

SQL Business Case – Target

Prepared By – Nabeel Abdul Rahman

Role – Data Analyst/Scientist

Date: 02-01-2024



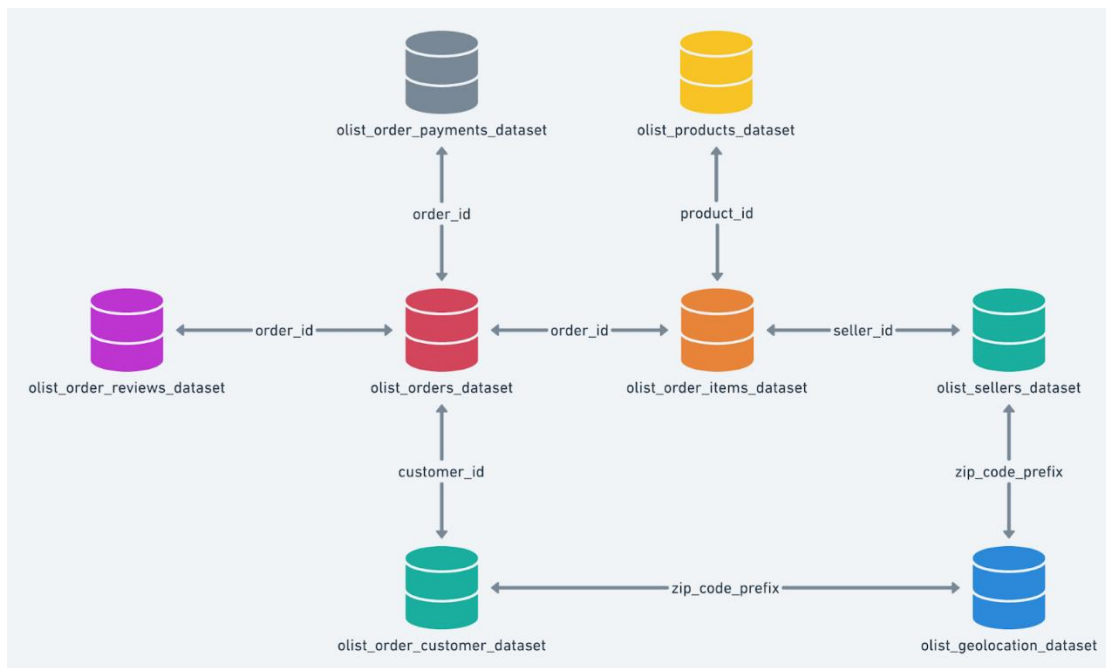
Profile Summary:

Target is a globally renowned brand and a prominent retailer in the United States. Target makes itself a preferred shopping destination by offering outstanding value, inspiration, innovation and an exceptional guest experience that no other retailer can deliver.

This business case focuses on the operations of Target in Brazil and provides insightful information about 100,000 orders placed between 2016 and 2018. The dataset offers a comprehensive view of various dimensions including the order status, price, payment and freight performance, customer location, product attributes, and customer reviews.

By analyzing this extensive dataset, it becomes possible to gain valuable insights into Target's operations in Brazil. The information can shed light on various aspects of the business, such as order processing, pricing strategies, payment and shipping efficiency, customer demographics, product characteristics, and customer satisfaction levels.

Target Dataset Schema:

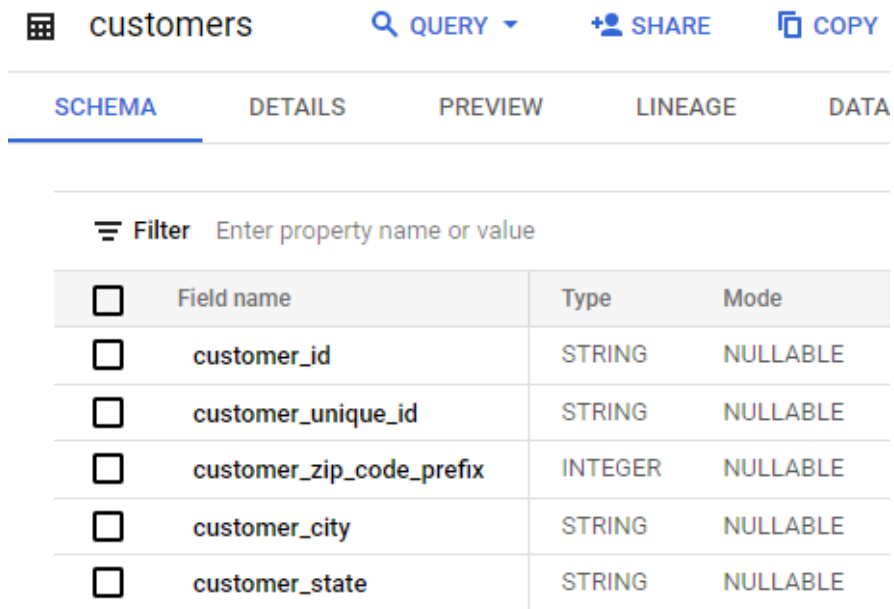







Analysis Section:

1. Import the dataset and do usual exploratory analysis steps like checking the structure & characteristics of the dataset:

A. Data type of all columns in the "customers" table.

Scheme Screenshot:



	customers	 QUERY ▾	 SHARE	 COPY
<hr/>				
<div>SCHEMA DETAILS PREVIEW LINEAGE DATA</div> <hr/>				
<div> Filter Enter property name or value</div>				
<input type="checkbox"/>	Field name	Type	Mode	
<input type="checkbox"/>	customer_id	STRING	NULLABLE	
<input type="checkbox"/>	customer_unique_id	STRING	NULLABLE	
<input type="checkbox"/>	customer_zip_code_prefix	INTEGER	NULLABLE	
<input type="checkbox"/>	customer_city	STRING	NULLABLE	
<input type="checkbox"/>	customer_state	STRING	NULLABLE	

Observation:

The customers table consists of 5 columns which particularly gives information on the table attributes such as customer id, unique id, city and state along with its data types. Here all columns except the customer zip code is string where zip code is integer.

B. Get the time range between which the orders were placed.

SQL Query Analysis 1 : Displaying when the first and last order were placed from dataset

```
SELECT MIN(order_purchase_timestamp) as first_order_purchase,  
max(order_purchase_timestamp) as last_order_purchase from `target.orders`
```

Screenshot:

Query results

JOB INFORMATION		RESULTS	CHART	PREVIEW	JSON
Row	//	first_order_purchase ▼	//	last_order_purchase ▼	//
1		2016-09-04 21:15:19 UTC		2018-10-17 17:30:18 UTC	

SQL Query Analysis 2 : Displaying the time range in seconds between every orders being purchased

```
SELECT order_id, customer_id, order_purchase_timestamp, previous_purchase_timestamp,  
TIMESTAMP_DIFF(order_purchase_timestamp,previous_purchase_timestamp,second) as  
diff_order_timestamp_seconds  
from (  
SELECT *, LAG(order_purchase_timestamp) over(order by order_purchase_timestamp) as  
previous_purchase_timestamp from `target.orders`  
order by order_purchase_timestamp)
```

Screenshot:

Query results

JOB INFORMATION		RESULTS	CHART	PREVIEW	JSON	EXECUTION DETAILS	EXECUTION GRAPH	
Row	order_id ▾	customer_id ▾	order_purchase_timestamp ▾	previous_purchase_timestamp ▾	diff_order_timestamp_seconds ▾			
1	2e7a8482f6fb09756ca50c10d...	08c5351a6aca1c1589a38f244...	2016-09-04 21:15:19 UTC	null	null			
2	e5fa5a7210941f7d56d0208e4...	683c54fc24d40ee9f8a6fc179f...	2016-09-05 00:15:34 UTC	2016-09-04 21:15:19 UTC	10815			
3	809a282bbd5dbcabb6f2f724fc...	622e13439d6b5a0b486c4356...	2016-09-13 15:24:19 UTC	2016-09-05 00:15:34 UTC	745725			
4	bfb0f9bdef84302105ad712d...	86dc2ffce2dff336de2f386a78...	2016-09-15 12:16:38 UTC	2016-09-13 15:24:19 UTC	161539			
5	71303d7e93b399f5bcd537d12...	b106b360fe2ef8849fbbd056f7...	2016-10-02 22:07:52 UTC	2016-09-15 12:16:38 UTC	1504274			
6	3b697a20d9e427646d925679...	355077684019f7f60a031656b...	2016-10-03 09:44:50 UTC	2016-10-02 22:07:52 UTC	41818			
7	be5bc2f0da14d8071e2d45451...	7ec40b22510fdbea1b08921dd...	2016-10-03 16:56:50 UTC	2016-10-03 09:44:50 UTC	25920			
8	65d1e226dfaeb8cdc42f66542...	70fc57eeae292675927697fe0...	2016-10-03 21:01:41 UTC	2016-10-03 16:56:50 UTC	14691			
9	a41c8759fbe7aab36ea07e038...	6f989332712d3222b6571b1cf...	2016-10-03 21:13:36 UTC	2016-10-03 21:01:41 UTC	715			
10	d207cc272675637bfe00062ed...	b8cf418e97ae795672d326288...	2016-10-03 22:06:03 UTC	2016-10-03 21:13:36 UTC	3147			

