



Launchpad 2024 - Analytics Case Study

Syed Ammad Ahmed

Table of Contents

Introduction.....	2
Tasks: 1. Data Exploration:.....	2
1.1 Total number of orders:	2
1.2 Total sales revenue:.....	2
1.3 Average order quantity:	3
1.4 Distribution of orders by warehouse and store:	3
1.5 Top selling items:.....	4
Tasks: 2. Analytical Questions:.....	5
2.1 Overall discount rate (average discount per item sold):	5
2.2 Warehouse with the highest average order value:	5
2.3 Total revenue generated by each store:.....	6
2.4 Top 5 customers based on total amount spent:.....	6
2.5 Month-over-month growth rate of sales revenue:	7
2.6 Percentage of canceled orders:.....	8
Task 3. Data Visualization:	9
3.1 Time series plot of sales revenue over time:	9
3.2 Bar chart showing total revenue by store:.....	9
3.3 Pie chart illustrating the distribution of orders by warehouse:	10
Task 4. Insights and Recommendations	10
Distribution of Orders by Warehouse:	10
How to Increase Customers Spending on Shopping:	11
Sales Revenue Over Time:.....	11
Total Revenue by Stores:	11

Introduction

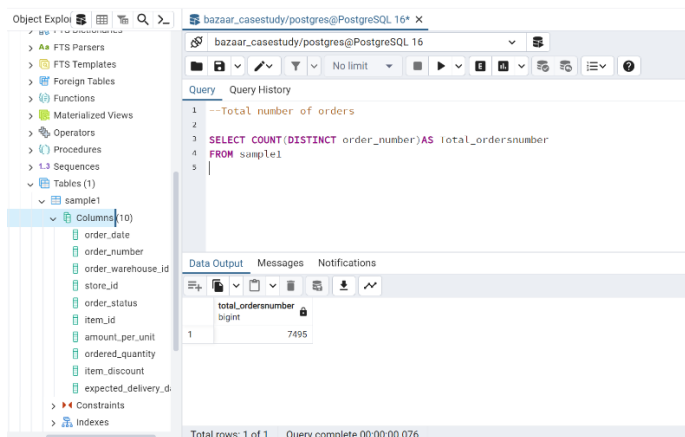
This report delves into a comprehensive exploration of key performance metrics and analytical questions, using a dataset from table sample1. We begin with a detailed examination of the total number of orders, sales revenue, and average order quantities, followed by an in-depth analysis of the distribution of orders by warehouse and store, and identification of top-selling items.

Drive link of working files: https://drive.google.com/drive/folders/1y1hFWHkrIN9nhixA6Uq_xh149-1uZ0rh?usp=drive_link

Tasks: 1. Data Exploration:

1.1 Total number of orders:

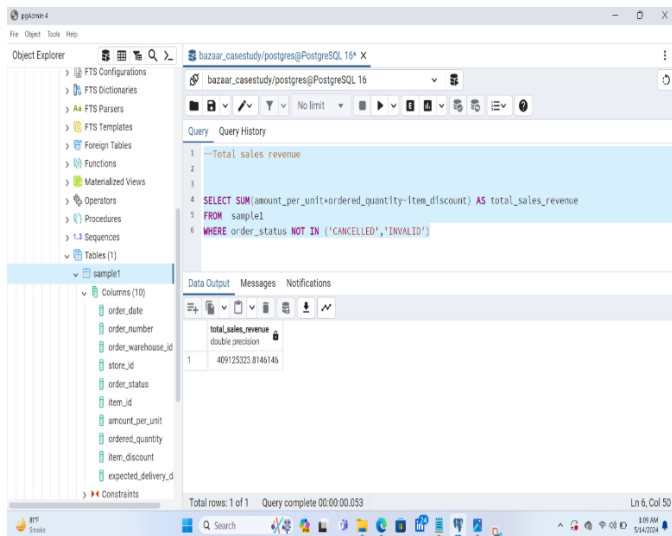
This query will count the distinct order numbers from the table sample1 to get the total number of orders.



