Target Business Case study

Executive summary by -Srikanth Reddy Gosala

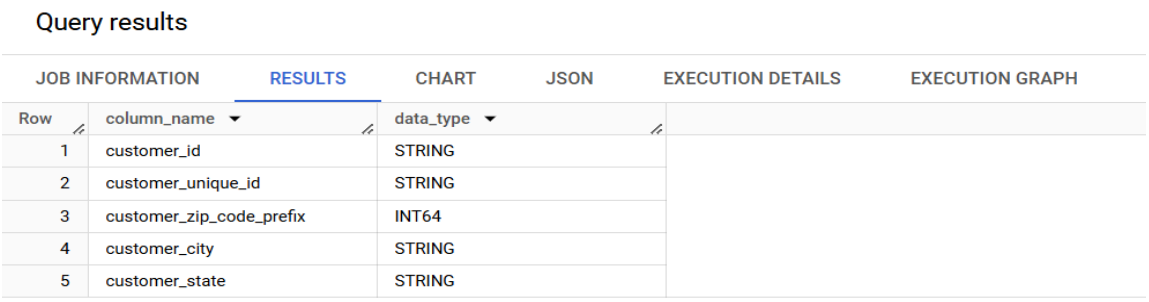
1. Import the dataset and do usual exploratory analysis steps like checking the structure & characteristics of the dataset:

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

Query:

```
select column_name, data_type

from target_data.INFORMATION_SCHEMA.COLUMNS

Where table_name = 'customers';
```

Result:



Query results

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

Explanation:

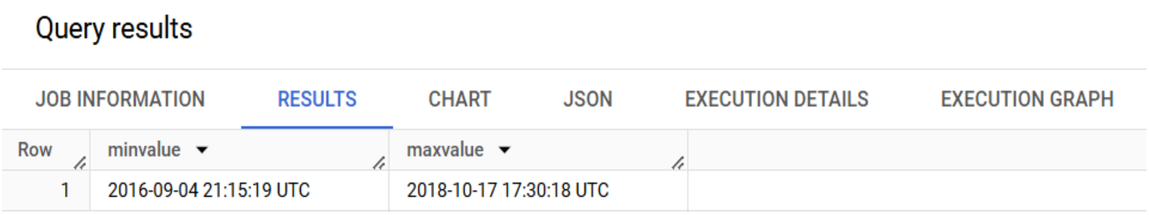
Data type of columns, with column names can be shown by using Information schema. This shows the structure of the table and the kind of attributes the table stores.

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

Query:

```
SELECT

min(order_purchase_timestamp) as minvalue,

max(order_purchase_timestamp) as maxvalue

from `target_data.orders`
```

Result:



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

Explanation:

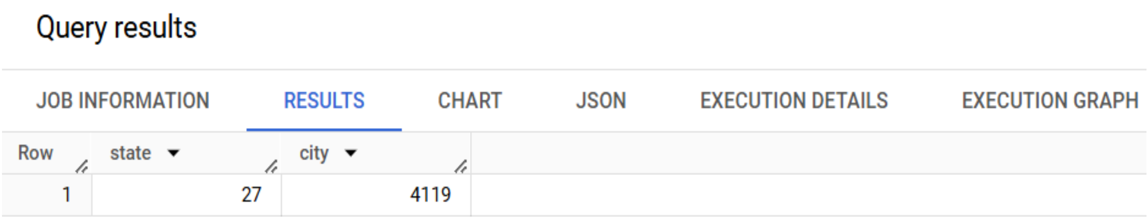
In the above query I used MIN() and MAX() functions to get time intervals of orders placed.

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

Query:

```
select count(distinct c.customer_state) state,

count(distinct customer_city) city

from

`target_data.customers` c

join

`target_data.orders` o

on c.customer_id = o.customer_id
```

Result:



Row	state	city
1	27	4119

Explanation:

In this query I used JOIN to combine CUSTOMERS and ORDERS tables to count NO.OF CITIES and STATES.

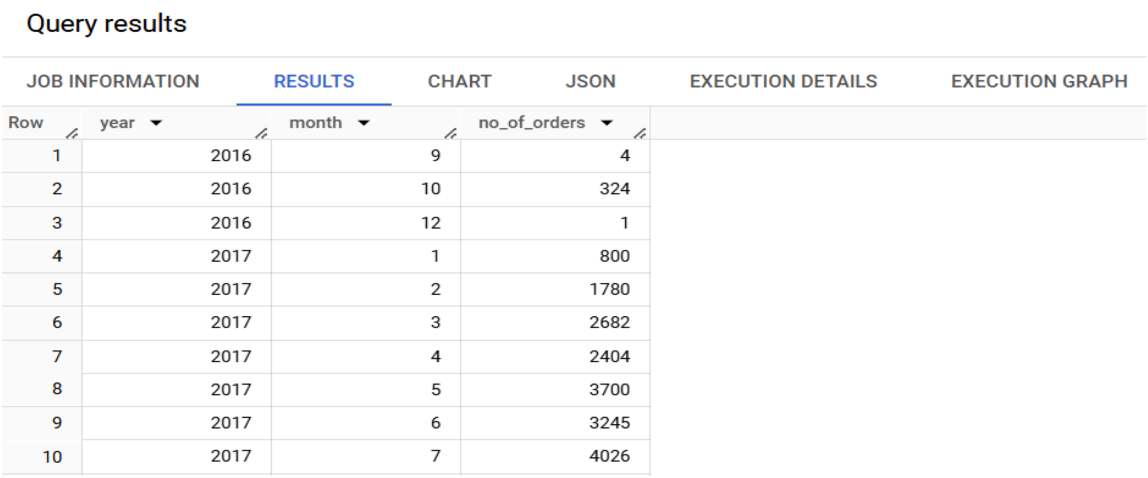
2. In-depth Exploration:

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

Query:

```
SELECT

extract(year from order_purchase_timestamp) as year,

extract(month from order_purchase_timestamp) as month,

 count(order_id) as no_of_orders

FROM `target_data.orders`

group by 1,2

order by 1,2
```

Result:



Row	year	month	no_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

Explanation:

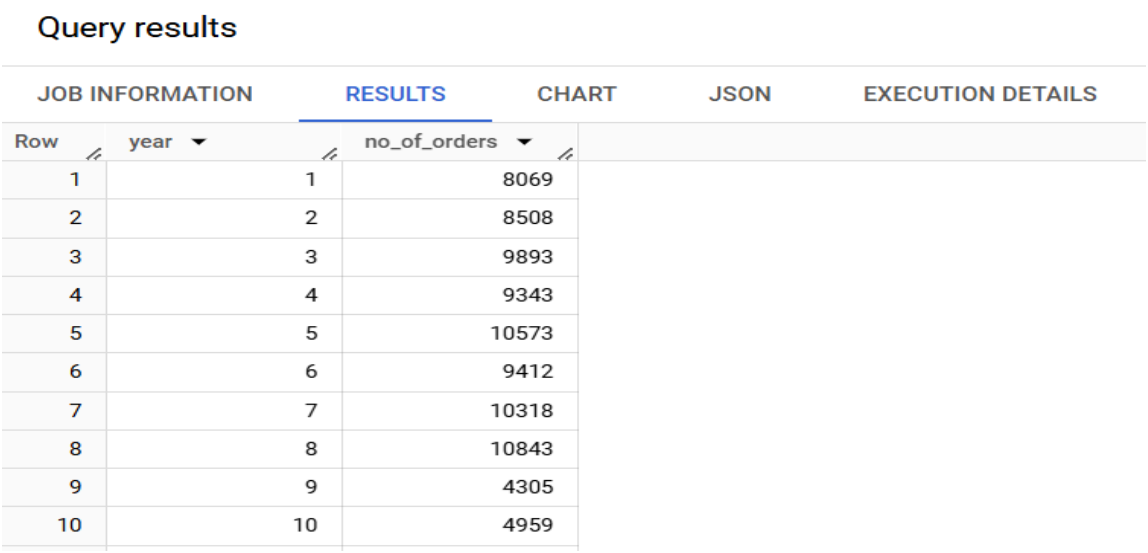
No: of orders can be calculated by count() function grouping by year and month.

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

Query:

```
SELECT

extract(month from order_purchase_timestamp) as year,

count(distinct order_id) as no_of_orders

FROM `target_data.orders`

group by 1

order by 1
```

Result:



Row	year	no_of_orders
1	1	8069
2	2	8508
3	3	9893
4	4	9343
5	5	10573
6	6	9412
7	7	10318
8	8	10843
9	9	4305
10	10	4959

Explanation:

By grouping on month, we can count the number of orders using count() aggregate function. When we sort them in an order, a comparison analysis can be done.

III.During what time of the day, do the Brazilian customers mostly place their orders? (Dawn, Morning, Afternoon or Night)

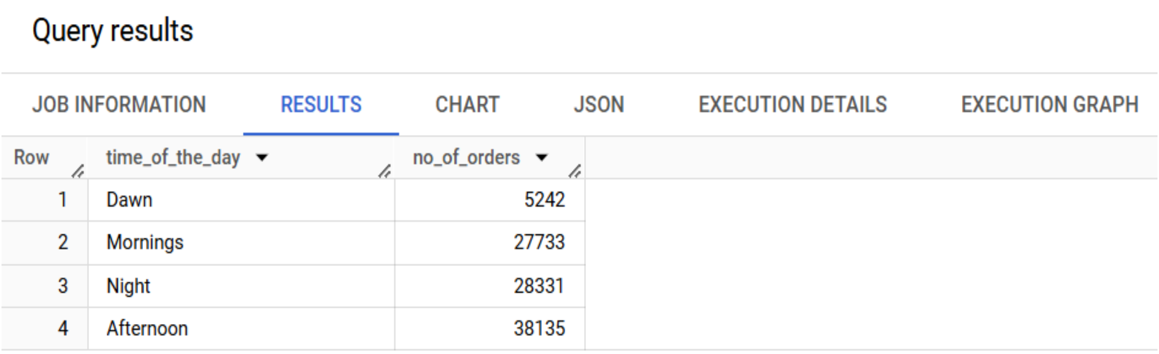
- 0-6 hrs : Dawn
- 7-12 hrs : Mornings
- 13-18 hrs : Afternoon
- 19-23 hrs : Night

Query:

```
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 time_of_the_day,
count(order_id) as no_of_orders
FROM `target_data.orders`
 group by time_of_the_day
order by no_of_orders
```

Result:



Row	time_of_the_day	no_of_orders
1	Dawn	5242
2	Mornings	27733
3	Night	28331
4	Afternoon	38135

Explanation:

From the timestamp of orders, we can get the time of placing an order. Using case when expression, we can check the condition if the time or order is during dawn or mornings or night or afternoon. No: of orders can be counted using count() aggregate function by grouping according to the time lap.

