Exploratory Data Analysis using SQL

Author: Nikos Papakostas
Created: September 2024
GitHub: https://github.com/papaknik
LinkedIn: https://www.linkedin.com/in/nikos-papakostas/

```
import pandas as pd
```

```
import sqlalchemy
```

```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

```
%sql postgresql://postgres:postgres@localhost/demo
```

We begin by examining the columns' data types for the tables to ensure that the data is correctly formatted and to understand the structure of the dataset.
This step is crucial for validating data integrity and preparing for subsequent analysis.

```
%%sql
SELECT column_name, data_type, is_nullable
FROM information_schema.columns
WHERE table_name = 'orders';
```

```
 * postgresql://postgres:***@localhost/demo
5 rows affected.
```

column_name	data_type	is_nullable
order_id	text	YES
order_date	text	YES
customername	text	YES
state	text	YES
city	text	YES

```
%%sql
SELECT column_name, data_type, is_nullable
FROM information_schema.columns
WHERE table_name = 'details';
```

```
 * postgresql://postgres:***@localhost/demo
7 rows affected.
```

column_name	data_type	is_nullable
quantity	bigint	YES
amount	bigint	YES
profit	bigint	YES
order_id	text	YES
category	text	YES
sub_category	text	YES
paymentmode	text	YES

Next, we check for missing values in the columns of both tables. This will help us identify any gaps in the dataset that need to be addressed before
proceeding with the analysis.

```
%%sql
SELECT
COUNT(*) AS total_rows,
COUNT(order_id) AS non_null_order_id,
COUNT(customername) AS non_null_customername,
COUNT(state) AS non_null_state,
COUNT(city) AS non_null_city
FROM orders;
```

```
 * postgresql://postgres:***@localhost/demo
1 rows affected.
```

total_rows	non_null_order_id	non_null_customername	non_null_state	non_null_city
500	500	500	500	500

```
%%sql
SELECT
COUNT(*) AS total_rows,
COUNT(order_id) AS non_null_order_id,
COUNT(amount) AS non_null_amount,
COUNT(quantity) AS non_null_quantity,
COUNT(profit) AS non_null_profit,
COUNT(sub_category) AS non_null_sub_category
FROM details;
```

```
 * postgresql://postgres:***@localhost/demo
1 rows affected.
```

total_rows	non_null_order_id	non_null_amount	non_null_quantity	non_null_profit	non_null_sub_cat
1500	1500	1500	1500	1500	

Last step before we begin our EDA is to check for orphan records in the tables
Orders in the details table that do not have a corresponding entry in the orders table.

```
%%sql
SELECT COUNT(*)
FROM details d
LEFT JOIN orders o ON d.order_id = o.order_id
```

```
WHERE o.order_id IS NULL;
```

```
 * postgresql://postgres:***@localhost/demo
1 rows affected.
```

count
0

Orders in the orders table that do not have a corresponding entry in the details table.

```
%%sql
SELECT COUNT(*)
FROM orders o
LEFT JOIN details d ON o.order_id = d.order_id
WHERE d.order_id IS NULL;
```

```
 * postgresql://postgres:***@localhost/demo
1 rows affected.
```

count
0

We are now prepared to conduct our exploratory data analysis (EDA) on the dataset by addressing a series of key questions.

1. List the states along with their corresponding total amount of orders made, from highest to lowest

SQL code comments
Selecting the columns of interest, aliasing the aggregate function sum() on the amount column using the clause 'as'.
Joining the tables which are also aliased as o and d, on their common column which is 'order_id'.
Whenever on our selection use aggregated columns like in our case the sum(d.amount) then we have to 'GROUP BY' all the other
non aggregated columns (in our case the o.state column).
'o.state' means that column state is extracted from the table orders which was aliased as o and the d.amount refers to the 'amount'
column that comes from the 'details' table aliased as d.
Finaly we order the resulting table by sorting the total_amount column in descending order(from highest to lowest)

```
%%sql
SELECT
o.state, sum(d.amount) as total_amount
FROM details as d
INNER JOIN orders as o
ON d.order_id = o.order_id
GROUP BY o.state
ORDER BY total_amount DESC;
```

