<u>CASE STUDY: TARGET CORPORATION</u>

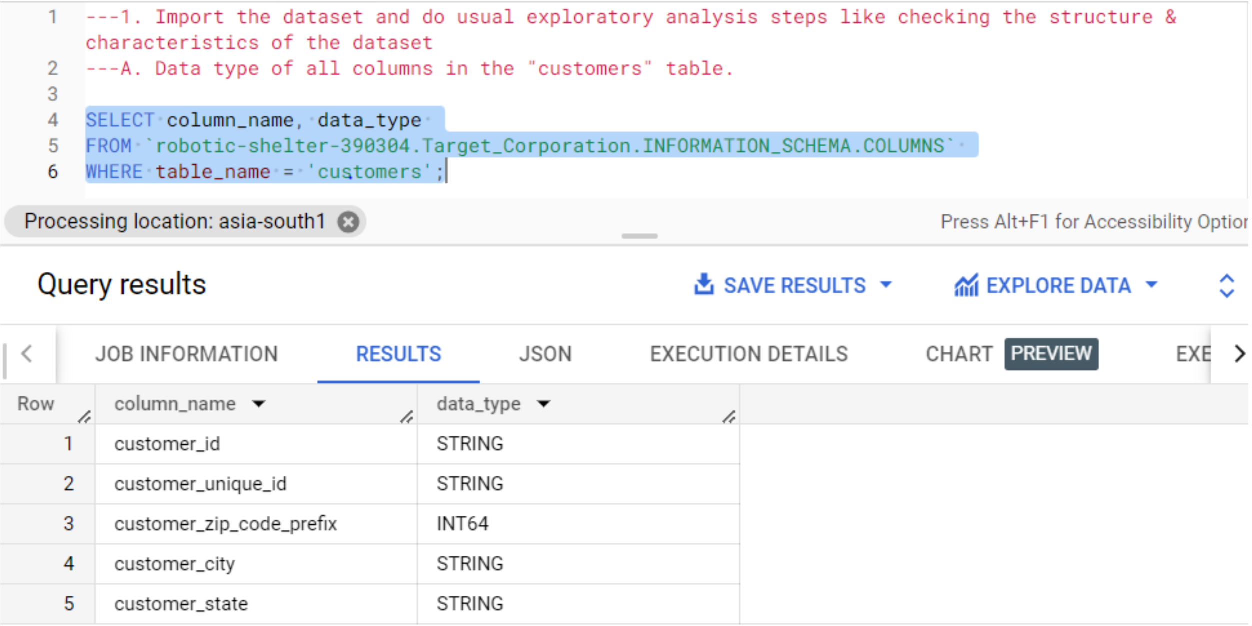
<u>Import the dataset and do usual exploratory analysis steps like checking the structure & characteristics of the dataset:</u>

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

- <u>Query</u>

```
SELECT column_name, data_type
FROM `robotic-shelter-
390304.Target_Corporation.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'customers';
```

- <u>Screenshot of Output</u>



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

SELECT column_name, data_type
FROM `robotic-shelter-390304.Target_Corporation.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'customers';
```

Processing location: asia-south1

Press Alt+F1 for Accessibility Optior

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

- <u>INSIGHTS</u>

  - ➢ By using this query, we can display the data type of each column present in the "customers" table.

- <u>Recommendations</u>

  - ➢ SQL data types can be broadly divided into the following categories.

1. Numeric data types such as: INT, TINYINT, BIGINT, FLOAT, REAL, etc.
2. Date and Time data types such as: DATE, TIME, DATETIME, etc.
3. Character and String data types such as: CHAR, VARCHAR, TEXT, etc.
4. Unicode character string data types such as: NCHAR, NVARCHAR, NTEXT, etc.
5. Binary data types such as: BINARY, VARBINARY, etc.
6. Miscellaneous data types - CLOB, BLOB, XML, CURSOR, TABLE, etc.

- Assumptions

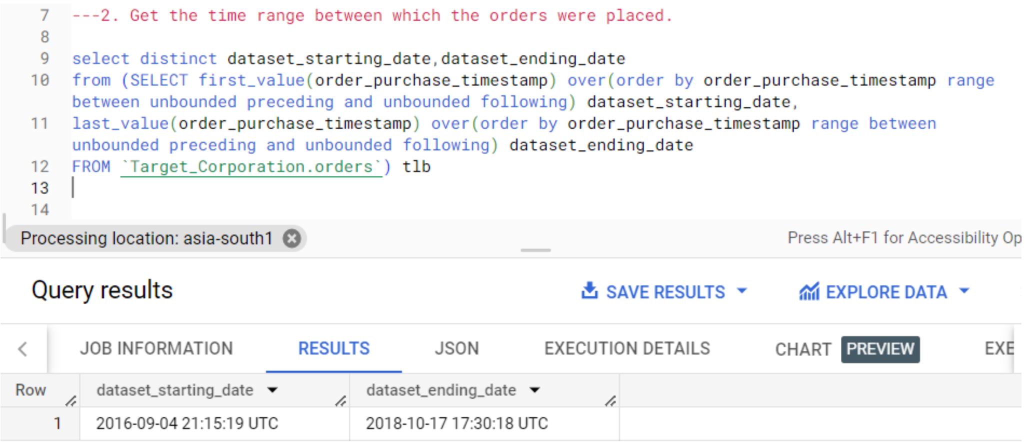
  ➢ By changing the table name in this query, we can display the data type of each column present in the other table also.

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

- Query

```
select distinct dataset_starting_date,dataset_ending_date
from (SELECT first_value(order_purchase_timestamp) over(order by
order_purchase_timestamp range between unbounded preceding and
unbounded following) dataset_starting_date,
last_value(order_purchase_timestamp) over(order by
order_purchase_timestamp range between unbounded preceding and
unbounded following) dataset_ending_date
FROM `Target_Corporation.orders`) tlb
```

- Screenshot of Output



```
---2. Get the time range between which the orders were placed.

select distinct dataset_starting_date,dataset_ending_date
from (SELECT first_value(order_purchase_timestamp) over(order by order_purchase_timestamp range
between unbounded preceding and unbounded following) dataset_starting_date,
last_value(order_purchase_timestamp) over(order by order_purchase_timestamp range between
unbounded preceding and unbounded following) dataset_ending_date
FROM `Target_Corporation.orders`) tlb
```

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

- Insights

➢ In this query, we can get the date & time when the first and last orders in our dataset were placed.

- Recommendation

➢ In this query, we used first_value and last_value window function to get the first and last value of the order_purchase_timestamp.

1. order_purchase_timestamp: - Timestamp of the purchase.
2. first_value(order_purchase_timestamp) over(order by order_purchase_timestamp range between unbounded preceding and unbounded following) dataset_starting_date: - this will provide first value in order_purchase_timestamp.
3. last_value(order_purchase_timestamp) over(order by order_purchase_timestamp range between unbounded preceding and unbounded following) dataset_ending_date: - this will provide last value in order_purchase_timestamp.

- Assumption

➢ if the dataset update with new values, the query will show updated date & time of last order placed.

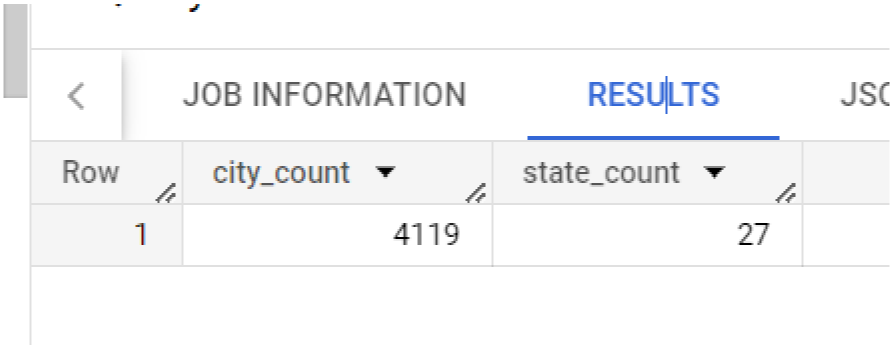
➢ By making minor changes in query like table name and timestamp of other dataset, we can get the date & time when the first and last orders in another dataset also.

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

- Query

