

# Target Case Study

## Problem Statement:

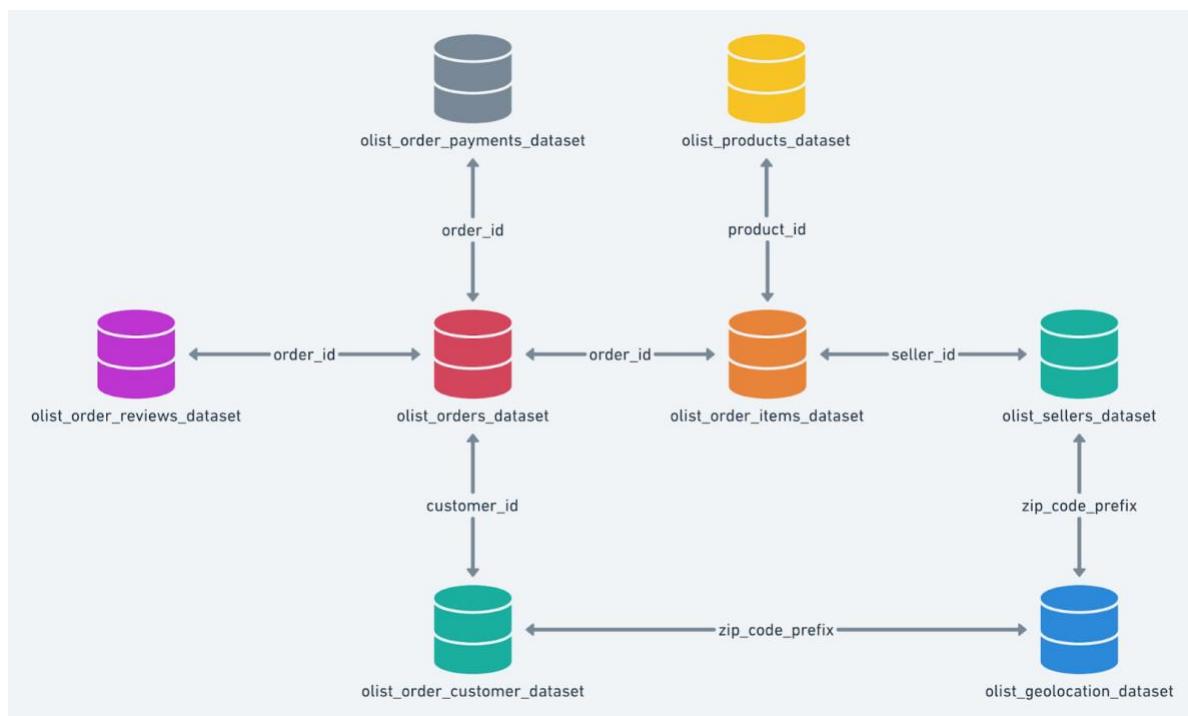
Assuming you are a data analyst/ scientist at Target, you have been assigned the task of analyzing the given dataset to extract valuable insights and provide actionable recommendations.

## Context:

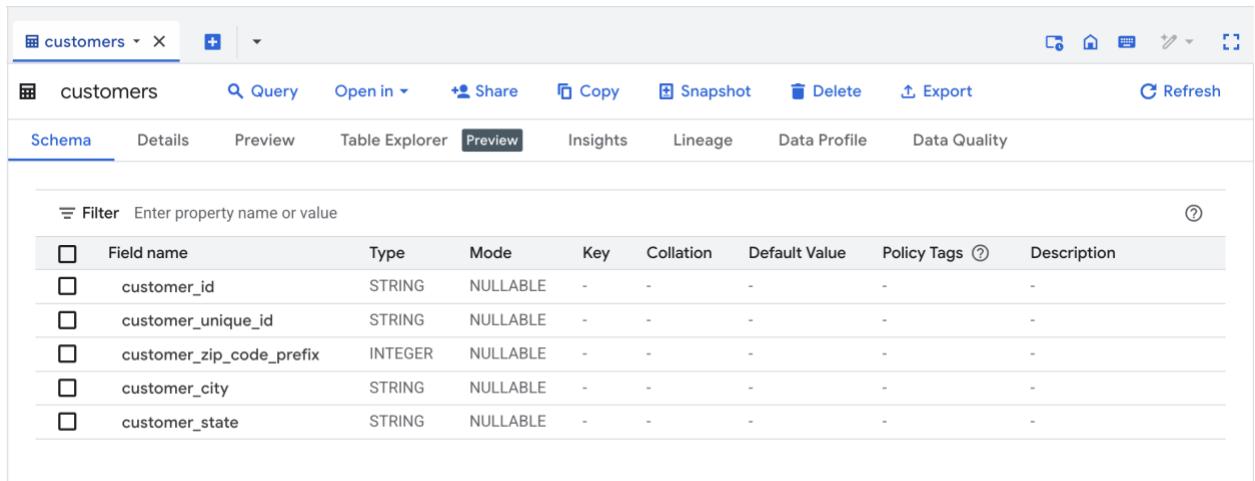
Target is a globally renowned brand and a prominent retailer in the United States. Target makes itself a preferred shopping destination by offering outstanding value, inspiration, innovation and an exceptional guest experience that no other retailer can deliver.

This particular 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.



1. Import the dataset and do usual exploratory analysis steps like checking the structure & characteristics of the dataset:
  - 1.a. Data type of all columns in the "customers" table.