```
 * postgresql://postgres:***@localhost/demo
19 rows affected.
```

state	total_amount
Maharashtra	102498
Madhya Pradesh	87463
Uttar Pradesh	38362
Delhi	22957
Rajasthan	22334
Gujarat	21371
Punjab	16786
West Bengal	14328
Kerala	13871
Bihar	13417
Andhra Pradesh	13256
Karnataka	12520
Nagaland	11993
Jammu and Kashmir	10829
Haryana	8863
Himachal Pradesh	8666
Goa	6705
Tamil Nadu	6276
Sikkim	5276

2. Find the customer with the highest total amount of order, along with his/her order_id, state and city

SQL code comments
This question cannot be answered by using only the two tables we have in our dataset. For this reason, we utilize 2 CTEs to reach the answer.
We create the first CTE naming it 'order_totals' with the total_amount for every 'order_id'.
Then, we create a second CTE named 'max_order', that selects the order_id from the first CTE that meets a specific criterion, that is the total_amount to be the maximum out of all orders.
This results in only one order_id. The one which has the maximum total_amount.
Finally, we select the columns of interest from our tables and join them with the result of the second CTE, so as to get the information only for the order with the maximum amount.
In our final selection, since we are computing an aggregate column 'SUM(d.amount) as total_amount' **we should not forget** to group by, all the other selected non aggregated columns

```
%%sql
WITH order_totals AS (
    SELECT
```

```
        o.order_id,
        SUM(d.amount) AS total_amount
    FROM orders AS o
    INNER JOIN details AS d
    ON o.order_id = d.order_id
    GROUP BY o.order_id
),
max_order AS (
    SELECT
        order_id
    FROM order_totals
    WHERE total_amount = (SELECT MAX(total_amount) FROM order_totals)
)
SELECT
    o.order_id,
    o.customername,
    SUM(d.amount) as total_amount,
    o.state,
    o.city
FROM details AS d
INNER JOIN orders AS o
ON d.order_id = o.order_id
INNER JOIN max_order AS mo
ON o.order_id = mo.order_id
GROUP BY o.order_id, o.customername, o.state, o.city;
```

```
 * postgresql://postgres:***@localhost/demo
1 rows affected.
```

order_id	customername	total_amount	state	city
B-26055	Harivansh	9902	Uttar Pradesh	Mathura

3. Find the top 3 most profitable customers in each state

SQL code comments

Using the same strategy as we did in the previous query. We create 2 CTEs, so as to make our final selection on the second CTE meeting a specific criterion.

A reasonable question would be, why do we need 2 CTEs to reach our answer?

The reason is that in the second CTE named ranked_customers we make use of the Window function DENSE_RANK() that includes in the ORDER BY clause an aggregated column, and this cannot be completed in one query.

For this reason we create the first CTE named customers_profit where we compute the aggregated column SUM(d.profit) AS total_profit, and in the second

CTE we make use of the window function DENSE_RANK() with the aid of the aggregated column that is already created in the previous CTE.

The rank is performed with respect to the total_profit, therefore in order to get the top 3 most profitable customers, we filter for the customers with

rank 1,2 or 3 using profit_rank <= 3.

```
%%sql
WITH customers_profit AS (
    SELECT
        o.state,
        o.city,
```

```
        o.customername,
        SUM(d.profit) AS total_profit
    FROM orders AS o
    INNER JOIN details AS d
    ON o.order_id = d.order_id
    GROUP BY o.state, o.city, o.customername
),
ranked_customers AS (
    SELECT
        state,
        city,
        customername,
        total_profit,
        DENSE_RANK() OVER (PARTITION BY state ORDER BY total_profit DESC) AS profit_rank
    FROM customers_profit
)
SELECT
    state,
    city,
    customername,
    total_profit
FROM ranked_customers
WHERE profit_rank <= 3
ORDER BY state, city, profit_rank;
```