```
select ct.city_count,st.state_count
from (select  count(customer_city) as city_count, row_number() over ()
as rows_number
from (Select customer_city
from `Target_Corporation.customers`  c
left join `Target_Corporation.orders`  o
on c.customer_id = o.customer_id
where o.customer_id is not null
group by customer_city
order by customer_city)) ct
join (select count(customer_state) as state_count, row_number() over
() as rows_number
from (Select customer_state
from `Target_Corporation.customers` c
left join `Target_Corporation.orders` o
on c.customer_id = o.customer_id
where o.customer_id is not null
group by customer_state
order by customer_state)) st
on ct.rows_number = st.rows_number;
```

- Screenshot of Output



Row	city_count	state_count
1	4119	27

- Insights

  - We can count the number of unique cities and states present in our dataset.

- Recommendation

  - We join customer and order table by left join to find customers who ordered from Target Corporation. Then group by city then find the count of cities.
  - In similar way, we join customer and order table by left join to find customers who ordered from Target Corporation. Then group by state then find the count of states.
  - At last, we join the tables to display both the result as same table.

- Assumption
  - ➢ In this similar way we can find the count the Cities and States of the customers who ordered for other datasets.

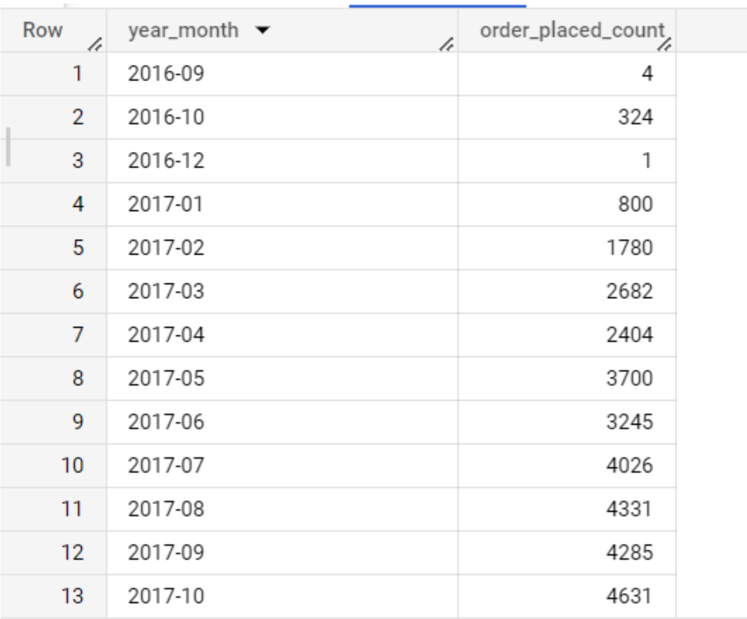
In-depth Exploration:

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

```
with yearmonth as
(SELECT order_id, customer_id,
FORMAT_TIMESTAMP("%Y-%m",order_purchase_timestamp) as year_month
FROM `Target_Corporation.orders`
order by order_purchase_timestamp)

select year_month, count(*) as order_placed_count
from yearmonth
group by year_month
order by year_month
```

- Screenshot of Output



Row	year_month	order_placed_count
1	2016-09	4
2	2016-10	324
3	2016-12	1
4	2017-01	800
5	2017-02	1780
6	2017-03	2682
7	2017-04	2404
8	2017-05	3700
9	2017-06	3245
10	2017-07	4026
11	2017-08	4331
12	2017-09	4285
13	2017-10	4631

- Insights

  - We can find out if no. of orders placed has increased gradually in each month, over the past years.

- Recommendation

  - FORMAT_TIMESTAMP("%Y-%m", order_purchase_timestamp) as year_month: - by using this query be extract Year and month part from purchase timestamp. Which comes out 2016-09 to 2018-10.
  - Make this as Common Table Expression(cte).
  - Then using this cte make group by year_month, then count the order placed.

- Assumption

  - We can see in our data order gradually increases with year starts.
  - We can see in our data order gradually decrease with year ends.

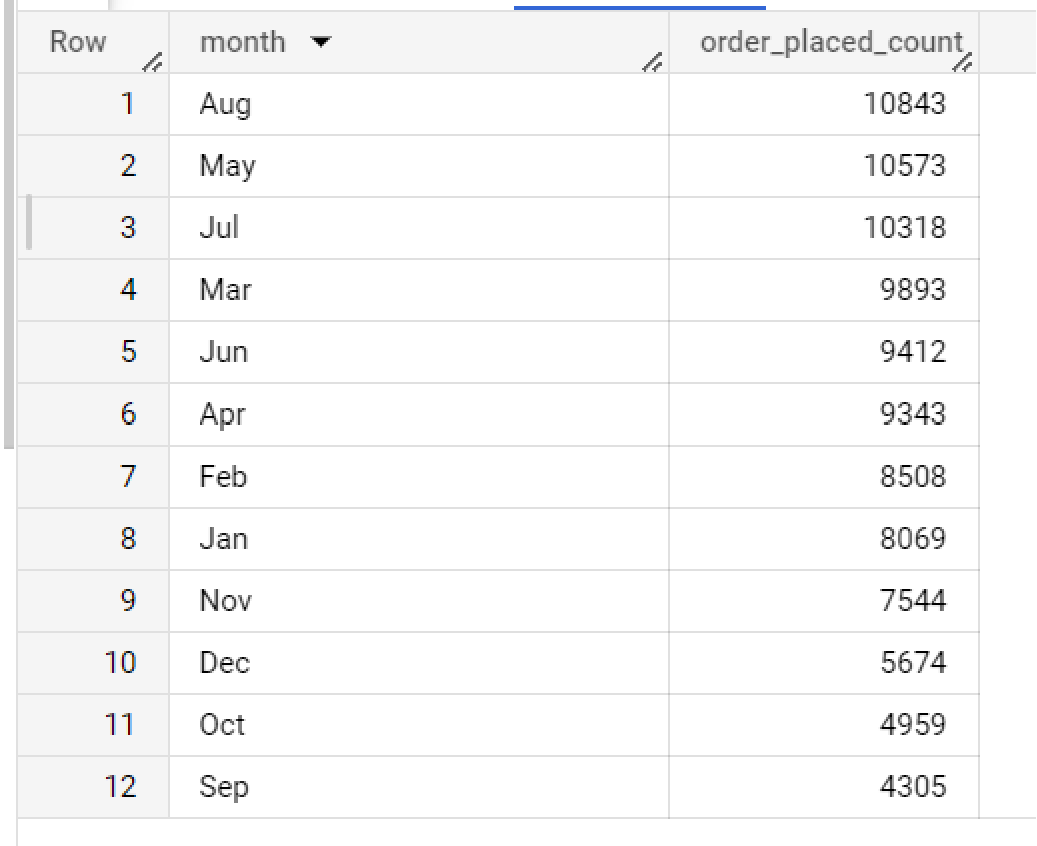
2. Can we see some kind of monthly seasonality in terms of the no. of orders being placed?

- Query

```
with yearmonth as
(SELECT order_id, customer_id,
FORMAT_TIMESTAMP("%h",order_purchase_timestamp) as month
FROM `Target_Corporation.orders`
order by order_purchase_timestamp)

select month, count(*) as order_placed_count
from yearmonth
group by month
order by order_placed_count desc;
```

- Screenshot of Output



Row	month	order_placed_count
1	Aug	10843
2	May	10573
3	Jul	10318
4	Mar	9893
5	Jun	9412
6	Apr	9343
7	Feb	8508
8	Jan	8069
9	Nov	7544
10	Dec	5674
11	Oct	4959
12	Sep	4305

- Insights
  - ➢ In the query, we can find out if the no. of orders placed are at peak during certain months.
- Recommendation
  - ➢ FORMAT_TIMESTAMP("%h", order_purchase_timestamp) as month: - by using this query be extract month part from purchase timestamp.
  - ➢ Make this as Common Table Expression(cte).
  - ➢ Then using this cte make group by month, then count the order placed in that month over years.
  - ➢ order by order_placed_count desc: - We order by order placed in descending order, to find the highest ordered month first then second and so on.
- Assumption
  - ➢ August, May and July have the highest ordered month.
  - ➢ December, October and September have lowest ordered month.

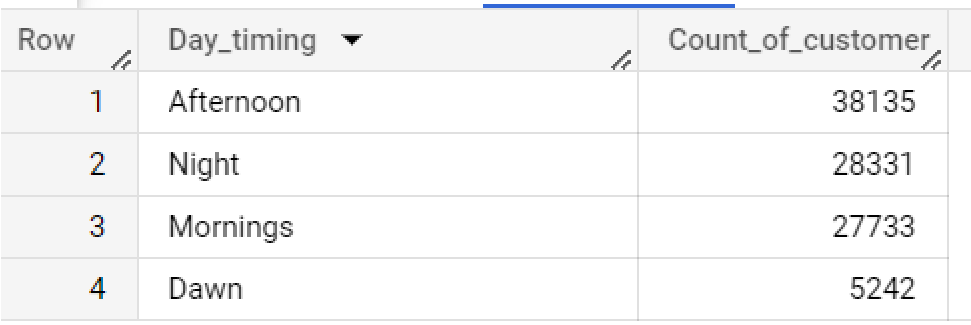
3. During what time of the day, do the Brazilian customers mostly place their orders? (Dawn, Morning, Afternoon or Night)

- Query