The screenshot shows the BigQuery UI with the 'customers' table selected. The 'Schema' tab is active, displaying the following schema:

Field name	Type	Mode	Key	Collation	Default Value	Policy Tags	Description
customer_id	STRING	NULLABLE	-	-	-	-	-
customer_unique_id	STRING	NULLABLE	-	-	-	-	-
customer_zip_code_prefix	INTEGER	NULLABLE	-	-	-	-	-
customer_city	STRING	NULLABLE	-	-	-	-	-
customer_state	STRING	NULLABLE	-	-	-	-	-

The “customers” table consists of 5 fields. Their data types are:

customer\_id -> STRING

customer\_unique\_id -> STRING

customer\_zip\_code\_prefix -> INTEGER

customer\_city -> STRING

customer\_state -> STRING

- 1.b. Get the time range between which the orders were placed.

```
select min(order_purchase_timestamp) as first_order,  
       max(order_purchase_timestamp) as latest_order,  
     from target.orders;
```

The screenshot shows the Google Cloud BigQuery interface. In the top navigation bar, it says "Google Cloud" and "Scaler-DSML-APR25-SQL". A search bar is present with the placeholder "Search (/) for resources, docs, products, and more". On the far right, there are icons for Dismiss and Upgrade.

The main area has a sidebar titled "Explorer" with a "Show starred only" checkbox checked. Below it is a tree view of datasets and tables under "scaler-dsml-apr25-sql": target (customers, geolocation, order\_items, order\_reviews, orders, payments, products, sellers). A "Search BigQuery resources" input field is also in the sidebar.

The central workspace shows an "Untitled query" tab with the following SQL code:

```
1 select min(order_purchase_timestamp) as first_order,
2      max(order_purchase_timestamp) as latest_order
3   from target.orders;
```

A green checkmark indicates "Query completed". Below the query editor is a "Query results" table with one row of data:

Row	first_order	latest_order
1	2016-09-04 21:15:19 UTC	2018-10-17 17:30:18 UTC

At the bottom, there are buttons for "Save results" and "Open in", and a footer showing "Results per page: 50" and "1 - 1 of 1".

### Alternative Way:

```
select min(order_purchase_timestamp) as first_order,
       max(order_purchase_timestamp) as latest_order,
       concat(
           floor(date_diff(max(order_purchase_timestamp),
               min(order_purchase_timestamp), day) / 365), ' Yrs, ',
           floor((mod(date_diff(max(order_purchase_timestamp),
               min(order_purchase_timestamp), day), 365)) / 30), ' Mnth '
       ) as time_range_btwn_ordrs
  from target.orders;
```

This screenshot shows the same Google Cloud BigQuery interface as the previous one, but with a different query. The sidebar and dataset structure are identical.

The central workspace shows an "Untitled query" tab with the alternative SQL code:

```
1 select min(order_purchase_timestamp) as first_order,
2      max(order_purchase_timestamp) as latest_order,
3      concat(
4          floor(date_diff(max(order_purchase_timestamp), min(order_purchase_timestamp), day)/365), ' Yrs, ',
5          floor((mod(date_diff(max(order_purchase_timestamp), min(order_purchase_timestamp), day), 365))/30), ' Mnth '
6      ) as time_range_btwn_ordrs
7   from target.orders;
```

A green checkmark indicates "Query completed". Below the query editor is a "Query results" table with one row of data:

Row	first_order	latest_order	time_range_btwn_ordrs
1	2016-09-04 21:15:19 UTC	2018-10-17 17:30:18 UTC	2 Yrs, 1 Mnth

At the bottom, there are buttons for "Save results" and "Open in", and a footer showing "Results per page: 50" and "1 - 1 of 1".

The first order was placed on 2016-09-04 21:15:19 UTC, and the latest order was placed on 2018-10-17 17:30:18 UTC. The difference, 2 years 1 month, represents the time range during which the orders were placed. **Note:** The reason for considering the first and latest order timestamps is that each order, regardless of its status, was initially in a "placed" status before it was canceled or approved or delivered etc.

### 1.c. Count the Cities & States of customers who ordered during the given period

```
with order_range as (
    select min(order_purchase_timestamp) as first_order,
           max(order_purchase_timestamp) as latest_order
    from target.orders
)
select count(distinct c.customer_city) as cust_city_count,
       count(distinct c.customer_state) as cust_state_count
from target.orders o
join target.customers c on o.customer_id = c.customer_id
where o.order_purchase_timestamp between (select first_order from order_range)
    and (select latest_order from order_range);
```

The screenshot shows the Google Cloud BigQuery interface. On the left is the Explorer sidebar with a tree view of datasets and tables. The main area is titled 'Untitled query' and contains the SQL code from the previous block. Below the code, a green checkmark indicates 'Query completed'. Under 'Results', there is a single row of data:

Row	cust_city_count	cust_state_count
1	4119	27

At the bottom, the results are paginated with '50' items per page, showing '1 - 1 of 1'.

#### Alternative Way:

```
select count(distinct c.customer_city) as cust_city_count,
       count(distinct c.customer_state) as cust_state_count
from target.orders o join target.customers c on o.customer_id = c.customer_id
where o.order_purchase_timestamp
between (select min(order_purchase_timestamp) as first_order from target.orders)
and
(select max(order_purchase_timestamp) as latest_order from target.orders);
```

The screenshot shows the Google Cloud BigQuery interface. On the left is the Explorer sidebar with a tree view of datasets and tables. The main area is titled "Untitled query" and contains the following SQL code:

```
1 select count(distinct c.customer_city) as cust_city_count,
2      ||| | count(distinct c.customer_state) as cust_state_count
3  from target.orders o join target.customers c on o.customer_id = c.customer_id
4  where o.order_purchase_timestamp
5    between (select min(order_purchase_timestamp) as first_order from target.orders)
6    and
7    (select max(order_purchase_timestamp) as latest_order from target.orders);
```

The status bar indicates "Query completed". Below the code, the "Query results" section shows a single row of data:

Row	cust_city_count	cust_state_count
1	4119	27

At the bottom, the results are summarized as "1 - 1 of 1".