SQL Query Analysis 3: Displaying the average time in seconds an order is punched or purchased.

`SELECT CEIL(AVG(diff_order_timestamp_seconds)) as average_order_purchase_seconds from`

`(SELECT order_id, customer_id, order_purchase_timestamp, previous_purchase_timestamp, TIMESTAMP_DIFF(order_purchase_timestamp,previous_purchase_timestamp,second) as diff_order_timestamp_seconds from (SELECT *, LAG(order_purchase_timestamp) over(order by order_purchase_timestamp) as previous_purchase_timestamp from `target.orders` order by order_purchase_timestamp))`

Query Result Screenshot:

Query results		
JOB INFORMATION		RESULTS
Row	average_order_purchase_seconds	
1	672.0	

Observation: First query we displayed when was the first and last order was purchased. Then we calculated the time range between every orders in seconds. After analyzing the 'orders' table, we found that there is an average time difference of 672 seconds between consecutive orders. This implies that, on average, customers place an order approximately every 11.2 minutes.

C. Count the Cities & States of customers who ordered during the given period.

SQL Query:

`SELECT COUNT(DISTINCT customer_id) as total_customers, count(DISTINCT customer_city) as total_city, count(DISTINCT customer_state) as total_state from `target.customers``

Query Result Screenshot:

Query results				
JOB INFORMATION		RESULTS	CHART	PREVIEW
Row	total_customers	total_city	total_state	
1	99441	4119	27	

Observation:

From the query results, we observe that over the span of 3 years, we had a total of 99,441 customers originating from 4,119 cities across 27 states. Total customer was just an add on which was not asked in the question. That's the insight I want to know.

2. In-depth Exploration:

a. Is there a growing trend in the no. of orders placed over the past years?

SQL Query:

```
SELECT year_, count(DISTINCT order_id) total_no_orders from (  
SELECT order_id, customer_id, order_purchase_timestamp, EXTRACT(year from  
order_purchase_timestamp) as year_ from `target.orders`)  
group by year_  
order by year_
```

Query Result Screenshot:

Query results

JOB INFORMATION		RESULTS	CHART	PRE
Row	year_	total_no_orders		
1	2016	329		
2	2017	45101		
3	2018	54011		

Observation: Indeed, there is a noticeable upward trend in the number of orders placed over the past years. It's important to note that this analysis solely focuses on the quantity of orders and does not account for their status, such as delivery, cancellation, etc.

b. Can we see some kind of monthly seasonality in terms of the no. of orders being placed?

SQL Query:

```
SELECT month_no, month_name, count(DISTINCT order_id) total_no_orders  
from (  
SELECT order_id, customer_id, order_purchase_timestamp, extract(month  
from order_purchase_timestamp) as month_no,  
format_datetime("%b",order_purchase_timestamp) as month_name from  
`target.orders`)  
group by 1,2  
order by 1,2
```

Query Result Screenshot:

Query results			
JOB INFORMATION		RESULTS	CHART
		PREVIEW	JSON
Row	month_no	month_name	total_no_orders
1	1	Jan	8069
2	2	Feb	8508
3	3	Mar	9893
4	4	Apr	9343
5	5	May	10573
6	6	Jun	9412
7	7	Jul	10318
8	8	Aug	10843
9	9	Sep	4305
10	10	Oct	4959
11	11	Nov	7544
12	12	Dec	5674

Observation: Upon analyzing the query results, we observed the trend to identify recurring patterns or seasonality monthly. This approach offers a detailed perspective on how order counts fluctuate month by month over the three-year period.