```
with hourday as
(SELECT customer_id, order_id, case
when cast(hour as int) between 0 and 6
then 'Dawn'
when  cast(hour as int) between 7 and 12
then 'Mornings'
when  cast(hour as int) between 13 and 18
then 'Afternoon'
else 'Night'
end as Day_timing
FROM (SELECT customer_id, order_id,
FORMAT_TIMESTAMP("%H",order_purchase_timestamp) as hour
FROM `Target_Corporation.orders`) tbl)

select Day_timing, count(customer_id) as Count_of_customer
from hourday
group by Day_timing
order by Count_of_customer desc;
```

- Screenshot of Output



Row	Day_timing	Count_of_customer
1	Afternoon	38135
2	Night	28331
3	Mornings	27733
4	Dawn	5242

- Insights
  - ➢ In this query, we can categorize the hours of a day into the given time brackets/ intervals and find out during which intervals the Brazilian customers usually order the most.

- Time intervals: -
    1. **1.** 0-6 hrs: Dawn
    2. **2.** 7-12 hrs: Mornings
    3. **3.** 13-18 hrs: Afternoon
    4. **4.** 19-23 hrs: Night

- Recommendation

    - FORMAT_TIMESTAMP("%H",order_purchase_timestamp) as hour: - Using this query we extract hour from purchase timestamp and make that as subquery.
    - Then using case when, we specify the Dawn, Mornings, Afternoon and Night, put it in column Day_timing.
    - Make is whole as cte.
    - Then using cte we group by Day_timing and count the order.

- Assumption

    - Most of the orders comes at Afternoon Day time.
    - Least of the orders comes at Dawn Day time.

Evolution of E-commerce orders in the Brazil region:

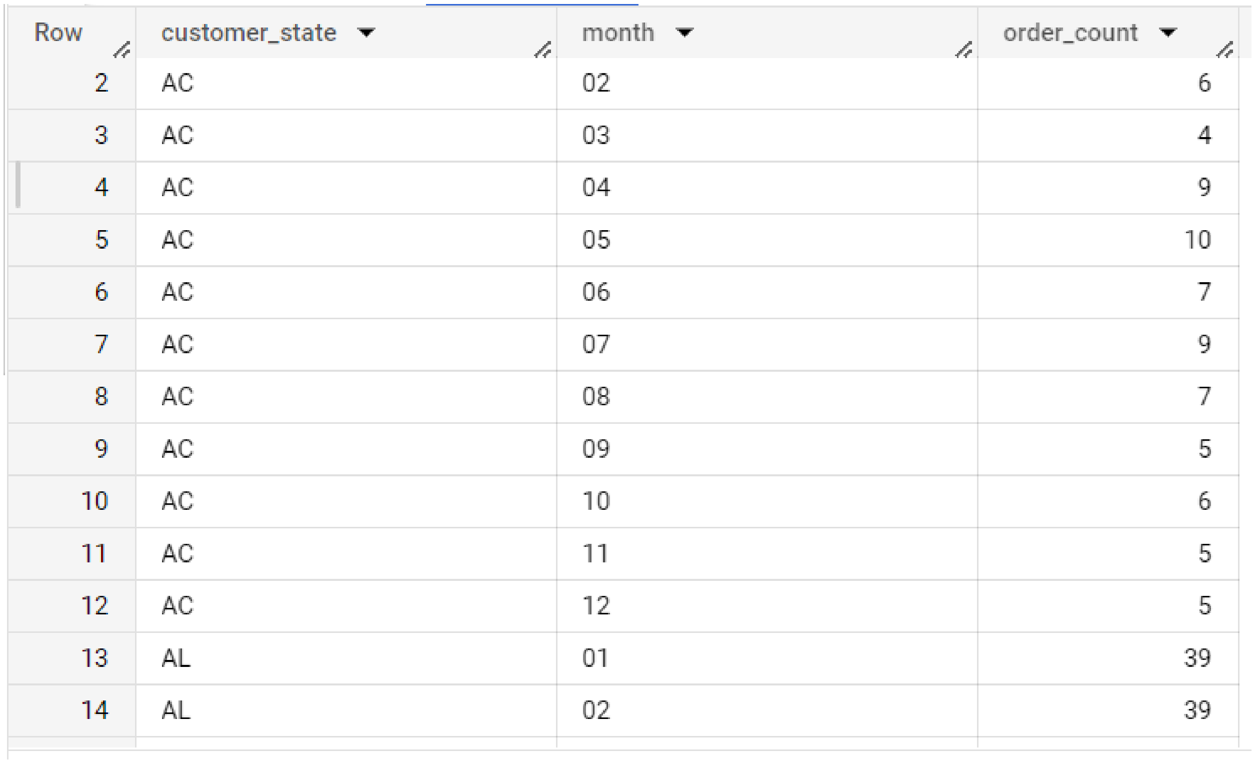
1. Get the month-on-month no. of orders placed in each state.

- Query

```
with city_month as
(SELECT o.customer_id, c.customer_state,
FORMAT_TIMESTAMP("%m",o.order_purchase_timestamp) as month
from `Target_Corporation.orders` o
join `Target_Corporation.customers` c
on o.customer_id = c.customer_id)

select customer_state, month, count(customer_id) as order_count
from city_month
group by customer_state,month
order by customer_state,month
```

- Screenshot of Output



Row	customer_state	month	order_count
2	AC	02	6
3	AC	03	4
4	AC	04	9
5	AC	05	10
6	AC	06	7
7	AC	07	9
8	AC	08	7
9	AC	09	5
10	AC	10	6
11	AC	11	5
12	AC	12	5
13	AL	01	39
14	AL	02	39

- Insights

  - ➢ In this query, we can get the no. of orders placed in each state, in each month by our customers.

- Recommendation

  - ➢ FORMAT_TIMESTAMP("%m", o.order_purchase_timestamp) as month: - Extract month from purchase timestamp.
  - ➢ Join orders table and customer table.
  - ➢ Make this as cte (Common table Expression).
  - ➢ Use cte and group by customer_state and month part.
  - ➢ Then count customer_id.
  - ➢ Order by customer_state and month.

- Assumption

  - ➢ We can observe in the screenshot: - state name, month and order count.

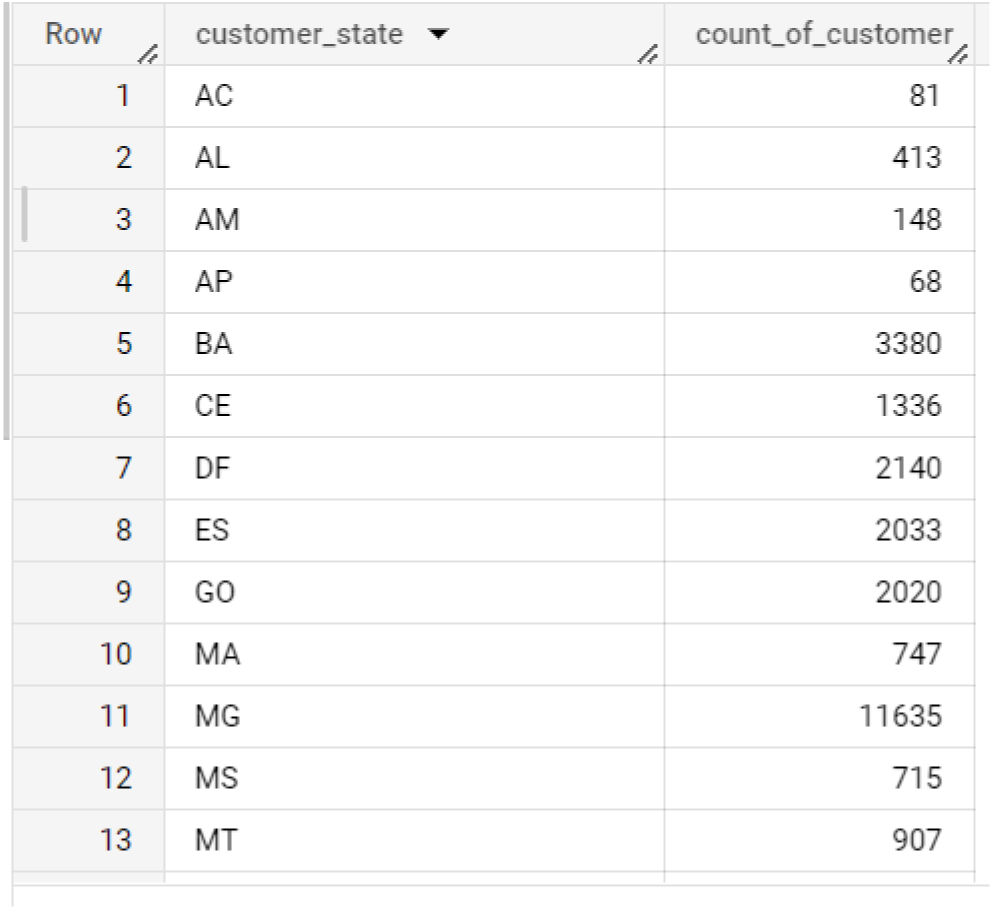
- State: - SP have highest no. of orders, then RJ State, then MG States and so on

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

- Query