Customers from 4119 cities and 27 states ordered during the period between the first order and the latest order.

## 2. In-depth Exploration:

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

```
select extract(year from order_purchase_timestamp) as order_year,
       count(order_id) as yrly_order_count
  from target.orders
 group by extract(year from order_purchase_timestamp)
 order by order_year;
```

The screenshot shows the Google Cloud BigQuery interface. The Explorer sidebar is visible on the left. The main area is titled "Untitled query" and contains the following SQL code:

```
1 select extract(year from order_purchase_timestamp) as order_year,
2      ||| | count(order_id) as yrly_order_count
3  from target.orders
4 group by extract(year from order_purchase_timestamp)
5 order by order_year;
```

The status bar indicates "Query completed". Below the code, the "Query results" section shows three rows of data:

Row	order_year	yrly_order_count
1	2016	329
2	2017	45101
3	2018	54011

At the bottom, the results are summarized as "1 - 3 of 3".

The result indicates that there is a growing trend in the number of orders placed over the past years as the yearly order count increased from 329 in 2016 to 45101 in 2017 and further increased to 54011 in 2018.

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

```
select extract(year from order_purchase_timestamp) as order_yr,
       extract(month from order_purchase_timestamp) as order_mnth,
       count(*) as number_of_orders
  from target.orders
 group by extract(year from order_purchase_timestamp), extract(month from
order_purchase_timestamp)
 order by order_yr, order_mnth;
```

The screenshot shows the Google Cloud BigQuery interface. On the left is the Explorer sidebar with a 'Sandbox' message. The main area shows an 'Untitled query' with the SQL code provided above. A green checkmark indicates 'Query completed'. Below it is the 'Query results' table:

Row	order_yr	order_mnth	number_of_orders
1	2016	9	4
2	2016	10	324
3	2016	12	1
4	2017	1	800
5	2017	2	1780

At the bottom, it says 'Results per page: 50 1 – 25 of 25'.

This screenshot is identical to the one above, showing the same query execution and results table. The table data is as follows:

Row	order_yr	order_mnth	number_of_orders
1	2016	9	4
2	2016	10	324
3	2016	12	1
4	2017	1	800
5	2017	2	1780
6	2017	3	2682
7	2017	4	2404
8	2017	5	3700
9	2017	6	3245
10	2017	7	4026

At the bottom, it says 'Results per page: 50 1 – 25 of 25'.

The number of orders in February – March showed a spike due to the onset of the Carnival, the biggest festival in Brazil. Towards the end of the year, the total number of orders also increased due to Christmas and the beginning of a new year. This indicates a trend of monthly seasonality among the customers to purchase products for the onset of the festivals and other relevant seasons.

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

```
select
    case when extract(hour from order_purchase_timestamp) between 0 and 6
then "Dawn"
        when extract(hour from order_purchase_timestamp) between 7 and 12
then "Mornings"
        when extract(hour from order_purchase_timestamp) between 13 and 18
then "Afternoon"
        when extract(hour from order_purchase_timestamp) between 19 and 23
then "Night"
    end as order_time_of_day,
    count(*) as order_count
from target.orders
group by order_time_of_day
order by order_count;
```

The screenshot shows the Google Cloud BigQuery interface. At the top, there's a navigation bar with 'Google Cloud' and a project name 'Scaler-DSML-APR25-SQL'. Below it is a search bar and a 'Run' button. The main area has an 'Explorer' sidebar on the left listing datasets like 'target' and its tables 'customers', 'geolocation', etc. The central pane displays the query results. The query itself is as follows:

```
1 select
2     case when extract(hour from order_purchase_timestamp) between 0 and 6
3 then "Dawn"
4         when extract(hour from order_purchase_timestamp) between 7 and 12
5 then "Mornings"
6         when extract(hour from order_purchase_timestamp) between 13 and 18
7 then "Afternoon"
8         when extract(hour from order_purchase_timestamp) between 19 and 23
9 then "Night"
10    end as order_time_of_day,
11    count(*) as order_count
12 from target.orders
13 group by order_time_of_day
14 order by order_count;
```

A green checkmark indicates 'Query completed'. Below the query text, the 'Results' tab is selected in the 'Query results' section. The results table shows the following data:

order_time_of_day	order_count
Dawn	5242
Mornings	27733
Night	28331
Afternoon	38135

The Brazilian customers mostly place their orders during “Afternoon” time with 38135 orders placed between 13-18 hours. Additionally, the customers have also placed more than 20,000 orders during the

morning as well as the night hours which suggests that majority of the orders in total are placed post dawn.

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

#### 3.a. Get the month-on-month no. of orders placed in each state.

```
select extract(year from o.order_purchase_timestamp) as order_year,
       extract(month from o.order_purchase_timestamp) as order_month,
       c.customer_state as customer_state,
       count(*) as mnth_on_mnth_ordr_count
  from target.orders o join target.customers c
    on o.customer_id = c.customer_id
 group by order_year, order_month, customer_state
 order by order_year, order_month, customer_state;
```

The screenshot shows the Google Cloud BigQuery interface. The left sidebar displays the project structure under 'scaler-dsml-apr25-sql' with a 'target' dataset containing tables like 'customers', 'geolocation', 'order\_items', etc. The main area shows an 'Untitled query' with the previously provided SQL code. A green checkmark indicates 'Query completed'. Below it, the 'Query results' section shows a table with the following data:

Row	order_year	order_month	customer_state	mnth_on_mnth_ordr
1	2016	9	RR	1
2	2016	9	RS	1
3	2016	9	SP	2
4	2016	10	AL	2
5	2016	10	BA	4
6	2016	10	CE	8

At the bottom, it says 'Results per page: 50 1 – 50 of 565'.

