Business Case: Target SQL

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 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 analysing 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.

Dataset:

https://drive.google.com/drive/folders/1TGEc66YKbD443nslRi1bWgVd238gJCnb

The data is available in 8 csv files:

1. customers.csv
2. sellers.csv
3. order_items.csv
4. geolocation.csv
5. payments.csv
6. reviews.csv
7. orders.csv
8. products.csv

Problem Statement

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

What does 'good' look like?

1. **Import the dataset and do usual exploratory analysis steps like checking the structure & characteristics of the dataset:**
   1.1. Data type of all columns in the "customers" table.
   Ans:
   - *Big Query and Results*:

```
SELECT
  column_name, data_type
FROM
  `scaler-dsml-sql-399417.Target.INFORMATION_SCHEMA.COLUMNS`
WHERE
  table_name = 'customers';
```

   - *Explanation/Insights:*
     In the customers table we have total 5 columns namely customer_id of datatype STRING (Text), customer_unique_id of datatype STRING (Text), customer_zip_code_prefix of datatype INT64 (Integer Value), customer_city of datatype STRING and customer_state of datatype STRING respectively

   - *Recommendations:*
     NA

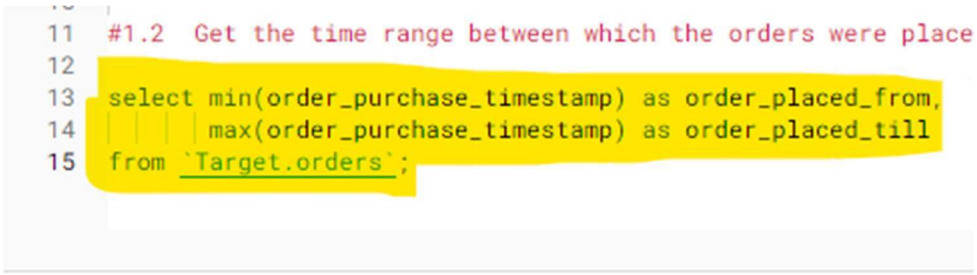
1.2. Get the time range between which the orders were placed.

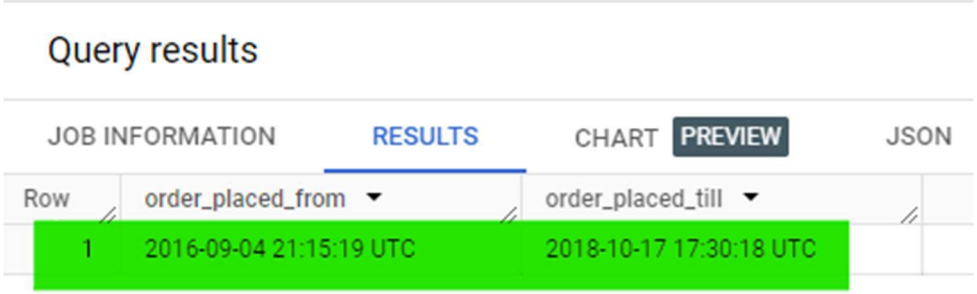
Ans:

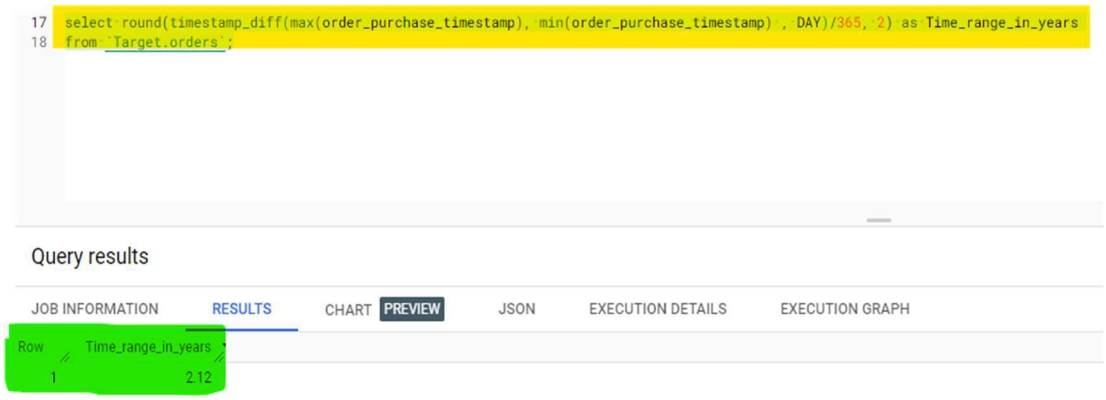
- *Big Query and Results*:

```
select min(order_purchase_timestamp) as order_placed_from,
       max(order_purchase_timestamp) as order_placed_till
from `Target.orders`;

select round(timestamp_diff(max(order_purchase_timestamp),
min(order_purchase_timestamp) , DAY)/365, 2) as Time_range_in_years
from `Target.orders`;
```







- *Explanation/Insights:*
  The time range between which the orders were placed is from 2026 to 2018 which in precise we can say that from 04-Sep-2016 to 17-Oct-2018 which will come upto 2.12 years.

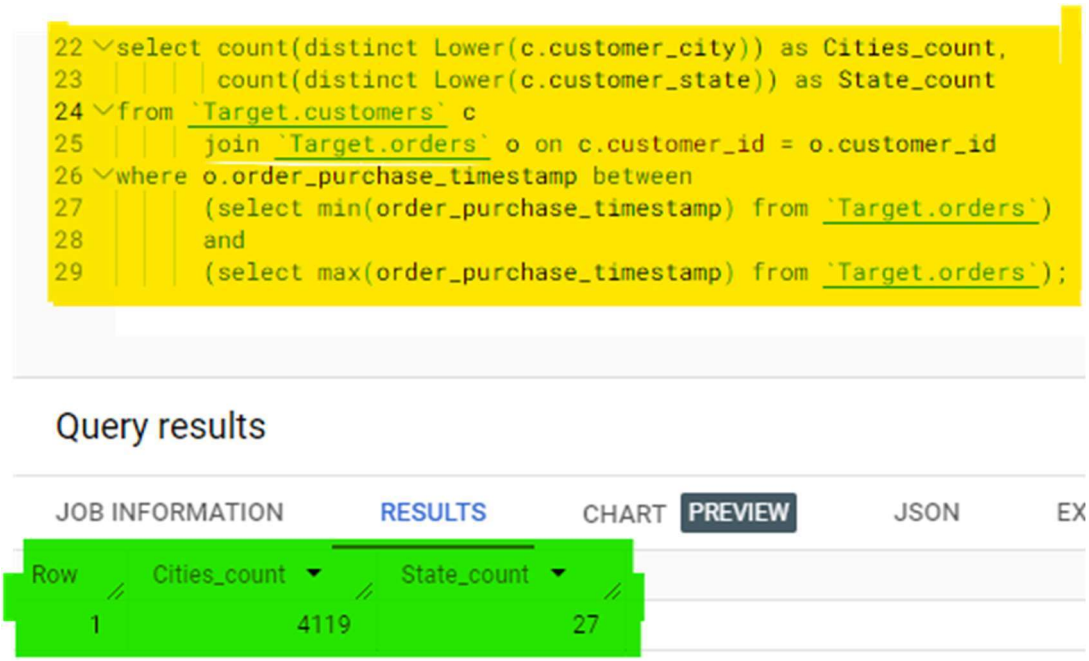
- *Recommendations:*
  NA

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

Ans:

- *Big Query and Results*:

```
select count(distinct Lower(c.customer_city)) as Cities_count,
count(distinct Lower(c.customer_state)) as State_count
from `Target.customers` c join `Target.orders` o on c.customer_id =
o.customer_id
where o.order_purchase_timestamp between (select
min(order_purchase_timestamp) from `Target.orders`) and (select
max(order_purchase_timestamp) from `Target.orders`)
```



- *Explanation/Insights:*

  Totally unique Cities and unique States of customers who ordered during the period from 04-Sep-2016 to 17-Oct-2018 are 4119 and 27 respectively. So we can say that "Target" has been spread over 4119 cities and 27 states.