```
SELECT customer_state, count(distinct customer_id) as
count_of_customer
from `Target_Corporation.customers`
group by customer_state
order by customer_state
```

- Screenshot of Output



Row	customer_state	count_of_customer
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
11	MG	11635
12	MS	715
13	MT	907

- Insights

  - In this query, we can get the no. of unique customers present in each state.

- Recommendation

  - Group by customer_state in table customers.
  - Count distinct customers_id.

- Assumption

  - SP state have highest no. of customer, then RJ have second highest customer and the MG state.
  - **Number of customers is directly proposal to Number order.**

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

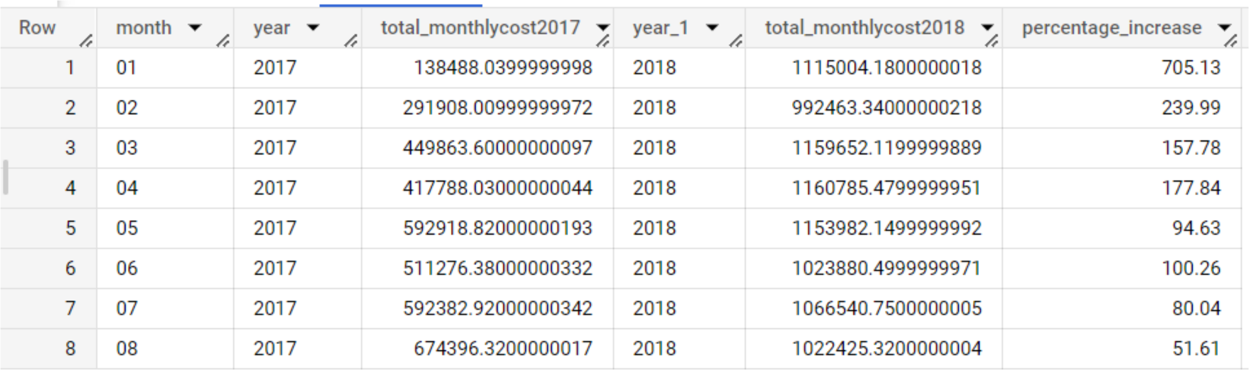
1. Get the % increase in the cost of orders from year 2017 to 2018 (include months between Jan to Aug only).

- Query

```
with monthyear as
(select o.order_id,FORMAT_TIMESTAMP("%m",o.order_purchase_timestamp)
as month,
FORMAT_TIMESTAMP("%Y",o.order_purchase_timestamp) as year,
payment_value
from `Target_Corporation.orders` o
join `Target_Corporation.payments` p
on o.order_id = p.order_id)

select tlb1.month, tlb1.year, tlb1.total_monthlycost2017, tlb2.year,
tlb2.total_monthlycost2018,
round((((tlb2.total_monthlycost2018-
tlb1.total_monthlycost2017)/tlb1.total_monthlycost2017)*100),2) as
percentage_increase,
from (select monthyear.month,monthyear.year, sum(payment_value) as
total_monthlycost2017
from monthyear
where year = '2017' and cast(monthyear.month as int) between 01 and 08
group by monthyear.month, monthyear.year
order by monthyear.month) tlb1
join (select monthyear.month,monthyear.year, sum(payment_value) as
total_monthlycost2018
from monthyear
where year = '2018' and cast(monthyear.month as int) between 01 and 08
group by monthyear.month, monthyear.year
order by monthyear.month) tlb2
on tlb1.month = tlb2.month
order by tlb1.month;
```

- Screenshot of Output



Row	month	year	total_monthlycost2017	year_1	total_monthlycost2018	percentage_increase
1	01	2017	138488.0399999998	2018	1115004.1800000018	705.13
2	02	2017	291908.00999999972	2018	992463.3400000218	239.99
3	03	2017	449863.60000000097	2018	1159652.1199999889	157.78
4	04	2017	417788.03000000044	2018	1160785.4799999951	177.84
5	05	2017	592918.82000000193	2018	1153982.1499999992	94.63
6	06	2017	511276.38000000332	2018	1023880.4999999971	100.26
7	07	2017	592382.92000000342	2018	1066540.7500000005	80.04
8	08	2017	674396.3200000017	2018	1022425.3200000004	51.61

- Insights

  - Percentage_ increase between months(January - August) over years.

- Recommendation

  - FORMAT_TIMESTAMP("%m",o.order_purchase_timestamp) as month: - Extract month from purchase timestamp.
  - FORMAT_TIMESTAMP("%Y",o.order_purchase_timestamp) as year: - Extract year from purchase time stamp.
  - Make this as cte.
  - In one subquery find sum of months payment over year 2017.
  - And in one subquery find sum of months payment over year 2018.
  - Join both sub query.

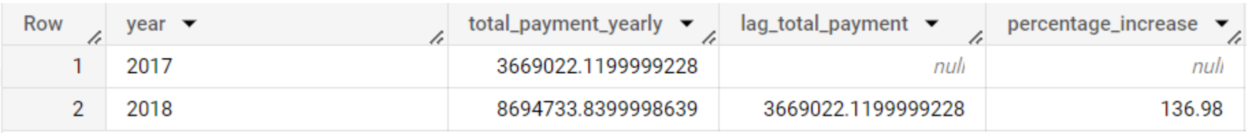
- ➢ And find percentage increase over months using formula: - round((((tlb2.total_monthlycost2018-tlb1.total_monthlycost2017)/tlb1.total_monthlycost2017)*100),2) as percentage_increase.
- ➢ Round function helps to round off 2 digits after point(.).
- ➢ Between Month January to August.

- <u>Assumption</u>

  - ➢ Highest percentage increase over month is in January: - 705.13%
  - ➢ Lowest percentage increase over month is in August: - 51.61%
  - ➢ We can find percentage increase over years.

```
with cte as(
select FORMAT_TIMESTAMP("%Y",o.order_purchase_timestamp) as year,
o.order_id,FORMAT_TIMESTAMP("%m",o.order_purchase_timestamp) as month,
payment_value,
from `Target_Corporation.orders` o
join `Target_Corporation.payments` p
on o.order_id = p.order_id)
,cte2 as(
select cte.year, sum(payment_value) total_payment_yearly
from cte
where cast(cte.month as int) between 01 and 08
group by cte.year
order by cte.year)
,cte3 as
(select *, lag(cte2.total_payment_yearly) over(order by cte2.year) as
lag_total_payment
from cte2)

select *,round((((cte3.total_payment_yearly-
cte3.lag_total_payment)/cte3.lag_total_payment)*100),2) as
percentage_increase
from cte3
order by cte3.year;
```



Row	year	total_payment_yearly	lag_total_payment	percentage_increase
1	2017	3669022.1199999228	null	null
2	2018	8694733.8399998639	3669022.1199999228	136.98

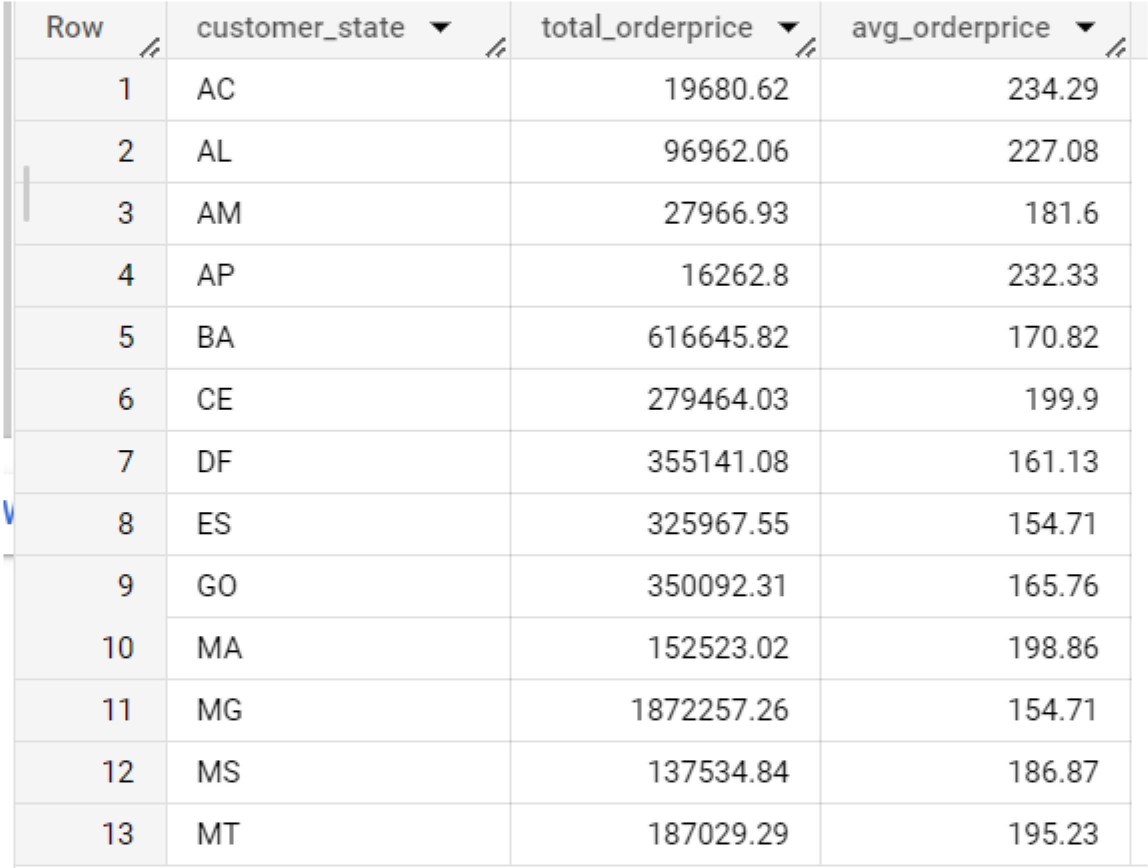
- ➢ Percentage Increase over Year 2017-2018 is 136.98%.

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

- Query

