Target_market_project_report

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

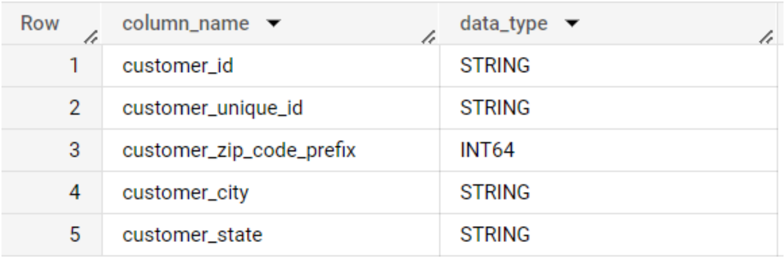
   a. Data type of all columns in the "customers" table.

   - **Interpretation -** Here we need to find the data type of the columns in the table provided.

   - The solution for the above problem statement is as follows:-

```
SELECT column_name, data_type
FROM `canvas-hook-394807.target_market.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'customers';
```

   - The output of the above query is as follows:-



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

   - **Insights -** The above query is used to extract the data type of the table (customers table). "INFORMATION_SCHEMA.COLUMNS" view allows you to get information about all columns for all table view databases.

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

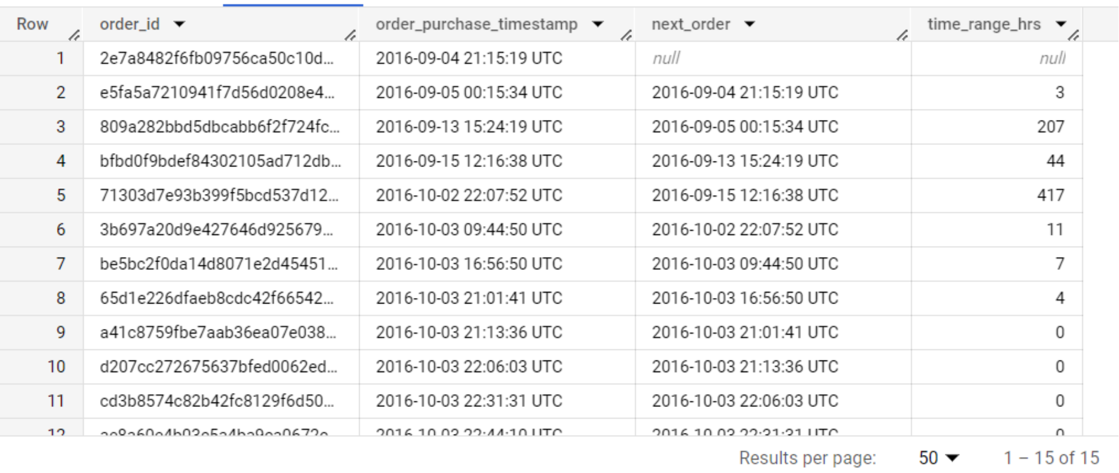
   - **Interpretation -** In the problem statement we need to get the time range between the two orders placed. We will take the time in hours.

   - The solution for the above problem statement is as follows:-

```
select
```

```
                order_id,
                order_purchase_timestamp,
                lag (order_purchase_timestamp,1) over (order by
order_purchase_timestamp) as next_order,
       date_diff(order_purchase_timestamp, lag
(order_purchase_timestamp,1) over (order by order_purchase_timestamp),
hour) as time_range_hrs
       from `target_market.orders`
       order by order_purchase_timestamp
       limit 15
```

- The output of the above query is as follows:-



Row	order_id	order_purchase_timestamp	next_order	time_range_hrs
1	2e7a8482f6fb09756ca50c10d...	2016-09-04 21:15:19 UTC	null	null
2	e5fa5a7210941f7d56d0208e4...	2016-09-05 00:15:34 UTC	2016-09-04 21:15:19 UTC	3
3	809a282bbd5dbcabb6f2f724fc...	2016-09-13 15:24:19 UTC	2016-09-05 00:15:34 UTC	207
4	bfbd0f9bdef84302105ad712db...	2016-09-15 12:16:38 UTC	2016-09-13 15:24:19 UTC	44
5	71303d7e93b399f5bcd537d12...	2016-10-02 22:07:52 UTC	2016-09-15 12:16:38 UTC	417
6	3b697a20d9e427646d925679...	2016-10-03 09:44:50 UTC	2016-10-02 22:07:52 UTC	11
7	be5bc2f0da14d8071e2d45451...	2016-10-03 16:56:50 UTC	2016-10-03 09:44:50 UTC	7
8	65d1e226dfaeb8cdc42f66542...	2016-10-03 21:01:41 UTC	2016-10-03 16:56:50 UTC	4
9	a41c8759fbe7aab36ea07e038...	2016-10-03 21:13:36 UTC	2016-10-03 21:01:41 UTC	0
10	d207cc272675637bfed0062ed...	2016-10-03 22:06:03 UTC	2016-10-03 21:13:36 UTC	0
11	cd3b8574c82b42fc8129f6d50...	2016-10-03 22:31:31 UTC	2016-10-03 22:06:03 UTC	0
12	ae8a60e4b03e5a4ba9ea0672e...	2016-10-03 22:44:10 UTC	2016-10-03 22:31:31 UTC	0

Results per page: 50 ▾ 1 – 15 of 15

- **Insights -** We use a Lag() function to access previous rows data as per defined offset value.
  "Date_diff" function is used to return the difference between the two dates.