By using **COUNT(DISTINCT order_number)** This function counts the unique instances of order_number in the table, ensuring that each order is only counted once, even if it appears multiple times.

1.2 Total sales revenue:

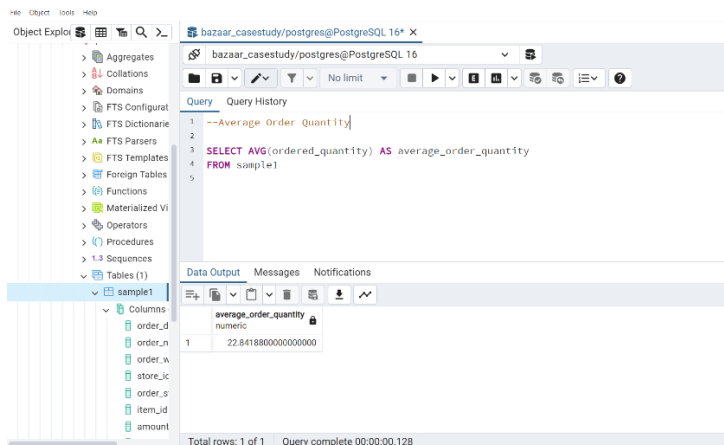
The result of this query will be a single value representing the total sales revenue, excluding cancelled or invalid orders.



First the **WHERE** clause filters the data to exclude orders with a status of '**CANCELLED**' or '**INVALID**' in table sample1 then **SUM(amount_per_unit * ordered_quantity - item_discount)** query calculates the total sales revenue by multiplying the amount per unit by the ordered quantity for each item, subtracting any discount, and then summing up these values for all orders.

1.3 Average order quantity:

This query will calculate the average ordered quantity from a table "sample1".



In this query **AVG(ordered_quantity)** function calculates the average value of the ordered_quantity column.

1.4 Distribution of orders by warehouse and store:

This query groups the transactions by warehouse and store, then counts the number of orders for each combination.

Query: Query History

```

--Distribution of orders by warehouse and store
SELECT order_warehouse_id AS warehouse, store_id AS store, COUNT(DISTINCT order_number) AS total_orders
FROM sample1
GROUP BY order_warehouse_id, store_id

```

Data Output

warehouse	store	total_orders
1	2	1
2	2	1
3	2	1
4	2	1
5	2	4
6	2	2
7	2	1

Total rows: 1000 of 6421 Query complete 00:00:00.191

order_warehouse_id AS warehouse this part selects the warehouse ID and renames it as “warehouse” for the output and **store_id AS store** this selects the store ID and renames it as “store” for the output. **GROUP BY order_warehouse_id, store_id** this groups the results by warehouse and store ID, so the count of orders is calculated for each unique combination.

1.5 Top selling items:

This query aggregates the total quantity sold for each item, then orders them in descending order and selects the top 5 items only.

Query: Query History

```

--Top selling items
SELECT item_id AS item, SUM(ordered_quantity) AS total_quantity_sold
FROM sample1
WHERE order_status NOT IN ('CANCELLED', 'INVALID')
GROUP BY item_id
ORDER BY total_quantity_sold DESC
LIMIT 5

```

Data Output

item	total_quantity_sold
P29wFPyKJioF9QvVm0Bm	400000
P9N14112688284	18457
P227644615428796458	11605
P5678188101788	7375
P19X35396797216954	5333

Total rows: 5 of 5 Query complete 00:00:00.097

SUM(ordered_quantity) AS total_quantity_sold function calculates the sum of the ordered_quantity column for each item, giving us the total quantity sold for that item and **ORDER BY total_quantity_sold DESC** function will show the result in descending order to get top 5 items.

Tasks: 2. Analytical Questions:

2.1 Overall discount rate (average discount per item sold):

This query calculates the average discount rate across all items sold.