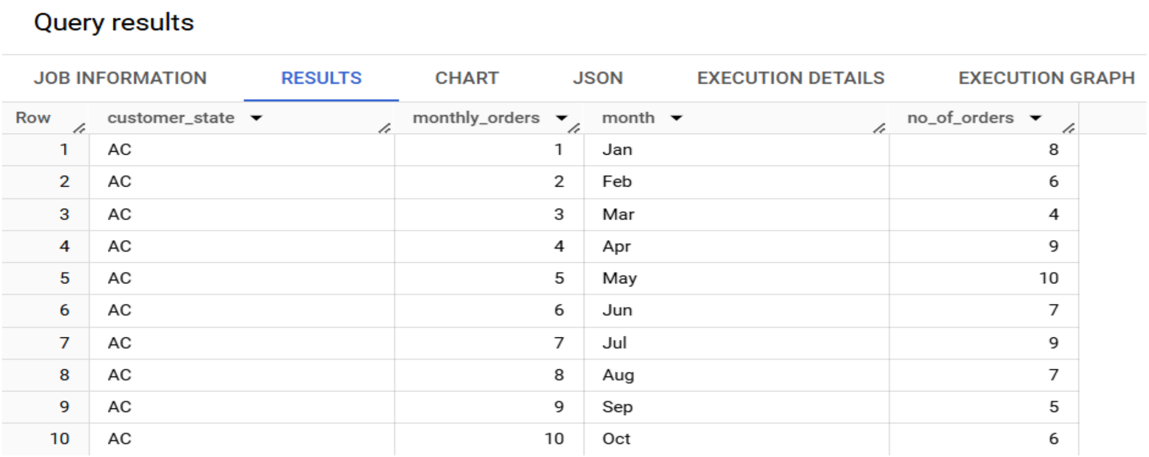
3. Evolution of E-commerce orders in the Brazil region:

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

Query:

```
SELECT

C.customer_state,

extract(month from O.order_purchase_timestamp) as
monthly_orders,

format_datetime('%b', O.order_purchase_timestamp) as month,

count(O.order_id) as no_of_orders

FROM `target_data.orders` O

inner join

`target_data.customers` C

on C.customer_id = O.customer_id

group by 1,2,3

order by 1,2
```

Result:



Row	customer_state	monthly_orders	month	no_of_orders
1	AC	1	Jan	8
2	AC	2	Feb	6
3	AC	3	Mar	4
4	AC	4	Apr	9
5	AC	5	May	10
6	AC	6	Jun	7
7	AC	7	Jul	9
8	AC	8	Aug	7
9	AC	9	Sep	5
10	AC	10	Oct	6

Explanation:

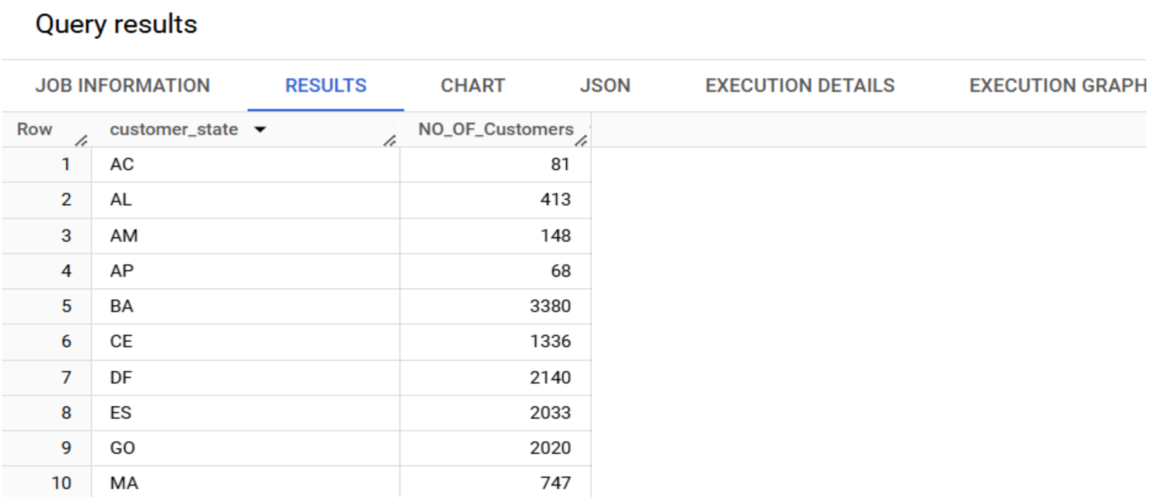
This analysis helps to get insights into customer purchase trends on a state by state basis. The state that has highest no. of orders in a given month or least no. of orders can be found out.

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

Query:

```
SELECT customer_state,

count(customer_unique_id) as NO_OF_Customers

FROM `target_data.customers`

group by 1

order by 1
```

Result:



Row	customer_state	NO_OF_Customers
1	AC	81
2	AL	413
3	AM	148
4	AP	68
5	BA	3380
6	CE	1336
7	DF	2140
8	ES	2033
9	GO	2020
10	MA	747

Explanation:

Grouping the states and counting the no: of customers would help us know from which state the orders are being placed more.