```
 * postgresql://postgres:***@localhost/demo
57 rows affected.
```

state	city	customername	total_profit
Andhra Pradesh	Hyderabad	Manju	799
Andhra Pradesh	Hyderabad	Krutika	391
Andhra Pradesh	Hyderabad	Devendra	254
Bihar	Patna	Amol	745
Bihar	Patna	Yogesh	594
Bihar	Patna	Akshay	378
Delhi	Delhi	Madhav	864
Delhi	Delhi	Diwakar	420
Delhi	Delhi	Amruta	377
Goa	Goa	Abhishek	685
Goa	Goa	Vikash	278
Goa	Goa	Kanak	72
Gujarat	Ahmedabad	Bharat	1143
Gujarat	Ahmedabad	Gaurav	970
Gujarat	Surat	Dashyam	609
Haryana	Chandigarh	Mukesh	610
Haryana	Chandigarh	Ankit	352
Haryana	Chandigarh	Aniket	296
Himachal Pradesh	Simla	Aastha	873
Himachal Pradesh	Simla	Pooja	434
Himachal Pradesh	Simla	Vandana	179
Jammu and Kashmir	Kashmir	Kirti	281
Jammu and Kashmir	Kashmir	Pinky	189
Jammu and Kashmir	Kashmir	Srishti	137
Karnataka	Bangalore	Arpita	573
Karnataka	Bangalore	Shruti	232
Karnataka	Bangalore	Ajay	215
Kerala	Thiruvananthapuram	Shourya	984
Kerala	Thiruvananthapuram	Hemant	455
Kerala	Thiruvananthapuram	Sharda	398
Madhya Pradesh	Bhopal	Abhijeet	948
Madhya Pradesh	Indore	Pournamasi	1027

state	city	customername	total_profit
Madhya Pradesh	Indore	Shivanshu	820
Maharashtra	Mumbai	Bhishm	1165
Maharashtra	Pune	Sarita	1183
Maharashtra	Pune	Priyanka	1043
Nagaland	Kohima	Sudhir	512
Nagaland	Kohima	Kushal	235
Nagaland	Kohima	Nidhi	205
Punjab	Chandigarh	Shrichand	1901
Punjab	Chandigarh	Nripraj	339
Punjab	Chandigarh	Monica	173
Rajasthan	Jaipur	Paridhi	570
Rajasthan	Udaipur	Manisha	240
Rajasthan	Udaipur	Soumya	234
Sikkim	Gangtok	Mahima	254
Sikkim	Gangtok	K	123
Sikkim	Gangtok	Vineet	92
Tamil Nadu	Chennai	Aarushi	2059
Tamil Nadu	Chennai	Amisha	438
Tamil Nadu	Chennai	Kalyani	78
Uttar Pradesh	Mathura	Madan Mohan	2166
Uttar Pradesh	Mathura	Madhav	984
Uttar Pradesh	Mathura	Lalita	400
West Bengal	Kolkata	Parishi	696
West Bengal	Kolkata	Jesal	516
West Bengal	Kolkata	Pranjali	486

4. List the top 5 cities with the highest volume of orders

```
%%sql
SELECT
city, count(order_id) as no_of_orders
FROM orders
GROUP BY city
ORDER BY no_of_orders DESC
LIMIT 5;
```

```
 * postgresql://postgres:***@localhost/demo
5 rows affected.
```

city	no_of_orders
Indore	71
Mumbai	67
Chandigarh	30
Pune	27
Delhi	24

5. Which state appears to be the most profitable? List it along with the least profitable state

```
%%sql
WITH states_profit AS (
                SELECT o.state, SUM(d.profit) AS total_profit
                FROM details AS d
                INNER JOIN orders AS o
                ON d.order_id = o.order_id
        GROUP BY o.state
)
SELECT s.state, s.total_profit
FROM states_profit AS s
WHERE s.total_profit = (SELECT MAX(total_profit) FROM states_profit)
   OR s.total_profit = (SELECT MIN(total_profit) FROM states_profit);
```

```
 * postgresql://postgres:***@localhost/demo
2 rows affected.
```

state	total_profit
Madhya Pradesh	7382
Rajasthan	-323

6. Which subcategories of products appear as the top 10 Sellers, with rescept to the volume of orders? List them along with their total profit

SQL code comment
Since we are sorting in descending order for the order volume, using LIMIT 10, will result to the top 10 subcategories with the most volume of orders.