The screenshot shows a PostgreSQL query editor with the following query:

```
--Overall Discount Rate (Average Discount per Item Sold)
SELECT AVG(item_discount) AS average_discount_rate
FROM sample1
WHERE order_status NOT IN ('CANCELLED', 'INVALID');
```

The Data Output tab shows the result:

average_discount_rate
-6807.162848239035

Total rows: 1 of 1 Query complete 00:00:00.063

2.2 Warehouse with the highest average order value:

This query calculates the average order value for each warehouse and selects the one with the highest value.

The screenshot shows a PostgreSQL query editor with the following query:

```
--Warehouse with the Highest Average Order Value
SELECT order_warehouse_id,
SUM(amount_per_unit * ordered_quantity - (item_discount)) / COUNT(DISTINCT order_number) AS average_order_value
FROM sample1
WHERE order_status NOT IN ('CANCELLED', 'INVALID') AND order_warehouse_id IS NOT NULL
GROUP BY order_warehouse_id
ORDER BY average_order_value DESC
LIMIT 1;
```

The Data Output tab shows the result:

order_warehouse_id	average_order_value
39	58587.5

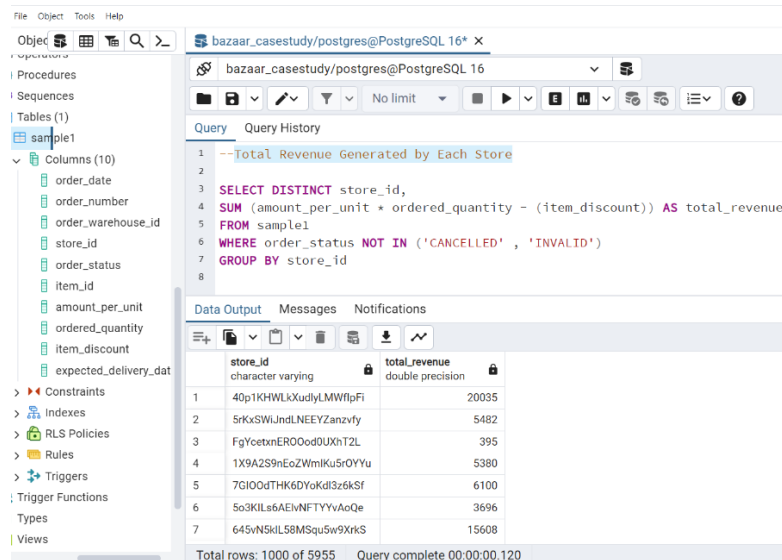
Total rows: 1 of 1 Query complete 00:00:00.223

SUM(amount_per_unit * ordered_quantity - item_discount) / COUNT(DISTINCT order_number) AS average_order_value: This part of the query calculates the average order value for each warehouse. It does so by summing the total sales (amount per unit times ordered quantity minus any item discount) and then dividing by the count of distinct orders. **WHERE order_status NOT IN ('CANCELLED', 'INVALID')** **AND order_warehouse_id IS NOT NULL:** The WHERE clause filters out orders that are cancelled or

invalid and ensures that the warehouse ID is not null and **LIMIT 1**: This limits the results to the warehouse with the highest average order value.

2.3 Total revenue generated by each store:

This query sums up the revenue for each store.



The screenshot shows a PostgreSQL query editor interface. The query is as follows:

```
--Total Revenue Generated by Each Store
SELECT DISTINCT store_id,
SUM (amount_per_unit * ordered_quantity - (item_discount)) AS total_revenue
FROM sample1
WHERE order_status NOT IN ('CANCELLED' , 'INVALID')
GROUP BY store_id
```

The query results are displayed in a table with two columns: **store_id** (character varying) and **total_revenue** (double precision). The results are as follows:

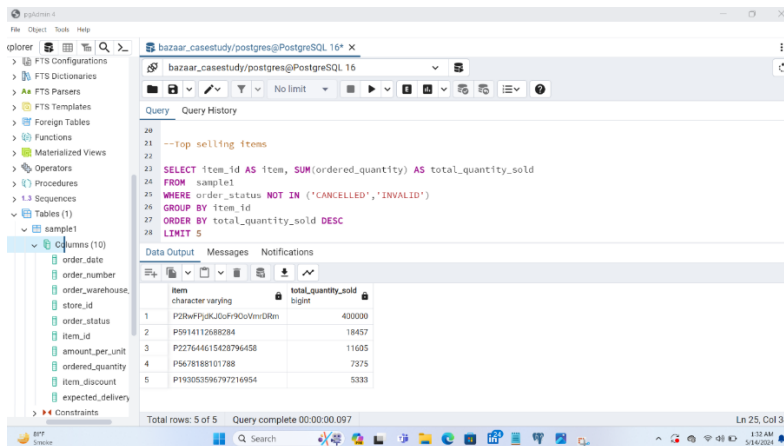
store_id	total_revenue
40p1KHwLxKudlyLMWflpFi	20035
5rKxSWJndLNEEYZanzvfy	5482
FgYcetxnEROOodUXhT2L	395
1X9A2S9nEozWmIkU5rOYYu	5380
7GIOdTHK6DYokd3z6kSf	6100
5o3KILs6AEInFTYYvAoQe	3696
645vN5kLL58MSqu5w9XrkS	15608

Total rows: 1000 of 5955 Query complete 00:00:00.120

SELECT DISTINCT store_id AS store: This part of the query selects unique store_id values from the sample1 table and renames them as “store” in the output. The **DISTINCT** keyword ensures that each store is listed only once. **GROUP BY store:** This groups the results by store_id, which has been renamed to “store”. This is necessary because the SUM function is an aggregate function that requires a GROUP BY clause to summarize data by each unique store.

2.4 Top 5 customers based on total amount spent:

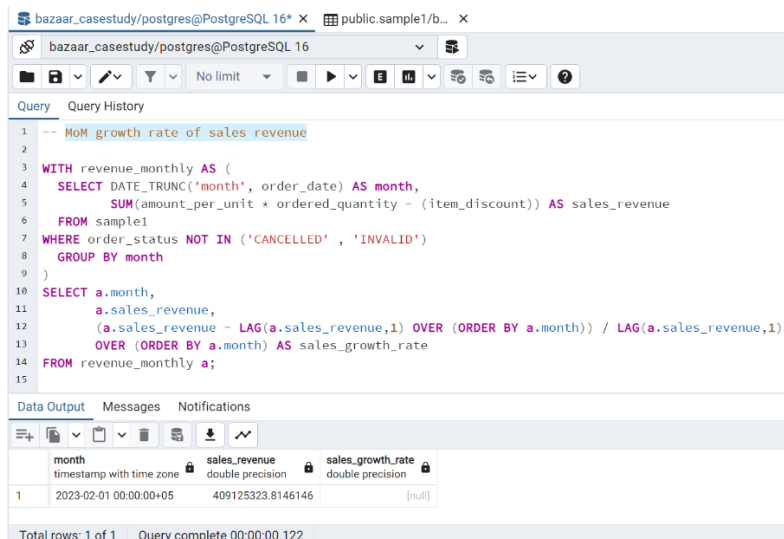
This query identifies the top 5 customers based on the total amount they've spent.



SELECT DISTINCT order_number AS customer: This part of the query selects unique order_number values from the sample1 table and renames them as “customer” in the output. The **DISTINCT** keyword ensures that each customer is listed only once.

2.5 Month-over-month growth rate of sales revenue:

This a query that calculates the Month-over-Month (MoM) growth rate of sales revenue.



WITH revenue_monthly AS: This CTE revenue_monthly is used to store the result of the subquery for use in the main query. **SELECT DATE_TRUNC('month', order_date) AS month:** This part truncates the order_date to the first day of the month, effectively grouping data by month.