```
SELECT distinct customer_state,
sum(payment_value) over(partition by customer_state order by
customer_state) as total_orderprice,
round(avg(payment_value) over(partition by customer_state order by
customer_state),2) as avg_orderprice
from `Target_Corporation.customers` c
join `Target_Corporation.orders` o
on c.customer_id = o.customer_id
join `Target_Corporation.payments` p
on o.order_id = p.order_id
order by customer_state;
```

- Screenshot of Output



Row	customer_state	total_orderprice	avg_orderprice
1	AC	19680.62	234.29
2	AL	96962.06	227.08
3	AM	27966.93	181.6
4	AP	16262.8	232.33
5	BA	616645.82	170.82
6	CE	279464.03	199.9
7	DF	355141.08	161.13
8	ES	325967.55	154.71
9	GO	350092.31	165.76
10	MA	152523.02	198.86
11	MG	1872257.26	154.71
12	MS	137534.84	186.87
13	MT	187029.29	195.23

- Insights

  - In this query, we can fetch the total price and the average price of orders for each state.

- Recommendation

  - round(avg(payment_value) over(partition by customer_state order by customer_state),2) as avg_orderprice: - using this window function of average of payment_value of states
  - We join three Tables customers, orders and payments.
  - Customer table helps in finding state.
  - Order table helps in order details.
  - Payments table helps in finding payment_value.

- Assumption

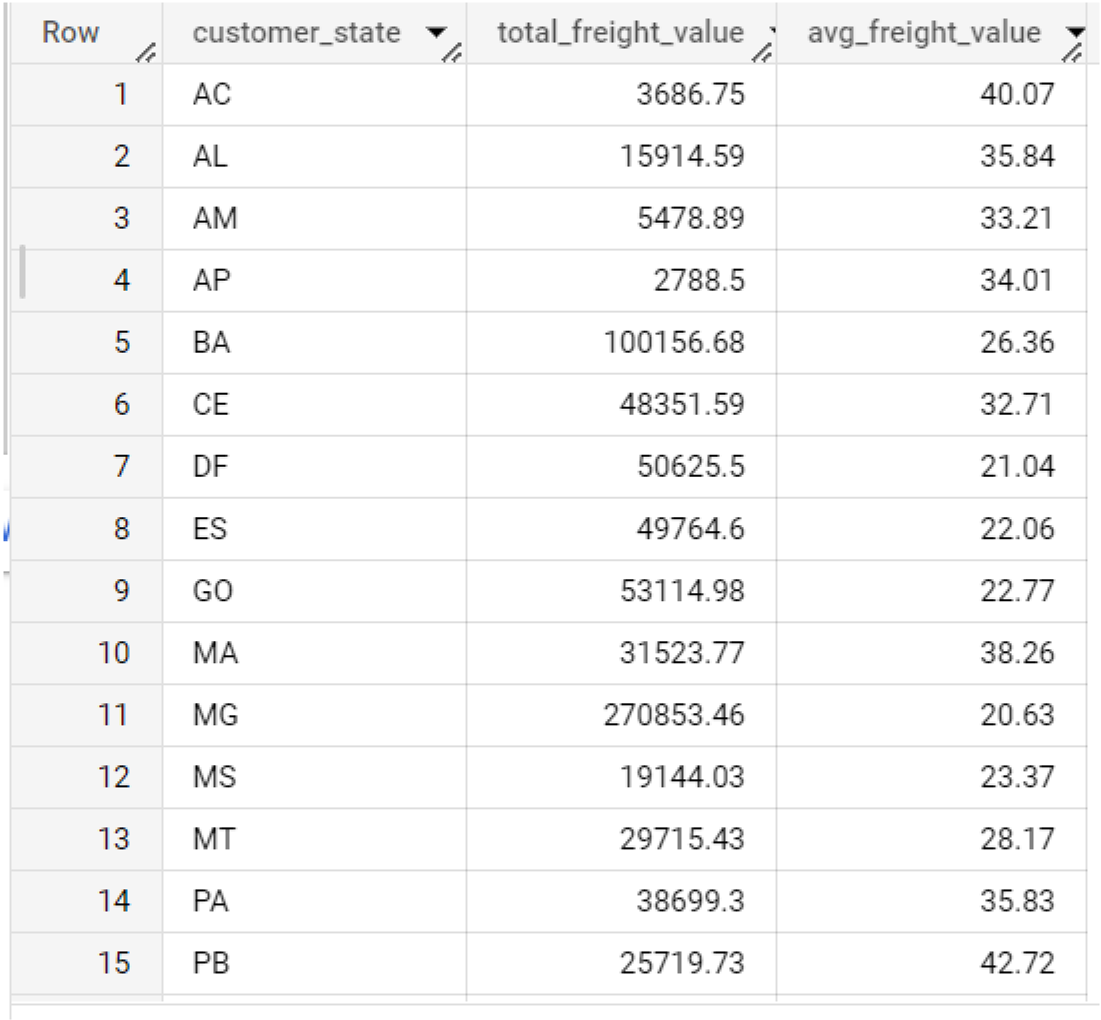
  - PB State have highest average price of order: - 248.33
  - Highest total_orderprice SP State: - 5998226.96
  - SP State have lowest average price of order: - 137.5
  - Lowest total_orderprice AP State: - 16262.8
  - We see SP State has high total sale but low Average

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

- Query