It is evident that during the months of May, July, and August, there was a notable increase in orders, possibly attributed to promotions or some offers. Conversely, the months of September and October, or Q3, recorded the fewest orders. Exploring strategies to boost order counts during this period may be worthwhile. Additionally, during the year end we observe the orders were good during the month of Nov but closing Dec was less.

- c. During what time of the day, do the Brazilian customers mostly place their orders? (Dawn, Morning, Afternoon or Night)
 - i. 0-6 hrs : Dawn
 - ii. 7-12 hrs : Mornings
 - iii. 13-18 hrs : Afternoon
 - iv. 19-23 hrs : Night

SQL Query –

```
SELECT Order_Placed, count(DISTINCT order_id) as count_orders from (
SELECT *, CASE
  WHEN(time_of_order_purchase between "00:00:00" AND "06:00:00") then "Dawn"
  WHEN(time_of_order_purchase between "07:00:00" AND "12:00:00") then"Mornings"
  WHEN(time_of_order_purchase between "13:00:00" AND "18:00:00")then"Afternoon"
  ELSE "Night"
END AS Order_Placed
from (

SELECT order_id, order_purchase_timestamp, extract(time from
order_purchase_timestamp) as time_of_order_purchase from `target.orders`
order by order_purchase_timestamp))
group by Order_Placed
```

Query Result Screenshot:

Query results

JOB INFORMATION		RESULTS	CHART	PREVIEW
Row	Order_Placed	count_orders		
1	Mornings	21738		
2	Dawn	4740		
3	Afternoon	32370		
4	Night	40593		

Observation: The query results indicate that a higher number of orders are placed during the night, followed by the afternoon, and then the morning. The least number of orders are observed during the dawn hours.

3. Evolution of E-commerce orders in the Brazil region:

1. Get the month-on-month no. of orders placed in each state.

SQL Query:

```
SELECT customer_state, month_, count(DISTINCT order_id) as total_orders from (
SELECT o.order_id, c.customer_state, o.order_purchase_timestamp, EXTRACT(month
from order_purchase_timestamp) as month_ from `target.orders` as o
LEFT JOIN `target.customers` as c
ON o.customer_id = c.customer_id
order by order_purchase_timestamp)
group by 1,2
order by 1,2
```

Query Result Screenshot:

Query results

JOB INFORMATION	RESULTS	CHART	PREVIEW	JSON
Row	customer_state	month_	total_orders	
1	AC	1	8	
2	AC	2	6	
3	AC	3	4	
4	AC	4	9	
5	AC	5	10	
6	AC	6	7	
7	AC	7	9	
8	AC	8	7	
9	AC	9	5	
10	AC	10	6	
11	AC	11	5	
12	AC	12	5	
13	AL	1	39	
14	AL	2	39	
15	AL	3	40	

Observation: From the query result, we can get valuable insights into the ordering patterns across different states over time. From this we can find the seasonal trends, customer engagements within states, operational consideration which may be affecting the shipments and being less orders for those states and finally the potential opportunities.

2. How are the customers distributed across all the states?

SQL Query:

```
SELECT customer_state, count(DISTINCT customer_id) as customer_count from
`target.customers`
group by customer_state
order by customer_count desc;
```

Screenshot:

Query results		
JOB INFORMATION		RESULTS
		CHART
		PREVIEW
Row	customer_state	customer_count
1	SP	41746
2	RJ	12852
3	MG	11635
4	RS	5466
5	PR	5045
6	SC	3637
7	BA	3380
8	DF	2140
9	ES	2033
10	GO	2020

Observation: The analysis of the query results reveals that the majority of customers are from Sao Paulo state, with over 40,000 customers. Following closely are Rio de Janeiro (RJ) and Minas Gerais (MG), also boasting substantial customer numbers exceeding 10,000. In contrast, other states have customer counts below 6,000. Notably, states such as Amapa (AP-68) and Roraima (RR) exhibit the least number of customers, with figures as low as 46. This distribution highlights the regional concentration of customers, with some states demonstrating significantly higher customer bases compared to others.