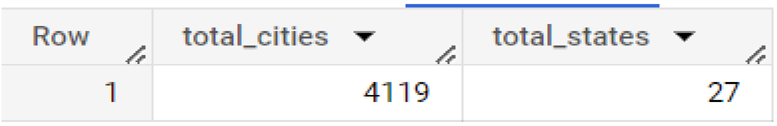
c. Count the Cities & States of customers who ordered during the given period.

- **Interpretation -** The given period is not specified so we will be working on the whole table as a given period. We need to count the cities and state of customers.

- The solution of the above problem is as follows:

```
SELECT
  COUNT(DISTINCT customer_city) AS total_cities,
  COUNT(DISTINCT customer_state) AS total_states
FROM
  `target_market.customers`
```

- The output for the above query is as follows:



Row	total_cities	total_states
1	4119	27

- **Insights -** In the above query, the “COUNT()” function returns the number of rows that matches the specific criterion.
  The “DISTINCT” keyword is used to return only the different values.

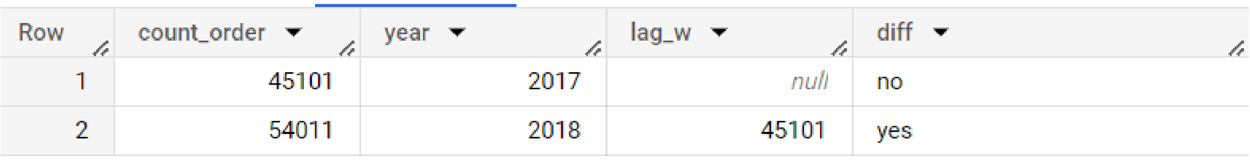
2. **In-depth Exploration:**
   a. Is there a growing trend in the no. of orders placed over the past years?

- **Interpretation -** In the above problem, we need to find if there is growth in the number of orders placed within the time range of a year. So the data for 2017 and 2018 is used.

- The query for the above problem statements is as follows:

```
select *,
        lag(x.count_order,1) over (order by x.year) as lag_w,
        if((x.count_order-lag(x.count_order,1) over (order by
x.year))>0,'yes','no') as diff
 from (
        select count(*) as count_order, extract(year from
order_purchase_timestamp) as year,
        from `target_market.orders` where extract(year from
order_purchase_timestamp)>2016
        group by year
        order by year desc) as x
```

- The output of the above query is as follows:



Row	count_order	year	lag_w	diff
1	45101	2017	*null*	no
2	54011	2018	45101	yes

- **Insights -** In the inner query of a sub-query, the total number of orders and the year is extracted where the year 2016 is excluded. Group by year is used to group the data into 2017 and 2018.
  In the outer query "LAG()" window function is used to make a new table so as to get the access of the previous row data. The conditional if statement is used to explain that if there was the growth in trend in the number of orders placed.
  If "Yes" then there is a growth.

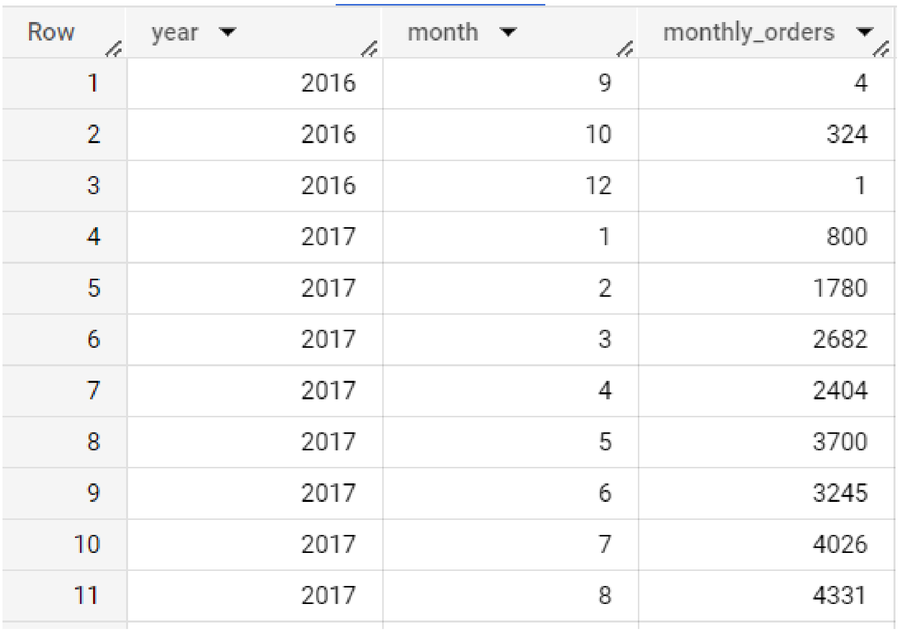
b. Can we see some kind of monthly seasonality in terms of the no. of orders being placed?

- **Interpretation -** we need to find the total number of orders placed on a monthly basis.