- *Recommendations:*

  We can list down all the states and cities neighbour to the all spread over cities/states and try to market in those areas for the same.

2. In-depth Exploration:

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

Ans:

- *Big Query and Results*:

```
select *, (count_orders - (lag(count_orders, 1) over(order by year)))
as trending_over_year,
       if((count_orders - (lag(count_orders, 1) over(order by year)))
is not null, round((count_orders - (lag(count_orders, 1) over(order
by year))))/(lag(count_orders, 1) over(order by year))*100, 2), 0) as
Growth_percentage
from
 (select distinct extract(YEAR FROM order_purchase_timestamp) as
year,
       count(order_id) over(partition by (extract(YEAR FROM
order_purchase_timestamp))) as count_orders,
  from `Target.orders`)
order by count_orders;
```

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

select *, (count_orders - (lag(count_orders, 1) over(order by year))) as trending_over_year,
       if((count_orders - (lag(count_orders, 1) over(order by year))) is not null,
                round((count_orders - (lag(count_orders, 1) over(order by year)))/(lag(count_orders, 1) over(order by year))*100, 2),
                0
                ) as Growth_percentage
from
(
  select distinct extract(YEAR FROM order_purchase_timestamp) as year,
       count(order_id) over(partition by (extract(YEAR FROM order_purchase_timestamp))) as count_orders,
  from `Target.orders`
)
order by count_orders;
```

Query results

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

Row	year	count_orders	trending_over_year	Growth_percentage
1	2016	329	null	0.0
2	2017	45101	44772	13608.51
3	2018	54011	8910	19.76

***Note*: In the snap, table name a is missed but I have updated in the query as results were same**

- *Explanation/Insights:*
  We can see clearly that the orders were placed in a growing trend over the years. In detail we can assume that 2016 was the starting year where only 329 orders were placed where in the next year 2017, orders were exponentially increased to 45101 where we can see 44772 extra orders were placed compared to 2016 which comes increase in % of 13608.51 and if we go further in 2018 54011 orders were placed in which increase in 8910 in numbers and with 19.76 % of growth.

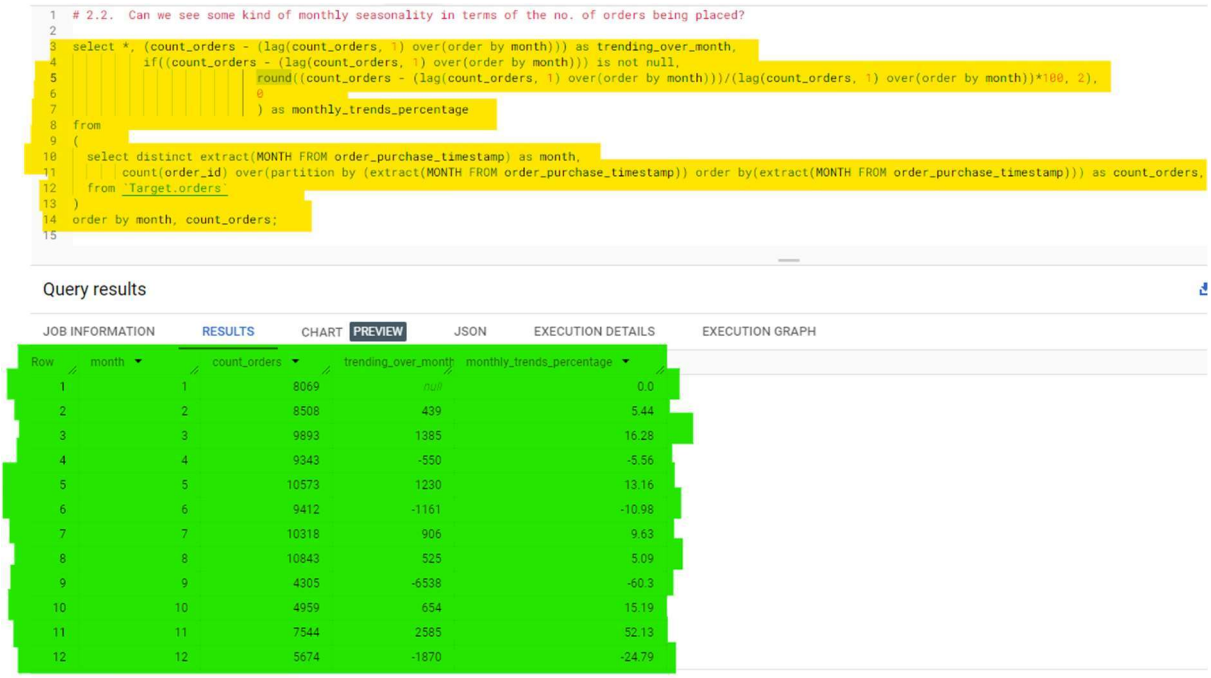
- *Recommendations:*
  Maintain the trust of employees with all required support. Expand the network with all remaining cities and states

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

Ans:

- *Big Query and Results*:

```
select *, (count_orders - (lag(count_orders, 1) over(order by
month))) as trending_over_month,
          if((count_orders - (lag(count_orders, 1) over(order by
month))) is not null,round((count_orders - (lag(count_orders, 1)
over(order by month)))/(lag(count_orders, 1) over(order by
month))*100, 2),0)
as monthly_trends_percentage
from
(select distinct extract(MONTH FROM order_purchase_timestamp) as
month,  count(order_id) over(partition by (extract(MONTH FROM
order_purchase_timestamp)) order by(extract(MONTH FROM
order_purchase_timestamp))) as count_orders,
  from `Target.orders`) a
order by month, count_orders;
```



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

select *, (count_orders - (lag(count_orders, 1) over(order by month))) as trending_over_month,
          if((count_orders - (lag(count_orders, 1) over(order by month))) is not null,
                    round((count_orders - (lag(count_orders, 1) over(order by month)))/(lag(count_orders, 1) over(order by month))*100, 2),
                    0
                    ) as monthly_trends_percentage
from
(
  select distinct extract(MONTH FROM order_purchase_timestamp) as month,
       count(order_id) over(partition by (extract(MONTH FROM order_purchase_timestamp)) order by(extract(MONTH FROM order_purchase_timestamp))) as count_orders,
  from `Target.orders`
)
order by month, count_orders;
```

Query results

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

Row	month	count_orders	trending_over_month	monthly_trends_percentage
1	1	8069	null	0.0
2	2	8508	439	5.44
3	3	9893	1385	16.28
4	4	9343	-550	-5.56
5	5	10573	1230	13.16
6	6	9412	-1161	-10.98
7	7	10318	906	9.63
8	8	10843	525	5.09
9	9	4305	-6538	-60.3
10	10	4959	654	15.19
11	11	7544	2585	52.13
12	12	5674	-1870	-24.79

***Note*: In the snap, table name a is missed but I have updated in the query as results were same**

- *Explanation/Insights:*

  We can see month by month orders irrespective of year here. We can assume that orders are placed highest in the month of May (05), July (07) and August (08) and orders placing comes down and hits the lowest in the month of September (09) and October (10). Also, we can see the trend that from August to September it drops around ~ 60%

- *Recommendations:*

  "Target" can concentrate mainly on non-seasonal months namely May, July and August, to provide more offers and do more marketing via various platforms so that orders placing can be increased.