4. Impact on Economy: Analyze the money movement by e-commerce by looking at order prices, freight and others.

1. Get the % increase in the cost of orders from year 2017 to 2018 (include months between Jan to Aug only).
You can use the "payment value" column in the payments table to get the cost of orders.

```
SELECT *, round(((pay_lead-total_payment)/total_payment)*100,2) as  
percentage_increase  
from (
```

```
SELECT *, LEAD(total_payment) over(order by year_) as pay_lead  
from (
```

```
SELECT extract(year from order_purchase_timestamp) as year_,  
round(sum(payment_value),2) as total_payment  
from
```

```
(SELECT  
p.order_id,o.order_purchase_timestamp,p.payment_type,p.payment_value  
from `target.orders` as o  
INNER JOIN `target.payments` as p ON o.order_id = p.order_id  
where extract(year from o.order_purchase_timestamp) = 2017 AND  
extract(month from o.order_purchase_timestamp) BETWEEN 1 AND 8  
OR  
extract(year from o.order_purchase_timestamp) = 2018 AND extract(month  
from o.order_purchase_timestamp)<=8  
order by o.order_purchase_timestamp  
)  
group by year_  
ORDER BY year_  
)
```

Screenshot:

Query results				
JOB INFORMATION		RESULTS	CHART	PREVIEW
		JSON	EXECUTION DE	
Row	year_	total_payment	pay_lead	percentage_increase
1	2017	3669022.12	8694733.84	136.98
2	2018	8694733.84	null	null

Observation:

The query results reveal a 137% increase in cost of orders from 2017 to 2018, inclusive of the months from January to August. This indicates a substantial growth in order volumes during this period, contributing to an overall increase in sales, revenue, and potentially profit year over year.

2. Calculate the Total & Average value of order price for each state.

SQL Query:

```
SELECT customer_state, ceil(sum(price)) as total_price,
round(avg(price),2) as average_price
from
(
SELECT ot.*,c.customer_state from `target.orders` as o
INNER JOIN `target.order_items` as ot ON o.order_id = ot.order_id
INNER JOIN `target.customers` as c ON o.customer_id = c.customer_id
order by customer_state)

group by customer_state
order by total_price desc
```

Screenshot:

Query results			
JOB INFORMATION		RESULTS	CHART
		PREVIEW	JSON
Row	customer_state	total_price	average_price
1	SP	5202956.0	109.65
2	RJ	1824093.0	125.12
3	MG	1585309.0	120.75
4	RS	750305.0	120.34
5	PR	683084.0	119.0
6	SC	520554.0	124.65
7	BA	511350.0	134.6
8	DF	302604.0	125.77
9	GO	294592.0	126.27
10	ES	275038.0	121.91
20	AL	80315.0	180.89
21	SE	58921.0	153.04
22	TO	49622.0	157.53
23	RO	46141.0	165.97
24	AM	22357.0	135.5
25	AC	15983.0	173.73
26	AP	13475.0	164.32
27	RR	7830.0	150.57

Observation:

The query results clearly indicate that the total cost price to the customer is predominantly contributed by the state of Sao Paulo, given the higher order volumes originating from this region. The average price per order is subsequently observed in other states such as RJ and MG.

This average price is calculated based on the number of orders or line items placed from each state, considering that each order has a specific order price.

3. Calculate the Total & Average value of order freight for each state.

SQL Query:

```
SELECT customer_state, ceil(sum(freight_value)) as total_freight_value,
round(avg(freight_value),2) as average_freight_value
from (
SELECT ot.*,c.customer_state from `target.orders` as o
INNER JOIN `target.order_items` as ot ON o.order_id = ot.order_id
INNER JOIN `target.customers` as c ON o.customer_id = c.customer_id
order by customer_state)

group by customer_state
order by total_freight_value desc
```

Screenshot:

Query results				
JOB INFORMATION		RESULTS	CHART	PREVIEW
JSON		EXECUTED		
Row	customer_state	total_freight_value	average_freight_value	
1	SP	718724.0	15.15	
2	RJ	305590.0	20.96	
3	MG	270854.0	20.63	
4	RS	135523.0	21.74	
5	PR	117852.0	20.53	
6	BA	100157.0	26.36	
7	SC	89661.0	21.47	
8	PE	59450.0	32.92	
9	GO	53115.0	22.77	
10	DF	50626.0	21.04	
20	AL	15915.0	35.84	
21	SE	14112.0	36.65	
22	TO	11733.0	37.25	
23	RO	11418.0	41.07	
24	AM	5479.0	33.21	
25	AC	3687.0	40.07	
26	AP	2789.0	34.01	
27	RR	2236.0	42.98	

Observation:

The query results clearly indicate that the total freight value price to the customer is predominantly contributed by the state of Sao Paulo, given the higher order volumes originating from this region. The average price per order is subsequently observed in other states such as RJ and MG.