This screenshot shows the same Google Cloud BigQuery interface as the previous one, but with a different set of results. The 'Query results' table now contains the following data:

Row	order_year	order_month	customer_state	mnth_on_mnth_ordr
1	2016	9	RR	1
2	2016	9	RS	1
3	2016	9	SP	2
4	2016	10	AL	2
5	2016	10	BA	4
6	2016	10	CE	8
7	2016	10	DF	6
8	2016	10	ES	4
9	2016	10	GO	9
10	2016	10	MA	4
11	2016	10	MG	40

At the bottom, it says 'Results per page: 50 1 – 50 of 565'.

São Paulo (SP) is the dominating state followed by Rio de Janeiro (RJ) and Minas Gerais (MG) in the state-wise distribution of the orders placed which is the expected e-commerce market in Brazil. The dominance of e-commerce in São Paulo can be leveraged in emerging and fast-growing states such as Minas Gerais. Regional growth in under-performing states such as Rio Grande do Sul (RS), Bahia (BA) and others is required through enhanced marketing, fast-delivery and product availability.

### 3.b. How are the customers distributed across all the states?

```
select customer_state,
       count(distinct customer_unique_id) as no_of_customers,
       round(count(distinct customer_unique_id) * 100.0 / sum(count(distinct
customer_unique_id)) over(), 2) as prcntge_of_total_cust
  from target.customers
 group by customer_state
 order by no_of_customers desc;
```

The screenshot shows the Google Cloud BigQuery interface. On the left is the Explorer sidebar with a tree view of datasets and tables. The main area is titled 'Untitled query' and contains the SQL code. Below the code is the 'Query results' section, which displays a table of data. The table has columns: Row, customer\_state, no\_of\_customers, and prcntge\_of\_total\_cust. The data is as follows:

Row	customer_state	no_of_customers	prcntge_of_total_cust
1	SP	40302	41.92
2	RJ	12384	12.88
3	MG	11259	11.71
4	RS	5277	5.49
5	PR	4882	5.08
6	SC	3534	3.68
7	BA	3277	3.41
8	DF	2075	2.16
9	ES	1964	2.04
10	GO	1952	2.03

The screenshot shows the Google Cloud BigQuery interface. On the left, the Explorer sidebar displays a project named 'scaler-dsml-apr25-sql' with a 'target' dataset containing tables like customers, geolocation, order\_items, etc. In the center, an 'Untitled query' window is open with the following SQL code:

```

1 select customer_state,
2       count(distinct customer_unique_id) as no_of_customers,
3       round(count(distinct customer_unique_id) * 100.0 / sum(count(distinct customer_unique_id)) over(), 2) as prcntge_of_total_cust

```

The status bar indicates 'Query completed'. Below the code, the 'Results' tab is selected, showing a table with 27 rows of data:

customer_state	no_of_customers	prcntge_of_total_cust
PI	482	0.5
RN	474	0.49
AL	401	0.42
SE	342	0.36
TO	273	0.28
RO	240	0.25
AM	143	0.15
AC	77	0.08
AP	67	0.07
RR	45	0.05

At the bottom, it says 'Results per page: 50 | 1 - 27 of 27'.

Majority of the customers are based in São Paulo and a significant proportion of them are based in emerging states of Rio de Janeiro and Minas Gerais as indicated by the percentage of total customers. Some of the states that require e-commerce growth and development are Acre (AC), Amapá (AP) and Roraima (RR).

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

4.a. 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 yearly_costs as (
    select extract(year from o.order_purchase_timestamp) as ordr_yr,
           sum(p.payment_value) as total_yrly_payment
    from target.payments p join target.orders o
    on p.order_id = o.order_id
    where extract(year from o.order_purchase_timestamp) in (2017, 2018)
        and
        extract(month from o.order_purchase_timestamp) between 1 and 8
    group by ordr_yr
    order by ordr_yr
)
select round(((select total_yrly_payment from yearly_costs where ordr_yr = 2018) -
              (select total_yrly_payment from yearly_costs where ordr_yr = 2017)) *
100.0 /
              (select total_yrly_payment from yearly_costs where ordr_yr = 2017),4)
as prcnt_inc_cost_of_ordrs;

```

The screenshot shows the Google Cloud BigQuery interface. On the left is the Explorer sidebar with a tree view of datasets and tables. The main area is titled "Untitled query" and contains the following SQL code:

```

1 with yearly_costs as (
2     select extract(year from o.order_purchase_timestamp) as ordr_yr,
3         round(sum(p.payment_value),4) as total_yrly_payment
4     from target.payments p join target.orders o
5     on p.order_id = o.order_id
6     where extract(year from o.order_purchase_timestamp) in (2017, 2018)
7         and
8             extract(month from o.order_purchase_timestamp) between 1 and 8
9     group by ordr_yr
10    order by ordr_yr
11 )

```

The status bar at the bottom indicates "Query completed". Below the code is a "Query results" section with a table:

Row	ordr_yr	total_yrly_payment
1	2017	3669022.12
2	2018	8695733.84

At the bottom right, there are buttons for "Save results" and "Open in".

This screenshot shows the same BigQuery interface after adding a new part to the query. The additional code calculates the percentage increase from 2017 to 2018.