```
%%sql
SELECT
sub_category, sum(quantity) as order_volume, sum(profit) as total_profit
FROM details
GROUP BY sub_category
ORDER BY order_volume DESC
LIMIT 10;
```

```
 * postgresql://postgres:***@localhost/demo
10 rows affected.
```

sub_category	order_volume	total_profit
Saree	795	4057
Hankerchief	741	1823
Stole	671	2431
Furnishings	310	-806
T-shirt	305	1500
Phones	304	1847
Electronic Games	297	-644
Bookcases	297	6516
Printers	291	8606
Chairs	277	1627

It seems that something went probably wrong in the 'Electronic Games' and 'Furnishings' subcategories.
Both sub-categories demonstrated negative profits, or operating loss Let's compare the average profits for this top 10

```
%%sql
SELECT
sub_category, ROUND(AVG(profit), 2) AS avg_profit
FROM details
WHERE sub_category IN (
    SELECT sub_category
    FROM (
        SELECT sub_category
        FROM details
        GROUP BY sub_category
        ORDER BY SUM(quantity) DESC
        LIMIT 10
    ) AS top_subcategories
)
GROUP BY sub_category
ORDER BY avg_profit DESC;
```

```
 * postgresql://postgres:***@localhost/demo
10 rows affected.
```

sub_category	avg_profit
Printers	116.30
Bookcases	82.48
Phones	22.25
Chairs	21.99
T-shirt	19.48
Saree	19.23
Stole	12.66
Hankerchief	9.25
Electronic Games	-8.15
Furnishings	-11.04

Among the top 10 subcategory sellers, only 'Electronic Games' and 'Furnishings' exhibit a negative average profit, indicating an average operational loss. Aggressive discounts and sell-offs on specific titles within these subcategories could be a reasonable cause. Further investigation is needed to review the pricing policy and promotional tactics. Adjustments should be considered to avoid similar issues in the future.

7. Which payment method is more popular in total?

SQL code comments
As we saw in previous queries, CTEs are valuable tools for helping us gain insights from the data tables.
The main characteristic of CTEs is that they live and die within the query they are used in. They are not stored, so if you need to use them in a later query, you must repeat the code for each query that requires it.
This is where a VIEW comes in handy. By creating a VIEW, you generate a 'virtual table' that is stored in the database as a query definition.
When you query the VIEW, the database executes the underlying SQL query, pulling the necessary data from the original tables in real-time. Unlike CTEs, a VIEW will exist beyond a single query but across multiple sessions if needed.
This allows you to reuse the VIEW in later queries or future sessions, without having to redefine it each time. You simply call it by its alias.

```
%%sql
DROP VIEW IF EXISTS payment_method_frequencies;
CREATE VIEW payment_method_frequencies AS
WITH payments_table AS (
    SELECT
                d.order_id,
        paymentmode,
        CASE WHEN paymentmode = 'COD' THEN 'Cash' ELSE NULL END AS cash_method,
        CASE WHEN paymentmode = 'Credit Card' THEN 'Credit Card' ELSE NULL END AS credit_ca
        CASE WHEN paymentmode = 'UPI' THEN 'UPI' ELSE NULL END AS web_payment,
```

```
        CASE WHEN paymentmode = 'Debit Card' THEN 'Debit Card' ELSE NULL END AS debit_card,
        CASE WHEN paymentmode = 'EMI' THEN 'EMI' ELSE NULL END AS monthly_instalments
    FROM details as d
)
SELECT
        order_id,
    COUNT(paymentmode) AS no_of_transcations,
    ROUND(COUNT(cash_method) * 100.0 / COUNT(*), 2) AS cash_percentage,
    ROUND(COUNT(credit_card) * 100.0 / COUNT(*), 2) AS credit_card_percentage,
    ROUND(COUNT(web_payment) * 100.0 / COUNT(*), 2) AS web_payment_percentage,
    ROUND(COUNT(debit_card) * 100.0 / COUNT(*), 2) AS debit_card_percentage,
    ROUND(COUNT(monthly_instalments) * 100.0 / COUNT(*), 2) AS monthly_instalments_percenta
FROM payments_table
GROUP BY order_id;

SELECT round(avg(cash_percentage), 2) as cash_percentage,
round(avg(credit_card_percentage), 2) as credit_card_percentage,
round(avg(web_payment_percentage), 2) as web_payment_percentage,
round(avg(debit_card_percentage), 2) as debit_card_percentage,
round(avg(monthly_instalments_percentage), 2) as monthly_instalments_percentage
FROM payment_method_frequencies;
```

```
 * postgresql://postgres:***@localhost/demo
Done.
Done.
1 rows affected.
```

cash_percentage	credit_card_percentage	web_payment_percentage	debit_card_percentage	monthly_
45.39	10.63	22.45	13.65	

Nearly half of the transactions are performed in cash, with the second most popular method being web payments (likely through mobile or internet banking), at around 22%.

8. Does the payment method frequency change across the states and cities?

