Business Case: Target SQL

Querying is done on BigQuery platform

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

   1.1. Tables present in the dataset

```
SELECT table_name
FROM `scaler-dsml-sql-406007.Business_case_Target_SQL.INFORMATION_SCHEMA.TABLES`
```

JOB INFORMATION | RESULTS

Row	table_name
1	order_items
2	sellers
3	geolocation
4	products
5	orders
6	payments
7	customers
8	order_reviews

   1.2. Data type of all columns in each table

   Data type of all columns in the "customers" table

```
SELECT column_name, data_type
FROM `scaler-dsml-sql406007.Business_case_Target_SQL.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'customers'
```

Query results | SAVE RESULTS | EXPLORE DATA

JOB INFORMATION | RESULTS | CHART PREVIEW | JSON | EXECUTION DETAILS | EXECUTION GRAPH

Row	column_name	data_type
1	customer_id	STRING
2	customer_unique_id	STRING
3	customer_zip_code_prefix	INT64
4	customer_city	STRING
5	customer_state	STRING

   Data type of all columns in the "sellers" table

```
SELECT column_name, data_type
FROM `scaler-dsml-sql-
406007.Business_case_Target_SQL.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'sellers'
```

Row	column_name	data_type
1	seller_id	STRING
2	seller_zip_code_prefix	INT64
3	seller_city	STRING
4	seller_state	STRING

Data type of all columns in the "geolocation" table

```
SELECT column_name, data_type
FROM `scaler-dsml-sql406007.Business_case_Target_SQL.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'geolocation
```

JOB INFORMATION | RESULTS | CHART | JSON | E

Row	column_name	data_type
1	geolocation_zip_code_prefix	INT64
2	geolocation_lat	FLOAT64
3	geolocation_lng	FLOAT64
4	geolocation_city	STRING
5	geolocation_state	STRING

Data type of all columns in the "products" table

```
SELECT column_name, data_type
FROM `scaler-dsml-sql406007.Business_case_Target_SQL.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'products'
```

JOB INFORMATION | RESULTS | CHART | JSON

Row	column_name	data_type
1	product_id	STRING
2	product category	STRING
3	product_name_length	INT64
4	product_description_length	INT64
5	product_photos_qty	INT64

Data type of all columns in the "orders" table

```
SELECT column_name, data_type
FROM `scaler-dsml-sql406007.Business_case_Target_SQL.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'orders'
```

JOB INFORMATION | RESULTS | CHART | JSON

Row	column_name	data_type
1	order_id	STRING
2	customer_id	STRING
3	order_status	STRING
4	order_purchase_timestamp	TIMESTAMP
5	order_approved_at	TIMESTAMP

Data type of all columns in the "payments" table

```
SELECT column_name, data_type
FROM `scaler-dsml-sql406007.Business_case_Target_SQL.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'payments'
```

JOB INFORMATION | RESULTS | CHART | JSON

Row	column_name	data_type
1	order_id	STRING
2	payment_sequential	INT64
3	payment_type	STRING
4	payment_installments	INT64
5	payment_value	FLOAT64

Data type of all columns in the "order_reviews" table

```
SELECT column_name, data_type
FROM `scaler-dsml-sql406007.Business_case_Target_SQL.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'order_reviews'
```

JOB INFORMATION | RESULTS | CHART | JSON

Row	column_name	data_type
1	review_id	STRING
2	order_id	STRING
3	review_score	INT64
4	review_comment_title	STRING
5	review_creation_date	TIMESTAMP

'

Data type of all columns in the "order_items" table

```
SELECT column_name, data_type
FROM `scaler-dsml-sql406007.Business_case_Target_SQL.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'order_items'
```

JOB INFORMATION | RESULTS | CHART | JSO

Row	column_name	data_type
1	order_id	STRING
2	order_item_id	INT64
3	product_id	STRING
4	seller_id	STRING
5	shipping_limit_date	TIMESTAMP

Insights:

The database Schema for different tables tells about the interrelation between different tables and primary key and foreign keys.