- The query for the above problem statements is as follows:

```
select *
from (select
        extract(year from order_purchase_timestamp) year,
        extract(month from order_purchase_timestamp) month,
        count(*) as monthly_orders
        from `target_market.orders`
        group by extract(year from order_purchase_timestamp),
extract(month from order_purchase_timestamp)
     ) as x

order by x.year, x.month
```

- The output of the above query is as follows:



Row	year	month	monthly_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
11	2017	8	4331

- **Insights -** “EXTRACT()” date function is used to extract the value from the corresponding date part.
  “COUNT()” is used to count the total orders placed on a monthly basis.
  We have used subquery for the above problem so as to perform multiple steps.

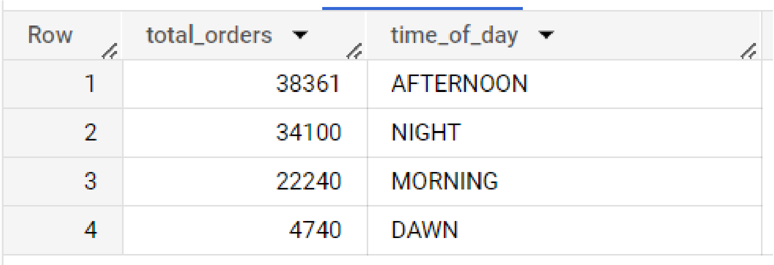
c. During what time of the day, do the Brazilian customers mostly place their orders?
   i. 0-6 hrs : Dawn
   ii. 7-12 hrs : Mornings
   iii. 13-18 hrs : Afternoon
   iv. 19-23 hrs : Night

- **Interpretation -** We need to justify in what time of the day the orders were maximum. The time of the day is described as Dawn, Morning, Afternoon or Night.
  - 0-6 hrs : Dawn
  - 7-12 hrs : Mornings
  - 13-18 hrs : Afternoon
  - 19-23 hrs : Night

- The query for the above problem statements is as follows:

```
select *
from
  (select  count(*) as total_orders,
   CASE
    WHEN EXTRACT(HOUR from order_purchase_timestamp) >= 0 and
EXTRACT(HOUR from order_purchase_timestamp) < 6
     THEN 'DAWN'
    WHEN EXTRACT(HOUR from order_purchase_timestamp) >= 6 and
EXTRACT(HOUR from order_purchase_timestamp) < 12
     THEN 'MORNING'
    WHEN EXTRACT(HOUR from order_purchase_timestamp) >= 12 and
EXTRACT(HOUR from order_purchase_timestamp) < 18
     THEN 'AFTERNOON'
    ELSE 'NIGHT'
   END AS time_of_day,

  from `target_market.orders`
  group by time_of_day)  as x
order by x.total_orders desc
```

- The output of the above query is as follows:



Row	total_orders	time_of_day
1	38361	AFTERNOON
2	34100	NIGHT
3	22240	MORNING
4	4740	DAWN

- **Insights -** Within the subquery there is a “CASE-WHEN” conditional function which extracts which part of the day it was when the orders were placed.
  Outside the subquery “GROUP BY” clause is used to group the different time zones of the day.
  “ORDER BY” clause is used to sort the columns based on maximum orders placed in a particular time zone in descending order.

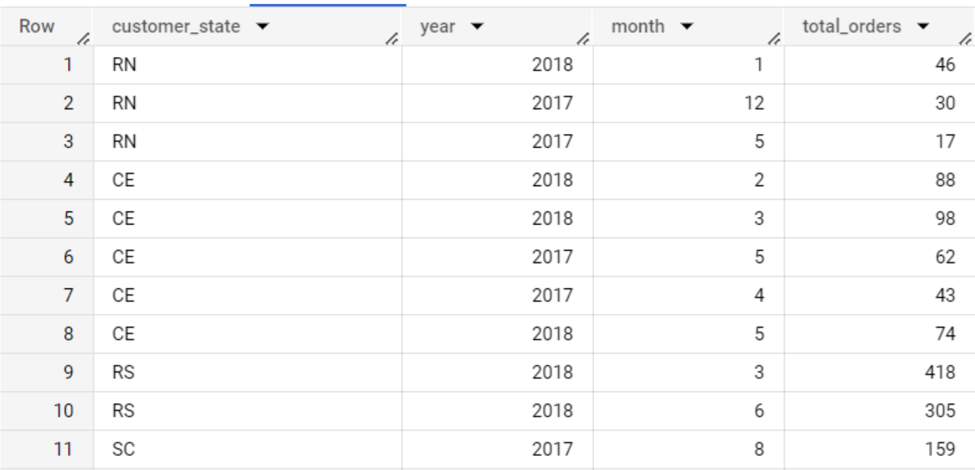
3. **Evolution of E-commerce orders in the Brazil region:**
    a. Get the month on month no. of orders placed in each state.

- **Interpretation -** We need to find the number of orders placed in on a monthly basis **per state.**

- The output of the above query is as follows:

```
select *
from (select distinct customer_state,
       extract(year from order_purchase_timestamp) as year,
       extract(month from order_purchase_timestamp) as month,
       count(order_id) as total_orders,
from `target_market.customers` cus
left join `target_market.orders` od
on cus.customer_id = od.customer_id

group by customer_state, year, month) as x
order by x.year, x.month
```

- The output of the above query is as follows:



Row	customer_state	year	month	total_orders
1	RN	2018	1	46
2	RN	2017	12	30
3	RN	2017	5	17
4	CE	2018	2	88
5	CE	2018	3	98
6	CE	2017	5	62
7	CE	2017	4	43
8	CE	2018	5	74
9	RS	2018	3	418
10	RS	2018	6	305
11	SC	2017	8	159

- **Insights -** Within the subquery we have extracted the year and month.