The main query then selects from the CTE:

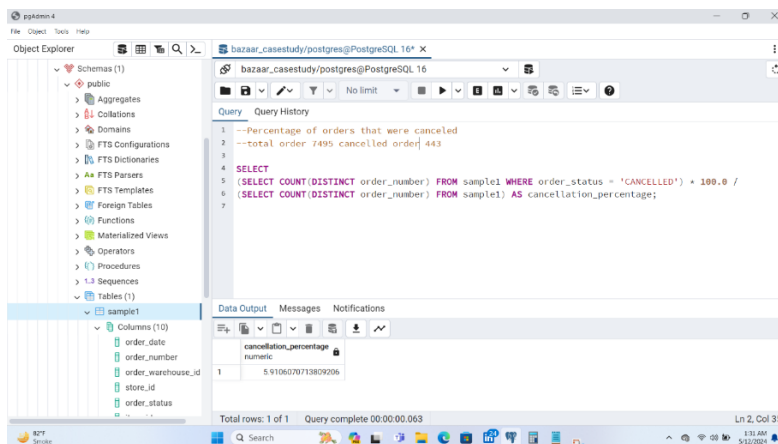
- **SELECT a.month, a.sales_revenue:** This selects the month and sales revenue from the CTE.
- **(a.sales_revenue - LAG(a.sales_revenue, 1) OVER (ORDER BY a.month)) / LAG(a.sales_revenue, 1) OVER (ORDER BY a.month) AS sales_growth_rate:** This part calculates the month-over-month growth rate.

growth rate. It subtracts the previous month's sales revenue (obtained using the LAG function) from the current month's sales revenue and divides the result by the previous month's sales revenue. The LAG function is a window function that provides access to a row at a given physical offset that comes before the current row.

Sales Growth Rate: This is the key result of the query. It will show the growth rate of sales revenue compared to the previous month as we only have transaction data of 1 month so sales growth rate can't be calculated.

2.6 Percentage of canceled orders:

The result of this query will be a single value representing the percentage of cancelled orders out of the total number of distinct orders.



- Subqueries:
 - Subquery 1: (SELECT COUNT(DISTINCT order_number) FROM sample1 WHERE order_status = 'CANCELLED') this subquery counts the number of cancelled orders.
 - Subquery 2: (SELECT COUNT(DISTINCT order_number) FROM sample1) this subquery calculates the total count of distinct order numbers from the entire dataset.
- Calculation:

The result of Subquery 1 (cancelled orders count) is multiplied by 100.0 to convert it to a percentage. The product is then divided by the result of Subquery 2 (total order count) to calculate the cancellation percentage.
- Alias:

The final result is given an alias named "cancellation_percentage".

Task 3. Data Visualization:

3.1 Time series plot of sales revenue over time:

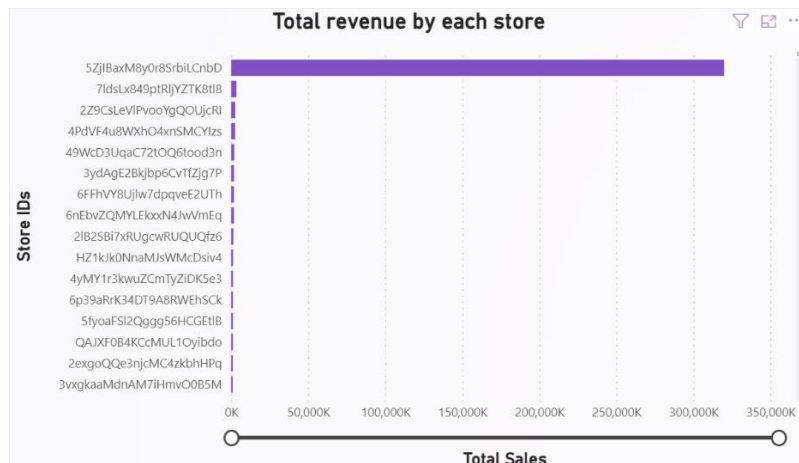
This visual report shows Time Series Plot of sales revenue over time, specifically for the month of February.



- **Order Date:** The x-axis represents the order date, which spans the entire month of February.
- **Total Sales:** The y-axis indicates the total sales revenue, with values ranging from 0K to 350K.
- **Sales Trend:** The purple line graph depicts the total sales revenue over the course of the month.
- **Significant Spike:** There is a notable spike in sales on February 3th, where the total sales reach approximately 350K.
- **Post-Spike Trend:** Following the spike, there is a sharp decline in sales, after which the sales level off and remain consistently low for the remainder of the month.