```
%%sql
SELECT
    o.state,
    ROUND(AVG(m.cash_percentage),2) AS cash_percentage,
    ROUND(AVG(m.credit_card_percentage),2) AS credit_card_percentage,
    ROUND(AVG(m.web_payment_percentage),2) AS web_payment_percentage,
    ROUND(AVG(m.debit_card_percentage),2) AS debit_card_percentage,
    ROUND(AVG(m.monthly_instalments_percentage),2) AS monthly_instalments_percentage
FROM orders AS o
INNER JOIN payment_method_frequencies AS m
ON o.order_id = m.order_id
GROUP BY o.state;
```

```
 * postgresql://postgres:***@localhost/demo
19 rows affected.
```

state	cash_percentage	credit_card_percentage	web_payment_percentage	debit_card_percenta
Haryana	54.40	8.57	18.93	16.
Madhya Pradesh	50.01	10.49	22.04	10.
Tamil Nadu	42.71	11.46	35.42	10.
Rajasthan	38.21	10.42	23.24	19.
Sikkim	27.78	0.00	34.44	26.
Maharashtra	47.67	8.71	23.57	13.
Bihar	39.69	10.68	16.35	17.
Jammu and Kashmir	51.94	14.74	9.60	21.
Kerala	38.30	12.01	18.89	14.
Gujarat	39.71	16.58	20.58	13.
Goa	53.12	9.52	31.80	4.
Punjab	46.33	7.33	29.33	8.
Himachal Pradesh	61.43	9.52	8.33	16.
Nagaland	34.00	15.74	21.26	17.
Karnataka	38.04	16.85	25.66	9.
Delhi	47.83	10.28	25.07	11.
Uttar Pradesh	49.65	16.46	18.25	9.
West Bengal	47.65	3.49	16.74	17.
Andhra Pradesh	25.85	13.36	29.63	17.

9. There is significant variation in payment methods across different states. This variation can be quantified by calculating the standard deviation, as well as the minimum and maximum average percentages for each payment method across the states.

```
%%sql
WITH state_averages AS (
    SELECT
        o.state,
        AVG(m.cash_percentage) AS avg_cash_p,
        AVG(m.credit_card_percentage) AS avg_credit_card_p,
        AVG(m.web_payment_percentage) AS avg_web_payment_p,
        AVG(m.debit_card_percentage) AS avg_debit_card_p,
        AVG(m.monthly_instalments_percentage) AS avg_monthly_instalments_p
    FROM orders AS o
    INNER JOIN payment_method_frequencies AS m
    ON o.order_id = m.order_id
```

```
    GROUP BY o.state
)
SELECT
    ROUND(STDDEV(avg_cash_p), 2) AS std_cash_p,
    ROUND(MIN(avg_cash_p), 2) AS min_cash_p,
    ROUND(MAX(avg_cash_p), 2) AS max_cash_p,
    ROUND(STDDEV(avg_credit_card_p), 2) AS std_credit_card_p,
    ROUND(MIN(avg_credit_card_p), 2) AS min_credit_p,
    ROUND(MAX(avg_credit_card_p), 2) AS max_credit_card_p,
    ROUND(STDDEV(avg_web_payment_p), 2) AS std_web_payment_p,
    ROUND(MIN(avg_web_payment_p), 2) AS min_web_payment_p,
    ROUND(MAX(avg_web_payment_p), 2) AS max_web_payment_p,
    ROUND(STDDEV(avg_debit_card_p), 2) AS std_debit_card_p,
    ROUND(MIN(avg_debit_card_p), 2) AS min_debit_card_p,
    ROUND(MAX(avg_debit_card_p), 2) AS max_debit_card_p,
    ROUND(STDDEV(avg_monthly_instalments_p), 2) AS std_monthly_instalments_p,
    ROUND(MIN(avg_monthly_instalments_p), 2) AS min_monthly_instalments_p,
    ROUND(MAX(avg_monthly_instalments_p), 2) AS max_monthly_instalments_p
FROM state_averages;
```

