Business Case:Target SQL

This particular business case focuses on the operations of Target in Brazil and provides insightful information about 100,000 orders placed between 2016 and 2018. The dataset offers a comprehensive view of various dimensions including the order status, price, payment and freight performance, customer location, product attributes, and customer reviews.

By analyzing this extensive dataset, it becomes possible to gain valuable insights into Target's operations in Brazil. The information can shed light on various aspects of the business, such as order processing, pricing strategies, payment and shipping efficiency, customer demographics, product characteristics, and customer satisfaction levels.

Analysis

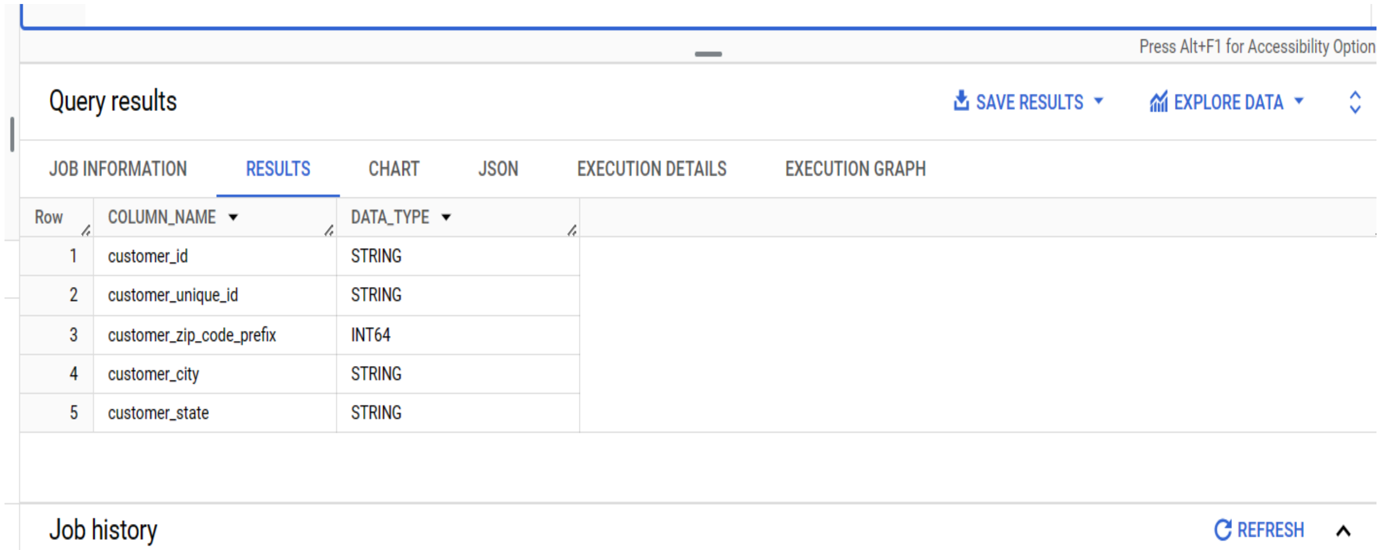
1.Import the dataset and do usual exploratory analysis steps like checking the structure & characteristics of the dataset:

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

Query:

```
SELECT COLUMN_NAME, DATA_TYPE
FROM `target-bcs-abhi.Target_abhi.INFORMATION_SCHEMA.COLUMNS`
WHERE TABLE_NAME = 'customers';
```

Result:



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

Inference:

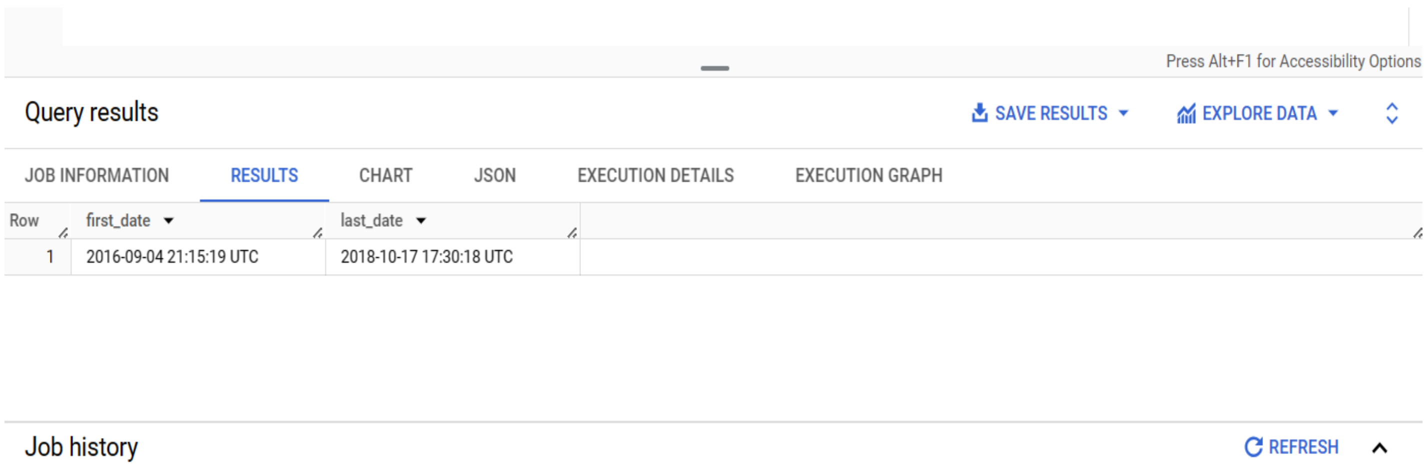
I used INFORMATION_SCHEMA.COLUMNS to find out the required information of a table.All columns are of **STRING** data type except “customer_zip_code_prefix” which is of **INT64** data type.