This average freight price is calculated based on the number of orders or line items placed from each state, considering that each state will have a different freight price depending on the distance, location, state laws and policy and other parameter and also no of orders being placed.

5. Analysis based on sales, freight and delivery time.

- Find the no. of days taken to deliver each order from the order's purchase date as delivery time.
Also, calculate the difference (in days) between the estimated & actual delivery date of an order.
Do this in a single query.

You can calculate the delivery time and the difference between the estimated & actual delivery date using the given formula:

- time_to_deliver** = order_delivered_customer_date - order_purchase_timestamp
- diff_estimated_delivery** = order_estimated_delivery_date - order_delivered_customer_date

SQL Query:

```
SELECT order_id, order_status, order_purchase_timestamp,
order_delivered_customer_date, order_estimated_delivery_date,

DATE_DIFF(order_delivered_customer_date, order_purchase_timestamp, day) as
days_taken_to_deliver,

DATE_DIFF(order_estimated_delivery_date, order_delivered_customer_date, day) as
difference_days_from_estimated

from `target.orders`
order by order_purchase_timestamp
```

Screenshot:

Query results								
JOB INFORMATION		RESULTS	CHART	PREVIEW	JSON	EXECUTION DETAILS	EXECUTION GRAPH	
Row	order_id	order_status	order_purchase_timestamp	order_delivered_customer_date	order_estimated_delivery_date	days_taken_to_deliver	difference_days_from_estimated	
1	2e7a8482f6fb09756ca50c10d...	shipped	2016-09-04 21:15:19 UTC	null	2016-10-20 00:00:00 UTC	null	null	
2	e5fa5a7210941f7d56d0208e4...	canceled	2016-09-05 00:15:34 UTC	null	2016-10-28 00:00:00 UTC	null	null	
3	809a282bdd5dbcabb6f2f724fc...	canceled	2016-09-13 15:24:19 UTC	null	2016-09-30 00:00:00 UTC	null	null	
4	bfb0f9bdef84302105ad712d...	delivered	2016-09-15 12:16:38 UTC	2016-11-09 07:47:38 UTC	2016-10-04 00:00:00 UTC	54	-36	
5	71303d7e93b399f5bcd537d12...	canceled	2016-10-02 22:07:52 UTC	null	2016-10-25 00:00:00 UTC	null	null	
6	3b697a20d9e427646d925679...	delivered	2016-10-03 09:44:50 UTC	2016-10-26 14:02:13 UTC	2016-10-27 00:00:00 UTC	23	0	
7	be5bc2f0da14d8071e2d45451...	delivered	2016-10-03 16:56:50 UTC	2016-10-27 18:19:38 UTC	2016-11-07 00:00:00 UTC	24	10	
8	65d1e226dfaeb8cdc42f66542...	canceled	2016-10-03 21:01:41 UTC	2016-11-08 10:58:34 UTC	2016-11-25 00:00:00 UTC	35	16	
9	a41c8759f7e7aab36ea07e038...	delivered	2016-10-03 21:13:36 UTC	2016-11-03 10:58:07 UTC	2016-11-29 00:00:00 UTC	30	25	
10	d207cc272675637bfd0062ed...	delivered	2016-10-03 22:06:03 UTC	2016-10-31 11:07:42 UTC	2016-11-23 00:00:00 UTC	27	22	

Observations:

- The "days-taken-to-deliver" column provides information on the number of days it took for an order to be delivered from the purchase stamp date to the delivered date to customer.
- The "difference-days-from-estimated" column indicates the variance in days between the estimated delivery date and the actual delivery date to the customer. A negative difference signifies a delay from the estimated delivery date, while a positive difference implies early delivery.
- Blank entries in both columns should prompt a check on the "order status" to identify canceled orders or those in transit but not yet delivered.
- These two columns serve as valuable metrics for analyzing customer service performance, including the average delivery time for orders and adherence to estimated delivery dates. Identification of frequent delays can guide efforts to improve service.
- State-level analysis can further reveal variations in delivery speed, helping distinguish regions with efficient delivery from those that may require improvements due to distance or other constraints.

