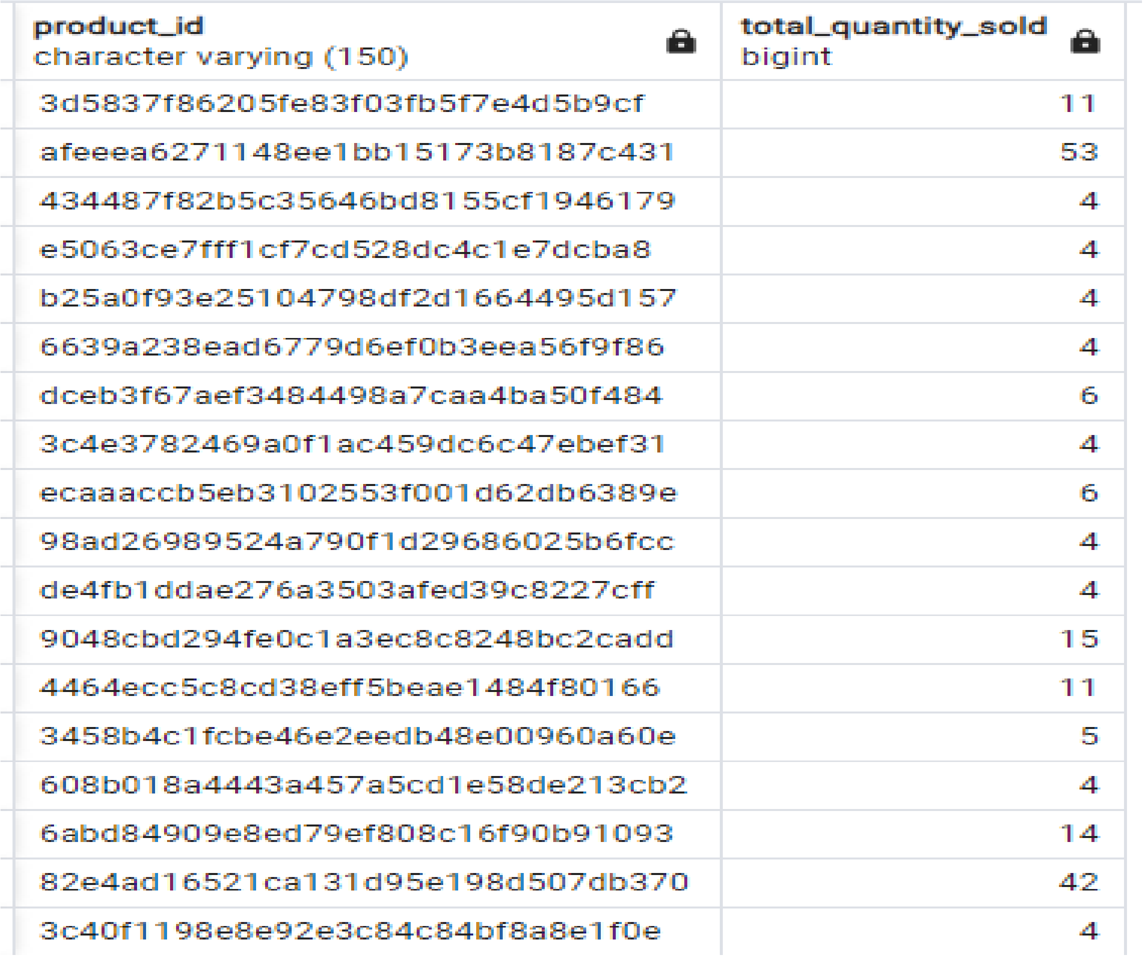
Approach:

- Filtered products exceeding average sales quantities using subqueries.