```
SELECT distinct customer_state,
sum(freight_value) over(partition by customer_state order by
customer_state) as total_freight_value,
round(avg(freight_value) over(partition by customer_state order by
customer_state),2) as avg_freight_value
from `Target_Corporation.customers` c
join `Target_Corporation.orders` o
on c.customer_id = o.customer_id
join `Target_Corporation.order_items` oi
on o.order_id = oi.order_id
order by customer_state;
```

- Screenshot of Output



Row	customer_state	total_freight_value	avg_freight_value
1	AC	3686.75	40.07
2	AL	15914.59	35.84
3	AM	5478.89	33.21
4	AP	2788.5	34.01
5	BA	100156.68	26.36
6	CE	48351.59	32.71
7	DF	50625.5	21.04
8	ES	49764.6	22.06
9	GO	53114.98	22.77
10	MA	31523.77	38.26
11	MG	270853.46	20.63
12	MS	19144.03	23.37
13	MT	29715.43	28.17
14	PA	38699.3	35.83
15	PB	25719.73	42.72

- Insights

  - In this Query, we can fetch the total freight value and the average freight value of orders for each state.

- Recommendation

  - round(avg(freight_value) over(partition by customer_state order by customer_state),2) as avg_freight_value
  - We join three Tables customers, orders and order_item.

- Customer table helps in finding state.
- Order table helps in order details.
- Payments table helps in finding freight_value.

- Assumption

  - RR State have highest average Freight of order: - 42.98
  - Highest Total Freight value: - SP State 718723.07
  - SP State have lowest average Freight of order: - 15.15
  - Lowest Total Freight value: - RR State   2235.19

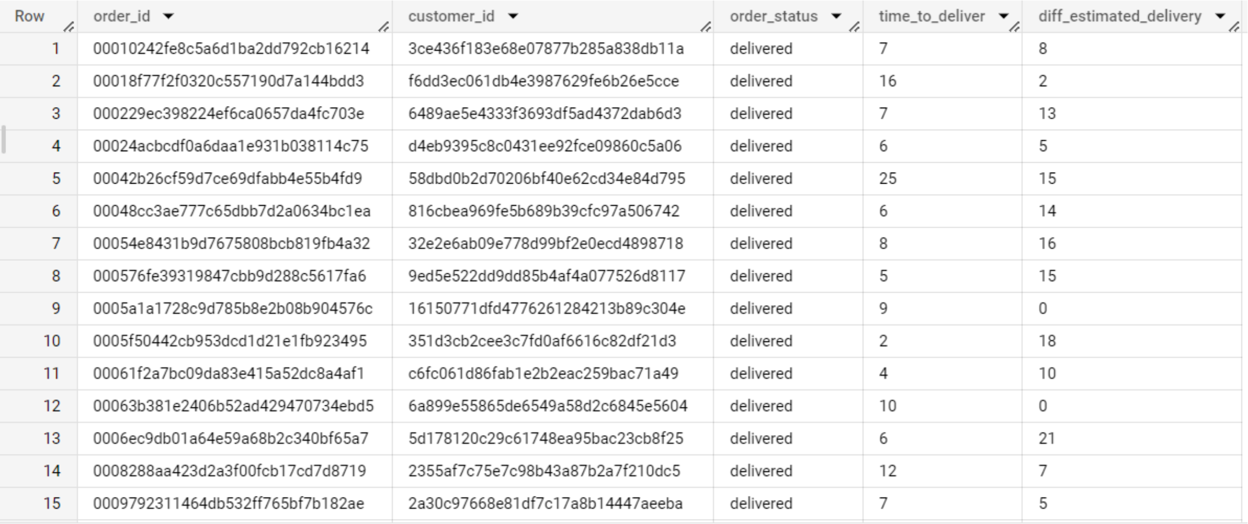
Analysis based on sales, freight and delivery time.

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.

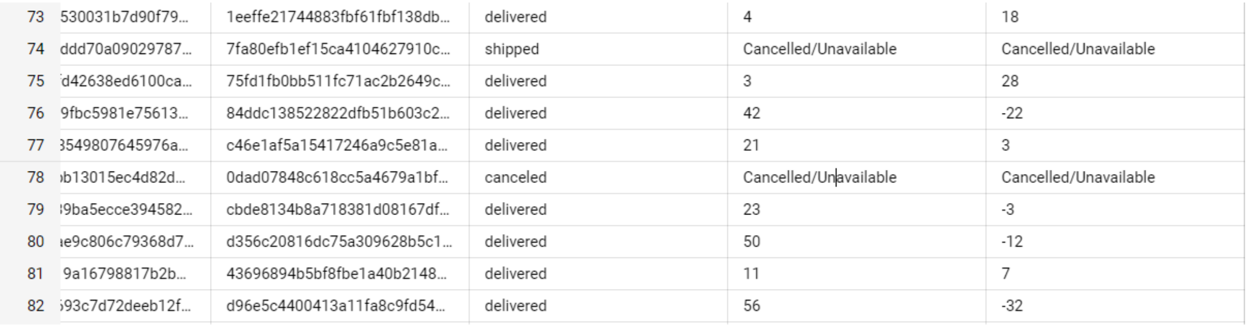
- Query

```
SELECT order_id, customer_id,order_status,
case
when order_delivered_customer_date is null
then 'Cancelled/Unavailable'
else cast(date_diff(order_delivered_customer_date,
order_purchase_timestamp, day) as string)
end as time_to_deliver,
case
when order_delivered_customer_date is null
then 'Cancelled/Unavailable'
else
cast(date_diff(order_estimated_delivery_date,
order_delivered_customer_date, day) as string)
end as diff_estimated_delivery
from `Target_Corporation.orders`
order by order_id
```

- <u>Screenshot of Output</u>



Row	order_id	customer_id	order_status	time_to_deliver	diff_estimated_delivery
1	00010242fe8c5a6d1ba2dd792cb16214	3ce436f183e68e07877b285a838db11a	delivered	7	8
2	00018f77f2f0320c557190d7a144bdd3	f6dd3ec061db4e3987629fe6b26e5cce	delivered	16	2
3	000229ec398224ef6ca0657da4fc703e	6489ae5e4333f3693df5ad4372dab6d3	delivered	7	13
4	00024acbcdf0a6daa1e931b038114c75	d4eb9395c8c0431ee92fce09860c5a06	delivered	6	5
5	00042b26cf59d7ce69dfabb4e55b4fd9	58dbd0b2d70206bf40e62cd34e84d795	delivered	25	15
6	00048cc3ae777c65dbb7d2a0634bc1ea	816cbea969fe5b689b39cfc97a506742	delivered	6	14
7	00054e8431b9d7675808bcb819fb4a32	32e2e6ab09e778d99bf2e0ecd4898718	delivered	8	16
8	000576fe39319847cbb9d288c5617fa6	9ed5e522dd9dd85b4af4a077526d8117	delivered	5	15
9	0005a1a1728c9d785b8e2b08b904576c	16150771dfd4776261284213b89c304e	delivered	9	0
10	0005f50442cb953dcd1d21e1fb923495	351d3cb2cee3c7fd0af6616c82df21d3	delivered	2	18
11	00061f2a7bc09da83e415a52dc8a4af1	c6fc061d86fab1e2b2eac259bac71a49	delivered	4	10
12	00063b381e2406b52ad429470734ebd5	6a899e55865de6549a58d2c6845e5604	delivered	10	0
13	0006ec9db01a64e59a68b2c340bf65a7	5d178120c29c61748ea95bac23cb8f25	delivered	6	21
14	0008288aa423d2a3f00fcb17cd7d8719	2355af7c75e7c98b43a87b2a7f210dc5	delivered	12	7
15	0009792311464db532ff765bf7b182ae	2a30c97668e81df7c17a8b14447aeeba	delivered	7	5



73	530031b7d90f79…	1eeffe21744883fbf61fbf138db…	delivered	4	18
74	ddd70a09029787…	7fa80efb1ef15ca4104627910c…	shipped	Cancelled/Unavailable	Cancelled/Unavailable
75	d42638ed6100ca…	75fd1fb0bb511fc71ac2b2649c…	delivered	3	28
76	9fbc5981e75613…	84ddc138522822dfb51b603c2…	delivered	42	-22
77	549807645976a…	c46e1af5a15417246a9c5e81a…	delivered	21	3
78	b13015ec4d82d…	0dad07848c618cc5a4679a1bf…	canceled	Cancelled/Unavailable	Cancelled/Unavailable
79	9ba5ecce394582…	cbde8134b8a718381d08167df…	delivered	23	-3
80	e9c806c79368d7…	d356c20816dc75a309628b5c1…	delivered	50	-12
81	9a16798817b2b…	43696894b5bf8fbe1a40b2148…	delivered	11	7
82	93c7d72deeb12f…	d96e5c4400413a11fa8c9fd54…	delivered	56	-32

- <u>Insights</u>

    - In this Query, we 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

- <u>Recommendation</u>

    - date_diff(order_delivered_customer_date, order_purchase_timestamp, day):- this function help in find the difference in dates.
    - Case when helps in finding cancelled and Unavailable.

- Assumption

  - time_to_deliver: - Differences in order_delivered_customer_date and order_purchase_timestamp.
  - diff_estimated_delivery: - Differences in order_estimated_delivery_date and order_delivered_customer_date
  - In row 74 order is shipped but never reach the customer.
  - In row 76,79,80 and 82 has negative values then it means order is deliver more days then estimated delivery time.

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

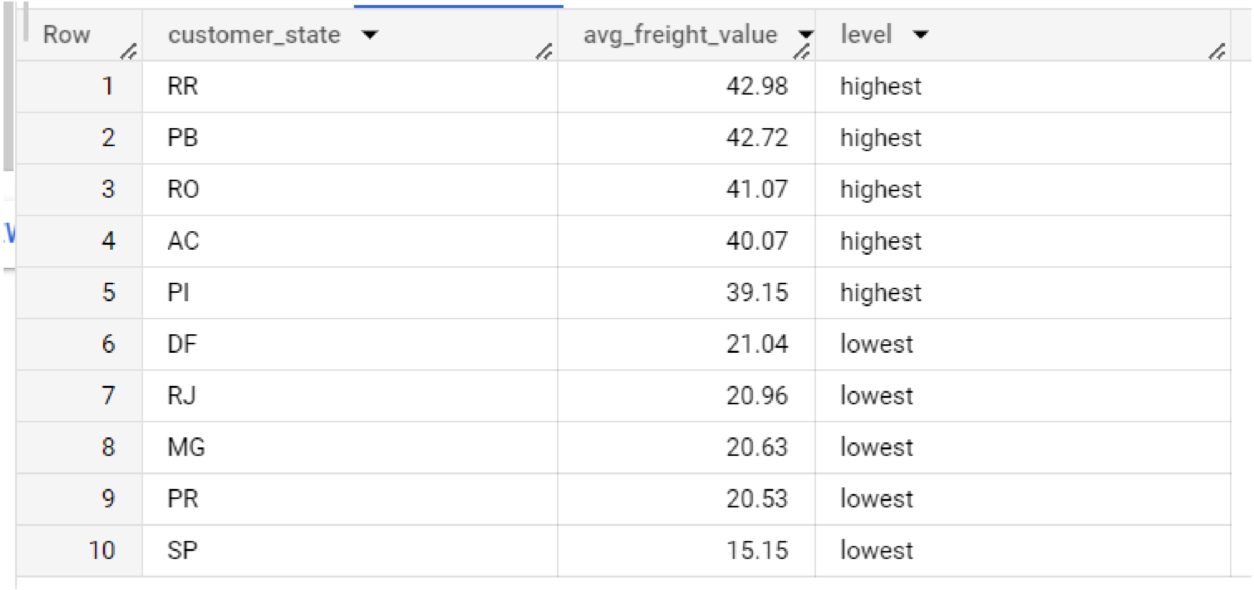
- Query