2.3. 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 : Mornings
13-18 hrs : Afternoon
19-23 hrs : Night

Ans:

- *Big Query and Results*:

```
select distinct time_of_the_day, count_of_timeof_the_day,
        dense_rank() over(order by count_of_timeof_the_day desc) as
Time_of_the_day_rank
from
(
  select *, count(hrs) over(partition by time_of_the_day) as
count_of_timeof_the_day,
  from
  (
    select *, case when hrs between 0 and 5 then 'Dawn'
               when hrs between 6 and 11 then 'Mornings'
               when hrs between 12 and 17 then 'Afternoon'
               when hrs between 18 and 23 then 'Night' end as
time_of_the_day
    from
      (
        select order_id, order_purchase_timestamp,
        extract(HOUR from order_purchase_timestamp) as hrs
        from `Target.orders`
      ) a
  ) b
) c
order by Time_of_the_day_rank;
```

In

```sql
select distinct time_of_the_day, count_of_timeof_the_day,
       dense_rank() over(order by count_of_timeof_the_day desc) as Time_of_the_day_rank
from
(
  select *, count(hrs) over(partition by time_of_the_day) as count_of_timeof_the_day,
  from
  (
    select *, case when hrs between 0 and 5 then 'Dawn'
                   when hrs between 6 and 11 then 'Mornings'
                   when hrs between 12 and 17 then 'Afternoon'
                   when hrs between 18 and 23 then 'Night' end as time_of_the_day
    from
      (
        select order_id, order_purchase_timestamp,
        extract(HOUR from order_purchase_timestamp) as hrs
        from `Target.orders`
      )
  )
)
order by Time_of_the_day_rank;
```

Query results

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

Row	time_of_the_day	count_of_timeof_the	Time_of_the_day_ran
1	Afternoon	38361	1
2	Night	34100	2
3	Mornings	22240	3
4	Dawn	4740	4

<u>Note</u>: In the snap, table name a, b, and c are missed but I have updated in the query as results were same

- *<u>Explanation/Insights:</u>*
  We can say that in the 'afternoon' time the most of the orders were placed and in the 'Dawn' time of the day least orders were placed

- *<u>Recommendations:</u>*
  We can do any website/app updating on the 'Dawn' time of the day as least orders were placed. Also we can include some extra benefits/offers on the same time zones so that more customers will be engaged and web or app traffic will be shared across all timings of the day

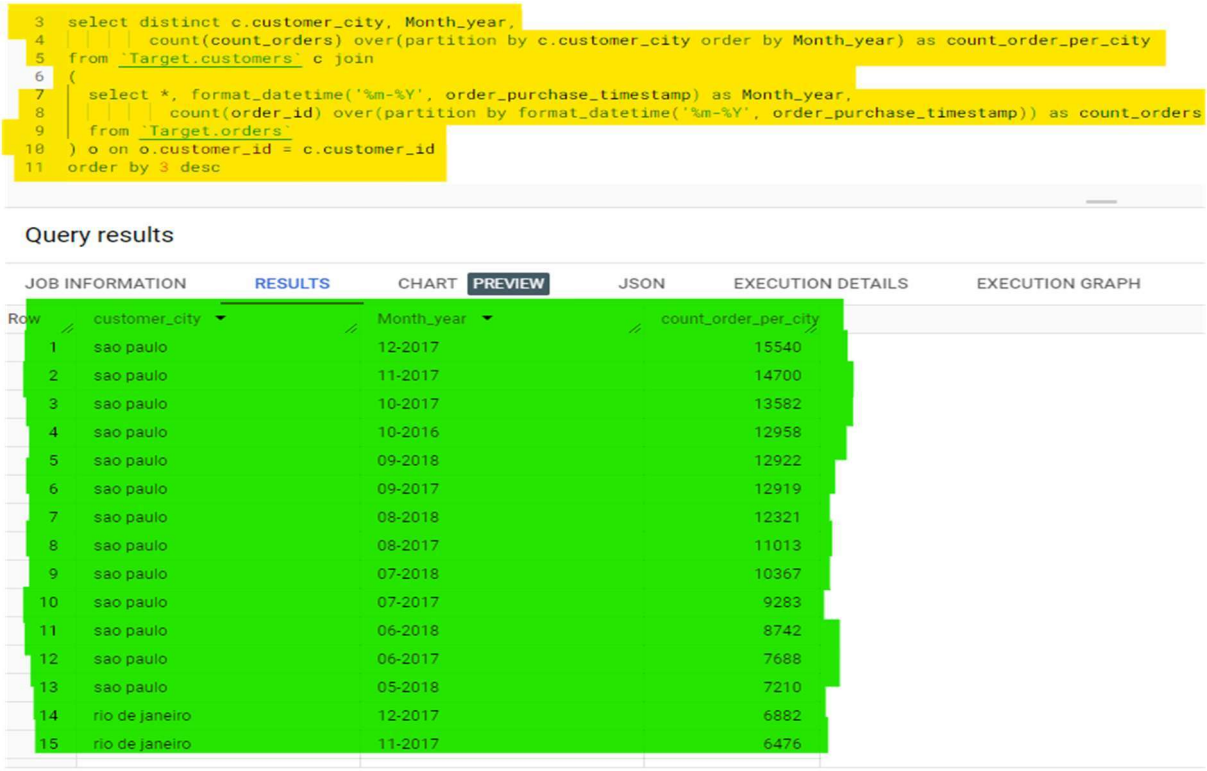
3. Evolution of E-commerce orders in the Brazil region:

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

Ans:

- *Big Query and Results*:

```
select distinct c.customer_city, Month_year,
        count(count_orders) over(partition by c.customer_city order by
Month_year) as count_order_per_city
from `Target.customers` c join
(
  select *, format_datetime('%m-%Y', order_purchase_timestamp) as
Month_year,
          count(order_id) over(partition by format_datetime('%m-%Y',
order_purchase_timestamp)) as count_orders
  from `Target.orders`
) o on o.customer_id = c.customer_id
order by 3 desc
```



Query results

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

Row	customer_city	Month_year	count_order_per_city
1	sao paulo	12-2017	15540
2	sao paulo	11-2017	14700
3	sao paulo	10-2017	13582
4	sao paulo	10-2016	12958
5	sao paulo	09-2018	12922
6	sao paulo	09-2017	12919
7	sao paulo	08-2018	12321
8	sao paulo	08-2017	11013
9	sao paulo	07-2018	10367
10	sao paulo	07-2017	9283
11	sao paulo	06-2018	8742
12	sao paulo	06-2017	7688
13	sao paulo	05-2018	7210
14	rio de janeiro	12-2017	6882
15	rio de janeiro	11-2017	6476