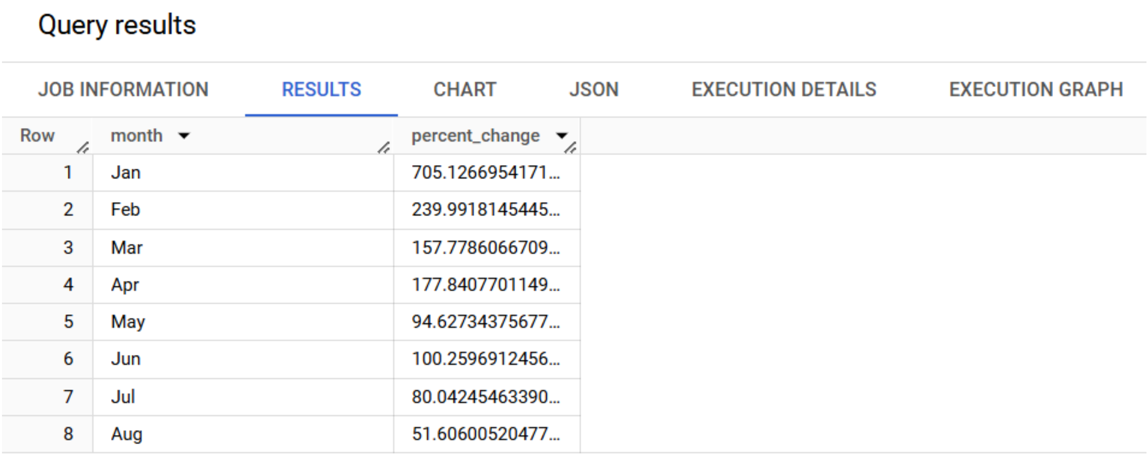
4. Impact on Economy: Analyze the money movement by e-commerce by looking at order prices, freight and others.

I. 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.

Query:

```
with cte1 as(

select

sum(p.payment_value) as orderval_2017,

extract(year from o.order_purchase_timestamp) as year,

format_datetime('%b', o.order_purchase_timestamp) as month,

from `target_data.orders` O

join

`target_data.payments` P

on o.order_id = p.order_id

where extract(year from o.order_purchase_timestamp) = 2017

and extract(month from order_purchase_timestamp) between 1
and 8

group by 2,3

order by 2,3),

 cte2 as (select

sum(p.payment_value) as orderval_2018,

extract(year from o.order_purchase_timestamp) as year,
format_datetime('%b', o.order_purchase_timestamp) as month,
extract(month from o.order_purchase_timestamp) as monthnum
from `target_data.orders` O
join `target_data.payments` P
on o.order_id = p.order_id
where extract(year from o.order_purchase_timestamp) = 2018
and extract(month from order_purchase_timestamp) between 1 and 8
group by 2,3,4
order by 2,3)
select
a.month,
((b.orderval_2018-a.orderval_2017)/a.orderva_l2017)*100 as
percent_change
from cte1 a inner join cte2 b
on a.month = b.month
order by b.monthnum
```

Result:



Row	month	percent_change
1	Jan	705.1266954171...
2	Feb	239.9918145445...
3	Mar	157.7786066709...
4	Apr	177.8407701149...
5	May	94.62734375677...
6	Jun	100.2596912456...
7	Jul	80.04245463390...
8	Aug	51.60600520477...

Explanation:

A common table expression is a temporary relational table which can be used later in a SQL statement. The table is called temp because it exists only during the scope of the sql statement written after the CTE. As attributes of 2 years are to be observed, 2 CTE tables are used to filter the data. Further % increase formula is applied to observe the change in the cost of orders.

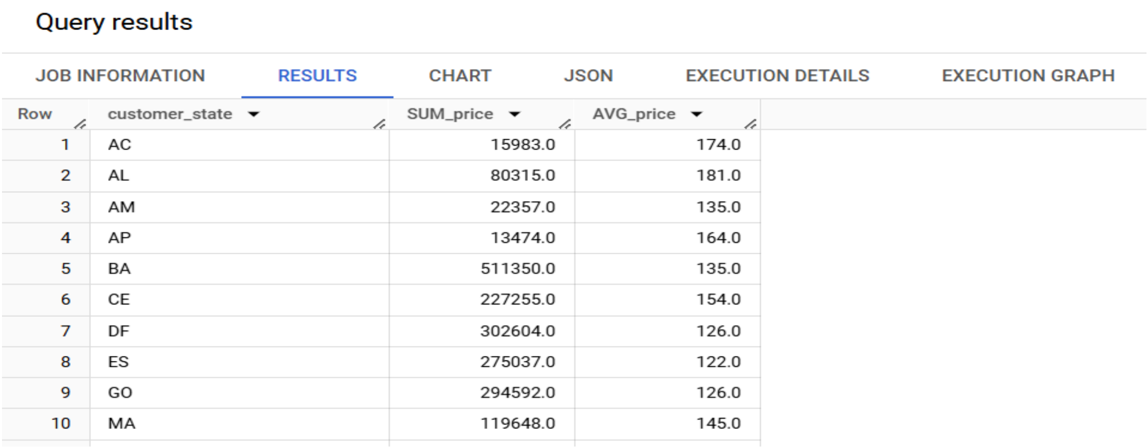
II. Calculate the Total & Average value of order price for each state.