Count is used to count the total number of orders placed in the given time period.
Inner join is used to join the two tables i.e. : the customers table and the orders table because in the orders table the timestamp for each order is given and in the customers table, the information of the state is provided.

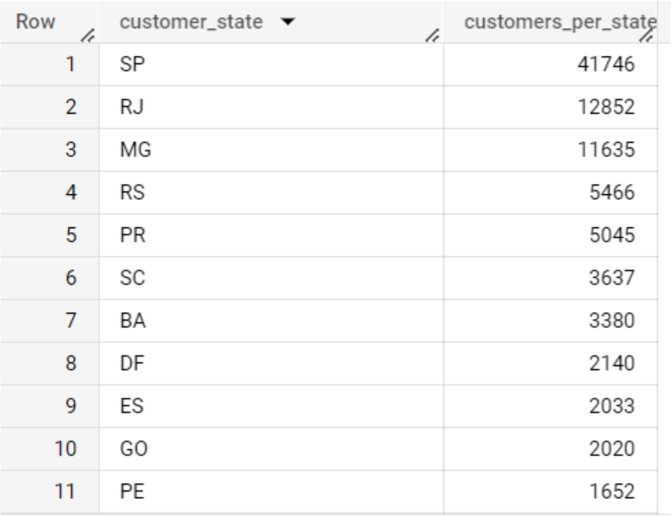
b. How are the customers distributed across all the states?

- **Interpretation -** Get the count of customers grouped by the states.

- The query for the above problem is as follows:

```
select *
from (select
      customer_state,
      count(*) as customers_per_state

from `target_market.customers`
group by customer_state) as x
order by x.customers_per_state desc
```

- The output of the above query is :



Row	customer_state	customers_per_state
1	SP	41746
2	RJ	12852
3	MG	11635
4	RS	5466
5	PR	5045
6	SC	3637
7	BA	3380
8	DF	2140
9	ES	2033
10	GO	2020
11	PE	1652

- **Insights -** Within the subquery we have “COUNT()” to count the number of customers and grouped by the customer_state.
  Outside the query “ORDER BY” clause is used to order by customer_state.

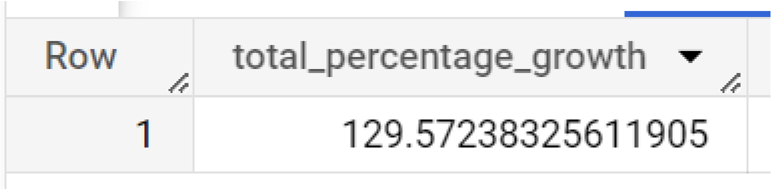
4. **Impact on Economy: Analyze the money movement by e-commerce by looking at order prices, freight and others.**
    a. Get the % increase in the cost of orders from year 2017 to 2018 (include months between Jan to Aug only).

- **Interpretation -** We need to find the percentage increase in the cost of orders in a year for a span of January to august.
  We can use the "payment_value" column in the payments table to get the cost of orders.
  The formula for percentage change is:
  ((New cost - old cost)/ old cost) * 100

- The query for the above problem is as follows:

```
select
      (x.total_order-Y.total_order)/Y.total_order*100 as
total_weight_loss
from(
      select
            count(*) as total_order,'D' as d
      from `target_market.payments` pay
      join `target_market.orders` od
      on pay.order_id = od.order_id
      where extract(month from order_purchase_timestamp) between 1
and 8
      and extract(year from order_purchase_timestamp)=2018
) as x
join
(
  select
        count(*) as total_order,'D' as d
  from `target_market.payments` pay
  join `target_market.orders` od
  on pay.order_id = od.order_id
  where extract(month from order_purchase_timestamp) between 1 and
8
  and extract(year from order_purchase_timestamp)=2017
) as Y
on x.d=y.d
```

- The output of the above query is as follows:



Row	total_percentage_growth
1	129.57238325611905

- **Insights -**
In the first sub-query the two tables are joined i.e.- payments and orders table. The data for January to August for 2018 is extracted using where clause.
SImilarly the second sub-query is created for the extraction of the same data as above for 2017.
Then Joined both sub-queries to form a single subquery.
In the outer query the formula to find the percentage is used to get the result of the total percentage growth between the year 2017 to 2018 for January to August.