```
 * postgresql://postgres:***@localhost/demo
1 rows affected.
```

std_cash_p	min_cash_p	max_cash_p	std_credit_card_p	min_credit_p	max_credit_card_p	std_web_pa
9.15	25.85	61.43	4.38	0.00	16.85	

Our initial observation of significant variation in payment methods across the states was confirmed by the computed standard deviation. Additional data on demographics and the financial status of each state and its residents would be required for a more granular analysis

10. Do product categories also influence the choice of payment methods? How the payment method ratios are distributed across the product categories

```
%%sql
SELECT d.category,
       COUNT(m.order_id) AS total_orders,
       ROUND(AVG(m.cash_percentage), 2) AS avg_cash_percentage,
       ROUND(AVG(m.credit_card_percentage), 2) AS avg_credit_card_percentage,
       ROUND(AVG(m.web_payment_percentage), 2) AS avg_web_payment_percentage,
       ROUND(AVG(m.debit_card_percentage), 2) AS avg_debit_card_percentage,
       ROUND(AVG(m.monthly_instalments_percentage), 2) AS avg_monthly_instalments_percentag
FROM payment_method_frequencies AS m
INNER JOIN details AS d
ON m.order_id = d.order_id
GROUP BY d.category
ORDER BY d.category;
```

```
 * postgresql://postgres:***@localhost/demo
3 rows affected.
```

category	total_orders	avg_cash_percentage	avg_credit_card_percentage	avg_web_payment_percent
Clothing	949	46.75	9.55	2
Electronics	308	43.28	13.26	2
Furniture	243	44.06	12.96	2

No significant difference is observed in the average percentage of payment methods across the different categories of products.

11. Could the amount of order affect the choice of payment method used?

```
%%sql
SELECT
    CASE
        WHEN d.amount < 100 THEN 'Small'
        WHEN d.amount BETWEEN 100 AND 1000 THEN 'Medium'
        ELSE 'High'
    END AS amount_category,
    COUNT(m.order_id) AS total_orders,
    ROUND(AVG(m.cash_percentage), 2) AS avg_cash_percentage,
    ROUND(AVG(m.credit_card_percentage), 2) AS avg_credit_card_percentage,
    ROUND(AVG(m.web_payment_percentage), 2) AS avg_web_payment_percentage,
    ROUND(AVG(m.debit_card_percentage), 2) AS avg_debit_card_percentage,
    ROUND(AVG(m.monthly_instalments_percentage), 2) AS avg_monthly_instalments_percentage
FROM payment_method_frequencies AS m
INNER JOIN details AS d
ON m.order_id = d.order_id
GROUP BY amount_category
ORDER BY amount_category;
```

```
 * postgresql://postgres:***@localhost/demo
3 rows affected.
```

amount_category	total_orders	avg_cash_percentage	avg_credit_card_percentage	avg_web_payment_
High	99	33.64	20.15	
Medium	739	45.28	11.71	
Small	662	47.75	8.54	

The results are interesting but somewhat anticipated.
As the order amount increases, the use of cash payments decreases.
Conversely, credit card usage increases with higher order amounts, becoming a more preferred method.
Debit cards and web payments show minimal changes for small and medium amounts but slightly decline for higher amounts.
Finally, the need for and use of monthly instalments nearly triples when the order amount becomes very high.

12. Rank the top 5 customers in each state, from the most to least profitable

```
%%sql
WITH RankedCustomers AS (
    SELECT
        customername,
        state,
        profit,
        DENSE_RANK() OVER (PARTITION BY state ORDER BY profit DESC) AS profit_rank
    FROM orders AS o
    INNER JOIN details AS d
```

```
    ON o.order_id = d.order_id
)
SELECT
state, customername, profit, profit_rank
FROM RankedCustomers
WHERE profit_rank <= 5
ORDER BY state, profit_rank;
```