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

```
SELECT  CONCAT(MIN(order_purchase_timestamp),"-",MAX(order_purchase_timestamp))
time_range,
DATE_DIFF(MAX(order_purchase_timestamp), MIN(order_purchase_timestamp), DAY)
day_Difference
FROM `scaler-dsml-sql-406007.Business_case_Target_SQL.orders`
```

JOB INFORMATION | RESULTS | CHART | JSON | EXI

Row	time_range	day_Difference
1	2016-09-04 21:15:19+00-2018-...	772

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

```
SELECT COUNT(DISTINCT customer_city) number_of_cities,
COUNT(DISTINCT customer_state) number_of_states
FROM `scaler-dsml-sql-406007.Business_case_Target_SQL.customers`
```

Query results | SAVE RESULTS | EXPLORE DATA

JOB INFORMATION | RESULTS | CHART PREVIEW | JSON | EXECUTION DETAILS | EXECUTION GRAPH

Row	number_of_cities	number_of_states
1	4119	27

2. **In-depth Exploration:**

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

```
WITH CTE_2017 AS (
```

```sql
  SELECT CASE WHEN purchase_month = 'January'THEN 1 WHEN purchase_month =
'February'THEN 2 WHEN purchase_month = 'March' THEN 3 WHEN purchase_month =
'April' THEN 4 WHEN purchase_month = 'May' THEN 5 WHEN purchase_month = 'June'
THEN 6 WHEN purchase_month = 'July' THEN 7 WHEN purchase_month = 'August' THEN 8
WHEN purchase_month = 'September' THEN 9 WHEN purchase_month = 'October' THEN 10
WHEN purchase_month = 'November' THEN 11 WHEN purchase_month = 'December'THEN 12
END month_no,
purchase_month, orders_per_month
FROM (
  SELECT FORMAT_DATE("%B", order_purchase_timestamp) purchase_month, count(*)
orders_per_month,
  FROM `scaler-dsml-sql-406007.Business_case_Target_SQL.orders`
  WHERE FORMAT_DATE("%Y", order_purchase_timestamp) like '2017'
  GROUP BY purchase_month ) as t
ORDER BY month_no
 ),
CTE_2016 AS (
SELECT FORMAT_DATE("%B", order_purchase_timestamp) as purchase_month, COUNT(*)
orders_per_month
FROM `scaler-dsml-sql-406007.Business_case_Target_SQL.orders`
WHERE FORMAT_DATE("%Y",order_purchase_timestamp) = '2016'
GROUP BY purchase_month),
CTE_2018 AS (
SELECT FORMAT_DATE("%B", order_purchase_timestamp) as purchase_month, COUNT(*)
orders_per_month
FROM `scaler-dsml-sql-406007.Business_case_Target_SQL.orders`
WHERE FORMAT_DATE("%Y",order_purchase_timestamp) = '2018'
GROUP BY purchase_month),
CTE AS (
  SELECT x.month_no, x.purchase_month, y.orders_per_month orders_per_month_2016,
x.orders_per_month AS orders_per_month_2017, z.orders_per_month AS
orders_per_month_2018
FROM CTE_2017 x
LEFT JOIN CTE_2016 y
ON x.purchase_month = y.purchase_month
LEFT JOIN CTE_2018 AS z
ON x.purchase_month = z.purchase_month
ORDER BY x.month_no )
SELECT AVG(CTE.orders_per_month_2016) as avg_orders_per_month_2016,
AVG(CTE.orders_per_month_2017) as avg_orders_per_month_2017,
AVG(CTE.orders_per_month_2018) as avg_orders_per_month_2018
FROM CTE
```

Query results

SAVE RESULTS EXP

JOB INFORMATION | RESULTS | CHART PREVIEW | JSON | EXECUTION DETAILS | EXECUTION GRAPH

Row	avg_orders_per_month_2016	avg_orders_per_month_2017	avg_orders_per_month_2018
1	109.66666666666667	3758.4166666666665	5401.1

Insights:

The average number of orders per month in 2016 was 109, in 2017 it was 3758, and in 2018 it was 5401. Every year, the average number of orders per month increased.

Recommendations:

- Understand customer needs, preferences to know the target audience well.
- Use social media effectively, and invest in digital marketing to increase visibility and reach a broader audience.
- Build good relationships with customers through personalized communication, feedback channels, and loyalty programs.
- Provide exceptional customer service to create positive experiences, leading to customer satisfaction.