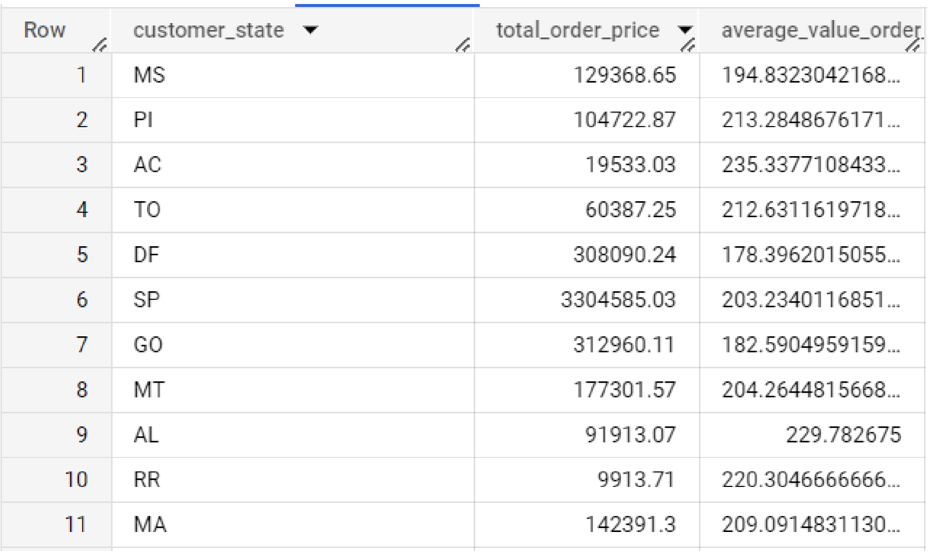
b. Calculate the Total & Average value of order price for each state.

- **Interpretation -** We need to find the total sum and average of order price per state.

```
select
      distinct cus.customer_state,
      sum(pay.payment_value) over (partition by cus.customer_state) as
total_order_price,
      avg(pay.payment_value) over (partition by cus.customer_state) as
average_value_order_price
from `target_market.payments` pay
join `target_market.orders` od
```

```
on pay.order_id = od.order_id
join `target_market.customers` cus
on cus.customer_id = od.customer_id
group by cus.customer_state, pay.payment_value
```

- The output of the above query is as follows:



Row	customer_state	total_order_price	average_value_order
1	MS	129368.65	194.8323042168...
2	PI	104722.87	213.2848676171...
3	AC	19533.03	235.3377108433...
4	TO	60387.25	212.6311619718...
5	DF	308090.24	178.3962015055...
6	SP	3304585.03	203.2340116851...
7	GO	312960.11	182.5904959159...
8	MT	177301.57	204.2644815668...
9	AL	91913.07	229.782675
10	RR	9913.71	220.3046666666...
11	MA	142391.3	209.0914831130...

- **Insights -** “SUM()” & “AVG()” aggregate function is used to extract the total number of total payment value and the average payment value for each customer state from the payments , orders, and customers table.
  Three tables are joined using inner join .

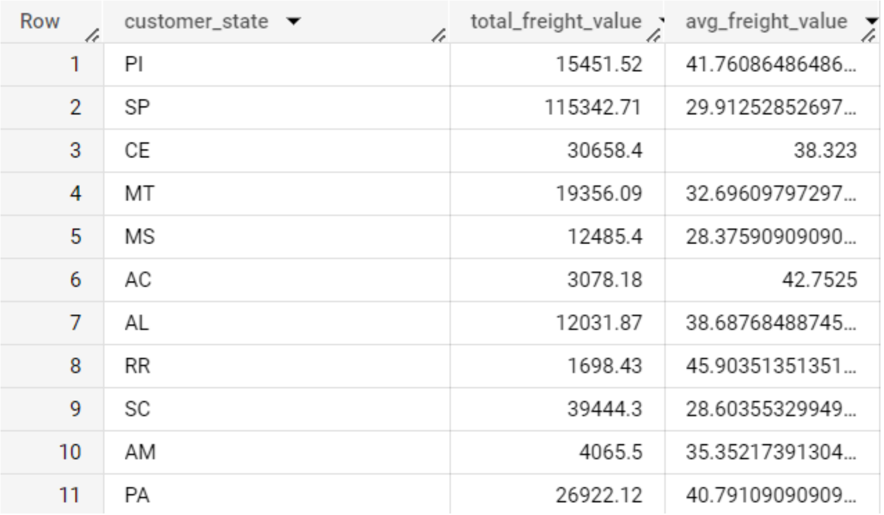
c. Calculate the Total & Average value of order freight for each state.

- **Interpretation -** We need to find the total sum and average of freight value per state.

- The query for the above problem is as follows :

```
select
      distinct cus.customer_state,
      sum(oditm.freight_value) over (partition by
cus.customer_state) as total_freight_value,
      avg(oditm.freight_value) over (partition by
cus.customer_state) as avg_freight_value
from `target_market.order_items` oditm
join `target_market.orders` od
on oditm.order_id = od.order_id
join `target_market.customers` cus
on cus.customer_id = od.customer_id
group by cus.customer_state, oditm.freight_value
```

- The output of the above query is as follows:



Row	customer_state	total_freight_value	avg_freight_value
1	PI	15451.52	41.76086486486...
2	SP	115342.71	29.91252852697...
3	CE	30658.4	38.323
4	MT	19356.09	32.69609797297...
5	MS	12485.4	28.37590909090...
6	AC	3078.18	42.7525
7	AL	12031.87	38.68768488745...
8	RR	1698.43	45.90351351351...
9	SC	39444.3	28.60355329949...
10	AM	4065.5	35.35217391304...
11	PA	26922.12	40.79109090909...

- **Insights -** "SUM()" & "AVG()" aggregate function is used to extract the total number of total freight value and the average payment value for each customer state from the order_items , orders, and customers table.
  Three tables are joined using inner join .