Query:

```
SELECT

c.customer_state,

round(sum(oi.price)) as SUM_price,

round(avg(oi.price)) as AVG_price,

FROM `target_data.order_items` OI

inner join

`target_data.orders` O
```

```
on o.order_id = oi.order_id

inner join

`target_data.customers` C

on o.customer_id = c.customer_id

group by c.customer_state

order by 1
```

Result:



Query results

Row	customer_state	SUM_price	AVG_price
1	AC	15983.0	174.0
2	AL	80315.0	181.0
3	AM	22357.0	135.0
4	AP	13474.0	164.0
5	BA	511350.0	135.0
6	CE	227255.0	154.0
7	DF	302604.0	126.0
8	ES	275037.0	122.0
9	GO	294592.0	126.0
10	MA	119648.0	145.0

Explanation:

The avg and sum of freight value can be calculated by joining the orders and customers table

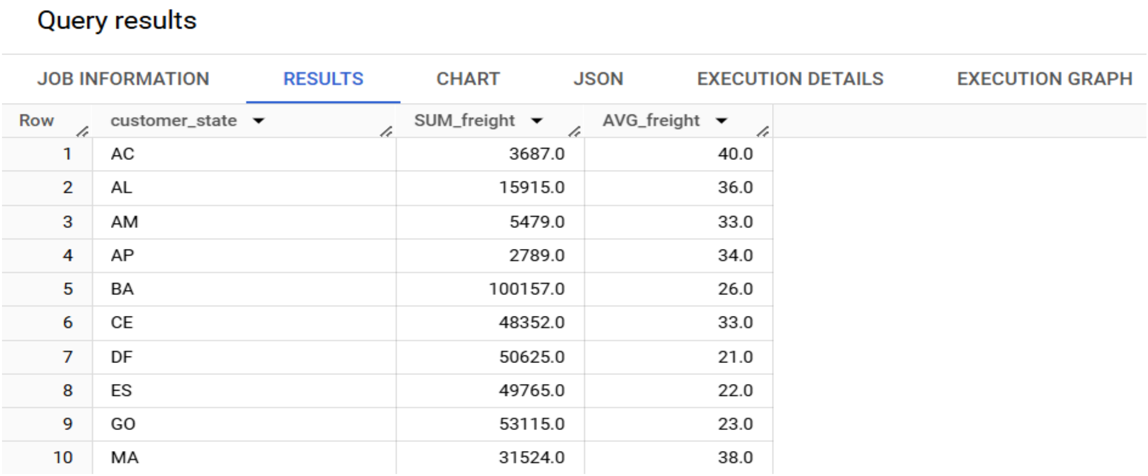
III.Calculate the Total & Average value of order freight for each state.

Query:

```
SELECT

c.customer_state,

round(sum(oi.freight_value)) as SUM_freight,

round(avg(oi.freight_value)) as AVG_freight
```

```
FROM `target_data.order_items` OI

inner join

`target_data.orders` O

on o.order_id = oi.order_id

inner join

`target_data.customers` C

on o.customer_id = c.customer_id

group by c.customer_state

order by 1
```

Result:



Query results

Row	customer_state	SUM_freight	AVG_freight
1	AC	3687.0	40.0
2	AL	15915.0	36.0
3	AM	5479.0	33.0
4	AP	2789.0	34.0
5	BA	100157.0	26.0
6	CE	48352.0	33.0
7	DF	50625.0	21.0
8	ES	49765.0	22.0
9	GO	53115.0	23.0
10	MA	31524.0	38.0

Explanation:

The avg and sum of freight value can be calculated by joining the orders and customers table

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

I.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.

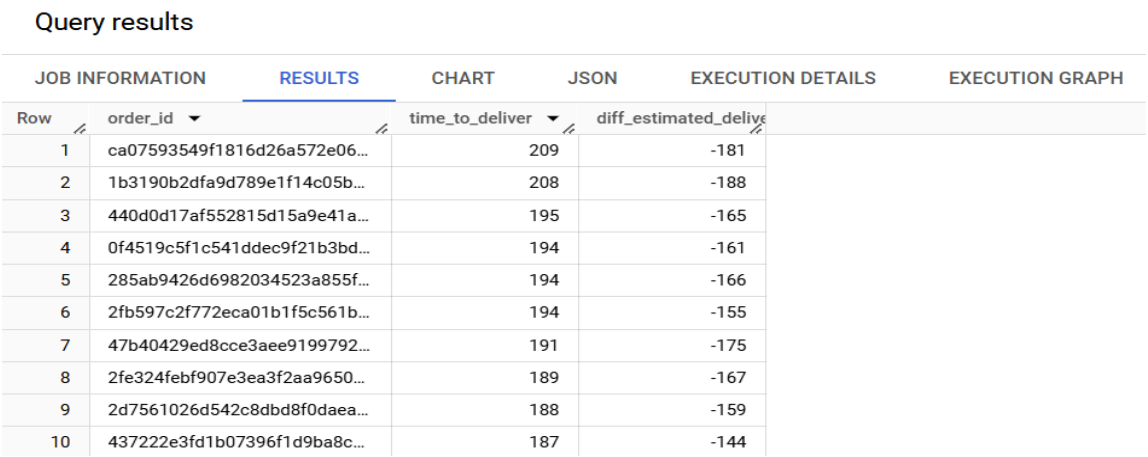
You can calculate the delivery time and the difference between the estimated & actual delivery date using the given formula:

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

Query:

```
SELECT

order_id,

date_diff(order_delivered_customer_date,
order_purchase_timestamp,

day) as time_to_deliver,

date_diff(order_estimated_delivery_date,

order_delivered_customer_date, day) as
diff_estimated_delivery

FROM `target_data.orders`

where order_delivered_customer_date is not null

and order_purchase_timestamp is not null

and order_estimated_delivery_date is not null

and order_delivered_customer_date is not null

order by time_to_deliver desc
```

Result:



Row	order_id	time_to_deliver	diff_estimated_delive
1	ca07593549f1816d26a572e06...	209	-181
2	1b3190b2dfa9d789e1f14c05b...	208	-188
3	440d0d17af552815d15a9e41a...	195	-165
4	0f4519c5f1c541ddec9f21b3bd...	194	-161
5	285ab9426d6982034523a855f...	194	-166
6	2fb597c2f772eca01b1f5c561b...	194	-155
7	47b40429ed8cce3aee9199792...	191	-175
8	2fe324febf907e3ea3f2aa9650...	189	-167
9	2d7561026d542c8dbd8f0daea...	188	-159
10	437222e3fd1b07396f1d9ba8c...	187	-144

Explanation:

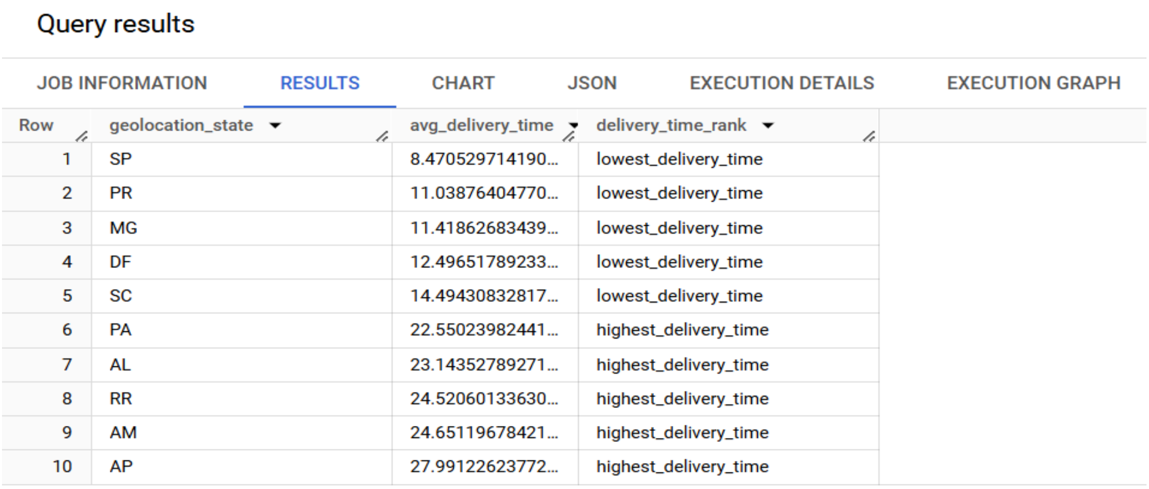
With the difference in the delivery time and estimated time, potential measures can be taken to enhance the fleet dispatching, route optimization and all activities that reduce chances of delays.

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

Query:

```
with del_time as

(SELECT g.geolocation_state,

avg(date_diff

 (o.order_delivered_customer_date,
o.order_purchase_timestamp, day))

as avg_delivery_time,

case when

dense_rank() over(order by

avg(date_diff
(o.order_delivered_customer_date,
o.order_purchase_timestamp, day))
desc) <=5 then 'highest_delivery_time'
when
dense_rank() over(order by
avg(date_diff(o.order_delivered_customer_date,
o.order_purchase_timestamp, day)))<=5 then
'lowest_delivery_time'
end as delivery_time_rank
FROM `target_data.orders` O
inner join `target_data.customers` C
on o.customer_id = c.customer_id
inner join `target_data.geolocation` G
on g.geolocation_zip_code_prefix =
c.customer_zip_code_prefix
group by 1)
select geolocation_state,
avg_delivery_time,
delivery_time_rank
from del_time
where delivery_time_rank is not null
order by del_time.avg_delivery_time
```

Result:


Query results

Row	geolocation_state	avg_delivery_time	delivery_time_rank
1	SP	8.470529714190...	lowest_delivery_time
2	PR	11.03876404770...	lowest_delivery_time
3	MG	11.41862683439...	lowest_delivery_time
4	DF	12.49651789233...	lowest_delivery_time
5	SC	14.49430832817...	lowest_delivery_time
6	PA	22.55023982441...	highest_delivery_time
7	AL	23.14352789271...	highest_delivery_time
8	RR	24.52060133630...	highest_delivery_time
9	AM	24.65119678421...	highest_delivery_time
10	AP	27.99122623772...	highest_delivery_time

Explanation:

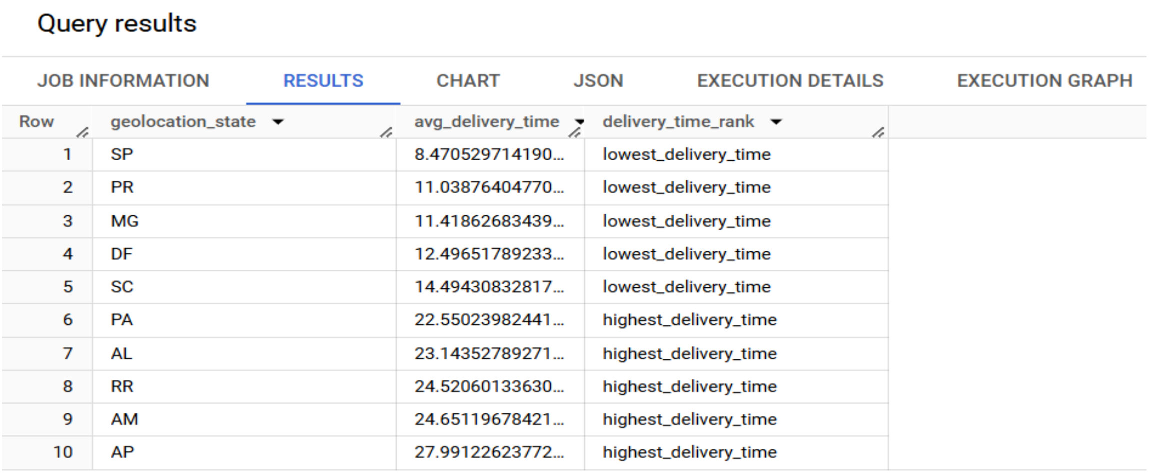
The result can be used to evaluate asset use, performance, baseline deviations and other focal points.

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

Query:

```
with del_time as

(SELECT g.geolocation_state,

avg(date_diff

(o.order_delivered_customer_date,
o.order_purchase_timestamp, day))

as avg_delivery_time,

case when

dense_rank() over(order by

avg(date_diff

(o.order_delivered_customer_date,
o.order_purchase_timestamp, day))
desc) <=5 then 'highest_delivery_time'
when
dense_rank() over(order by
avg(date_diff(o.order_delivered_customer_date,
o.order_purchase_timestamp, day)))<=5 then
'lowest_delivery_time'
end as delivery_time_rank
FROM `target_data.orders` O
inner join `target_data.customers` C
on o.customer_id = c.customer_id
inner join `target_data.geolocation` G
on g.geolocation_zip_code_prefix =
c.customer_zip_code_prefix
group by 1)
select geolocation_state,
avg_delivery_time,
delivery_time_rank
from del_time
where delivery_time_rank is not null
order by del_time.avg_delivery_time
```

Result:



Row	geolocation_state	avg_delivery_time	delivery_time_rank
1	SP	8.470529714190...	lowest_delivery_time
2	PR	11.03876404770...	lowest_delivery_time
3	MG	11.41862683439...	lowest_delivery_time
4	DF	12.49651789233...	lowest_delivery_time
5	SC	14.49430832817...	lowest_delivery_time
6	PA	22.55023982441...	highest_delivery_time
7	AL	23.14352789271...	highest_delivery_time
8	RR	24.52060133630...	highest_delivery_time
9	AM	24.65119678421...	highest_delivery_time
10	AP	27.99122623772...	highest_delivery_time

Explanation:

Identifying the state through geolocation_state is preferred as customer_state always might not be a valid value.

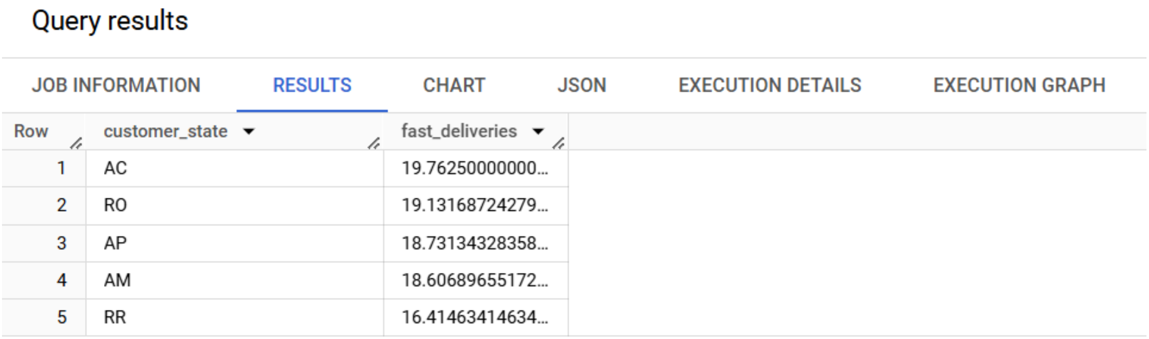
IV.Find out the top 5 states where the order delivery is really fast as compared to the estimated date of delivery.
You can use the difference between the averages of actual & estimated delivery date to figure out how fast the delivery was for each state.

Query:

```
select c.customer_state,

avg(datetime_diff(order_estimated_delivery_date,
order_delivered_customer_date, day)) as

fast_deliveries

from `target_data.orders` O

inner join `target_data.customers` C

on o.customer_id = c.customer_id

where order_delivered_customer_date is not null

group by 1
```

```
order by 2 desc

limit 5
```

Result:



Query results

Row	customer_state	fast_deliveries
1	AC	19.76250000000…
2	RO	19.13168724279…
3	AP	18.73134328358…
4	AM	18.60689655172…
5	RR	16.41463414634…

Explanation:

Avg delivery time and avg estimated time are calculated by filtering according to the state.