```

1 with yearly_costs as (
2     select extract(year from o.order_purchase_timestamp) as ordr_yr,
3         round(sum(p.payment_value),4) as total_yrly_payment
4     from target.payments p join target.orders o
5     on p.order_id = o.order_id
6     where extract(year from o.order_purchase_timestamp) in (2017, 2018)
7         and
8             extract(month from o.order_purchase_timestamp) between 1 and 8
9     group by ordr_yr
10    order by ordr_yr
11 )
12 select round(((select total_yrly_payment from yearly_costs where ordr_yr = 2018) -
13                 (select total_yrly_payment from yearly_costs where ordr_yr = 2017)) * 100.0 /
14                         (select total_yrly_payment from yearly_costs where ordr_yr = 2017),4) as prcnt_inc_cost_of_ordrs;

```

The status bar indicates "Query completed". The "Query results" section shows a single row:

Row	prcnt_inc_cost_of_ordrs
1	136.9769

The total cost of orders was \$3669022.12 and \$8695733.84 in the year 2017 and 2018 respectively (include months between Jan to Aug only). The percentage increase between the defined period was 136.9769% which indicates a significant growth in the e-commerce industry in Brazil.

#### 4.b. Calculate the Total & Average value of order price for each state.

```

select c.customer_state,
       round(sum(oi.price),4) as total_price,
       round(avg(oi.price),4) as average_price
  from target.order_items oi
 join target.orders o on oi.order_id = o.order_id
 join target.customers c on o.customer_id = c.customer_id
 group by c.customer_state
 order by total_price desc, average_price desc, c.customer_state;

```

The screenshot shows the Google Cloud BigQuery interface. On the left, the Explorer sidebar displays a project named "Scaler-DSML-APR25-SQL" with a "target" dataset containing tables like customers, geolocation, order\_items, etc. In the main area, an "Untitled query" is running, displaying the following SQL code:

```

1 select c.customer_state,
2     round(sum(oi.price),4) as total_price,
3     round(avg(oi.price),4) as average_price
4 from target.order_items oi
5 join target.orders o on oi.order_id = o.order_id
6 join target.customers c on o.customer_id = c.customer_id
7 group by c.customer_state
8 order by total_price desc, average_price desc, c.customer_state;

```

The "Query completed" message is visible. Below the code, the "Results" tab is selected, showing a table with the following data:

customer_state	total_price	average_price
SP	5202955.05	109.6536
RJ	1824092.67	125.1178
MG	1585308.03	120.7486
RS	750304.02	120.3375
PR	683083.76	119.0041
SC	520553.34	124.6536

At the bottom, it says "Results per page: 50 1 - 27 of 27".

This screenshot is identical to the one above, showing the same BigQuery interface, query code, and results table. The data in the table is the same, listing the top 5 states by total price.

The total value and average value of order price for each state was calculated as shown in the image above. São Paulo (SP), Rio de Janeiro (RJ), Minas Gerais (MG), Rio Grande do Sul (RS) and Paraná (PR) are the top 5 states (Ordered by total\_price in descending order followed by average\_price in descending order and then by customer\_state alphabetically).

#### 4.c. Calculate the Total & Average value of order freight for each state.

```

select c.customer_state,
       round(sum(oi.freight_value),4) as total_order_freight,
       round(avg(oi.freight_value),4) as average_order_freight
from target.order_items oi
join target.orders o on oi.order_id = o.order_id
join target.customers c on o.customer_id = c.customer_id
group by c.customer_state
order by total_order_freight desc, average_order_freight desc, c.customer_state;

```

The screenshot shows the Google Cloud BigQuery interface. On the left, the Explorer sidebar displays a project named "Scaler-DSML-APR25-SQL" with a "target" dataset containing tables like customers, geolocation, order\_items, etc. In the center, an "Untitled query" tab is open with the following SQL code:

```

1 select c.customer_state,
2      round(sum(oi.freight_value),4) as total_order_freight,
3      round(avg(oi.freight_value),4) as average_order_freight
4  from target.order_items oi
5  join target.orders o on o.order_id = o.order_id
6  join target.customers c on o.customer_id = c.customer_id
7  group by c.customer_state
8  order by total_order_freight desc, average_order_freight desc, c.customer_state;

```

The "Query results" section shows the following data:

customer_state	total_order_freight	average_order_freight
SP	718723.07	15.1473
RJ	305589.31	20.9609
MG	270853.46	20.6302
RS	135522.74	21.7358
PR	117851.68	20.5317
BA	100156.68	26.364

At the bottom, the results are paginated from 1 to 27.

This screenshot is identical to the one above, showing the same BigQuery interface and query results. However, the results table contains 10 rows instead of 6, indicating a different execution or a different set of data being processed.

customer_state	total_order_freight	average_order_freight
SP	718723.07	15.1473
RJ	305589.31	20.9609
MG	270853.46	20.6302
RS	135522.74	21.7358
PR	117851.68	20.5317
BA	100156.68	26.364
SC	89660.26	21.4704
PE	59449.66	32.9179
GO	53114.98	22.7668
DF	50625.5	21.0414

The total value and average value of order freight for each state was calculated as shown in the image above. São Paulo (SP), Rio de Janeiro (RJ), Minas Gerais (MG), Rio Grande do Sul (RS) and Paraná (PR) are the top 5 states (Ordered by total\_order\_freight in descending order followed by average\_order\_freight in descending order and then by customer\_state alphabetically).

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

- i.  $\text{time\_to\_deliver} = \text{order\_delivered\_customer\_date} - \text{order\_purchase\_timestamp}$
- ii.  $\text{diff\_estimated\_delivery} = \text{order\_delivered\_customer\_date} - \text{order\_estimated\_delivery\_date}$