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

Query:

```
select min(order_purchase_timestamp) as first_date, max(order_purchase_timestamp) as
last_date
from `Target_abhi.orders`;
```

Result:



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

Inference:

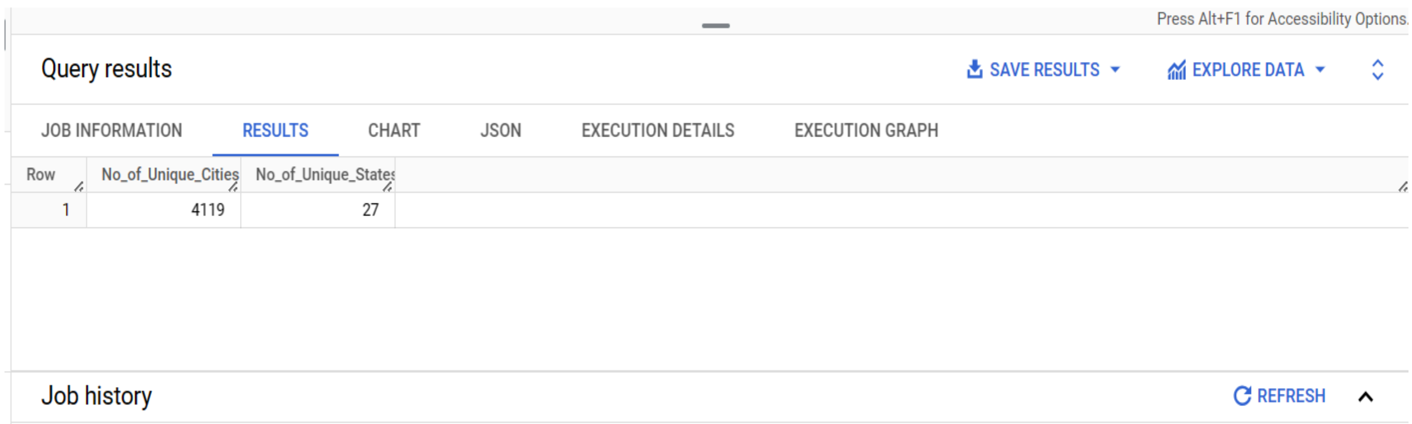
I used min() function to find the lowest(earliest) date and max() to find the most highest(recent) date
First Date of purchase(along with timestamp in UTC): 2016-09-04 21:15:19 UTC
Last Date of purchase(along with timestamp in UTC): 2018-10-17 17:30:18 UTC

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

Query:

```
select count( distinct customer_city) as No_of_Unique_Cities, count( distinct
customer_state) as No_of_Unique_States
from `Target_abhi.customers` c join `Target_abhi.orders` o on
c.customer_id=o.customer_id;
```

Result:



Inference:

I used **Distinct** to find unique values and **count()** to find out the count of cities and states. We further used inner join between customers and orders as we want to retain orders placed by customers during the given time period.
We got 1 row(because of **aggregate function**)
No. of unique States=27
No.of unique Cities=4119

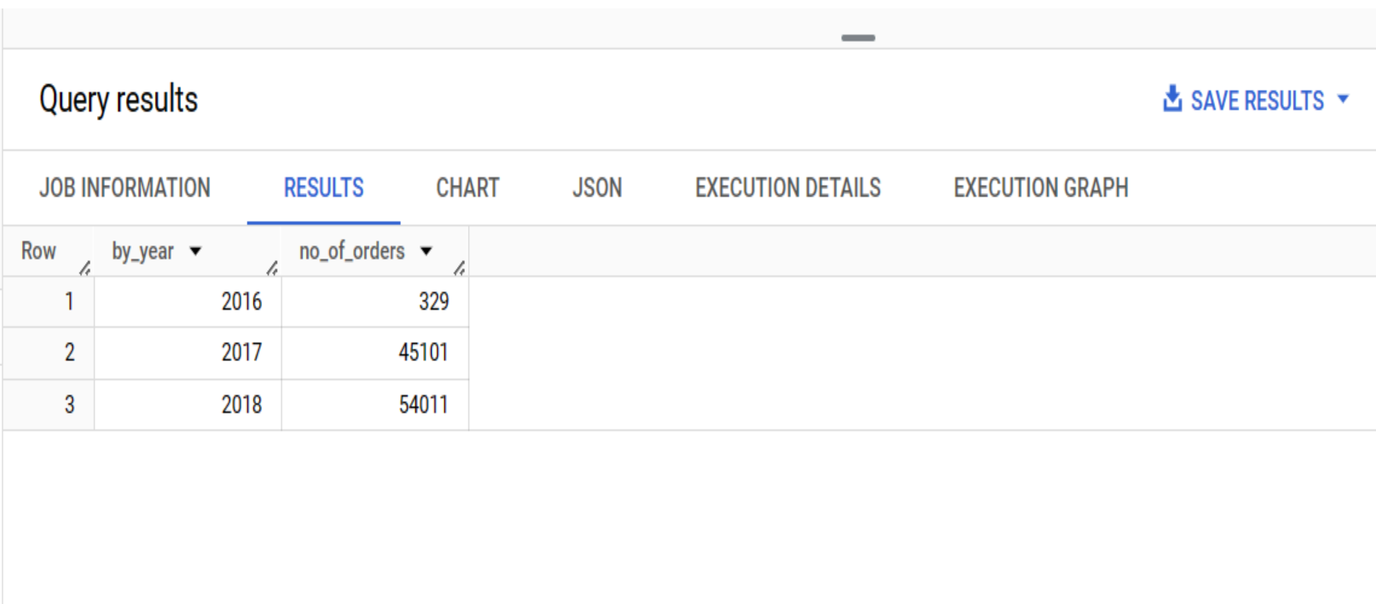
2. In-depth Exploration:

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

Query:

```
select extract(year from order_purchase_timestamp) as by_year, count(distinct
order_id) as no_of_orders
from `Target_abhi.orders`
group by 1 order by 1;
```

Result:



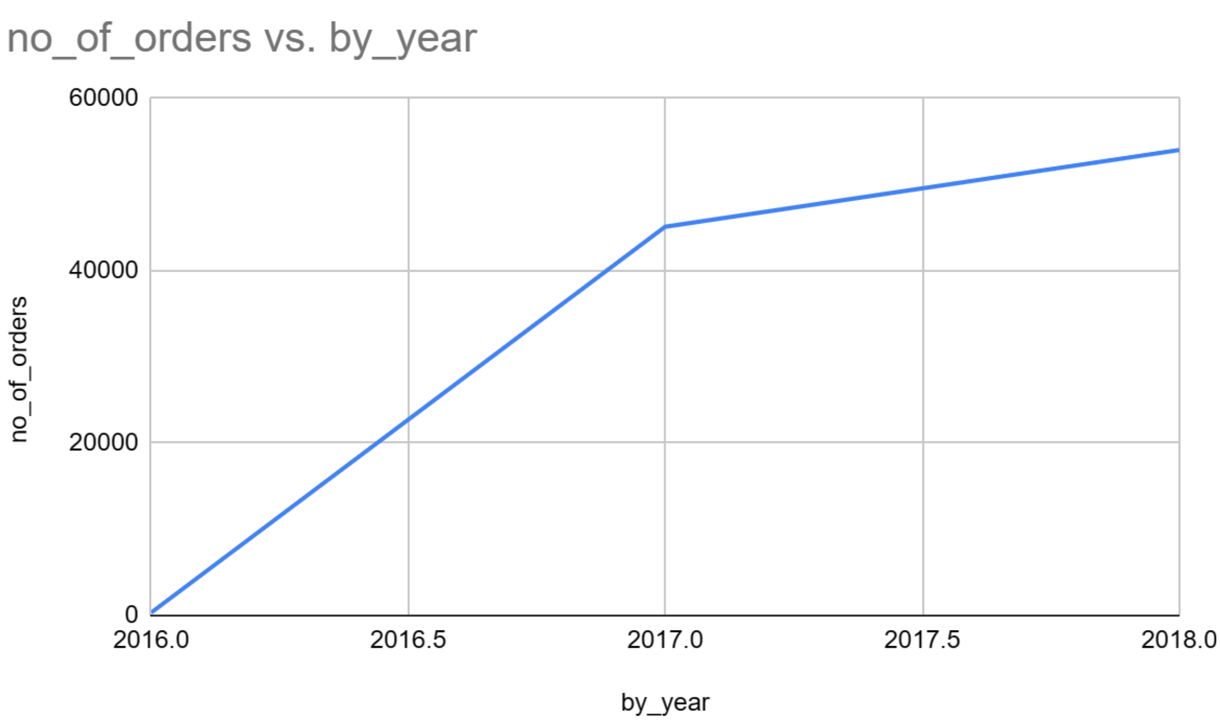
Row	by_year	no_of_orders
1	2016	329
2	2017	45101
3	2018	54011

Inference:

We had to analyze the trend of order placements over the years. I used the order table and extracted the years and applied the aggregate function count() to determine the number of orders in each year using a group by. Subsequently, I sorted the rows based on the year column(which will be by default ascending if not mentioned). As per the output of the query above the number of orders placed annually has increased. Additionally, it's important to note that for the year 2016, we only have records for September, October and December. Therefore, drawing conclusions from 2016-2017 is limited due to this data gap.(check image below)

Row	by_year	by_month	no_of_orders
1	2016	Sep	4
2	2016	Oct	324
3	2016	Dec	1
4	2017	Jan	800
5	2017	Feb	1780
6	2017	Mar	2682
7	2017	Apr	2404
8	2017	May	3700