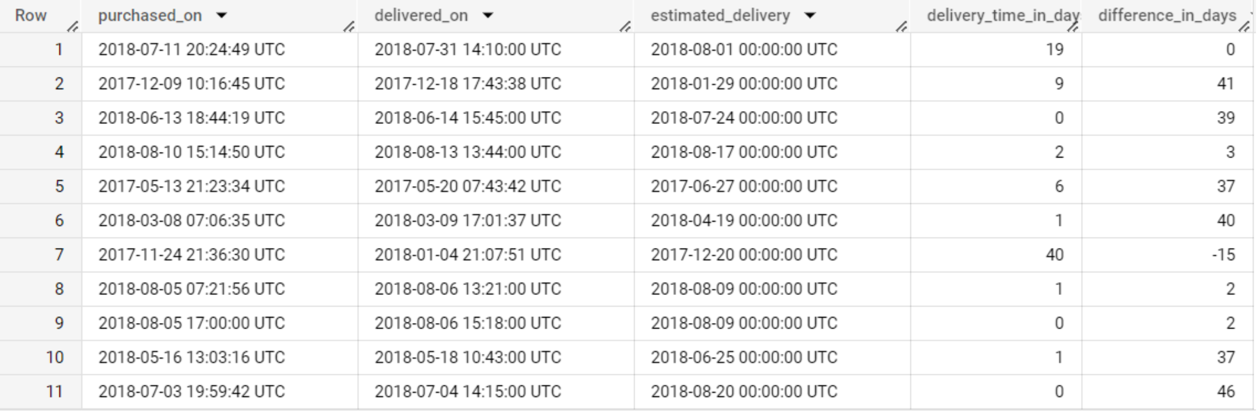
5. **Analysis based on sales, freight and delivery time.**
   a. Find the no. of days taken to deliver each order from the order's purchase date as delivery time.
      Also, calculate the difference (in days) between the estimated & actual delivery date of an order.

- **Interpretation -** We need to find the difference in the number of days when the order was placed and the delivery time.
  In addition we need to find the difference in the estimated delivery time and the actual delivery date.

- The query for the above problem is as follows :

```
select
      order_purchase_timestamp as purchased_on,
      order_delivered_carrier_date as delivered_on,
      order_estimated_delivery_date as estimated_delivery,

timestamp_diff(order_delivered_carrier_date,order_purchase_timesta
mp, day) as delivery_time_in_days,
      timestamp_diff(order_estimated_delivery_date,
order_delivered_carrier_date, day) as difference_in_days,

from `target_market.orders`
```

- The output of the above query is as follows:



Row	purchased_on	delivered_on	estimated_delivery	delivery_time_in_day	difference_in_days
1	2018-07-11 20:24:49 UTC	2018-07-31 14:10:00 UTC	2018-08-01 00:00:00 UTC	19	0
2	2017-12-09 10:16:45 UTC	2017-12-18 17:43:38 UTC	2018-01-29 00:00:00 UTC	9	41
3	2018-06-13 18:44:19 UTC	2018-06-14 15:45:00 UTC	2018-07-24 00:00:00 UTC	0	39
4	2018-08-10 15:14:50 UTC	2018-08-13 13:44:00 UTC	2018-08-17 00:00:00 UTC	2	3
5	2017-05-13 21:23:34 UTC	2017-05-20 07:43:42 UTC	2017-06-27 00:00:00 UTC	6	37
6	2018-03-08 07:06:35 UTC	2018-03-09 17:01:37 UTC	2018-04-19 00:00:00 UTC	1	40
7	2017-11-24 21:36:30 UTC	2018-01-04 21:07:51 UTC	2017-12-20 00:00:00 UTC	40	-15
8	2018-08-05 07:21:56 UTC	2018-08-06 13:21:00 UTC	2018-08-09 00:00:00 UTC	1	2
9	2018-08-05 17:00:00 UTC	2018-08-06 15:18:00 UTC	2018-08-09 00:00:00 UTC	0	2
10	2018-05-16 13:03:16 UTC	2018-05-18 10:43:00 UTC	2018-06-25 00:00:00 UTC	1	37
11	2018-07-03 19:59:42 UTC	2018-07-04 14:15:00 UTC	2018-08-20 00:00:00 UTC	0	46

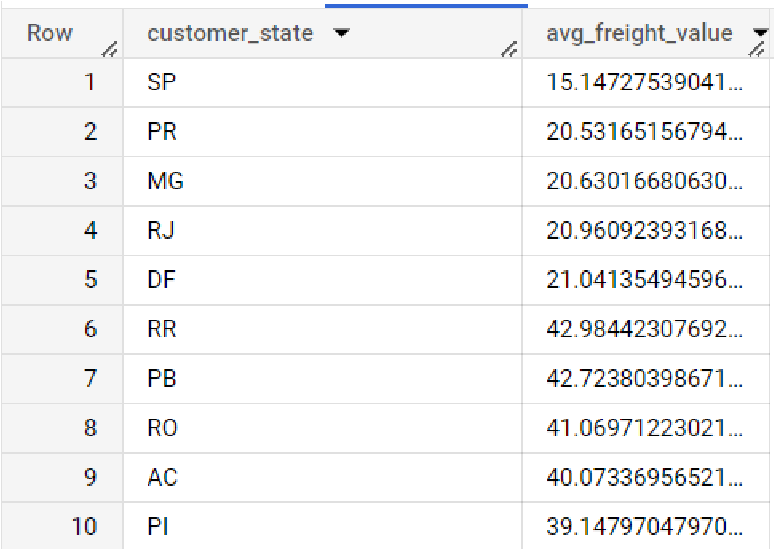
- **Insights -** “TIMESTAMP_DIFF” is used to get the difference between the two dates and their corresponding time.
  The first “TIMESTAMP_DIFF” is used for getting the total number of days from placing the order to its delivery date.
  The second “TIMESTAMP_DIFF” in the query is used to get the difference between the estimated delivery date and the actual delivery date.