```
(select customer_state,avg_freight_value,'highest' as level
from(select*,dense_rank() over(order by avg_freight_value desc) as
ranking
from(SELECT distinct customer_state,
round(avg(freight_value) over(partition by customer_state order by
customer_state),2) as avg_freight_value,
from `Target_Corporation.customers` c
join `Target_Corporation.orders` o
on c.customer_id = o.customer_id
join `Target_Corporation.order_items` oi
on o.order_id = oi.order_id
order by avg_freight_value) a
order by ranking) b
where ranking <= 5)

UNION distinct

(select customer_state,avg_freight_value,'lowest' as level
from(select*,dense_rank() over(order by avg_freight_value) as ranking
from(SELECT distinct customer_state,
round(avg(freight_value) over(partition by customer_state order by
customer_state),2) as avg_freight_value,
from `Target_Corporation.customers` c
join `Target_Corporation.orders` o
on c.customer_id = o.customer_id
join `Target_Corporation.order_items` oi
on o.order_id = oi.order_id
order by avg_freight_value) a
order by ranking) b
where ranking <= 5)
order by avg_freight_value desc;
```

- Screenshot of Output



Row	customer_state	avg_freight_value	level
1	RR	42.98	highest
2	PB	42.72	highest
3	RO	41.07	highest
4	AC	40.07	highest
5	PI	39.15	highest
6	DF	21.04	lowest
7	RJ	20.96	lowest
8	MG	20.63	lowest
9	PR	20.53	lowest
10	SP	15.15	lowest

- Insights

  - ➢ In This Query, we can find the top 5 & the bottom 5 states arranged in increasing order of the average freight value.

- Recommendation

  - ➢ round(avg(freight_value) over(partition by customer_state order by customer_state),2) as avg_freight_value: - find the average of freight_value.
  - ➢ Make this in subquery and the put dense_rank() on subquery to find top 5 highest avg freight value. dense_rank() over(order by avg_freight_value desc) as ranking.
  - ➢ In Similar way we find top 5 lowest avg freight value. dense_rank() over(order by avg_freight_value) as ranking.
  - ➢ And at last union both of them.

- Assumption

  - ➢ 1-5 top 5 highest avenge freight value states.

- RR State have highest avenged freight value state.
- 6-10 top 5 lowest avenge freight value states
- SP State have lowest avenged freight value state.

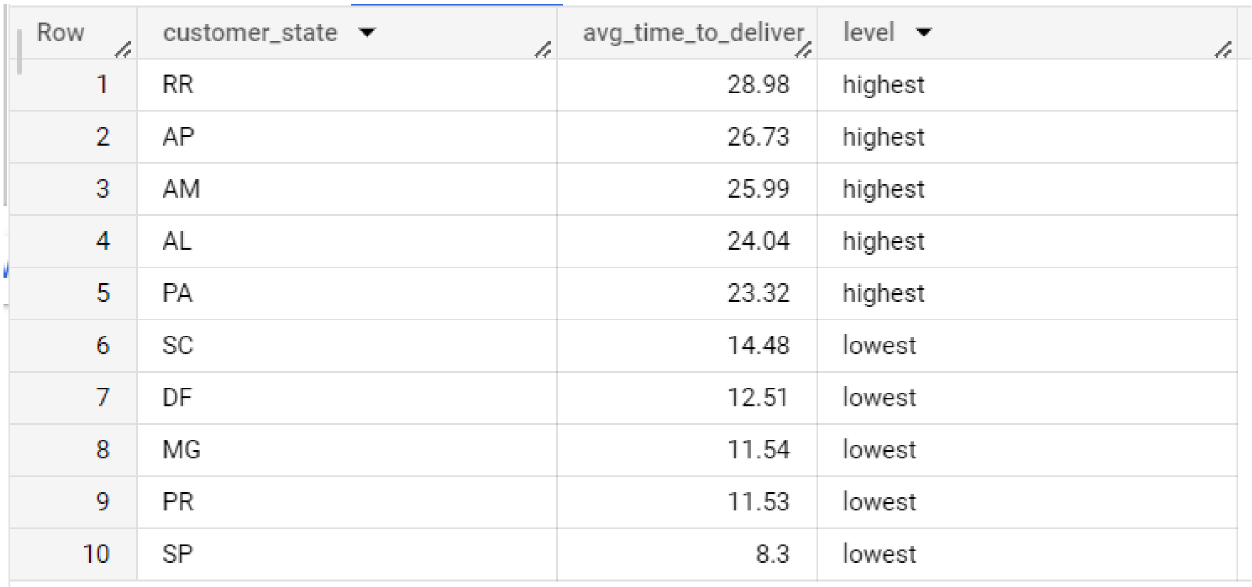
3. Find out the top 5 states with the highest & lowest average delivery time.

- Query

```
with cte as
(select distinct customer_state, round(avg(c.time_to_deliver),2) as
avg_time_to_deliver
from (SELECT customer_state, date_diff(order_delivered_customer_date,
order_purchase_timestamp, day) as time_to_deliver
from `Target_Corporation.customers` c
join `Target_Corporation.orders` o
on c.customer_id = o.customer_id) c
group by customer_state)

(select customer_state, avg_time_to_deliver,'highest' as level
from(select customer_state, cte.avg_time_to_deliver,dense_rank()
over(order by cte.avg_time_to_deliver desc) as ranking
from cte)
where ranking <= 5)
UNION ALL
(select customer_state, avg_time_to_deliver,'lowest' as level
from(select customer_state, cte.avg_time_to_deliver,dense_rank()
over(order by cte.avg_time_to_deliver) as ranking
from cte)
where ranking <= 5)
order by avg_time_to_deliver desc
```

- Screenshot of Output



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

- Insights

  - In this Query, we can find the top 5 & the bottom 5 states arranged in increasing order of the average delivery time.

- Recommendation

  - date_diff(order_delivered_customer_date, order_purchase_timestamp, day) as time_to_deliver: - date difference to find delivery time.
  - Avg() on delivery time of states.
  - Make this in cte and the put dense_rank() on cte to find top 5 highest avg freight value. dense_rank() over(order by cte.avg_time_to_deliver desc) as ranking
  - In Similar way we find top 5 lowest avg freight value. dense_rank() over(order by cte.avg_time_to_deliver) as ranking
  - And at last union both of them.

- Assumption

  - 1-5 top 5 highest avenge time to deliver states.
  - RR State have Highest delivery time average.
  - 6-10 top 5 lowest avenge time to deliver states
  - SP State have lowest average Delivery time.

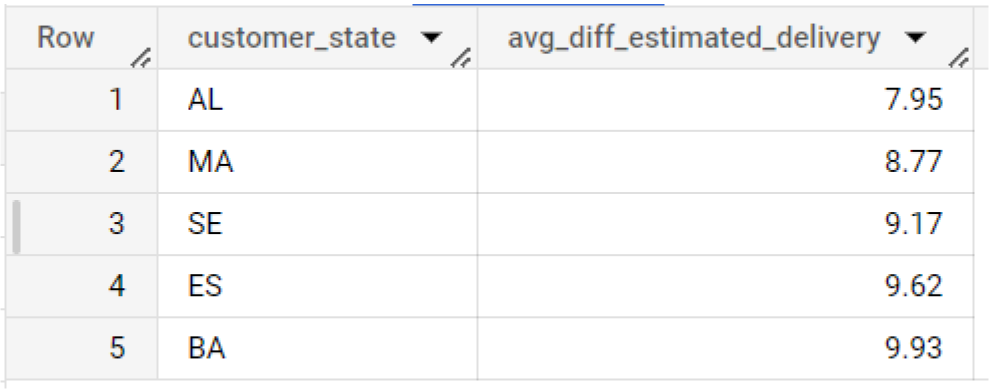
4. Find out the top 5 states where the order delivery is really fast as compared to the estimated date of delivery.

- Query