```
select order_id,
       timestamp_diff(order_delivered_customer_date, order_purchase_timestamp, day)
  as time_to_deliver,
       timestamp_diff(order_delivered_customer_date, order_estimated_delivery_date,
day) as diff_estimated_delivery
 from target.orders
where order_delivered_customer_date is not null
order by order_id;
```

The screenshot shows the Google Cloud BigQuery interface. On the left, the Explorer sidebar displays a project named 'scaler-dsml-apr25-sql' with a 'target' dataset containing tables like 'customers', 'geolocation', 'order\_items', etc. In the center, an 'Untitled query' window is open with the SQL code provided above. Below the code, a green circular icon indicates 'Query completed'. The 'Results' tab is selected, showing a table with 7 rows of data. The columns are 'Row', 'order\_id', 'time\_to\_deliver', and 'diff\_estimated\_delivery'. The data is as follows:

Row	order_id	time_to_deliver	diff_estimated_delivery
1	00010242fe8c5a6d1ba2dd792cb16214	7	-8
2	00018f77f2f0320c557190d7a14bbdd3	16	-2
3	000229ec398224ef6ca0657da4fc703e	7	-13
4	00024acbcdf0ada1e931b038114c75	6	-5
5	00042b26cf59d7ce69dfab4e55b4fd9	25	-15
6	00048cc3ae777c65dbb7d2a0634bc1ea	6	-14
7	00054e8431b9d7675808bc819fb4a32	8	-16

Google Cloud Scaler-DSML-APR25-SQL Search (/) for resources, docs, products, and more Search

Sandbox Set up billing to upgrade to the full BigQuery experience. Learn more Dismiss Upgrade

Explorer + Add data Untitled query \* Untitled...ery Run Save Download Share Schedule Open in More

Untitled query

```
1 select order_id,
2       timestamp_diff(order_delivered_customer_date,order_purchase_timestamp, day) as time_to_deliver,
3       timestamp_diff(order_delivered_customer_date, order_estimated_delivery_date, day) as diff_estimated_delivery
4
5 Query completed
```

Query results

Row	order_id	time_to_deliver	diff_estimated_delivery
1	00010242fe8c5a6d1ba2dd792cb16214	7	-8
2	00018f77f2f0320c557190d7a144bdd3	16	-2
3	000229ec398224ef6ca0657da4fc703e	7	-13
4	00024acbcdf0a6daa1e931b038114c75	6	-5
5	00042b26cf59dc669afabb4e55b4fd9	25	-15
6	00048cc3ae777c65db7d2a0634bc1ea	6	-14
7	00054e84319d7675808bcc819fb4a32	8	-16
8	000576f639319847ccb9d28c5617fa6	5	-15
9	0005a1a1728c9d785b8e2b08b904576c	9	0
10	0005f50442cb953cd1d121e1fb923495	2	-18

Results per page: 50 1 - 50 of 96476 < > >>

```
with time_delivery as (
    select timestamp_diff(order_delivered_customer_date,order_purchase_timestamp,
day) as time_to_deliver,
    timestamp_diff(order_delivered_customer_date,
order_estimated_delivery_date, day) as diff_estimated_delivery
    from target.orders
    where order_delivered_customer_date is not null
)
select count(*) as time_to_deliver_lessthan_0 from time_delivery where
time_to_deliver < 0;
```

Google Cloud Scaler-DSML-APR25-SQL Search (/) for resources, docs, products, and more Search

Sandbox Set up billing to upgrade to the full BigQuery experience. Learn more Dismiss Upgrade

Explorer + Add data Untitled query \* Untitled...ery Run Save Download Share Schedule Open in More

Untitled query

```
1 with time_delivery as (
2     select timestamp_diff(order_delivered_customer_date,order_purchase_timestamp, day) as time_to_deliver,
3           timestamp_diff(order_delivered_customer_date, order_estimated_delivery_date, day) as diff_estimated_delivery
4     from target.orders
5     where order_delivered_customer_date is not null
6 )
7 select count(*) as time_to_deliver_lessthan_0 from time_delivery where time_to_deliver < 0;
```

Query completed

Query results

Row	time_to_deliver_lessthan_0
1	0

Results per page: 50 1 - 1 of 1 < < > >>

Time to deliver is the total time taken from when the customer placed the order to when the customer received the product, showing how long customers wait for their orders. This difference has to be zero or positive since the date when the customer receives the order is always greater than or equal to the purchase date. If this value is negative, it means that the delivery occurred before purchase, indicating a data error. There are no such values where time to deliver is less than 0 which indicates that there is no error.

```
with diff_delivery as (
  select order_id,
    timestamp_diff(order_delivered_customer_date, order_purchase_timestamp, day) as time_to_deliver,
    timestamp_diff(order_delivered_customer_date, order_estimated_delivery_date, day) as diff_estimated_delivery
  from target.orders
  where order_delivered_customer_date is not null
)
select count(*) as late_orders from diff_delivery where diff_estimated_delivery > 0;
```

The screenshot shows the Google Cloud BigQuery interface. The top navigation bar includes 'Google Cloud' and 'Scaler-DSML-APR25-SQL'. A search bar and a 'Search' button are on the right. Below the navigation is a 'Sandbox' message: 'Set up billing to upgrade to the full BigQuery experience. [Learn more](#)' with 'Dismiss' and 'Upgrade' buttons. The main area has two tabs: 'Explorer' (selected) and 'Query'. The 'Query' tab shows an 'Untitled query' with the SQL code provided above. The 'Run' button is highlighted. Below the code is a message: 'Query completed'. The 'Results' tab is selected under 'Query results', showing a single row: 'Row 1 late\_orders 6535'. Other tabs include 'Job information', 'Chart', 'JSON', 'Execution details', and 'Execution graph'. At the bottom, there are buttons for 'Save results' and 'Open in', and a footer with 'Results per page: 50 ▾ 1 - 1 of 1 ▶ ▷ ▶ ▶'.

On the other hand, `diff_estimated_delivery` measures the difference between the actual delivery date and the estimated delivery date. If this difference is negative, it means that the order was delivered early, zero means delivered on time and positive indicates that the order was delivered late. Out of total orders (96,476 orders), 6,535 orders have a positive `diff_estimated_delivery` value. Target should ensure that year-on-year, this value remains zero or below zero.

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