2. Find out the top 5 states with the highest & lowest average freight value.

SQL Query for top 5 with **highest** average freight value:

```
SELECT customer_state, round(avg(freight_value),2) as
Top_Highest_Freight_Value
from (
SELECT ot.*,c.customer_state from `target.orders` as o
INNER JOIN `target.order_items` as ot ON o.order_id = ot.order_id
INNER JOIN `target.customers` as c ON o.customer_id = c.customer_id
order by customer_state)

group by customer_state
order by Top_Highest_Freight_Value DESC LIMIT 5
```

Screenshot:

Query results		
JOB INFORMATION RESULTS CHART PREVIEW JSQ		
Row	customer_state	Top_Highest_Freight_Value
1	RR	42.98
2	PB	42.72
3	RO	41.07
4	AC	40.07
5	PI	39.15

SQL Query for top 5 with **lowest** average freight value:

```
SELECT customer_state, round(avg(freight_value),2) as  
Top_Lowest_Freight_Value  
from (  
SELECT ot.*,c.customer_state from `target.orders` as o  
INNER JOIN `target.order_items` as ot ON o.order_id = ot.order_id  
INNER JOIN `target.customers` as c ON o.customer_id = c.customer_id  
order by customer_state)  
  
group by customer_state  
order by Top_Lowest_Freight_Value ASC LIMIT 5
```

Screenshot :

Query results

JOB INFORMATION		RESULTS	CHART	PREVIEW
Row	customer_state	Top_Lowest_Freight_Value		
1	SP	15.15		
2	PR	20.53		
3	MG	20.63		
4	RJ	20.96		
5	DF	21.04		

Observation:

- The states with the highest average freight values are RR, PB, RO, AC, and PI.
- The states with the lowest average freight values are SP, PR, MG, RJ, and DF.
- Freight values can be influenced by various factors, including distance, location, freight volume and demand, fuel costs, and other market conditions and state-level rules.
- Sao Paulo has the lowest freight value, suggesting that this location experiences a higher volume of orders and demand, resulting in cost-effective shipping for bulk orders.
- It will be just opposite for the state RR where cost is high possibly due to less orders and other state level constraints which resulting in higher freight cost.

3. Find out the top 5 states with the highest & lowest average delivery time.

SQL Query: Top 5 **highest** average delivery time

```
SELECT customer_state,  
ROUND(AVG(DATE_DIFF(order_delivered_customer_date,order_purchase_timestamp,  
p,day)),2) as avg_high_delivery_days  
from (  
SELECT o.*,ot.*,c.* from `target.orders` as o  
INNER JOIN `target.order_items` as ot ON o.order_id = ot.order_id  
INNER JOIN `target.customers` as c ON o.customer_id = c.customer_id  
order by customer_state)  
  
group by customer_state  
order by avg_high_delivery_days DESC LIMIT 5
```

Screenshot:

Query results		
JOB INFORMATION		
RESULTS		
CHART		
PREVIEW		
JS		
Row	customer_state	avg_high_delivery_days
1	RR	27.83
2	AP	27.75
3	AM	25.96
4	AL	23.99
5	PA	23.3

SQL Query: Top 5 **lowest** average delivery time

```
SELECT customer_state,  
ROUND(AVG(DATE_DIFF(order_delivered_customer_date,order_purchase_timestamp,  
p,day)),2) as avg_low_delivery_days  
from (  
SELECT o.*,ot.*,c.* from `target.orders` as o  
INNER JOIN `target.order_items` as ot ON o.order_id = ot.order_id  
INNER JOIN `target.customers` as c ON o.customer_id = c.customer_id  
order by customer_state)  
  
group by customer_state  
order by avg_low_delivery_days ASC LIMIT 5
```


Screenshot:

Query results

JOB INFORMATION		RESULTS	CHART	PREVIEW
Row	customer_state	avg_low_delivery_days		
1	SP	8.26		
2	PR	11.48		
3	MG	11.52		
4	DF	12.5		
5	SC	14.52		

Observation:

As one can anticipate there may be a correlation between the freight value and lead time of delivery. As we can see sau paulo is where it takes less no of days in average to deliver the orders whereas RR take on an average 28 days to deliver an order. Same RR have high freight value as well. So its connecting that we have potential opportunity to look at the operating cost from the state RR and additionally how better we can service from SP and improve the servicing along with other states with high and low average lead times.

- Find out the top 5 states where the order delivery is really fast as compared to the estimated date of delivery.
You can use the difference between the averages of actual & estimated delivery date to figure out how fast the delivery was for each state.