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

- The query for the above problem is as follows:

```
with highest_avg as
    (
     select
     distinct cus.customer_state,
     AVG(oditm.freight_value) as avg_freight_value
     from `target_market.order_items` oditm
     join `target_market.orders` od
     on oditm.order_id = od.order_id
     join `target_market.customers` cus
     on od.customer_id = cus.customer_id
     group by cus.customer_state
     ORDER BY avg_freight_value desc
     limit 5
    ),
    lowest_average as
     (
      select
      distinct cus.customer_state,
      AVG(oditm.freight_value) as avg_freight_value
      from `target_market.order_items` oditm
      join `target_market.orders` od
      on oditm.order_id = od.order_id
      join `target_market.customers` cus
      on od.customer_id = cus.customer_id
      group by cus.customer_state
      ORDER BY avg_freight_value
      limit 5
     )
select * from highest_avg
union all
select * from lowest_average
```

- The output of the above query is as follows:



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

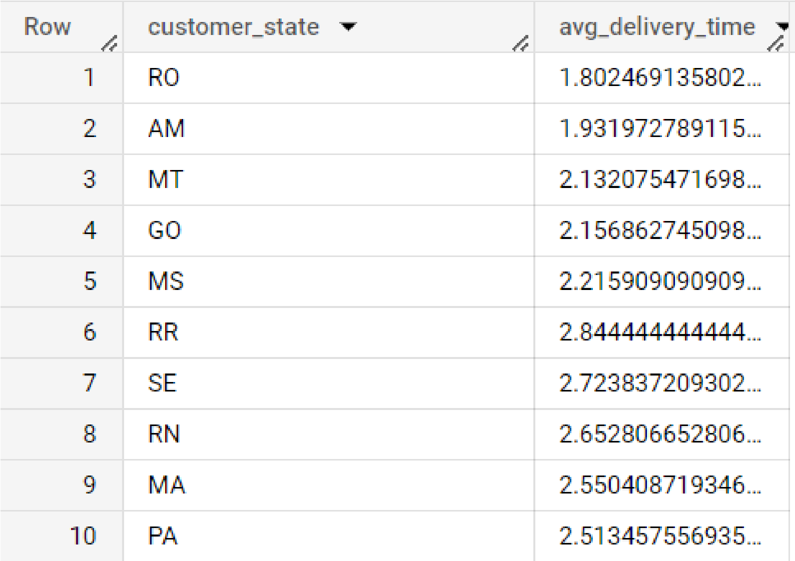
c. **Interpretation -** Find out the top 5 states with the highest & lowest average delivery time.

- The query for the above problem is as follows:

```
with highest_avg as
        (
          select
          distinct cus.customer_state,

AVG(timestamp_diff(order_delivered_carrier_date,order_approved_at,
day)) as avg_delivery_time
          from `target_market.orders` od
          join `target_market.customers` cus
          on od.customer_id = cus.customer_id
          group by cus.customer_state
          ORDER BY avg_delivery_time desc
          limit 5
        ),
        lowest_average as
          (
            select
            distinct cus.customer_state,

AVG(timestamp_diff(order_delivered_carrier_date,order_approved_at,
day)) as avg_delivery_time
```

```
      from `target_market.orders` od
      join `target_market.customers` cus
      on od.customer_id = cus.customer_id
      group by cus.customer_state
      ORDER BY avg_delivery_time
      limit 5
     )

select * from highest_avg
union all
select * from lowest_average
```

- The output of the above query is as follows:



Row	customer_state	avg_delivery_time
1	RO	1.802469135802...
2	AM	1.931972789115...
3	MT	2.132075471698...
4	GO	2.156862745098...
5	MS	2.215909090909...
6	RR	2.844444444444...
7	SE	2.723837209302...
8	RN	2.652806652806...
9	MA	2.550408719346...
10	PA	2.513457556935...

- **Insights -**

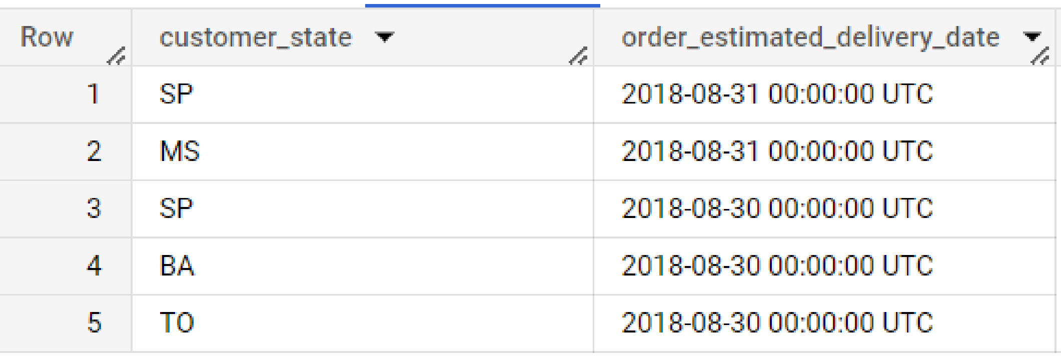
d. Find out the top 5 states where the order delivery is really fast as compared to the estimated date of delivery.