6. Analysis based on the payments:

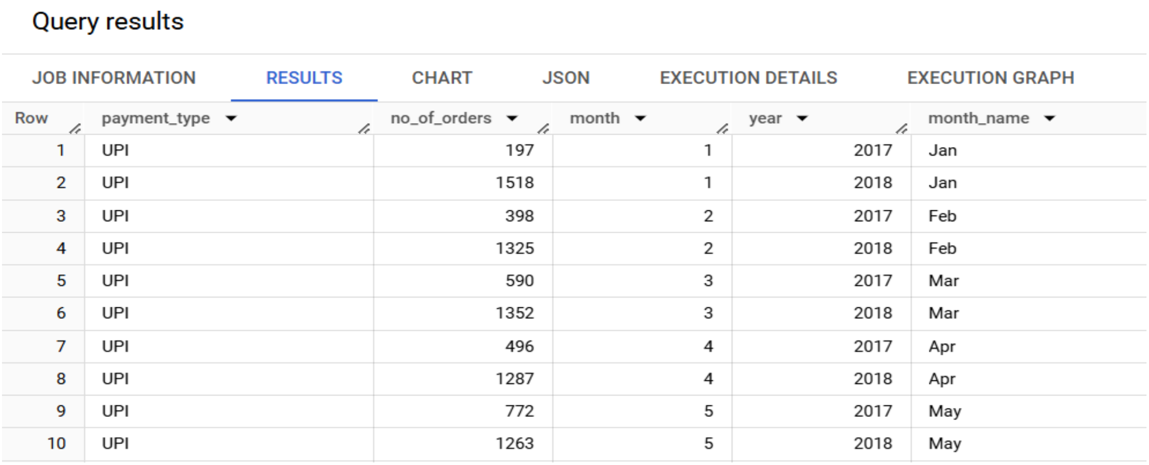
I.Find the month on month no. of orders placed using different payment types.

Query:

```
SELECT

p.payment_type,

count(o.order_id) as no_of_orders,

extract(month from o.order_purchase_timestamp) as month,

extract(year from o.order_purchase_timestamp) as year,

format_datetime('%b', o.order_purchase_timestamp) as
month_name

FROM `target_data.payments` P
```

```
inner join

`target_data.orders` O

on p.order_id = o.order_id

group by 1,3,4,5

order by 1,3,4
```

Result:



Row	payment_type	no_of_orders	month	year	month_name
1	UPI	197	1	2017	Jan
2	UPI	1518	1	2018	Jan
3	UPI	398	2	2017	Feb
4	UPI	1325	2	2018	Feb
5	UPI	590	3	2017	Mar
6	UPI	1352	3	2018	Mar
7	UPI	496	4	2017	Apr
8	UPI	1287	4	2018	Apr
9	UPI	772	5	2017	May
10	UPI	1263	5	2018	May

Explanation:

To understand the trends in payment types, analysis on month-over month count of orders for different payment types is done.

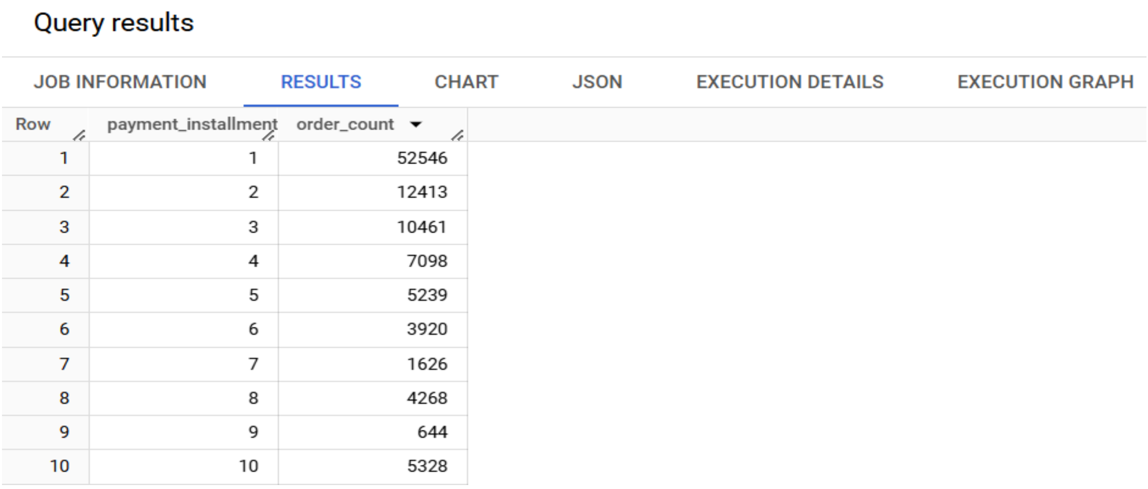
II.Find the no. of orders placed on the basis of the payment installments that have been paid.

Query:

```
SELECT

p.payment_installments,

count(o.order_id) as order_count,

FROM `target_data.payments` P
```

```
inner join

`target_data.orders` O

on p.order_id = o.order_id

where p.payment_installments!=0

group by 1

order by 1
```

Result:



Row	payment_installment	order_count
1	1	52546
2	2	12413
3	3	10461
4	4	7098
5	5	5239
6	6	3920
7	7	1626
8	8	4268
9	9	644
10	10	5328

Explanation:

Status of the payments instalments can be found by joining orders and payments tables.