- *Explanation/Insights:*
  We can clearly see that the most orders were placed in “Sao Paulo” and followed by “Rio de Janerio”. The least number of orders were placed in the multiple cities on respective month - year which is counted as 1

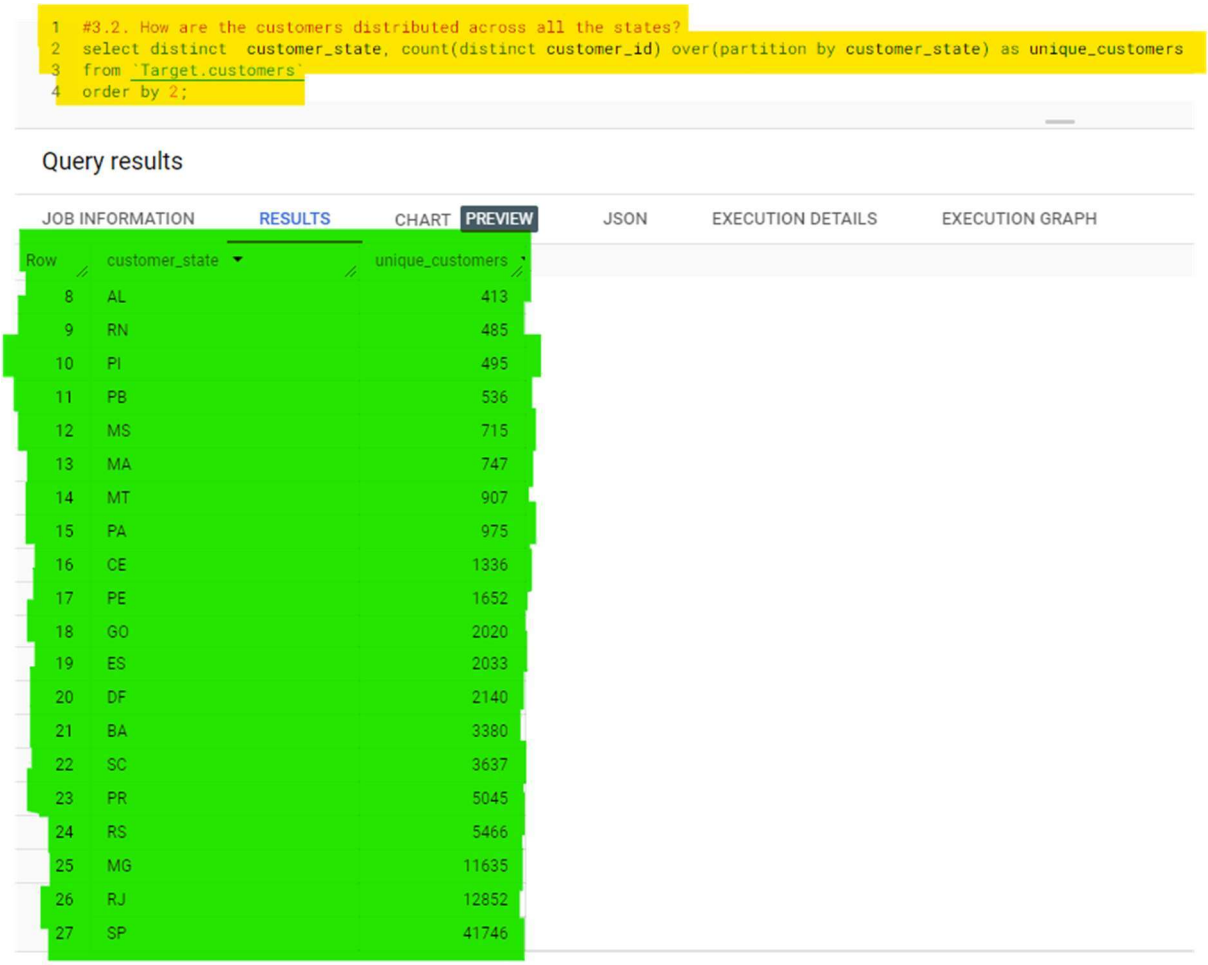
- *Recommendations:*
  Need to focus on the cities where a smaller number of orders are placed. Needs to increase the marketing and increasing in the sellers are required and based on the need of the city people we must adjust the benefits and offers.

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

Ans:

- *Big Query and Results*:

```
select distinct  customer_state,
count(distinct customer_id) over(partition by customer_state) as
unique_customers
from `Target.customers`
order by 2;
```



```
#3.2. How are the customers distributed across all the states?
select distinct  customer_state, count(distinct customer_id) over(partition by customer_state) as unique_customers
from `Target.customers`
order by 2;
```

Query results

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

Row	customer_state	unique_customers
8	AL	413
9	RN	485
10	PI	495
11	PB	536
12	MS	715
13	MA	747
14	MT	907
15	PA	975
16	CE	1336
17	PE	1652
18	GO	2020
19	ES	2033
20	DF	2140
21	BA	3380
22	SC	3637
23	PR	5045
24	RS	5466
25	MG	11635
26	RJ	12852
27	SP	41746

- *Explanation/Insights:*
  The highest Unique Customers are present in the state “SP” and lowest unique Customers are present in the state “AL” followed by “RN”

- *Recommendations:*
  “Target” can check what is the criteria or what strategies giving them the highest customers in the state “SP”. It may be mostly because of offers, price or due to specific marketing. So same must be implemented in the other states where customer centric marketing and offers needs to be given.

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.

Ans:

- *Big Query and Results*:

```
select *,
    round(((sum_of_order_cost - lag(sum_of_order_cost, 1) over(order by
sum_of_order_cost))*100/(lag(sum_of_order_cost, 1) over(order by
sum_of_order_cost))), 2)
    as percentage_increase_of_order_cost
from
(
  select distinct extract(YEAR from o.order_purchase_timestamp) as Year,
                  sum(p.payment_value) over(partition by extract(YEAR from
order_purchase_timestamp)) as sum_of_order_cost,
                  count(p.payment_value) over(partition by extract(YEAR from
order_purchase_timestamp)) as count_of_order_year
  from `Target.orders` o join `Target.payments` p on o.order_id = p.order_id
  where EXTRACT(MONTH from o.order_purchase_timestamp) between 1 and 8
) op
where Year in (2017, 2018)
order by Year
```

Query results

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

Row	Year	sum_of_order_cost	count_of_order_year	percentage_increase
1	2017	3669022.12	24391	null
2	2018	8694733.84	55995	136.98

- *Explanation/Insights:*
This is one finding where cost of the orders were increased by ~136% with increasing in the number of orders also.