- The query for the above problem is as follows:

```
select distinct cus.customer_state,
o.order_estimated_delivery_date
 from `target_market.orders` as o
join `target_market.customers` as cus
on o.customer_id = cus.customer_id
```

```
where o.order_status='delivered' and
o.order_estimated_delivery_date < o.order_delivered_customer_date
order by o.order_estimated_delivery_date desc
limit 5
```

- The output of the above query is as follows :



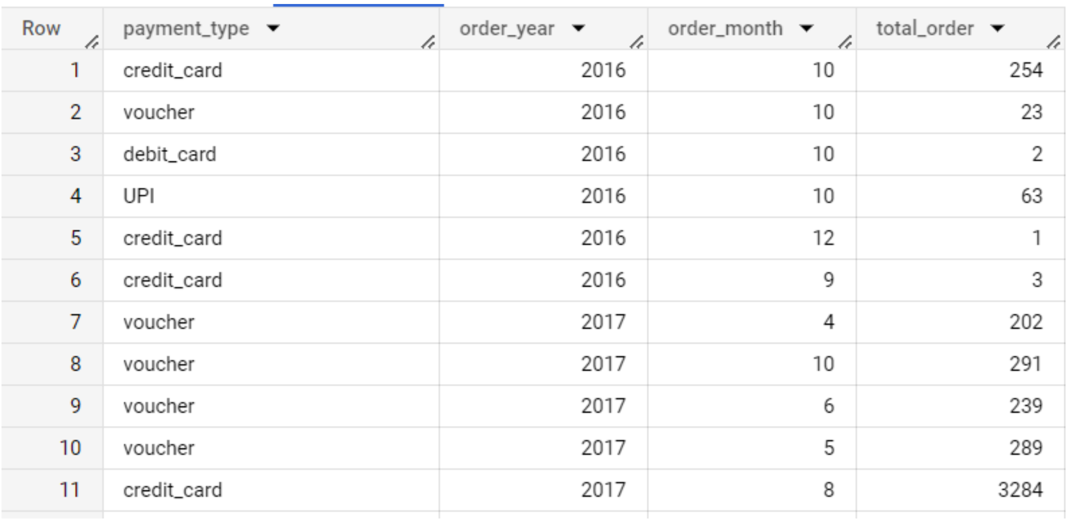
Row	customer_state	order_estimated_delivery_date
1	SP	2018-08-31 00:00:00 UTC
2	MS	2018-08-31 00:00:00 UTC
3	SP	2018-08-30 00:00:00 UTC
4	BA	2018-08-30 00:00:00 UTC
5	TO	2018-08-30 00:00:00 UTC

- **Insights -** Joined two tables i.e : - orders and customers table. Where condition is used for extracting the orders that are delivered AND comparison is done where estimated time of delivery is greater than the delivery time.
  Order by and limit clause is used to get the top 5 states.

6. **Analysis based on the payments:**
   a. Find the month on month no. of orders placed using different payment types.
      - The query for the above problem is as follows :

```
select *
       from
              (select
                     pay.payment_type,
                     extract(year from
od.order_purchase_timestamp) as order_year,
                     extract(month from
od.order_purchase_timestamp) as order_month,
                     count(od.order_id) as total_order
              from `target_market.payments`
pay,`target_market.orders` od
              where pay.order_id = od.order_id
              group by extract(month from
od.order_purchase_timestamp),order_year,pay.payment_type) as x
       order by x.order_year
```

- The output of the above query is as follows:



Row	payment_type	order_year	order_month	total_order
1	credit_card	2016	10	254
2	voucher	2016	10	23
3	debit_card	2016	10	2
4	UPI	2016	10	63
5	credit_card	2016	12	1
6	credit_card	2016	9	3
7	voucher	2017	4	202
8	voucher	2017	10	291
9	voucher	2017	6	239
10	voucher	2017	5	289
11	credit_card	2017	8	3284

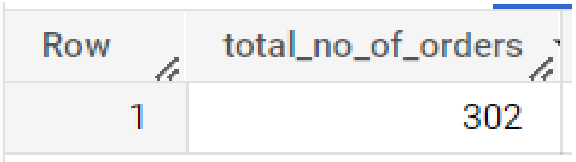
- **Insights -**

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

- The query for the above problem is as follows:

```
select count(*) as total_no_of_orders
from (select sum(p.payment_value) as sum_payment,p.order_id
      from `target_market.payments` as p
      group by p.order_id) as x
,`target_market.order_items` as o
where o.order_id=x.order_id and o.price=x.sum_payment
```

- The output of the above query is as follows:



Row	total_no_of_orders
1	302

- **Insights -** In the inner query we are taking the sum of the payment_value and the order_id from the payments table and grouping by order_id.
  In the outer query we are taking the count of the orders as total_no_of_orders and specifying the condition in where clause i.e. `o.order_id=x.order_id and o.price=x.sum_payment.`

Recommendations - My all over recommendation is, in the description of the table if the comments were added it would have been much easier for us to read the raw data.
As in some questions I have assumed some conditions.