SQL Query:

```
WITH base as (  
SELECT o.*,c.* from `target.orders` as o  
LEFT JOIN `target.customers` as c ON o.customer_id = c.customer_id  
order by customer_state),  
  
delivered as (  
SELECT order_id,  
order_delivered_customer_date,order_estimated_delivery_date,customer_state,  
date_diff(order_estimated_delivery_date,order_delivered_customer_date,day) as  
difference from base where order_status = "delivered")  
  
select customer_state, round(avg(difference),2) avg_days from delivered  
group by customer_state  
order by avg_days desc LIMIT 5
```

Screenshot:

Query results

JOB INFORMATION		RESULTS	CHART	PREVIEW
Row	customer_state	avg_days		
1	AC	19.76		
2	RO	19.13		
3	AP	18.73		
4	AM	18.61		
5	RR	16.41		

Observation:

- Using a Common Table Expression (CTE), I established the base table by joining the orders and customers tables.
- A subquery within the base table, named "delivered," computes the number of days as the difference between the estimated delivery date and the actual delivered-to-customer date.
- The average of the days difference is then calculated and grouped on the state level in descending order.
- The result shows the top 5 states where orders are delivered earlier than the estimated delivery date. For instance, State AC exhibits an average delivery 19.76 days ahead of the estimated date, followed by RO with a similar early delivery period. Other states in this category include AP, AM, and RR.

6. Analysis based on the payments:

- Find the month-on-month no. of orders placed using different payment types.

SQL Query:

```
With base as (  
    SELECT o.order_id, o.order_purchase_timestamp, p.payment_type,  
           extract(month from order_purchase_timestamp) as month_no,  
           format_datetime("%B", order_purchase_timestamp) as Month_ from  
    `target.orders` as o  
    INNER JOIN `target.payments` as p ON o.order_id = p.order_id  
)  
  
select Month_, month_no, payment_type, count(order_id) as  
count_orders from base  
group by month_no, Month_, payment_type  
order by month_no
```

Screenshot:

Query results

JOB INFORMATION		RESULTS	CHART	PREVIEW	JSON	EXECUTION DETAILS	E
Row	Month_	month_no	payment_type	count_orders			
1	January	1	credit_card	6103			
2	January	1	UPI	1715			
3	January	1	voucher	477			
4	January	1	debit_card	118			
5	February	2	UPI	1723			
6	February	2	credit_card	6609			
7	February	2	voucher	424			
8	February	2	debit_card	82			
9	March	3	credit_card	7707			
10	March	3	UPI	1942			
11	March	3	debit_card	109			

Observation: From the resulted query, we can clearly see the for every month that, majority of orders are being placed using credit card followed by UPI and others.

- ii. Find the no. of orders placed on the basis of the payment installments that have been paid.

SQL Query:

```
SELECT payment_installments, count(DISTINCT o.order_id) as
orders_count from `target.orders` as o
INNER JOIN `target.payments` as p ON o.order_id = p.order_id
where payment_installments >0
group by payment_installments
order by payment_installments
```

Screenshot:

Query results

JOB INFORMATION		RESULTS	CHART
Row	payment_installments	orders_count	
1	1	49060	
2	2	12389	
3	3	10443	
4	4	7088	
5	5	5234	
6	6	3916	
7	7	1623	
8	8	4253	
9	9	644	
10	10	5315	
11	11	23	
12	12	133	

Observation:

From the resulted query, we can observe how many distinct orders fall into different categories of payment installment. The grouping allows you to see which values of payment installments are more common or prevalent among the orders. This provides insights into the popular choices for payment plans.

Conclusion:

The analysis of the SQL Target business case has provided valuable insights. Through the execution of various queries, we have uncovered order patterns by year, month, payment trends, and noteworthy observations within the dataset. The key findings have been put on as observation after every query.

These insights contribute to a deeper understanding of specific areas of focus and offer actionable information for potential decisions, optimizations, or strategies. The utilization of SQL queries has proven to be effective in extracting meaningful information from the dataset, demonstrating the value of data-driven decision-making for Target in Brazil states operations.

Moving forward, these findings can serve as a foundation for potential actions or further analysis. As the dataset evolves or new data becomes available, ongoing SQL analysis will be instrumental in gaining continuous insights and adapting to dynamic business needs.