```
select c.customer_state,
       round(avg(oi.freight_value),4) as avg_freight_value
  from target.order_items oi join target.orders o on oi.order_id = o.order_id
 join target.customers c on o.customer_id = c.customer_id
 where o.order_delivered_customer_date is not null
 group by c.customer_state
 order by avg_freight_value desc
 limit 5;
```

The screenshot shows the Google Cloud BigQuery interface. In the left sidebar, there's an 'Explorer' section with a tree view of datasets and tables. The main area is titled 'Untitled query' and contains the SQL code provided above. Below the code, a green circular icon indicates 'Query completed'. Underneath, the 'Query results' section displays a table with two columns: 'customer\_state' and 'avg.freight\_value'. The results are:

customer_state	avg.freight_value
PB	43.0917
RR	43.088
RO	41.3305
AC	40.0479
PI	39.1151

At the bottom of the results table, it says 'Results per page: 50 ▾ 1 – 5 of 5 |< < > >|'. The top navigation bar includes 'Google Cloud', 'Scaler-DSML-APR25-SQL', a search bar, and various icons for upgrade and dismiss.

The top 5 states with the highest average freight value are Paraíba (PB), Roraima (RR), Rondônia (RO), Acre (AC) and Piauí (PI) with Paraíba (PB) having highest the average freight value of 43.0917 and Piauí (PI) with the 5<sup>th</sup> highest average freight value of 39.1151.

```
select c.customer_state,
       round(avg(oi.freight_value),4) as avg_freight_value
  from target.order_items oi join target.orders o on oi.order_id = o.order_id
 join target.customers c on o.customer_id = c.customer_id
 where o.order_delivered_customer_date is not null
 group by c.customer_state
 order by avg_freight_value
 limit 5;
```

The screenshot shows the Google Cloud BigQuery interface. On the left, the Explorer sidebar displays a project named "scaler-dsml-apr25-sql" with a "target" dataset containing tables like customers, geolocation, order\_items, etc. In the center, an "Untitled query" window is open with the following SQL code:

```

1 select c.customer_state,
2      round(avg(oi.freight_value),4) as avg_freight_value
3  from target.order_items oi join target.orders o on oi.order_id = o.order_id
4    join target.customers c on o.customer_id = c.customer_id
5   where o.order_delivered_customer_date is not null
6   group by c.customer_state
7   order by avg_freight_value
8   limit 5;

```

The status bar indicates "Query completed". Below the code, the "Results" tab is selected, showing the following data:

customer_state	avg_freight_value
SP	15.115
PR	20.4718
MG	20.6258
RJ	20.9098
DF	21.0722

At the bottom, there are navigation links for "Save results" and "Open in".

The top 5 states with the lowest average freight value are São Paulo (SP), Paraná (PR), Minas Gerais (MG), Rio de Janeiro (RJ) and Distrito Federal (DF) with São Paulo (SP) having the lowest average freight value of 15.115 and Distrito Federal (DF) with the 5<sup>th</sup> lowest average freight value of 21.0722.

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

```

select c.customer_state,
       round(avg(timestamp_diff(order_delivered_customer_date,order_purchase_timestamp,
                                day))) as delivery_time
  from target.orders o join target.customers c on o.customer_id = c.customer_id
 where o.order_delivered_customer_date is not null
 group by c.customer_state
 order by delivery_time
 limit 5;

```

The screenshot shows the Google Cloud BigQuery interface. On the left is the Explorer sidebar with a tree view of datasets and tables. In the center is the Query Editor window titled "Untitled query". The query code is:

```

1 select c.customer_state,
2      round(avg(timestamp_diff(order_delivered_customer_date,order_purchase_timestamp, day))) as delivery_time
3 from target.orders o join target.customers c on o.customer_id = c.customer_id
4 where o.order_delivered_customer_date is not null
5 group by c.customer_state
6 order by delivery_time
7 limit 5;

```

The status bar indicates "Query completed". Below the code is the "Query results" section with a table:

customer_state	delivery_time
SP	8.0
MG	12.0
PR	12.0
DF	13.0
SC	14.0

At the bottom of the results table, it says "Results per page: 50 1 - 5 of 5".

The top 5 states with the lowest average delivery time are the southern/southeastern states of São Paulo (SP), Minas Gerais (MG), Paraná (PR), Distrito Federal (DF) and Santa Catarina (SC). That benefit from better infrastructure. Rio de Janeiro (RJ) didn't make it to the top 5 (6<sup>th</sup> place) given its proximity to São Paulo (SP).