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

```
 WITH CTE_2017 AS (
  SELECT CASE WHEN purchase_month = 'January'THEN 1 WHEN purchase_month =
'February'THEN 2 WHEN purchase_month = 'March' THEN 3 WHEN purchase_month =
'April' THEN 4 WHEN purchase_month = 'May' THEN 5 WHEN purchase_month = 'June'
THEN 6 WHEN purchase_month = 'July' THEN 7 WHEN purchase_month = 'August' THEN 8
WHEN purchase_month = 'September' THEN 9 WHEN purchase_month = 'October' THEN 10
WHEN purchase_month = 'November' THEN 11 WHEN purchase_month = 'December'THEN 12
END month_no,
purchase_month, orders_per_month
FROM (
  SELECT FORMAT_DATE("%B", order_purchase_timestamp) purchase_month, count(*)
orders_per_month,
  FROM `scaler-dsml-sql-406007.Business_case_Target_SQL.orders`
  WHERE FORMAT_DATE("%Y", order_purchase_timestamp) like '2017'
  GROUP BY purchase_month ) as t
ORDER BY month_no
 ),
CTE_2016 AS (
SELECT FORMAT_DATE("%B", order_purchase_timestamp) as purchase_month, COUNT(*)
orders_per_month
FROM `scaler-dsml-sql-406007.Business_case_Target_SQL.orders`
WHERE FORMAT_DATE("%Y",order_purchase_timestamp) = '2016'
GROUP BY purchase_month),
CTE_2018 AS (
SELECT FORMAT_DATE("%B", order_purchase_timestamp) as purchase_month, COUNT(*)
orders_per_month
FROM `scaler-dsml-sql-406007.Business_case_Target_SQL.orders`
WHERE FORMAT_DATE("%Y",order_purchase_timestamp) = '2018'
GROUP BY purchase_month),
CTE AS (SELECT x.month_no, x.purchase_month, y.orders_per_month AS
orders_per_month_2016, x.orders_per_month as orders_per_month_2017,
z.orders_per_month AS orders_per_month_2018
FROM CTE_2017 x
LEFT JOIN CTE_2016 y
ON x.purchase_month = y.purchase_month
LEFT JOIN CTE_2018 AS z
ON x.purchase_month = z.purchase_month
ORDER BY x.month_no )
SELECT CTE.purchase_month, CTE.orders_per_month_2016, CTE.orders_per_month_2017,
CTE.orders_per_month_2018,
CASE WHEN CTE.orders_per_month_2016 > CTE.orders_per_month_2017 AND
CTE.orders_per_month_2016>CTE.orders_per_month_2018 THEN
CTE.orders_per_month_2016 WHEN
CTE.orders_per_month_2017>CTE.orders_per_month_2016 AND
```

```
CTE.orders_per_month_2017>CTE.orders_per_month_2018 THEN
CTE.orders_per_month_2017 WHEN CTE.orders_per_month_2016 IS NULL AND
CTE.orders_per_month_2018 IS NULL THEN CTE.orders_per_month_2017 ELSE
CTE.orders_per_month_2018 END AS max_orders_for_month
FROM CTE
ORDER BY max_orders_for_month DESC
```

Query results

SAVE RESULTS | EXPLORE DATA

JOB INFORMATION | RESULTS | CHART PREVIEW | JSON | EXECUTION DETAILS | EXECUTION GRAPH

Row	purchase_month	orders_per_month_2016	orders_per_month_2017	orders_per_month_2018	max_orders_for_month
1	November	null	7544	null	7544
2	January	null	800	7269	7269
3	March	null	2682	7211	7211
4	April	null	2404	6939	6939
5	May	null	3700	6873	6873
6	February	null	1780	6728	6728
7	August	null	4331	6512	6512
8	July	null	4026	6292	6292
9	June	null	3245	6167	6167
10	October	324	4631	4	4631
11	September	4	4285	16	4285

Results per page: 100 | 1 – 12 of 12

Insights:

By looking at maximum orders per month over the range in which the orders were placed. November has the highest orders per month, which is 7544. The number of orders per month for January, March, April, and May falls after November. These are seasonal months in which the order purchase rate was high.

Recommendations:

- Introduce package related products or services together at a discounted price, providing added value and enticing customers to buy more.
- Offer seasonal promotions, discounts, first purchase discounts, referral discounts.
- Introduce limited time offers, flash sales.

2.3. During what time of the day, do the Brazilian customers mostly place their orders?

```
WITH CTE AS (SELECT CASE WHEN time BETWEEN '00:00' AND '06:00' THEN 'dawn'
WHEN time BETWEEN '07:00' AND '12:00' THEN 'morning'
WHEN time BETWEEN '13:00' AND '18:00' THEN 'afternoon'
ELSE 'night' END time_of_day
FROM(
SELECT order_purchase_timestamp, FORMAT_DATETIME("%H:%M",
order_purchase_timestamp) as time
FROM `scaler-dsml-sql-406007.Business_case_Target_SQL.orders` ) temp )
SELECT  CTE.time_of_day, COUNT(*) as no_of_orders_placed
FROM CTE
GROUP BY CTE.time_of_day
```

```
ORDER BY no_of_orders_placed desc
```

Query results SAVE

JOB INFORMATION | RESULTS | CHART PREVIEW | JSON | EXECUTION DETAILS | EXECUTION GRAPH

Row	time_of_day	no_of_orders_placed
1	night	40402
2	afternoon	32446
3	morning	21848
4	dawn	4745

Insight:

In the night, a greater number of orders were placed as compared to other times.