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) AS product_sales 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) AS avg_sales);
```

Output:



product_id character varying (150)	total_quantity_sold bigint
3d5837f86205fe83f03fb5f7e4d5b9cf	11
afeeea6271148ee1bb15173b8187c431	53
434487f82b5c35646bd8155cf1946179	4
e5063ce7fff1cf7cd528dc4c1e7dcba8	4
b25a0f93e25104798df2d1664495d157	4
6639a238ead6779d6ef0b3eea56f9f86	4
dceb3f67aef3484498a7caa4ba50f484	6
3c4e3782469a0f1ac459dc6c47ebef31	4
ecaaaccb5eb3102553f001d62db6389e	6
98ad26989524a790f1d29686025b6fcc	4
de4fb1ddae276a3503afed39c8227cff	4
9048cbd294fe0c1a3ec8c8248bc2cadd	15
4464ecc5c8cd38eff5beae1484f80166	11
3458b4c1fcbe46e2eedb48e00960a60e	5
608b018a4443a457a5cd1e58de213cb2	4
6abd84909e8ed79ef808c16f90b91093	14
82e4ad16521ca131d95e198d507db370	42
3c40f1198e8e92e3c84c84bf8a8e1f0e	4

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

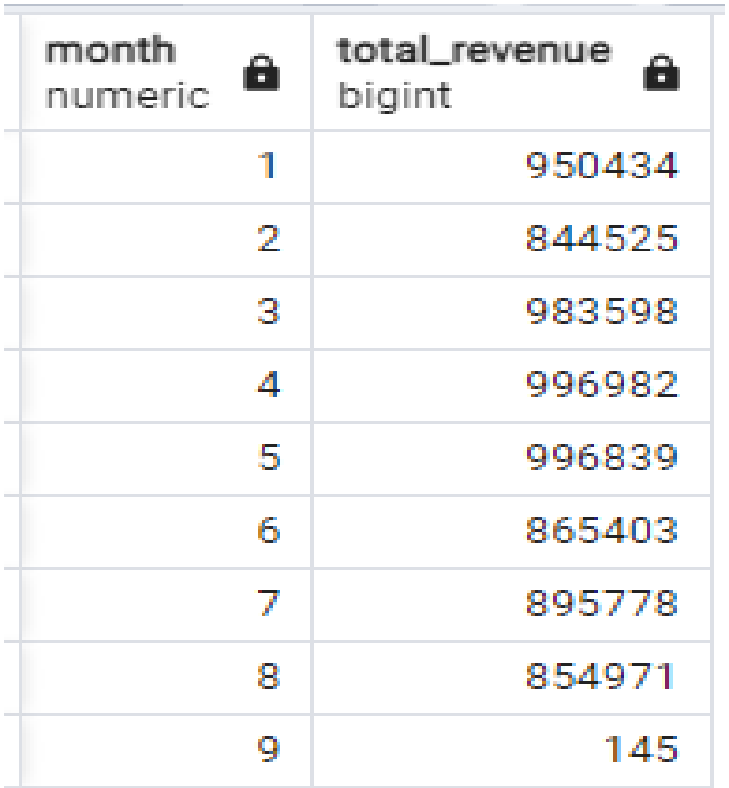
Approach:

- Extracted monthly revenue for the year 2018.
- Exported results to Excel and visualized trends in a graph.

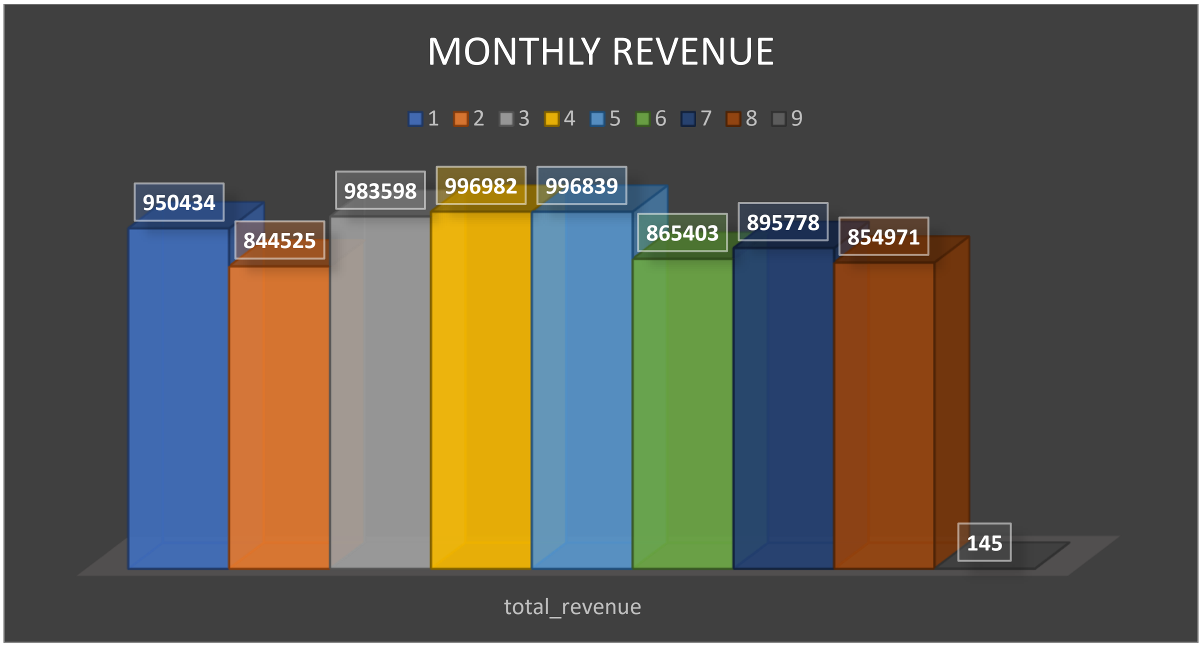
SQL Query:

```
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 bigint
1	950434
2	844525
3	983598
4	996982
5	996839
6	865403
7	895778
8	854971
9	145



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

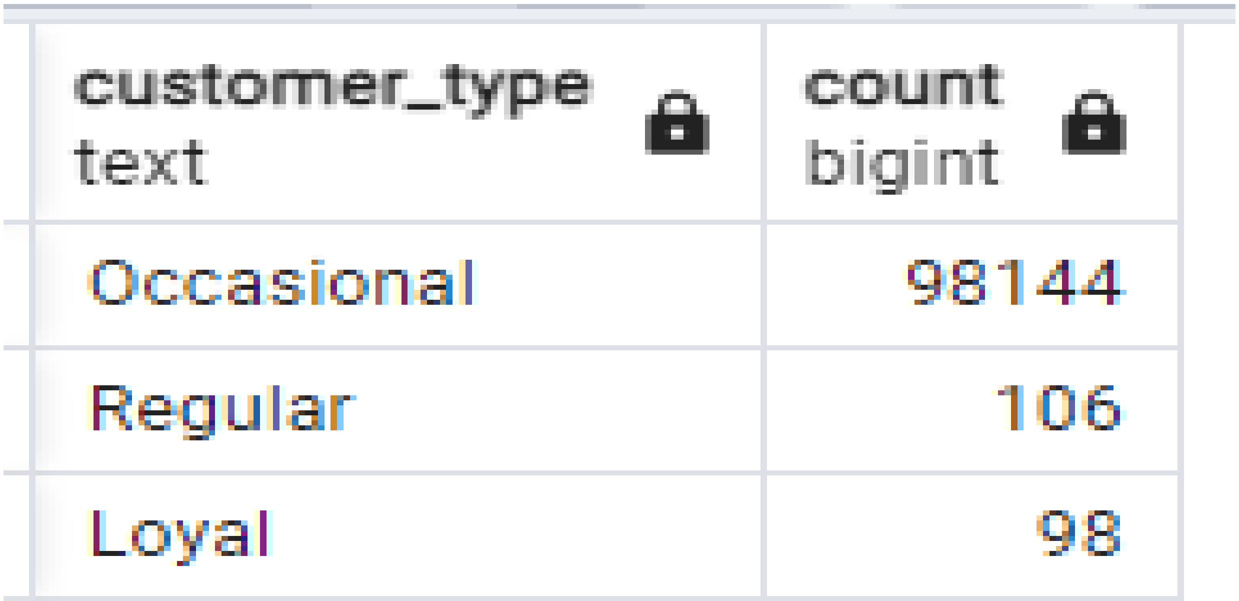
Approach:

- Used CTEs to segment customers into "Occasional," "Regular," and "Loyal" groups.
- Counted customers in each segment and visualized proportions in a chart.

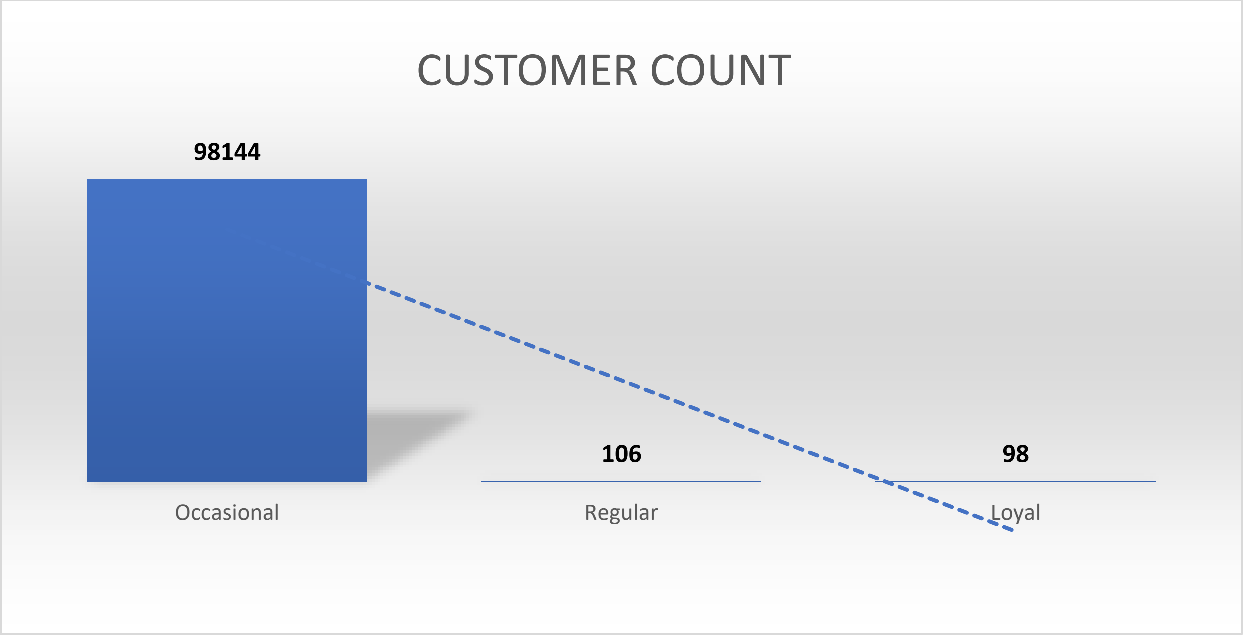
SQL Query:

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

Output:



customer_type text	count bigint
Occasional	98144
Regular	106
Loyal	98



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

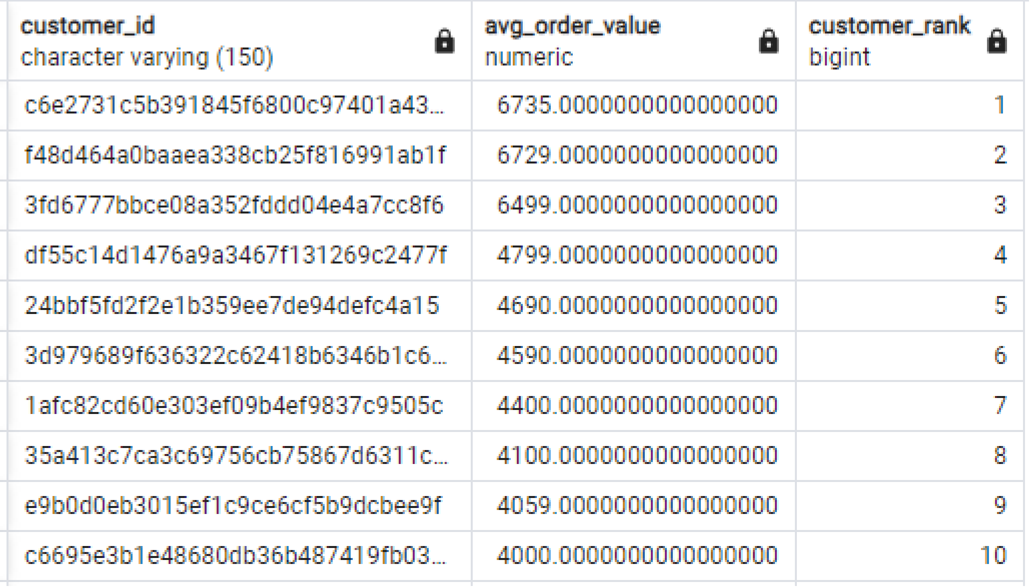
Approach:

- Ranked customers based on their average order values using window functions.
- Identified the top 20 customers for potential rewards programs.

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 (150)	avg_order_value numeric	customer_rank bigint
c6e2731c5b391845f6800c97401a43...	6735.0000000000000000	1
f48d464a0baaea338cb25f816991ab1f	6729.0000000000000000	2
3fd6777bbce08a352fddd04e4a7cc8f6	6499.0000000000000000	3
df55c14d1476a9a3467f131269c2477f	4799.0000000000000000	4
24bbf5fd2f2e1b359ee7de94defc4a15	4690.0000000000000000	5
3d979689f636322c62418b6346b1c6...	4590.0000000000000000	6
1afc82cd60e303ef09b4ef9837c9505c	4400.0000000000000000	7
35a413c7ca3c69756cb75867d6311c...	4100.0000000000000000	8
e9b0d0eb3015ef1c9ce6cf5b9dcbee9f	4059.0000000000000000	9
c6695e3b1e48680db36b487419fb03...	4000.0000000000000000	10

6. **Amazon wants to analyze sales growth trends for its key products over their lifecycle.** Calculate monthly cumulative sales for each product from the date of its first sale. Use a recursive CTE to compute the cumulative sales (total_sales) for each product month by month.