```

select c.customer_state,
       round(avg(timestamp_diff(order_delivered_customer_date,order_purchase_timestamp,
day))) as delivery_time
  from target.orders o join target.customers c on o.customer_id = c.customer_id
 where o.order_delivered_customer_date is not null
 group by c.customer_state
 order by delivery_time desc
 limit 5;

```

This screenshot shows the Google Cloud BigQuery interface again. The Explorer sidebar is identical. The Query Editor window is titled "Untitled query" and contains the same query as the previous screenshot:

```

1 select c.customer_state,
2      round(avg(timestamp_diff(order_delivered_customer_date,order_purchase_timestamp, day))) as delivery_time
3 from target.orders o join target.customers c on o.customer_id = c.customer_id
4 where o.order_delivered_customer_date is not null
5 group by c.customer_state
6 order by delivery_time
7 limit 5;

```

The status bar indicates "Query completed". Below the code is the "Query results" section with a table:

customer_state	delivery_time
RR	29.0
AP	27.0
AM	26.0
AL	24.0
PA	23.0

At the bottom of the results table, it says "Results per page: 50 1 - 5 of 5".

On the other hand, the top 5 states with the highest average delivery time are the northeastern/northern states of Roraima (RR), Amapá (AP), Amazonas (AM), Alagoas (AL) and Pará (PA) reflecting logistical challenges in those regions of Brazil.

**5.d. 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.**

```
select c.customer_state,
       round(avg(timestamp_diff(o.order_delivered_customer_date,o.order_estimated_delivery_date,day))) as avg_diff_fast_delivery
  from target.orders o join target.customers c on o.customer_id = c.customer_id
 where o.order_delivered_customer_date is not null and order_status = "delivered"
 group by c.customer_state
 order by avg_diff_fast_delivery
 limit 5;
```

The screenshot shows the Google Cloud BigQuery interface. The left sidebar displays the project structure under 'Scalier-DSML-APR25-SQL' with a 'target' dataset containing tables like 'customers', 'geolocation', 'order\_items', etc. The main area shows an 'Untitled query' with the SQL code provided above. Below it, the 'Query results' section displays a table with the following data:

customer_state	avg_diff_fast_delivery
AC	-20.0
AM	-19.0
RO	-19.0
AP	-19.0
RR	-16.0

Acre (AC), Amazonas (AM), Rondônia (RO), Amapá (AP) and Roraima (RR) are the top 5 states where the order delivery is really fast as compared to the estimated date of delivery. A higher negative value of the difference between `order_delivered_customer_date` and `order_estimated_delivery_date` indicates much faster delivery. Customers in the state of Acre (AC) have reported fastest delivery of their orders with an approx. average of 20 days.

## 6. Analysis based on the payments:

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

```
select format_timestamp("%Y - %B", o.order_purchase_timestamp) as order_mnth,
       p.payment_type,
       count(p.order_id) as total_no_of_orders
  from target.payments p join target.orders o on p.order_id = o.order_id
 group by order_mnth, p.payment_type
 order by order_mnth, p.payment_type;
```

The screenshot shows the Google Cloud BigQuery interface. On the left, the Explorer sidebar displays a project named 'scaler-dsml-apr25-sql' with a 'target' dataset containing tables like customers, geolocation, order\_items, etc. In the center, an 'Untitled query' window is open with the SQL code provided above. Below the code, a 'Query completed' message is shown. The 'Results' tab is selected, displaying a table with the following data:

order_mnth	payment_type	total_no_of_orders
2016 - December	credit_card	1
2016 - October	UPI	63
2016 - October	credit_card	254
2016 - October	debit_card	2
2016 - October	voucher	23
2016 - September	credit_card	3
2017 - April	UPI	496

The screenshot shows the Google Cloud BigQuery interface. On the left, the Explorer sidebar displays a project named 'scaler-dsml-apr25-sql' with a 'target' dataset containing tables like customers, geolocation, order\_items, etc. In the center, an 'Untitled query' window is open with the SQL code provided above. Below the code, a 'Query completed' message is shown. The 'Results' tab is selected, displaying a table with the following data:

order_mnth	payment_type	total_no_of_orders
2016 - December	credit_card	1
2016 - October	UPI	63
2016 - October	credit_card	254
2016 - October	debit_card	2
2016 - October	voucher	23
2016 - September	credit_card	3
2017 - April	UPI	496
2017 - April	credit_card	1846
2017 - April	debit_card	27
2017 - April	voucher	202

The result indicates that customers can comfortably place orders using a credit card, debit card, UPI or vouchers as the payment\_type and majority of the orders have been placed with a credit card.

## 6.b. Find the no. of orders placed on the basis of the payment installments that have been paid.

```
select p.payment_installments,
       count(distinct o.order_id) as total_no_of_orders
  from target.orders o join target.payments p on o.order_id = p.order_id
 group by p.payment_installments
 order by p.payment_installments;
```

The screenshot shows the Google Cloud BigQuery interface. On the left is the Explorer sidebar with a tree view of datasets and tables. The main area is titled 'Untitled query' with a 'Run' button. Below it is a 'Query results' table with the following data:

Row	payment_installments	total_no_of_orders
1	0	2
2	1	49060
3	2	12389
4	3	10443
5	4	7088
6	5	5234
7	6	3916
8	7	1623

This screenshot shows the same BigQuery interface after a refresh or re-execution. The 'Query results' table now contains 10 rows, indicating a change in the data or a different execution environment. The data is as follows:

Row	payment_installments	total_no_of_orders
1	0	2
2	1	49060
3	2	12389
4	3	10443
5	4	7088
6	5	5234
7	6	3916
8	7	1623
9	8	4253
10	9	644

Customers that choose 0 or 1 installment indicates a strong preference for full payment or short-term installments. Many customers opt for flexible installments plan of 3-12 months suggesting that the full amount will be paid within a year of purchase.

However, long term installments plan (12+ months) indicate that the customers rely heavily on credit depending on the product. This means delayed revenue realization which affects cash flow. Hence, Target can opt for optimized payment plans with incentives for faster cash flow and offer flexibility to the customers.