Recommendations:

- Focus on displaying more ads during night time.
- Schedule online and campaigns to run more frequently during light time hours.
- Create promotions or discounts that are exclusively available during night time.

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

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

```
SELECT customer_state, year_month, current_month_orders,
LEAD(current_month_orders) over(PARTITION BY customer_state ORDER BY year_month
desc) previous_month_orders
FROM(
  SELECT c.customer_state, FORMAT_DATE("%Y-%m", o.order_purchase_timestamp)
year_month, COUNT(*) current_month_orders
FROM `scaler-dsml-sql-406007.Business_case_Target_SQL.orders` o
JOIN `scaler-dsml-sql-406007.Business_case_Target_SQL.customers` c
ON o.customer_id = c.customer_id
GROUP BY c.customer_state, year_month
)
ORDER BY customer_state, year_month desc
```

Query results

SAVE RESULTS

JOB INFORMATION | RESULTS | CHART PREVIEW | JSON | EXECUTION DETAILS | EXECUTION GRAPH

Row	customer_state	year_month	current_month_orders	previous_month_orders
1	AC	2018-08	3	4
2	AC	2018-07	4	3
3	AC	2018-06	3	2
4	AC	2018-05	2	4
5	AC	2018-04	4	2
6	AC	2018-03	2	3
7	AC	2018-02	3	6
8	AC	2018-01	6	5
9	AC	2017-12	5	5
10	AC	2017-11	5	6
11	AC	2017-10	6	5
12	AC	2017-09	5	4
13	AC	2017-08	4	5

Results per page: 50 1 – 50 of 565

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

```
SELECT customer_state, COUNT(*) no_of_customers
FROM `scaler-dsml-sql-406007.Business_case_Target_SQL.customers`
GROUP BY customer_state
ORDER BY no_of_customers DESC
```

Query results

JOB INFORMATION | RESULTS | CHART PREVIEW | JSON | EXECUTION DETAILS

Row	customer_state	no_of_customers
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
11	PE	1652
12	CE	1336
13	PA	975

22	TO	280
23	RO	253
24	AM	148
25	AC	81
26	AP	68
27	RR	46

Insight:

- SP, RJ, MG have relatively large number of customers as compared with other states.
- RR, AP and AC have relatively large number of customers as compared with other states.

Recommendations:

- To increase customer count for Target in the states of RR, AP, and AC in Brazil, implement targeted marketing campaigns, establish local partnerships, offer region-specific products, enhance the customer experience, and optimize the online presence with localized content and promotions. Understanding the local market is essential.
- To increase customer count in RJ, MG, and SP, Target should prioritize improving the store experience, tailoring product selection, enhancing online presence, implementing competitive pricing and promotions, and engaging with local communities. These strategies aim to create a positive shopping environment, cater to regional preferences, optimize the online shopping experience, attract customers with attractive pricing, and build customer loyalty through community engagement .

4. **Impact on Economy: Analyze the money movement by e-commerce by looking at order prices, freight and others.**
   4.1. 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

```
WITH CTE_2017 AS (
SELECT DISTINCT o.order_id, FORMAT_DATE("%Y-%m", o.order_purchase_timestamp)
yr_month, p.payment_value,
FROM `scaler-dsml-sql-406007.Business_case_Target_SQL.orders` o
JOIN `scaler-dsml-sql-406007.Business_case_Target_SQL.payments` p
ON o.order_id = p.order_id
WHERE o.order_purchase_timestamp BETWEEN "2017-01-01 00:00:00 UTC" AND "2017-08-
31 00:00:00 UTC"
ORDER BY yr_month),
CTE_2018 AS (
SELECT DISTINCT o.order_id, FORMAT_DATE("%Y-%m", o.order_purchase_timestamp)
yr_month, p.payment_value,
FROM `scaler-dsml-sql-406007.Business_case_Target_SQL.orders` o
JOIN `scaler-dsml-sql-406007.Business_case_Target_SQL.payments` p
ON o.order_id = p.order_id
WHERE o.order_purchase_timestamp BETWEEN "2018-01-01 00:00:00 UTC" AND "2018-08-
31 00:00:00 UTC"
```

```
ORDER BY yr_month)
SELECT x.cost_of_orders_2017, y.cost_of_orders_2018,
((y.cost_of_orders_2018- x.cost_of_orders_2017)/x.cost_of_orders_2017)*100 AS
percentage_increase_in_cost_of_orders
FROM ( SELECT ROW_NUMBER() OVER() num,  SUM(CTE_2017.payment_value)
cost_of_orders_2017
FROM CTE_2017 ) x
JOIN (SELECT ROW_NUMBER() OVER() num, SUM(CTE_2018.payment_value)
cost_of_orders_2018
FROM CTE_2018) y
ON x.num = y.num
```

JOB INFORMATION | RESULTS | CHART | JSON | EXECUTION DETAILS

Row	cost_of_orders_2017	cost_of_orders_2018	percentage_increase
1	3639846.860000...	8686950.559999...	138.6625287856...