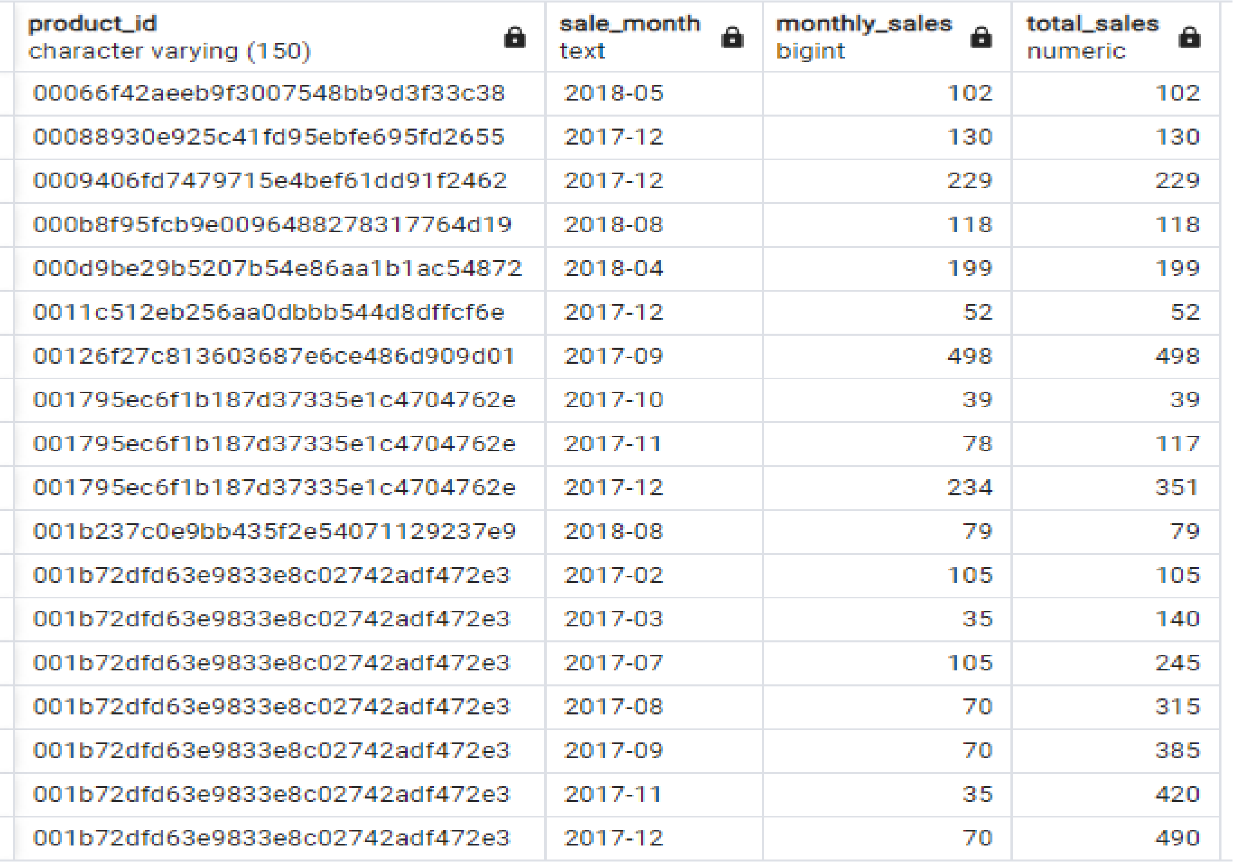
Approach:

- Used recursive CTEs to calculate monthly cumulative sales for each product.

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 (150)	sale_month text	monthly_sales bigint	total_sales numeric
00066f42aeeb9f3007548bb9d3f33c38	2018-05	102	102
00088930e925c41fd95ebfe695fd2655	2017-12	130	130
0009406fd7479715e4bef61dd91f2462	2017-12	229	229
000b8f95fcb9e0096488278317764d19	2018-08	118	118
000d9be29b5207b54e86aa1b1ac54872	2018-04	199	199
0011c512eb256aa0dbbb544d8dffcf6e	2017-12	52	52
00126f27c813603687e6ce486d909d01	2017-09	498	498
001795ec6f1b187d37335e1c4704762e	2017-10	39	39
001795ec6f1b187d37335e1c4704762e	2017-11	78	117
001795ec6f1b187d37335e1c4704762e	2017-12	234	351
001b237c0e9bb435f2e54071129237e9	2018-08	79	79
001b72dfd63e9833e8c02742adf472e3	2017-02	105	105
001b72dfd63e9833e8c02742adf472e3	2017-03	35	140
001b72dfd63e9833e8c02742adf472e3	2017-07	105	245
001b72dfd63e9833e8c02742adf472e3	2017-08	70	315
001b72dfd63e9833e8c02742adf472e3	2017-09	70	385
001b72dfd63e9833e8c02742adf472e3	2017-11	35	420
001b72dfd63e9833e8c02742adf472e3	2017-12	70	490