- *Recommendations:*
So, there is considerable increase in the cost of the order. But we have keep in mind the cost of each product (selling price) apart from that we can continue to monitor the service and support to each customer and seller perspective

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

Ans:

- *Big Query and Results*:

```
select c.customer_city, sum(oi.price) as total_prices_of_city, avg(oi.price)
avg_price_of_the_city
from `Target.orders` o join `Target.customers` c on c.customer_id =
o.customer_id
join `Target.order_items` oi on oi.order_id = o.order_id
group by 1
order by 3 desc;
```

```
6  select c.customer_city, sum(oi.price) as total_prices_of_city, avg(oi.price) avg_price_of_the_city
7  from `Target.orders` o join `Target.customers` c on c.customer_id = o.customer_id
8  join `Target.order_items` oi on oi.order_id = o.order_id
9  group by 1
10 order by 3 desc;
11
```

Query results

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

Row	customer_city	total_prices_of_city	avg_price_of_the_city
1	pianco	2200.0	2200.0
2	nova esperanca do piria	2199.0	2199.0
3	engenheiro navarro	1997.0	1997.0
4	agrestina	1989.0	1989.0
5	mariental	1799.0	1799.0
6	loreto	1599.99	1599.99
7	ibitita	1450.0	1450.0
8	pirpirituba	1340.0	1340.0
9	barao ataliba nogueira	1300.0	1300.0
10	barra longa	1300.0	1300.0
11	bom jesus do araguaia	1199.9	1199.9
12	passagem	1199.0	1199.0
13	bom jesus do galho	4795.2899999999991	1198.8225
14	coari	1190.65	1190.65
15	brejo do cruz	1160.0	1160.0

- *Explanation/Insights:*

  "Pianco" has the highest order_price as there is only one order and multiple city having the one order. "Sao Paulo" is having highest total of the price.

- *Recommendations:*

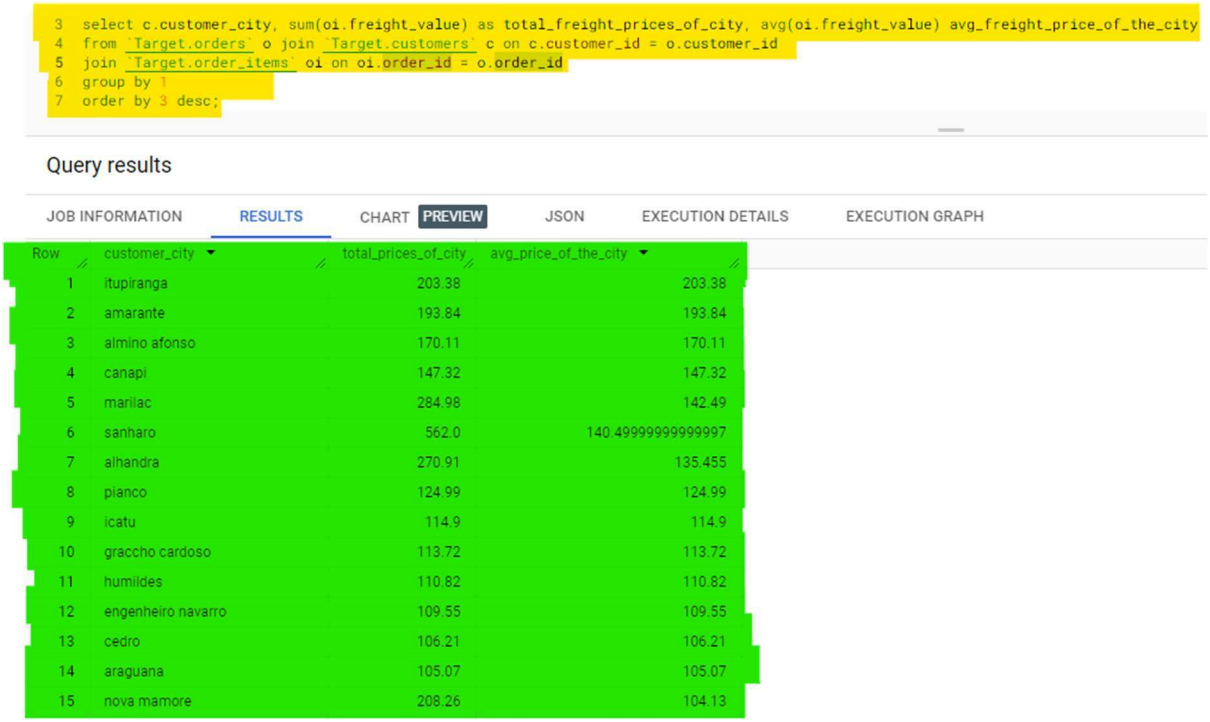
  Concentrating on the cities where lowest order price is found. Try to increase the order by marketing and offers. Once the number of orders increases, will it will increase the sales

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

Ans:

- *Big Query and Results*:

```
select c.customer_city, sum(oi.freight_value) as
total_freight_prices_of_city, avg(oi.freight_value)
avg_freight_price_of_the_city
from `Target.orders` o join `Target.customers` c on c.customer_id =
o.customer_id
join `Target.order_items` oi on oi.order_id = o.order_id
group by 1
order by 3 desc;
```



```
3  select c.customer_city, sum(oi.freight_value) as total_freight_prices_of_city, avg(oi.freight_value) avg_freight_price_of_the_city
4  from `Target.orders` o join `Target.customers` c on c.customer_id = o.customer_id
5  join `Target.order_items` oi on oi.order_id = o.order_id
6  group by 1
7  order by 3 desc;
```

Query results

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

Row	customer_city	total_prices_of_city	avg_price_of_the_city
1	itupiranga	203.38	203.38
2	amarante	193.84	193.84
3	almino afonso	170.11	170.11
4	canapi	147.32	147.32
5	marilac	284.98	142.49
6	sanharo	562.0	140.49999999999997
7	alhandra	270.91	135.455
8	pianco	124.99	124.99
9	icatu	114.9	114.9
10	graccho cardoso	113.72	113.72
11	humildes	110.82	110.82
12	engenheiro navarro	109.55	109.55
13	cedro	106.21	106.21
14	araguana	105.07	105.07
15	nova mamore	208.26	104.13

- *Explanation/Insights:*
  Highest “freight value” is in the city “itupiranga”. Several cities are having 0 freight value where it is very beneficial for both customers and sellers.

- *Recommendations:*
  Try to reduce the freight value by having more sellers and concentrate on the cities where freight value is lesser or zero to sell more orders

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. 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

Ans:

- *Big Query and Results*:

```
select order_id, customer_id,
      datetime_diff(order_delivered_customer_date, order_purchase_timestamp,
DAY) AS time_to_deliver,
      datetime_diff(order_estimated_delivery_date,
order_delivered_customer_date, DAY) AS diff_estimated_delivery
from `Target.orders`
where lower(order_status) = 'delivered' and order_delivered_customer_date is
not null
order by 4 desc
```

```
8  select order_id, customer_id,
9      datetime_diff(order_delivered_customer_date, order_purchase_timestamp, DAY) AS time_to_deliver,
10     datetime_diff(order_estimated_delivery_date, order_delivered_customer_date, DAY) AS diff_estimated_delivery
11 from `Target.orders`
12 where lower(order_status) = 'delivered' and order_delivered_customer_date is not null
13 order by 4 desc
14
```

Query results

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

Row	order_id	customer_id	time_to_deliver	diff_estimated_delivery
1	0607f0efea4b566f1eb8f7d3c2397320	a5fbb6579eacbeb02752a143b...	3	146
2	c72727d29cde4cf870d569bf65edabfd	964253ff0e4e08180064764a4...	6	139
3	eec7f369423b033e549c02f3c5381205	32cef4bdd6bfa50612d81dc77...	20	134
4	c2bb89b5c1dd978d507284be78a04cb2	6357fffb5704244d552615bbfc...	16	123
5	40dc2ba6f322a17626aac6244332828c	6210a37f9d6a265a4f3fbe2c21...	7	108
6	1a695d543b7302aa9446c8d5fbd632bf	b882cbae40f34e60a3ec3efefa...	12	83
7	39e0115911bf404857e14baa7f097feb	816642e9995c2461f2172469e...	11	82
8	38930f76efb00b138f4d632e4d557341	0f9043c635f86f7eb1e8fc0770...	11	77
9	c5132855100a12d63ed4e8ae05f9594d	6e5b6ba2e8de70d9e920b541a...	12	77
10	559eea5a72341a4c82dbce9884277cb7	5fd5cb96f515996e88d84c610...	13	77

- *Explanation/Insights:*
  Time taken for several orders are way earlier if you see top first row in which order is delivered 146 days earlier compared to the estimated delivery time.