```
with cte5 as
(select customer_state, round(avg(diff_estimated_delivery),2) as
avg_diff_estimated_delivery
from (select customer_state, date_diff(order_estimated_delivery_date,
order_delivered_customer_date, day) as diff_estimated_delivery
from `Target_Corporation.customers` c
join `Target_Corporation.orders` o
on c.customer_id = o.customer_id
where order_status = 'delivered') f
group by customer_state)

select customer_state, avg_diff_estimated_delivery,
from(select customer_state, avg_diff_estimated_delivery,dense_rank()
over(order by avg_diff_estimated_delivery) as ranking
from cte5)
where ranking <= 5
order by avg_diff_estimated_delivery;
```

- Screenshot of Output



Row	customer_state	avg_diff_estimated_delivery
1	AL	7.95
2	MA	8.77
3	SE	9.17
4	ES	9.62
5	BA	9.93

- Insights

  - In this Query, we find top 5 states where the order delivery is really fast as compared to the estimated date of delivery.
  - We can use the difference between the averages of actual & estimated delivery date to figure out how fast the delivery was for each state.

- Recommendation

  - date_diff(order_estimated_delivery_date, order_delivered_customer_date, day) as diff_estimated_delivery: - Date Difference to find estimated date of delivery.
  - Join two table customers and orders.
  - Where clause to filter delivered ordered
  - Make it as cte.
  - Using cte us dense_rank() we find the top 5 states where the order delivery is really fast as compared to the estimated date of delivery.

- Assumption

  - AL State fastest order delivery

Analysis based on the payments:

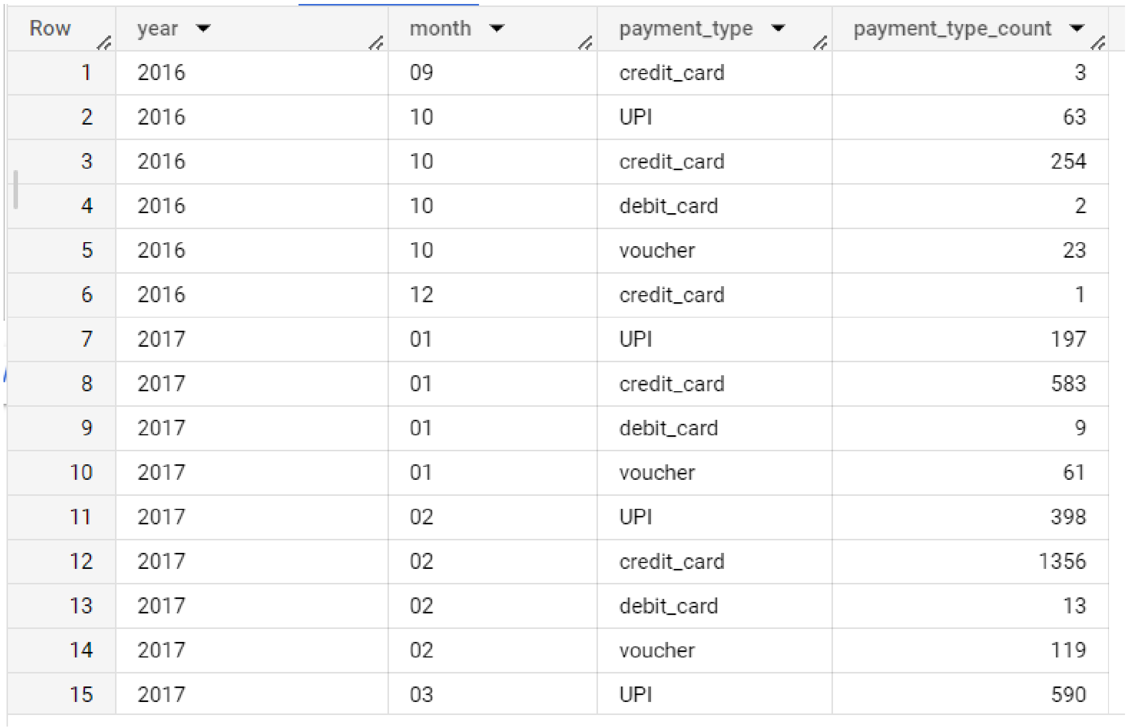
1. **Find the month on month no. of orders placed using different payment types.**

- Query

```
with cte5 as
(SELECT FORMAT_TIMESTAMP("%Y",o.order_purchase_timestamp) as year,
FORMAT_TIMESTAMP("%m",o.order_purchase_timestamp) as month,
p.payment_type
from `Target_Corporation.orders` o
join `Target_Corporation.payments` p
on o.order_id = p.order_id
order by year, month,p.payment_type)

select cte5.year,cte5.month,payment_type, count(payment_type) as
payment_type_count
from cte5
group by cte5.year, cte5.month, payment_type
order by year, month,payment_type;
```

- Screenshot of Output



Row	year	month	payment_type	payment_type_count
1	2016	09	credit_card	3
2	2016	10	UPI	63
3	2016	10	credit_card	254
4	2016	10	debit_card	2
5	2016	10	voucher	23
6	2016	12	credit_card	1
7	2017	01	UPI	197
8	2017	01	credit_card	583
9	2017	01	debit_card	9
10	2017	01	voucher	61
11	2017	02	UPI	398
12	2017	02	credit_card	1356
13	2017	02	debit_card	13
14	2017	02	voucher	119
15	2017	03	UPI	590

- Insights

  - In this Query, we can count the no. of orders placed using different payment methods in each month over the past years.

- Recommendation

  - FORMAT_TIMESTAMP("%Y",o.order_purchase_timestamp) as year, FORMAT_TIMESTAMP("%m",o.order_purchase_timestamp) as month: - Extract year and month from purchase timestamp.
  - Join tables orders and payments.
  - We take out payment_type from payments table.
  - Make it as cte.
  - Using cte we group by year, month and payment_type.
  - Count payment_type.

- Assumption

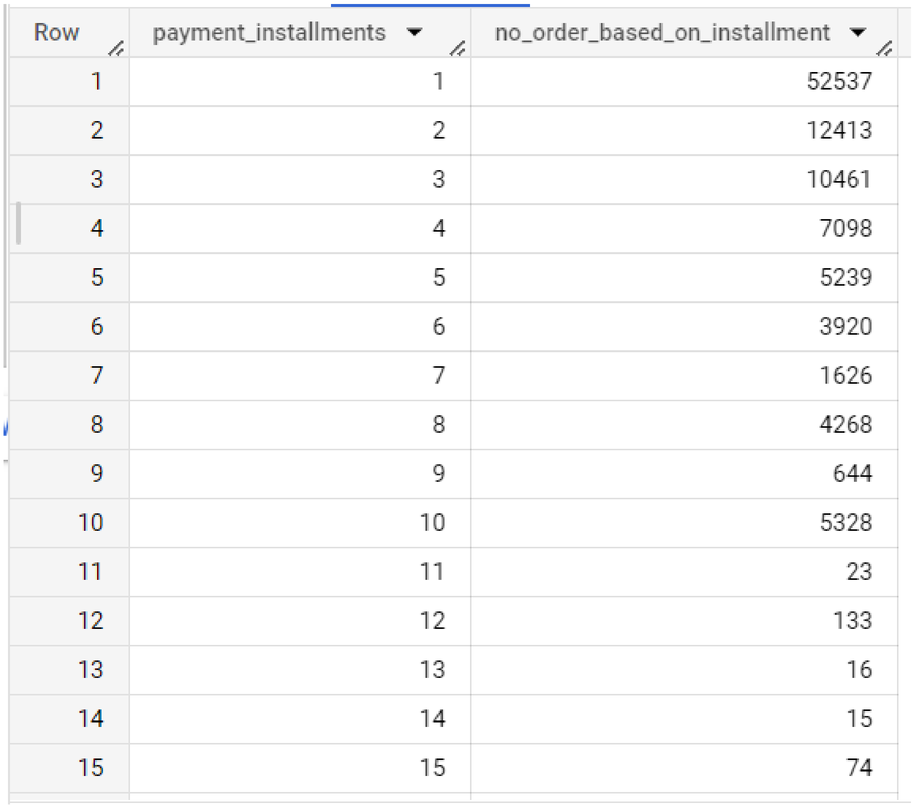
  - Mostly people are using Credit Card to pay.
  - UPI Second highest.
  - Third is Vouches.

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

- Query

```
 select payment_installments, count(order_id) as
no_order_based_on_installment
 from `Target_Corporation.payments`
 where payment_installments >= 1 and payment_value > 0
 group by payment_installments
 order by payment_installments;
```

- Screenshot of Output



Row	payment_installments	no_order_based_on_installment
1	1	52537
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
11	11	23
12	12	133
13	13	16
14	14	15
15	15	74

- Insights
  - In this Query, we can count the no. of orders placed based on the no. of payment installments where at least one installment has been successfully paid.

- Recommendation
  - Using Table payment.
  - where clause for filter payment_installments >= 1 and payment_value > 0
  - Group by payment installment.
  - Count order id.

- Assumption
  - Most of the people paid in 1-2 instalments

Submitted BY: - Smit Singh
Batch: - dsml-june-23-beginner-mor-mon-batch