Results per page: 50 1 – 25 of 25



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

Query:

```
with cte as (
    select
        extract(year from order_purchase_timestamp) as by_year,
        format_date("%B", order_purchase_timestamp) as by_month,
        count(order_id) as no_of_orders
    from
        `Target_abhi.orders`
    group by
        1, 2
    order by
        1, parse_date('%B', by_month)
)

select
    *,
    dense_rank() over (partition by by_year order by no_of_orders desc) as ranking
from
    cte
order by
    by_year, parse_date('%B', by_month) asc;
```

Result:

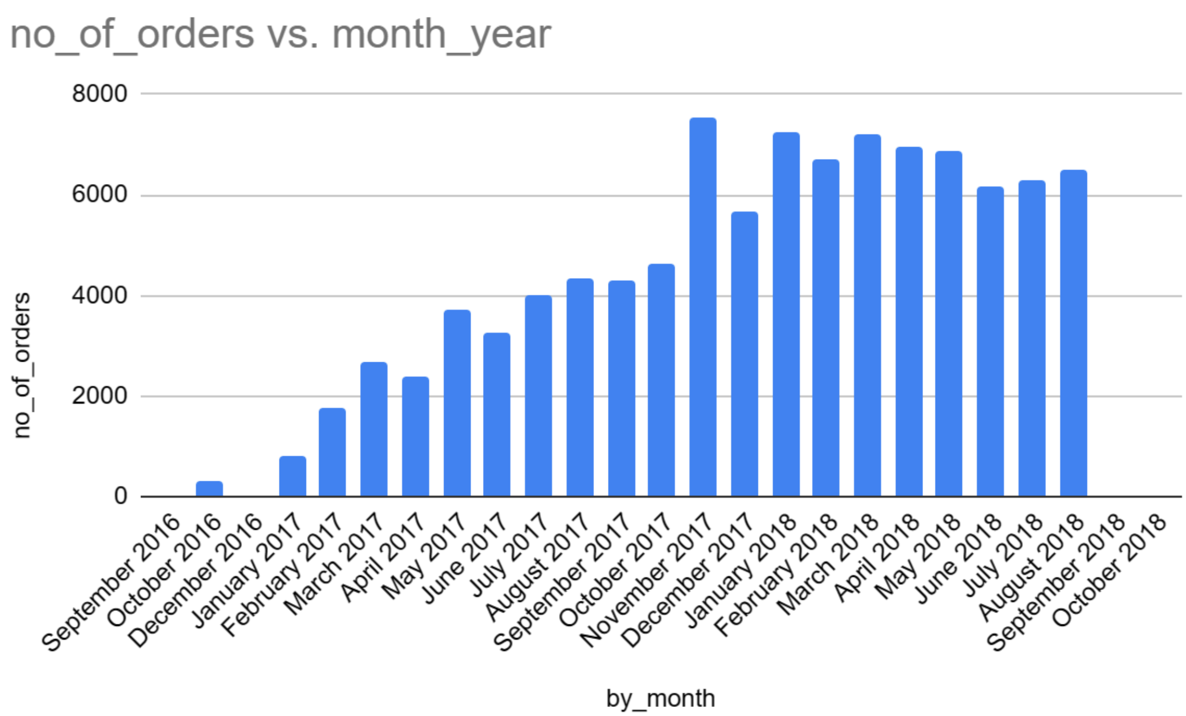
Query results

JOB INFORMATION RESULTS CHART JSON EXECUTION DETAILS EXECUTION GRAPH

Row	by_year	by_month	no_of_orders	ranking
1	2016	September	4	2
2	2016	October	324	1
3	2016	December	1	3
4	2017	January	800	12
5	2017	February	1780	11
6	2017	March	2682	9
7	2017	April	2404	10
8	2017	May	3700	7
9	2017	June	3245	8
10	2017	July	4026	6
11	2017	August	4331	4
12	2017	September	4285	5
13	2017	October	4631	3
14	2017	November	7544	1
15	2017	December	5673	2
16	2018	January	7269	1
17	2018	February	6728	5
18	2018	March	7211	2
19	2018	April	6939	3
20	2018	May	6873	4
21	2018	June	6167	8

Inference:

In the provided query, we analyzed the total number of orders placed each month across different years. I have used parse_date() to convert a string to a date value. By applying the DENSE_RANK() function, we ranked the months based on order volume (from highest to lowest). For instance, in 2016, October held the top rank due to the highest order count, followed by September with four orders, and December with the fewest. However, this ranking alone does not reveal any clear monthly seasonality for 2016. Moving to 2017, November saw the highest order volume, while January dominated in 2018. Overall, 2017 exhibited a positive trend with increasing orders each month, albeit with minor fluctuations. As for 2018, no distinct trend emerges.