Insights:

- The percentage increase in the cost of orders over the years 2017 and 2018 is 138%, which means that e-commerce is rapidly growing and becoming a necessity in Brazil.
- E-commerce growth can be attributed to several factors, including improved internet access, increased smartphone penetration, convenience, a wider range of products, and competitive pricing. As more people embrace online shopping and businesses expand their digital presence, the e-commerce industry in Brazil is likely to continue growing.

Recommendations:

- Improve the online shopping experience with a user-friendly interface and efficient checkout process.
- Enhance customer service through prompt responses and personalized assistance.
- Expand product assortment to cater to a wider range of customer preferences.
- Strengthen marketing efforts to increase brand visibility and attract more customers.
- Invest in logistics, data analytics, and customer retention strategies to support growth and maintain competitiveness.

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

```
SELECT customer_state state, SUM(price) total_price , AVG(price) average_price
FROM (
SELECT c.customer_id, o.order_id, c.customer_state, oi.price
FROM `scaler-dsml-sql-406007.Business_case_Target_SQL.customers` c
JOIN `scaler-dsml-sql-406007.Business_case_Target_SQL.orders` o
ON c.customer_id = o.customer_id
JOIN `scaler-dsml-sql-406007.Business_case_Target_SQL.order_items` oi
ON o.order_id = oi.order_id ) temp
GROUP BY customer_state
ORDER BY total_price DESC, average_price DESC
```

Query results

SAVE RESULTS

JOB INFORMATION | RESULTS | CHART PREVIEW | JSON | EXECUTION DETAILS | EXECUTION GRAPH

Row	state	total_price	average_price
1	SP	5202955.05000...	109.6536291597...
2	RJ	1824092.66999...	125.1178180945...
3	MG	1585308.02999...	120.7485741488...
4	RS	750304.020000...	120.3374530874...
5	PR	683083.760000...	119.0041393728...
6	SC	520553.340000...	124.6535775862...
7	BA	511349.990000...	134.6012082126...
8	DF	302603.939999...	125.7705486284...
9	GO	294591.949999...	126.2717316759...
10	ES	275037.309999...	121.9137012411...
11	PE	262788.029999...	145.5083222591...

Results per page: 50 1 – 27 of 27

Insights:

- The total price of orders in SP is higher than in other states, but the average value of orders is not the highest since SP has a larger number of customers.

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

```
SELECT customer_state state, SUM(freight_value) total_freight_value ,
AVG(freight_value) average_freight_value
FROM (
SELECT c.customer_id, o.order_id, c.customer_state, oi.freight_value
FROM `scaler-dsml-sql-406007.Business_case_Target_SQL.customers` c
JOIN `scaler-dsml-sql-406007.Business_case_Target_SQL.orders` o
ON c.customer_id = o.customer_id
JOIN `scaler-dsml-sql-406007.Business_case_Target_SQL.order_items` oi
ON o.order_id = oi.order_id ) temp
GROUP BY customer_state
ORDER BY total_freight_value DESC, average_freight_value DESC
```

JOB INFORMATION | RESULTS | CHART | JSON | EXECUTION DETAILS | EXECUTION GRAPH

Row	state	total_price_value	average_price_value	total_freight_value	average_freight_valu
1	SP	5202955.050001...	109.6536291597...	718723.0699999...	15.14727539041...
2	RJ	1824092.669999...	125.1178180945...	305589.3100000...	20.96092393168...
3	MG	1585308.029999...	120.7485741488...	270853.4600000...	20.63016680630...
4	RS	750304.0200000...	120.3374530874...	135522.7400000...	21.73580433039...
5	PR	683083.7600000...	119.0041393728...	117851.6800000...	20.53165156794...
6	BA	511349.9900000...	134.6012082126...	100156.6799999...	26.36395893656...
7	SC	520553.3400000...	124.6535775862...	89660.26000000...	21.47036877394...
8	PE	262788.0299999...	145.5083222591...	59449.65999999...	32.91786267995...
9	GO	294591.9499999...	126.2717316759...	53114.97999999...	22.76681525932...
10	DF	302603.9399999...	125.7705486284...	50625.49999999...	21.04135494596...
11	ES	275037.3099999...	121.9137012411...	49764.59999999...	22.05877659574...