- *Recommendations:*
  We can concentrate on the top 1st order id and compare with other tables to get how was the fast delivery happened and must implement the same for other orders also. For ex: finding out which seller delivering orders fast or any city and implement same procedure for others as well

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

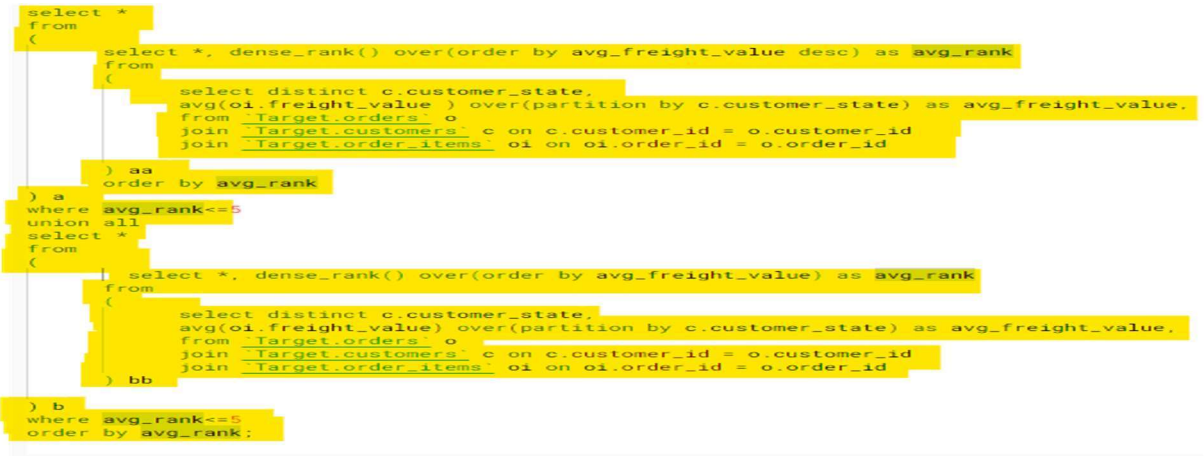
Ans:

- *Big Query and Results*:

```
select *
from
(
	select *, dense_rank() over(order by avg_freight_value desc) as
avg_rank
	from
	(
		select distinct c.customer_state,
		avg(oi.freight_value ) over(partition by c.customer_state) as
avg_freight_value,
		from `Target.orders` o
		join `Target.customers` c on c.customer_id = o.customer_id
		join `Target.order_items` oi on oi.order_id = o.order_id

	) aa
	order by avg_rank
) a
where avg_rank<=5
union all
select *
from
(
	  select *, dense_rank() over(order by avg_freight_value) as avg_rank
	from
	(
		select distinct c.customer_state,
		avg(oi.freight_value) over(partition by c.customer_state) as
avg_freight_value,
		from `Target.orders` o
		join `Target.customers` c on c.customer_id = o.customer_id
		join `Target.order_items` oi on oi.order_id = o.order_id
	) bb

) b
where avg_rank<=5
order by avg_rank;
```



Query results

JOB INFORMATION | RESULTS | CHART | PREVIEW | JSON

Row	customer_state	avg_freight_value	avg_rank
1	SP	15.14727539041...	1
2	PR	20.53165156794...	2
3	MG	20.63016680630...	3
4	RJ	20.96092393168...	4
5	DF	21.04135494596...	5
6	PI	39.14797047970...	5
7	AC	40.07336956521...	4
8	RO	41.06971223021...	3
9	PB	42.72380398671...	2
10	RR	42.98442307692...	1

- *Explanation/Insights:*

  From the Query results we can clearly see that “SP” state having the lowest avg freight value and “RR” is having the highest avg freight value. First 5 states are having the lowest avg freight value and next 5 states are having the highest avg freight value

- *Recommendations:*

  We can plan to reduce the freight value on the states were having the highest freight value so that orders from respective states will be increased as freight value reduces for the same we can increase the sellers in those states. The states where lowest freight value is there, we must utilise those states supply products through those states more.

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

Ans:

- *Big Query and Results*:

```
select *
from
(
      select *, dense_rank() over(order by avg_delivery_time desc) as
avg_rank
      from
      (
            select distinct c.customer_state,
            round(avg(o.time_to_deliver) over(partition by
c.customer_state), 2) as avg_delivery_time,
            from
            (
                  select order_id, customer_id,
                  datetime_diff(order_delivered_customer_date,
order_purchase_timestamp, DAY) AS time_to_deliver,
                  from `Target.orders`
                  where lower(order_status) = 'delivered' and
order_delivered_customer_date is not null

            ) o
            join `Target.customers` c on c.customer_id = o.customer_id

      ) aa
   order by avg_delivery_time

) a
where avg_rank<=5
union all
select *
from
(
        select *, dense_rank() over(order by avg_delivery_time) as avg_rank
      from
      (
            select distinct c.customer_state,
            round(avg(o.time_to_deliver) over(partition by
c.customer_state), 2) as avg_delivery_time,
            from
            (
                  select order_id, customer_id,
                  datetime_diff(order_delivered_customer_date,
order_purchase_timestamp, DAY) AS time_to_deliver,
                  from `Target.orders`
                  where lower(order_status) = 'delivered' and
order_delivered_customer_date is not null

            ) o
            join `Target.customers` c on c.customer_id = o.customer_id
      ) bb

) b
where avg_rank<=5
order by avg_delivery_time;
```

Query results

JOB INFORMATION | RESULTS | CHART | PREVIEW | JSON

Row	customer_state	avg_delivery_time	avg_rank
1	SP	8.3	1
2	PR	11.53	2
3	MG	11.54	3
4	DF	12.51	4
5	SC	14.48	5
6	PA	23.32	5
7	AL	24.04	4
8	AM	25.99	3
9	AP	26.73	2
10	RR	28.98	1

- *Explanation/Insights:*

  We can see that “SP” having the lowest avg delivery time and “RR” having the highest avg delivery time. First 5 represents the lowest avg delivery time while next 5 represents the highest avg delivery time.
  We can also see that SP is having lowest avg freight value and lowest avg delivery time and RR is having highest avg freight value and highest avg delivery time. We can say that delivery time is directly proportional to the freight value of the state

- *Recommendations:*

Try to reduce the freight value through increasing the sellers and other startergies so that deliver time can be decreased.