3.2 Bar chart showing total revenue by store:

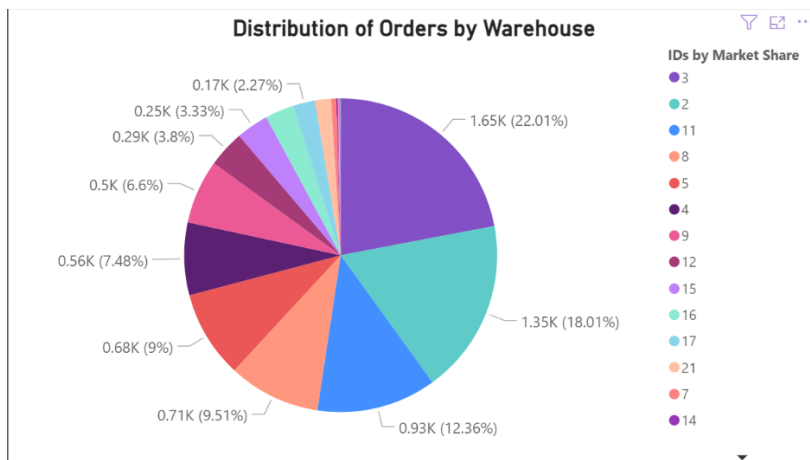
The visual report shows bar graph titled "Total revenue by each store."



- **X-Axis (Total Sales):** The horizontal axis represents the total sales in currency, ranging from 0K to 350,000K.
- **Y-Axis (Store IDs):** The vertical axis lists various Store IDs.
- **Sales Figures:** Each store's revenue is represented by a vertical dashed line. All stores have identical sales figures, as indicated by the dashed lines reaching up to approximately 350,000K on the x-axis.
- **Uniformity in Revenue:** The equal length of all dashed lines suggests that every store generated the same amount of revenue during the time period analyzed excluding store ID "5ZjlBaxM8yOr8SrbiLCnbD" its revenue reach between 300K to 350K by single order.

3.3 Pie chart illustrating the distribution of orders by warehouse:

The visual report shows pie chart that illustrates the distribution of orders by warehouse. Each slice of the pie represents a different warehouse, identified by unique IDs and colors.



- **Largest Share:** Warehouse ID 3 has the largest share of orders, accounting for 22.01% of the total, as indicated by the light purple slice.
- **Second Largest:** Warehouse ID 2 follows with 18.01% of orders, represented by the light green slice.
- **Other Warehouses:** The remaining warehouses have smaller percentages of the total orders, each represented by different colors and sizes of slices in the pie chart.

Task 4. Insights and Recommendations

Distribution of Orders by Warehouse:

The data suggests that warehouses ID 3 and ID 2 handle a significant portion of the orders, and strategies could be developed to manage the workload or to increase efficiency in other warehouses.

How to Increase Customers Spending on Shopping:

Customers spending can be increase by offering bundle package with mirror discount on specifically **Complementary Products** (This means product which are used together to achieve a common purpose).

Example:

Paint box and paint brush can be sell together by offering some discount in it, so if customer wants to buy paint box, he/she will attract and buy paint brush as well.

Some more complementary product are mention below.

1. Smartphones and Cases
2. Printers and Ink Cartridges
3. Coffee Machines and Coffee Pods/Beans
4. Bicycles and Helmets
5. Cereal and Milk

Sales Revenue Over Time:

The spike on February 3th could be due to a successful marketing campaign, a seasonal event, or other factors that temporarily boosted sales. The subsequent decline suggests that the factors contributing to the spike were short-lived. Understanding the reasons behind such spikes and drops in sales is crucial for making informed business decisions.

Total Revenue by Stores:

The uniformity in low revenue across all stores as depicted in this graph is unusual and only 1 store took responsibility of major part of revenue is alarming. When each store receive order frequently then revenue will be much higher and workload, inventory will also be organized.