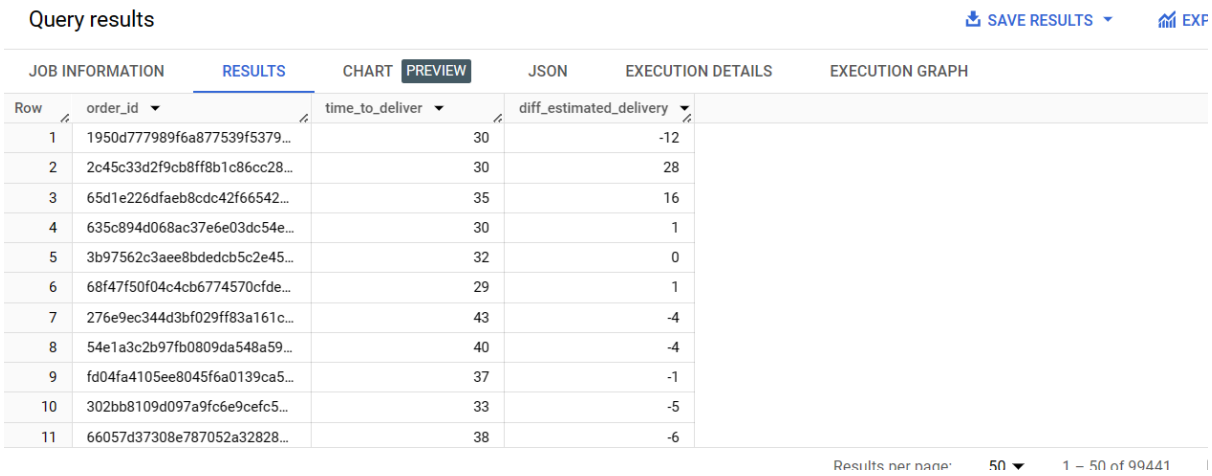
Insights and Recommendations:

- Average Order Value: The average price across different customer states varies significantly, ranging from 109.65 (SP) to 191.48 (PB). Target could focus on increasing the average order value by implementing strategies such as upselling, cross-selling, and offering bundled or complementary products.
- Freight Costs: The average freight value also varies across states, with AL having the highest average at 35.84 and SP having the lowest at 15.15. Target could optimize logistics and negotiate better shipping rates to reduce freight costs, which would make their products more attractive to customers and potentially increase sales.
- Market Potential: Consider the sum of prices and sum of freight values to gauge market potential. SP has the highest sum of prices (5,202,955.05) and sum of freight (718,723.07), indicating a large market with significant revenue potential. Target could focus on expanding its presence and marketing efforts in SP to tap into this lucrative market.
- Regional Opportunities: Identify states with higher average prices and lower competition, such as PB (191.48) and AL (180.89). Target could strategically target these regions with tailored marketing campaigns and promotions to attract customers who are willing to spend more, potentially increasing sales and profitability.
- Customer Retention: Analyse customer behaviour in each state to identify opportunities for improving customer retention. For instance, states like MG and RJ have significantly higher sums of prices and freight values, indicating potential loyal customer bases. Target could implement loyalty programs, personalized offers, and exceptional customer service to retain customers in these regions and encourage repeat purchases.

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

5.1. 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.

```
SELECT order_id,
DATE_DIFF(order_delivered_customer_date, order_purchase_timestamp, DAY)
time_to_deliver,
DATE_DIFF(order_estimated_delivery_date, order_delivered_customer_date, DAY)
diff_estimated_delivery
FROM `scaler-dsml-sql-406007.Business_case_Target_SQL.orders`
```


Query results

SAVE RESULTS EXP

JOB INFORMATION | RESULTS | CHART PREVIEW | JSON | EXECUTION DETAILS | EXECUTION GRAPH

Row	order_id	time_to_deliver	diff_estimated_delivery
1	1950d777989f6a877539f5379...	30	-12
2	2c45c33d2f9cb8ff8b1c86cc28...	30	28
3	65d1e226dfaeb8cdc42f66542...	35	16
4	635c894d068ac37e6e03dc54e...	30	1
5	3b97562c3aee8bdedcb5c2e45...	32	0
6	68f47f50f04c4cb6774570cfde...	29	1
7	276e9ec344d3bf029ff83a161c...	43	-4
8	54e1a3c2b97fb0809da548a59...	40	-4
9	fd04fa4105ee8045f6a0139ca5...	37	-1
10	302bb8109d097a9fc6e9cefc5...	33	-5
11	66057d37308e787052a32828...	38	-6

Results per page: 50 1 – 50 of 99441

Insights:

- some orders delivered to the customer within the estimated duration.
- But some orders were delayed, reaching after the given estimated delivery date.
- The negative sign in diff_estimated_delivery column indicates that the order were delayed.