```
 * postgresql://postgres:***@localhost/demo
95 rows affected.
```

state	customername	profit	profit_rank
Andhra Pradesh	Manju	547	1
Andhra Pradesh	Krutika	352	2
Andhra Pradesh	Devendra	254	3
Andhra Pradesh	Shishu	148	4
Andhra Pradesh	Soumyabrata	144	5
Bihar	Amol	701	1
Bihar	Yogesh	594	2
Bihar	Akshay	312	3
Bihar	Akshay	215	4
Bihar	Sonal	161	5
Delhi	Yohann	342	1
Delhi	Diwakar	323	2
Delhi	Madhav	305	3
Delhi	Amruta	251	4
Delhi	Madhav	208	5
Goa	Abhishek	215	1
Goa	Vikash	206	2
Goa	Abhishek	199	3
Goa	Abhishek	192	4
Goa	Kanak	76	5
Gujarat	Bharat	1148	1
Gujarat	Gaurav	721	2
Gujarat	Dashyam	568	3
Gujarat	Shardul	340	4
Gujarat	Kartik	225	5
Haryana	Mukesh	447	1
Haryana	Ankit	352	2
Haryana	Aniket	225	3
Haryana	Anjali	206	4
Haryana	Mukesh	101	5
Himachal Pradesh	Aastha	782	1
Himachal Pradesh	Pooja	503	2

state	customername	profit	profit_rank
Himachal Pradesh	Aastha	64	3
Himachal Pradesh	Vandana	56	4
Himachal Pradesh	Vandana	54	5
Jammu and Kashmir	Kirti	208	1
Jammu and Kashmir	Vijay	193	2
Jammu and Kashmir	Pinky	179	3
Jammu and Kashmir	Pinky	151	4
Jammu and Kashmir	Srishti	137	5
Karnataka	Arpita	573	1
Karnataka	Shruti	262	2
Karnataka	Vini	183	3
Karnataka	Ajay	170	4
Karnataka	Noopur	99	5
Kerala	Shourya	564	1
Kerala	Girase	536	2
Kerala	Hemant	421	3
Kerala	Snel	371	4
Kerala	Sharda	346	5
Madhya Pradesh	Sauptik	802	1
Madhya Pradesh	Pournamasi	742	2
Madhya Pradesh	Bhawna	658	3
Madhya Pradesh	Abhijeet	567	4
Madhya Pradesh	Anudeep	527	5
Maharashtra	Vrinda	1151	1
Maharashtra	Savi	980	2
Maharashtra	Sarita	712	3
Maharashtra	Priyanka	680	4
Maharashtra	Uudhav	665	5
Nagaland	Sudhir	460	1
Nagaland	Farah	318	2
Nagaland	Farah	102	3
Nagaland	Nidhi	77	4

state	customername	profit	profit_rank
Nagaland	Nidhi	75	5
Punjab	Shrichand	1303	1
Punjab	Shrichand	624	2
Punjab	Nripraj	199	3
Punjab	Paromita	154	4
Punjab	Monica	119	5
Rajasthan	Manisha	240	1
Rajasthan	Paridhi	239	2
Rajasthan	Soumya	234	3
Rajasthan	Paridhi	143	4
Rajasthan	Divsha	97	5
Sikkim	Mahima	254	1
Sikkim	K	123	2
Sikkim	Amit	50	3
Sikkim	Amit	48	4
Sikkim	Vineet	41	5
Tamil Nadu	Aarushi	1864	1
Tamil Nadu	Amisha	392	2
Tamil Nadu	Aarushi	385	3
Tamil Nadu	Amisha	153	4
Tamil Nadu	Aarushi	107	5
Uttar Pradesh	Madan Mohan	1698	1
Uttar Pradesh	Madhav	1050	2
Uttar Pradesh	Madan Mohan	433	3
Uttar Pradesh	Lalita	247	4
Uttar Pradesh	Manshul	212	5
West Bengal	Pranjali	486	1
West Bengal	Ayush	292	2
West Bengal	Jesal	198	3
West Bengal	Parishi	195	4
West Bengal	Atharv	170	5

13. How has the profit evolved over the year, quarter by quarter?