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

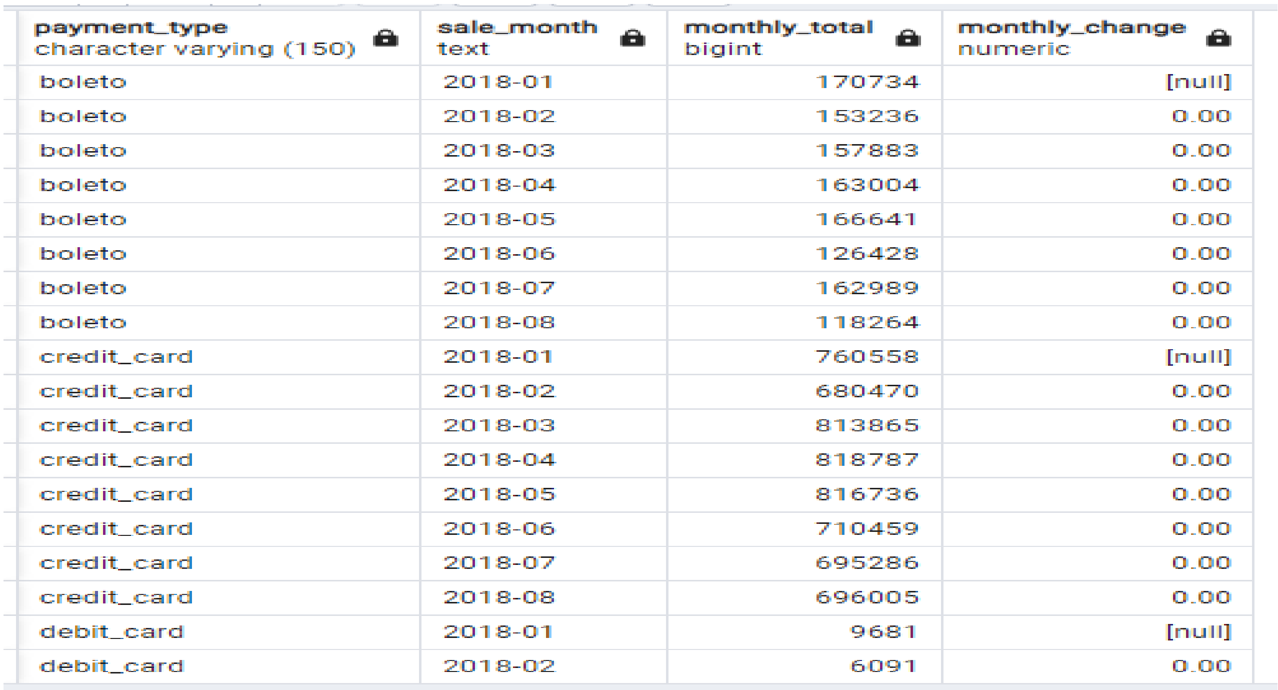
Approach:

Computed total sales and month-over-month growth rates for each payment method in 2018.

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 (150)	sale_month text	monthly_total bigint	monthly_change numeric
boleto	2018-01	170734	[null]
boleto	2018-02	153236	0.00
boleto	2018-03	157883	0.00
boleto	2018-04	163004	0.00
boleto	2018-05	166641	0.00
boleto	2018-06	126428	0.00
boleto	2018-07	162989	0.00
boleto	2018-08	118264	0.00
credit_card	2018-01	760558	[null]
credit_card	2018-02	680470	0.00
credit_card	2018-03	813865	0.00
credit_card	2018-04	818787	0.00
credit_card	2018-05	816736	0.00
credit_card	2018-06	710459	0.00
credit_card	2018-07	695286	0.00
credit_card	2018-08	696005	0.00
debit_card	2018-01	9681	[null]
debit_card	2018-02	6091	0.00

Recommendations:

- Utilize seasonal insights to optimize inventory and promotions during high-sales periods.
- Identify and prioritize high-performing products and customers for exclusive programs.
- Address low-growth periods with strategic campaigns.
- Encourage consistent payment methods to ensure steady revenue growth.