Recommendations:

- Optimize operations: Streamline warehouse and inventory management systems, collaborate with reliable shipping partners, and establish multiple delivery fulfilment centres strategically to improve delivery efficiency.
- Enhance last-mile delivery: Focus on improving the final leg of the delivery process by utilizing local delivery services, optimizing routes, and implementing real-time tracking and communication with customers.
- Prioritize customer experience: Offer express delivery options for faster service, maintain clear and proactive communication with customers regarding order updates and potential delays, and utilize predictive analytics to forecast demand and optimize inventory levels.

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

```
WITH CTE AS (
SELECT customer_state state, ROUND(AVG(freight_value),4) average_freight_value
FROM (
SELECT c.customer_id, o.order_id, c.customer_state, oi.freight_value
FROM `scaler-dsml-sql-406007.Business_case_Target_SQL.customers` c
JOIN `scaler-dsml-sql-406007.Business_case_Target_SQL.orders` o
ON c.customer_id = o.customer_id
JOIN `scaler-dsml-sql-406007.Business_case_Target_SQL.order_items` oi
ON o.order_id = oi.order_id ) temp
GROUP BY customer_state ),
highest AS(
SELECT CTE.state, CTE.average_freight_value,
DENSE_RANK() OVER(ORDER BY CTE.average_freight_value DESC) den_rank
FROM CTE
ORDER BY den_rank),
lowest AS(
SELECT CTE.state, CTE.average_freight_value,
DENSE_RANK() OVER(ORDER BY CTE.average_freight_value) den_rank
FROM CTE
ORDER BY den_rank)
SELECT h.state highest_avg_freight_value_states, l.state
lowest_avg_freight_value_states, h.den_rank
FROM highest h
JOIN lowest l
ON h.den_rank = l.den_rank
ORDER BY h.den_rank
LIMIT 5
```

Query results

JOB INFORMATION | RESULTS | CHART PREVIEW | JSON | EXECUTION DETAILS | EXECUTION GR

Row	highest_avg_freight_value_states	lowest_avg_freight_value_states	den_rank
1	RR	SP	1
2	PB	PR	2
3	RO	MG	3
4	AC	RJ	4
5	PI	DF	5

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

```
WITH CTE AS(
SELECT c.customer_state,
AVG(DATE_DIFF(o.order_delivered_customer_date, o.order_purchase_timestamp,
DAY)) avg_delivery_time
FROM `scaler-dsml-sql-406007.Business_case_Target_SQL.orders` o
JOIN `scaler-dsml-sql-406007.Business_case_Target_SQL.customers` c
ON c.customer_id = o.customer_id
GROUP BY c.customer_state),
highest AS (
  SELECT CTE.customer_state, CTE.avg_delivery_time, DENSE_RANK() OVER(ORDER
BY CTE.avg_delivery_time DESC) den_rank
  FROM CTE ORDER BY den_rank LIMIT 5 ),
lowest AS (
  SELECT CTE.customer_state, CTE.avg_delivery_time, DENSE_RANK() OVER(ORDER
BY CTE.avg_delivery_time) den_rank
  FROM CTE ORDER BY den_rank LIMIT 5
)
SELECT h.customer_state highest_avg_delivery_time_states, l.customer_state
lowest_avg_delivery_time_states, h.den_rank
FROM highest h
JOIN lowest l
ON h.den_rank = l.den_rank
ORDER BY h.den_rank
```

Query results SAV

JOB INFORMATION | RESULTS | CHART PREVIEW | JSON | EXECUTION DETAILS | EXECUTION GRAPH

Row	highest_avg_delivery_time_states	lowest_avg_delivery_time_states	den_rank
1	RR	SP	1
2	AP	PR	2
3	AM	MG	3
4	AL	DF	4
5	PA	SC	5

5.4. Find out the top 5 states where the order delivery is really fast as compared to the estimated date of delivery.

```sql
WITH CTE AS (
SELECT order_id, customer_id,
DATE_DIFF(order_delivered_customer_date, order_purchase_timestamp, DAY) y,
DATE_DIFF(order_estimated_delivery_date, order_purchase_timestamp, DAY) x
FROM `scaler-dsml-sql-406007.Business_case_Target_SQL.orders`)
SELECT c.customer_state, AVG(o.x) - AVG(o.y) avg_diff_estimated_actual_delivered
FROM CTE o
JOIN `scaler-dsml-sql-406007.Business_case_Target_SQL.customers` c
ON o.customer_id = c.customer_id
GROUP BY c.customer_state
ORDER BY avg_diff_estimated_actual_delivered DESC LIMIT 5
```