```
%%sql
WITH quarter_profit AS (
    SELECT
        EXTRACT(QUARTER FROM TO_DATE(order_date, 'DD-MM-YYYY')) AS quarter,
        SUM(profit) AS total_quarter_profit
    FROM details AS d
    INNER JOIN orders AS o
    ON d.order_id = o.order_id
    GROUP BY
        EXTRACT(QUARTER FROM TO_DATE(order_date, 'DD-MM-YYYY'))
),
cumulative_profits AS (
    SELECT
        quarter,
        total_quarter_profit,
        SUM(total_quarter_profit) OVER (ORDER BY quarter) AS cumulative_quarter_profit
    FROM quarter_profit
)
SELECT
quarter, total_quarter_profit, cumulative_quarter_profit
FROM cumulative_profits
ORDER BY quarter;
```

```
 * postgresql://postgres:***@localhost/demo
4 rows affected.
```

quarter	total_quarter_profit	cumulative_quarter_profit
1	25942	25942
2	882	26824
3	-1469	25355
4	11608	36963

The first quarter significantly impacts overall profitability, accounting for around 70% of the total profit. Profitability declines in the second quarter and turns into a loss in the third quarter. The company returns to profitability in the fourth quarter. Without more information/data about the company's main activity, we cannot explore the causes of this seasonality further. Also diving deeper and computing the company's result over time across all 19 states will not be an easy task without the aid of visualizations.
One more thing we can do is to check monthly profit trends and examine how profitability varies across different product categories.

14. How has the company's profit evolved over the year, month by month?

```
%%sql
WITH MonthlyProfits AS (
    SELECT
        EXTRACT(MONTH FROM TO_DATE(order_date, 'DD-MM-YYYY')) AS month,
        SUM(profit) AS total_monthly_profit
    FROM orders AS o
    INNER JOIN details AS d
```

```
    ON o.order_id = d.order_id
    GROUP BY EXTRACT(MONTH FROM TO_DATE(order_date, 'DD-MM-YYYY'))
),
CumulativeProfits AS (
    SELECT
        month,
        total_monthly_profit,
        SUM(total_monthly_profit) OVER (ORDER BY month) AS cumulative_profit
    FROM MonthlyProfits
)
SELECT
month, total_monthly_profit, cumulative_profit
FROM CumulativeProfits
ORDER BY month;
```

```
 * postgresql://postgres:***@localhost/demo
12 rows affected.
```

month	total_monthly_profit	cumulative_profit
1	9684	9684
2	8465	18149
3	7793	25942
4	4192	30134
5	-3730	26404
6	420	26824
7	-2138	24686
8	2068	26754
9	-1399	25355
10	2959	28314
11	10253	38567
12	-1604	36963

During the months of May, July, September, and December, the company produced a negative result(loss).
Let's move one step down and see what happened during these months.

15. Compute the total profit for the identified problematic months, categorized by product category

```
%%sql
SELECT
SUM(profit) AS total_profit, category
FROM details AS d
INNER JOIN orders AS o
ON d.order_id = o.order_id
WHERE EXTRACT(MONTH FROM TO_DATE(order_date, 'DD-MM-YYYY')) IN (5, 7, 9, 12)
GROUP BY category;
```

```
 * postgresql://postgres:***@localhost/demo
3 rows affected.
```

total_profit	category
-1046	Furniture
-6922	Electronics
-903	Clothing

The result of the above query indicates a severe loss impact associated with the 'Electronics' category. This issue has been previously identified in earlier queries. A detailed investigation into this category is recommended to address the underlying causes and develop targeted strategies for improvement.

Closing EDA comments and proposals

Based on the exploratory data analysis, we recommend the following business proposals:

Re-evaluate Pricing Policy

Conduct a thorough assessment of the current pricing strategy to ensure alignment with profitability objectives. Consider adjusting pricing structures to improve margins.

Adjust Promotional Campaigns

Review and optimize promotional strategies, with particular attention to discount practices. Ensure that discount strategies are effective in attracting customers while maintaining profitability.

Promote Non-Cash Payment Methods

Increase efforts to promote non-cash payment methods, which are associated with higher order amounts. This could involve providing incentives for using credit cards, web payments, or monthly installments