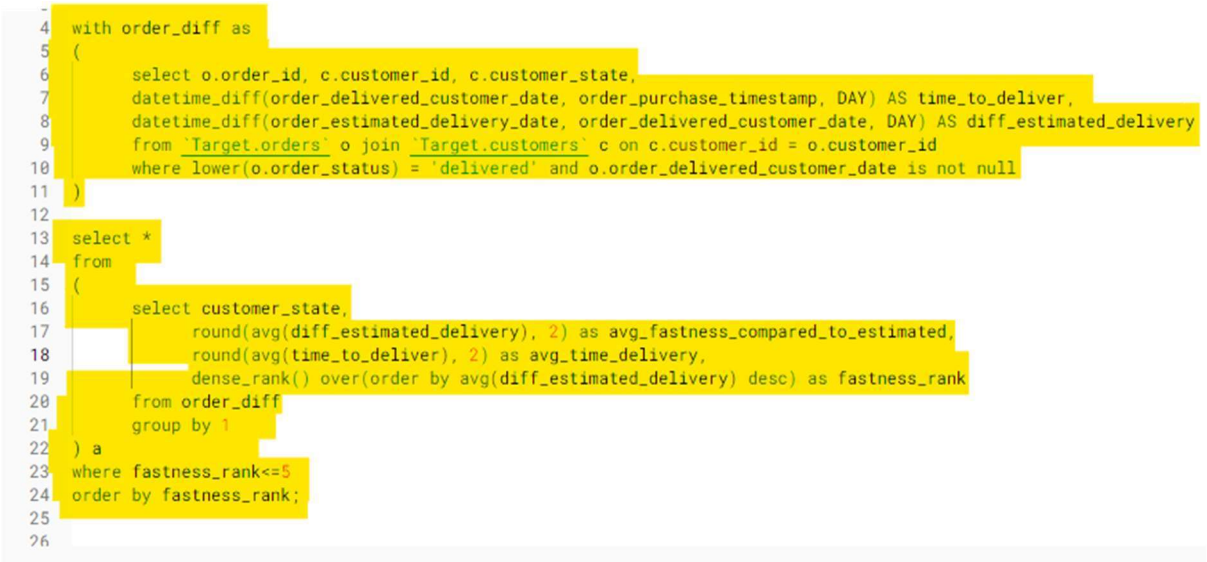
5.4. Find out the top 5 states where the order delivery is really fast as compared to the estimated date of delivery.
You can use the difference between the averages of actual & estimated delivery date to figure out how fast the delivery was for each state.

Ans:

- *Big Query and Results*:

```
with order_diff as
(
      select o.order_id, c.customer_id, c.customer_state,
      datetime_diff(order_delivered_customer_date, order_purchase_timestamp,
DAY) AS time_to_deliver,
      datetime_diff(order_estimated_delivery_date,
order_delivered_customer_date, DAY) AS diff_estimated_delivery
      from `Target.orders` o join `Target.customers` c on c.customer_id =
o.customer_id
      where lower(o.order_status) = 'delivered' and
o.order_delivered_customer_date is not null
)

select *
from
(
      select customer_state,
            round(avg(diff_estimated_delivery), 2) as
avg_fastness_compared_to_estimated,
            round(avg(time_to_deliver), 2) as avg_time_delivery,
            dense_rank() over(order by avg(diff_estimated_delivery) desc) as
fastness_rank
      from order_diff
      group by 1
) a
where fastness_rank<=5
order by fastness_rank;
```



Query results

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

Row	customer_state	avg_fastness_compared_to_estimated	avg_time_delivery	fastness_rank
1	AC	19.76	20.64	1
2	RO	19.13	18.91	2
3	AP	18.73	26.73	3
4	AM	18.61	25.99	4
5	RR	16.41	28.98	5

- *Explanation/Insights:*

  We can see that in the state “AC” the fastness of the delivery compared to estimated is fast. And followed by RO, AP, AM, and RR

- *Recommendations:*

  We can see all the orders data of the state “AC” and implement the same in other states so that the fastness of the delivery will be can be better in other states also.

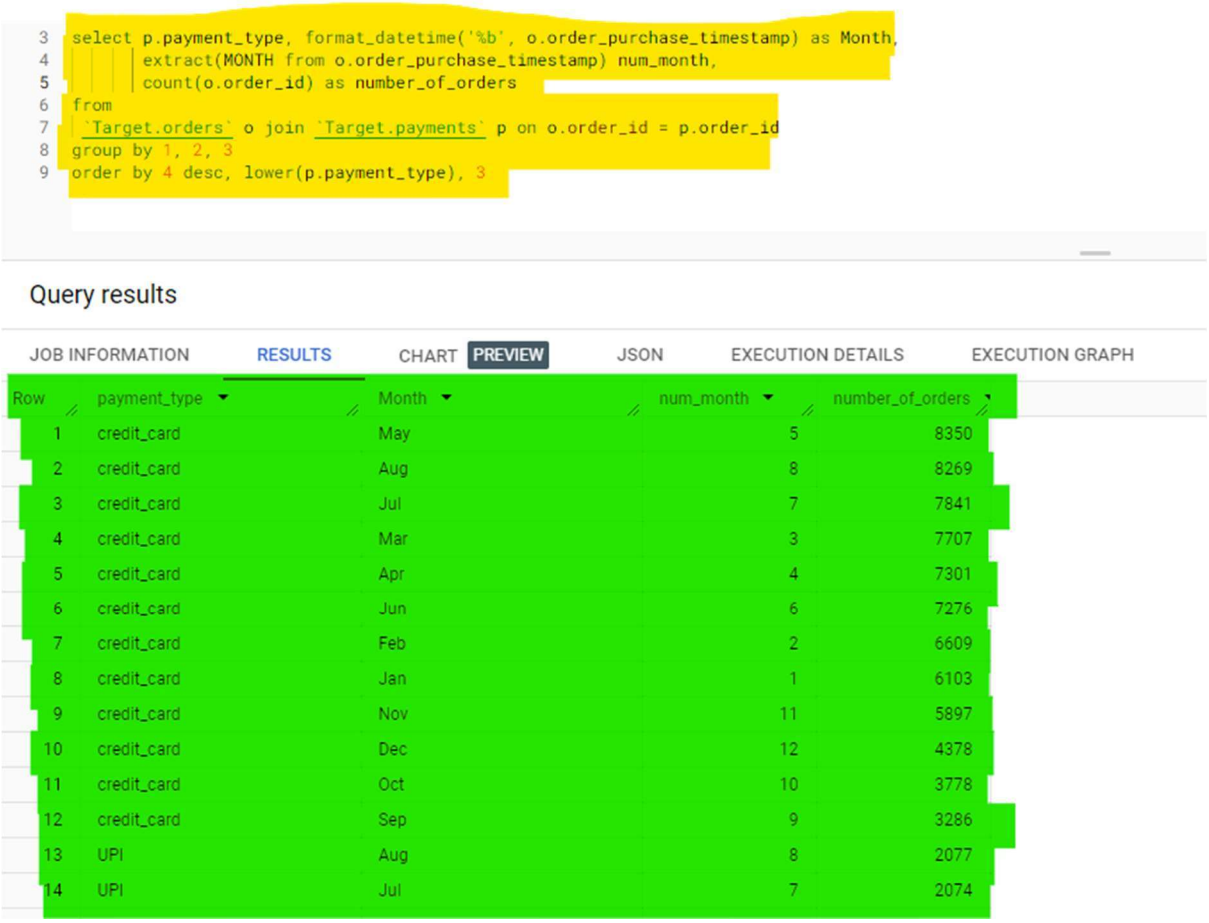
6. Analysis based on the payments:

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

Ans:

- *Big Query and Results*:

```
select p.payment_type, format_datetime('%b', o.order_purchase_timestamp) as
Month,
       extract(MONTH from o.order_purchase_timestamp) num_month,
       count(o.order_id) as number_of_orders
from
 `Target.orders` o join `Target.payments` p on o.order_id = p.order_id
group by 1, 2, 3
order by 4 desc, lower(p.payment_type), 3
```



Query results

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

Row	payment_type	Month	num_month	number_of_orders
1	credit_card	May	5	8350
2	credit_card	Aug	8	8269
3	credit_card	Jul	7	7841
4	credit_card	Mar	3	7707
5	credit_card	Apr	4	7301
6	credit_card	Jun	6	7276
7	credit_card	Feb	2	6609
8	credit_card	Jan	1	6103
9	credit_card	Nov	11	5897
10	credit_card	Dec	12	4378
11	credit_card	Oct	10	3778
12	credit_card	Sep	9	3286
13	UPI	Aug	8	2077
14	UPI	Jul	7	2074

- *Explanation/Insights:*
  Highest number of orders were placed using credit cards. Also, highest number of orders were placed in the month of May. Less number of orders were mostly in September.

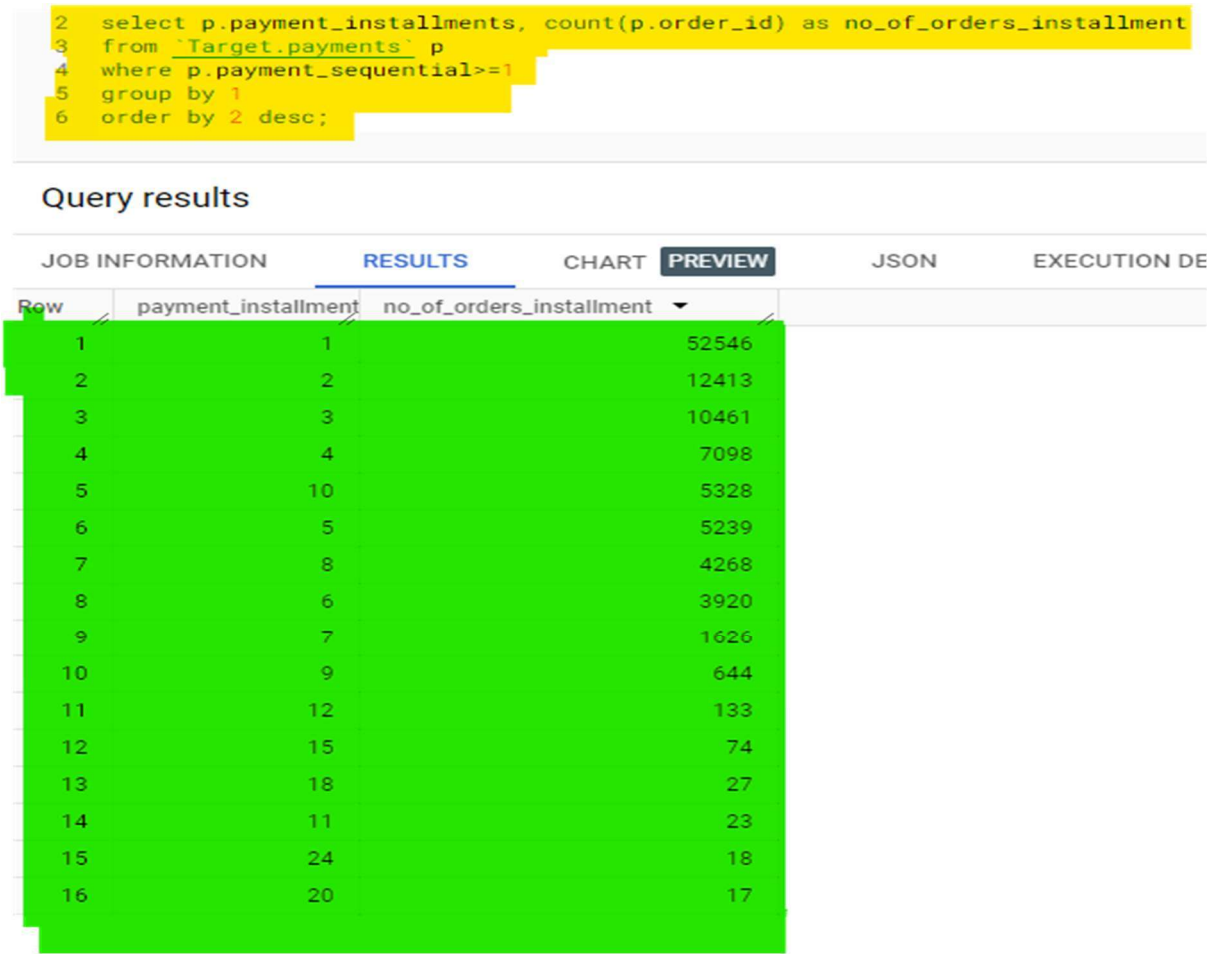
- *Recommendations:*
  As less number orders were placed in September and highest number of orders were placed through credit card, we can give more offers on credit card on the month of September particularly.

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

Ans:

- *Big Query and Results*:

```
select p.payment_installments, count(p.order_id) as no_of_orders_installment
from `Target.payments` p
where p.payment_sequential>=1
group by 1
order by 2 desc;
```



Row	payment_installment	no_of_orders_installment
1	1	52546
2	2	12413
3	3	10461
4	4	7098
5	10	5328
6	5	5239
7	8	4268
8	6	3920
9	7	1626
10	9	644
11	12	133
12	15	74
13	18	27
14	11	23
15	24	18
16	20	17

- *Explanation/Insights:*
  Highest number of orders were paid in one installment and 0 also with same meaning.

- *Recommendations:*
  We can see almost all the installments being paid. We can concentrate on the orders which are not paid fully with opted installment.

Initial Queries before finalising:

2.1 :

```
select *, (count_orders - (lag(count_orders, 1) over(order by
count_orders))) as trending_over_year
from
(
  select distinct extract(YEAR FROM order_purchase_timestamp) as year,
       count(order_id) over(partition by (extract(YEAR FROM
order_purchase_timestamp))) as count_orders,
  from `Target.orders`
)
order by count_orders
```

```
select *, (count_orders - (lag(count_orders, 1) over(order by count_orders))) as trending_over_year
from
(
  select distinct extract(YEAR FROM order_purchase_timestamp) as year,
       count(order_id) over(partition by (extract(YEAR FROM order_purchase_timestamp))) as count_orders,
  from `Target.orders`
)
order by count_orders
```

Query results

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

Row	year	count_orders	trending_over_year
1	2016	329	null
2	2017	45101	44772
3	2018	54011	8910