Query results

JOB INFORMATION | RESULTS | CHART PREVIEW | JSON | EXECUTION DETAILS

Row	customer_state	avg_diff_estimated_actual_delivered
1	AC	20.12793209876542
2	RO	19.493534377592315
3	AP	18.974539069359089
4	AM	18.770549860205037
5	RR	17.198303287380689

Insights:

- The order delivery is the fastest in the AC state.
- RO, AP, AM, and RR are the next states, respectively, after the AC.

6. **Analysis based on the payments:**

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

```sql
SELECT payment_type, year_month, current_month_orders,
LAG(current_month_orders) OVER(PARTITION BY payment_type ORDER BY year_month )
previous_month_orders
FROM(
SELECT p.payment_type, FORMAT_DATE("%Y-%m", o.order_purchase_timestamp)
year_month, COUNT(*) current_month_orders
FROM `scaler-dsml-sql-406007.Business_case_Target_SQL.payments`  p
JOIN `scaler-dsml-sql-406007.Business_case_Target_SQL.orders` o
ON p.order_id = o.order_id
GROUP BY p.payment_type, year_month
ORDER BY p.payment_type, year_month ) temp_table
ORDER BY year_month
```

Query results

SAVE RESULTS

JOB INFORMATION | RESULTS | CHART PREVIEW | JSON | EXECUTION DETAILS | EXECUTION GRAPH

Row	payment_type	year_month	current_month_orders	previous_month_orders
1	credit_card	2016-09	3	null
2	credit_card	2016-10	254	3
3	debit_card	2016-10	2	null
4	voucher	2016-10	23	null
5	UPI	2016-10	63	null
6	credit_card	2016-12	1	254
7	credit_card	2017-01	583	1
8	debit_card	2017-01	9	2
9	voucher	2017-01	61	23
10	UPI	2017-01	197	63
11	credit_card	2017-02	1356	583
12	debit_card	2017-02	13	9

Results per page: 50 1 – 50 of 90

Insights:

- We can observe that year by year credit card payments are increasing followed by UPI payments as it is easy and can be used for faster transactions.
- Debit card payments are not much popular as there is a very low probability that customer uses a debit card.
- Vouchers are increasingly used by people more and more.

Recommendations:

- To retain credit card customers Target can provide offers on credit cards frequently so that the credit card payments can increase more and tie up with banks to issue Target Credit cards which can offer cash backs.
- Simplify the payment process and remove any unnecessary steps or friction points to make it easy and convenient for customers to use debit card and UPI payments.
- Promote the benefits of debit card and UPI payments, highlighting their convenience, security, and instant transfer capabilities, and offer incentives such as cashbacks, discounts, or rewards to encourage customers to choose these payment options.
- Optimize the mobile experience, collaborate with banks and UPI providers, leverage social media and email marketing, and provide excellent customer support to further enhance the adoption of debit card and UPI payments for Target e-commerce

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

```
SELECT  p.payment_installments, COUNT(o.order_id)
no_of_orders_based_on_installments
FROM `scaler-dsml-sql-406007.Business_case_Target_SQL.payments` p
JOIN `scaler-dsml-sql-406007.Business_case_Target_SQL.orders` o
ON p.order_id = o.order_id
WHERE p.payment_installments > 0 AND
(o.order_delivered_customer_date IS NOT NULL OR o.order_status =
'delivered'OR o.order_approved_at IS NOT NULL) AND
```

```
FORMAT_DATE("%Y-%m",o.order_purchase_timestamp) < (SELECT FORMAT_DATE("%Y-
%m",MAX(order_purchase_timestamp)) FROM `scaler-dsml-sql-
406007.Business_case_Target_SQL.orders`)
GROUP BY p.payment_installments
```

Query results

JOB INFORMATION | RESULTS | CHART PREVIEW | JSON | EXECUTIO

Row	payment_installments	no_of_orders_based_on_installments
1	1	52542
2	7	1626
3	10	5328
4	6	3920
5	2	12413
6	4	7098
7	3	10461
8	8	4268
9	9	644
10	5	5239
11	12	133
12	20	17
13	15	74

Insights :

- The majority share is taken by one-time payments followed by 2 installments and 3 installments payments.
- Instalments more than 4 are considerably less as people don’t want long-term EMI.