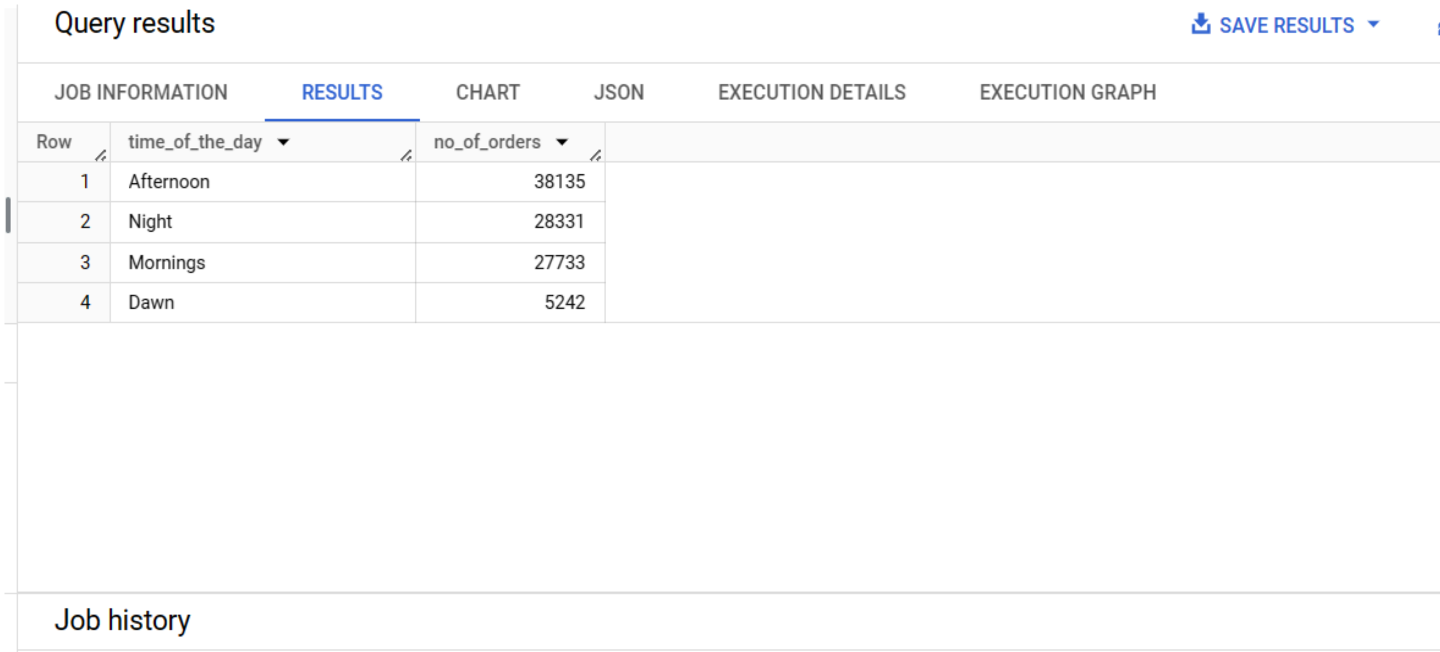


(3) During what time of the day, do the Brazilian customers mostly place their orders? (Dawn, Morning, Afternoon or Night)
0-6 hrs : Dawn
7-12 hrs : Mornings
13-18 hrs : Afternoon
19-23 hrs : Night

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_abhi.orders`
group by 1
order by 2 desc
```

Result:



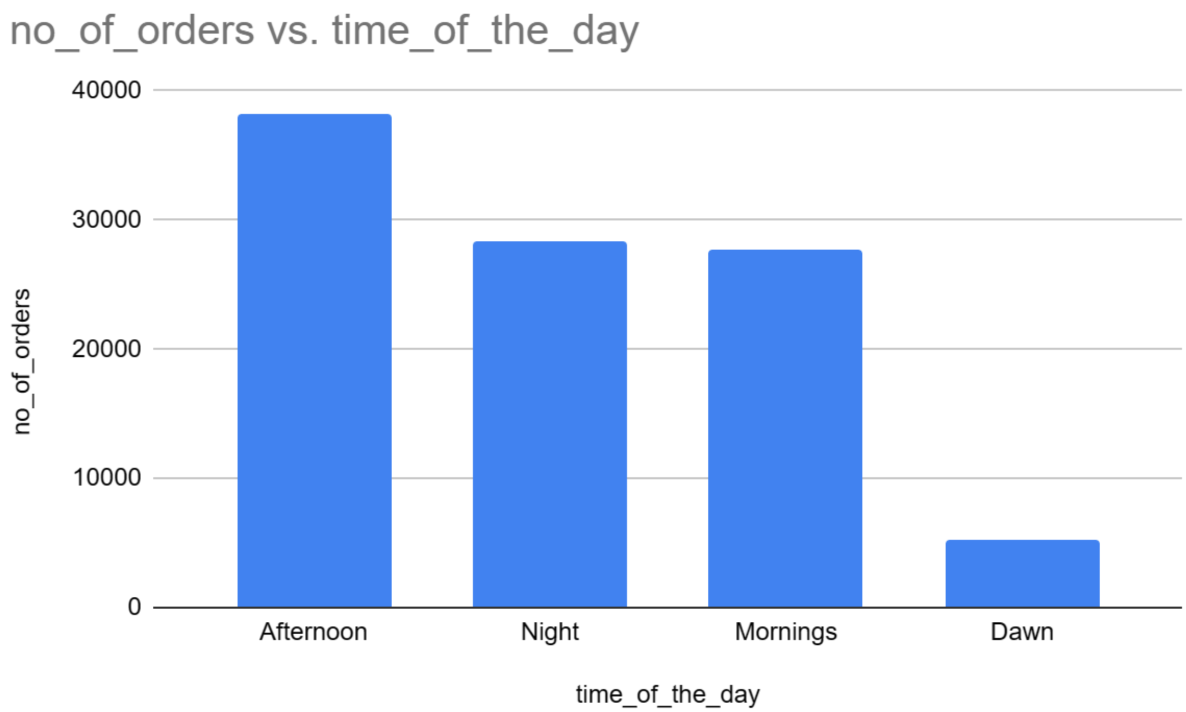
Row	time_of_the_day	no_of_orders
1	Afternoon	38135
2	Night	28331
3	Mornings	27733
4	Dawn	5242

Inference:

I wrote an SQL query to analyze order purchase times based on the order_purchase_timestamp.
The query categorized purchase times into four parts of the day: “Dawn,” “Morning,” “Afternoon,” and “Night.”
It counted the total number of orders made during each time period.
The result was grouped by purchase_time and ordered in descending order of total orders.

By using the CASE statement, we efficiently classified orders into different time periods.
This query helps us understand customer behavior based on when they make purchases.
For example, we can infer which part of the day has the highest order volume.

From the results, we can see that the “Afternoon” time period has the highest order volume.


no_of_orders vs. time_of_the_day
40000
30000
20000
10000
0
no_of_orders
Afternoon
Night
Mornings
Dawn
time_of_the_day

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

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

Query:

```
WITH cte AS (
  SELECT
    g.geolocation_state AS State,
    FORMAT_DATE("%B %Y", order_purchase_timestamp) AS Month,
    COUNT(*) AS No_of_orders_placed
  FROM `Target_abhi.customers` c
  JOIN `Target_abhi.orders` o ON o.customer_id = c.customer_id
  JOIN `Target_abhi.geolocation` g ON g.geolocation_zip_code_prefix =
c.customer_zip_code_prefix
  GROUP BY 1, 2
),
cte2 AS (
  SELECT
    State,
    Month,
```

```
    No_of_orders_placed,
    LAG(No_of_orders_placed) OVER (PARTITION BY State ORDER BY PARSE_DATE('%B %Y',
Month)) AS Lag_No_of_orders_placed
  FROM cte
)
SELECT
  State,
  Month,
  No_of_orders_placed,
  Lag_No_of_orders_placed,
  concat(ROUND((No_of_orders_placed - Lag_No_of_orders_placed) /
Lag_No_of_orders_placed * 100, 2),'%') AS Percent_change_over_month
FROM cte2
ORDER BY State, PARSE_DATE('%B %Y', Month) ASC;
```

Result:

Query results

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

Row	State	Month	No_of_orders_placed	Lag_No_of_orders_pl	Percent_Growth
1	AC	January 2017	45	null	null
2	AC	February 2017	179	45	297.78%
3	AC	March 2017	329	179	83.8%
4	AC	April 2017	362	329	10.03%
5	AC	May 2017	886	362	144.75%
6	AC	June 2017	432	886	-51.24%
7	AC	July 2017	605	432	40.05%
8	AC	August 2017	657	605	8.6%
9	AC	September 2017	161	657	-75.49%
10	AC	October 2017	535	161	232.3%
11	AC	November 2017	368	535	-31.21%
12	AC	December 2017	389	368	5.71%
13	AC	January 2018	649	389	66.84%
14	AC	February 2018	336	649	-48.23%
15	AC	March 2018	187	336	-44.35%
16	AC	April 2018	427	187	128.34%
17	AC	May 2018	275	427	-35.6%

Inference:

The query begins with two CTEs: cte and cte2.
cte calculates the number of orders placed per state and month. It joins the customers, orders, and geolocation tables, grouping the results by state and formatted month.
cte2 builds on cte by adding a column that computes the lagged value of the number of orders placed for each state. This is done using the LAG() window function.
:
The main query selects columns from cte2:
State: The state name.
Month: The formatted month.
No_of_orders_placed: The number of orders placed in the current month.
Lag_No_of_orders_placed: The number of orders placed in the previous month (lagged value).

Percent_change_over_month: The percentage change in orders placed from the previous month to the current month.

The query aims to analyze the trend in order placements over time for different states. By calculating the percentage change, it provides insights into whether order volumes are increasing or decreasing.
For example, if the Percent_change_over_month is positive, it indicates growth in orders, while a negative value suggests a decline.

(2) How are the customers distributed across all the states?

Query:

```
select customer_state, count(distinct customer_unique_id) as Customers from
`Target_abhi.customers` group by 1 order by 2 desc
```

Result:

Query results

JOB INFORMATION RESULTS CHART JSON EXECUTION DETAILS EXECUTION GRAPH

Row	customer_state	Customers
1	SP	40302
2	RJ	12384
3	MG	11259
4	RS	5277
5	PR	4882
6	SC	3534
7	BA	3277
8	DF	2075
9	ES	1964
10	GO	1952
11	PE	1609
12	CE	1313
13	PA	949
14	MT	876
15	MA	726
16	MS	694
17	PB	519

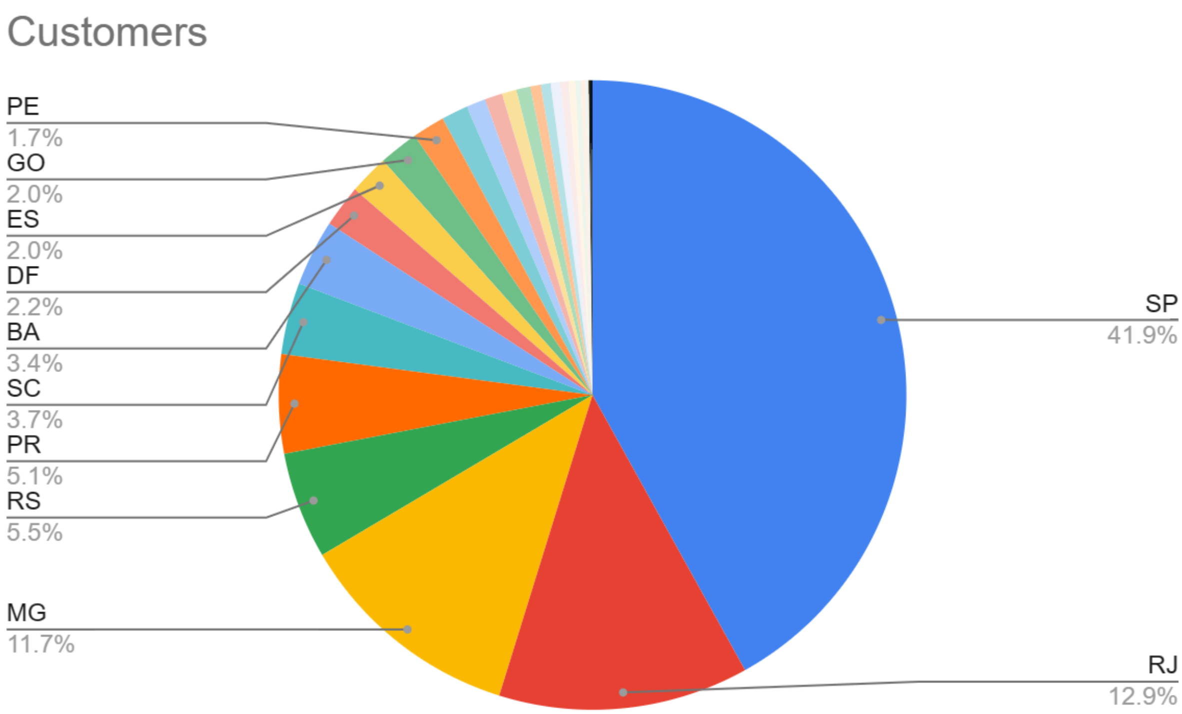
Results per page: 50

Inference:

The query involves determining the number of customers in each state. My approach:
-Grouping by State: We start by grouping the data based on the state column.

-Counting Unique Customers: To find the unique number of customers in each state, we apply the COUNT(DISTINCT customer_id) function.
-Ordering the Results: Finally, we sort the results by the Customers column in descending order.

This analysis provides insights into the distribution of customers across different states and we found that the most no of customers are from SP state.



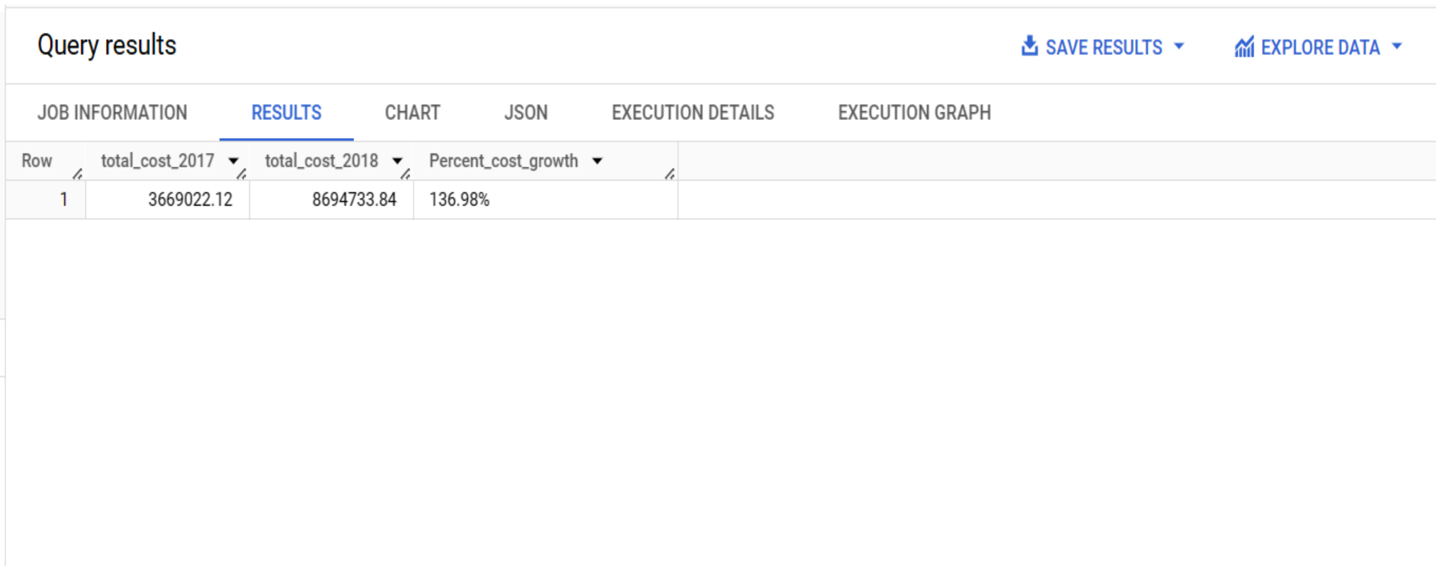
4. Impact on Economy: Analyze the money movement by e-commerce by looking at order prices, freight and others:

(1) Get the % increase in the cost of orders from year 2017 to 2018 (include months between Jan to Aug only). You can use the "payment_value" column in the payments table to get the cost of orders.

Query:

```
SELECT
  round(SUM(CASE WHEN EXTRACT(YEAR FROM o.order_purchase_timestamp) = 2017 THEN
p.payment_value ELSE 0 END),2) AS total_cost_2017,
  round(SUM(CASE WHEN EXTRACT(YEAR FROM o.order_purchase_timestamp) = 2018 THEN
p.payment_value ELSE 0 END),2) AS total_cost_2018,
  concat(round((SUM(CASE WHEN EXTRACT(YEAR FROM o.order_purchase_timestamp) = 2018
THEN p.payment_value ELSE 0 END)
   - SUM(CASE WHEN EXTRACT(YEAR FROM o.order_purchase_timestamp) = 2017 THEN
p.payment_value ELSE 0 END))
  * 100.0 / IFNULL(SUM(CASE WHEN EXTRACT(YEAR FROM o.order_purchase_timestamp) = 2017
THEN p.payment_value ELSE 0 END), 0),2),'%') AS Percent_cost_growth
FROM `Target_abhi.orders` o
JOIN `Target_abhi.payments` p ON o.order_id = p.order_id
WHERE EXTRACT(MONTH FROM o.order_purchase_timestamp) IN (1, 2, 3, 4, 5, 6, 7, 8);
```

Result:



Row	total_cost_2017	total_cost_2018	Percent_cost_growth
1	3669022.12	8694733.84	136.98%

Inference:

1.We calculate the total order costs for the years 2017 and 2018.
For 2017: Sum the payment values for orders placed in that year.
For 2018: Sum the payment values for orders placed in that year.
2. Next, we compute the percentage growth in costs from 2017 to 2018:
Subtract the total cost in 2017 from the total cost in 2018.
Divide the difference by the total cost in 2017 (avoiding division by zero using IFNULL).
Multiply by 100 and round to two decimal places.
Append a percentage sign to the result.
3.The query joins the orders and payments tables based on the order_id.
4. We filter the results to include only orders placed in the first eight months of the year.

The analysis of order costs reveals interesting trends. In 2018, the total order costs increased significantly compared to 2017. The percentage growth indicates a positive trajectory, suggesting improved business performance.

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

Query:

```
select distinct customer_state,
       round(sum(price) over(partition by customer_state),2) as Total_order_price,
       round(avg(price) over(partition by customer_state),2) as Average_order_price
from `Target_abhi.order_items` i join `Target_abhi.orders` o on o.order_id=i.order_id
join `Target_abhi.customers` c using(customer_id)
order by 1
```

Result:

Query results

JOB INFORMATION RESULTS CHART JSON EXECUTION DETAILS EXECUTION GRAPH

Row	customer_state	Total_order_price	Average_order_price
1	AC	15982.95	173.73
2	AL	80314.81	180.89
3	AM	22356.84	135.5
4	AP	13474.3	164.32
5	BA	511349.99	134.6
6	CE	227254.71	153.76
7	DF	302603.94	125.77
8	ES	275037.31	121.91
9	GO	294591.95	126.27
10	MA	119648.22	145.2
11	MG	1585308.03	120.75
12	MS	116812.64	142.63
13	MT	156453.53	148.3
14	PA	178947.81	165.69
15	PB	115268.08	191.48
16	PE	262788.03	145.51

Results per page: 50

Inference:

We had to calculate the average and total price of orders for each state. We have two tables: Customers (containing state information) and Order_Items (with order prices). Unfortunately, there's no direct common column between these tables for an inner join. However, both tables share a common link with the Orders table. Here's what I did:

We start by performing an inner join between the Customers and Orders tables using the customer_id as the connecting column.
Next, we perform another inner join, this time with the Order_Items table, using the order_id column.
We retrieve distinct customer_state values.
The SUM(price) OVER (PARTITION BY customer_state) calculates the cumulative sum of order prices for each state.
Similarly, AVG(price) OVER (PARTITION BY customer_state) computes the average order price for each state.

Finally, we round the calculated values to two decimal places for clarity.

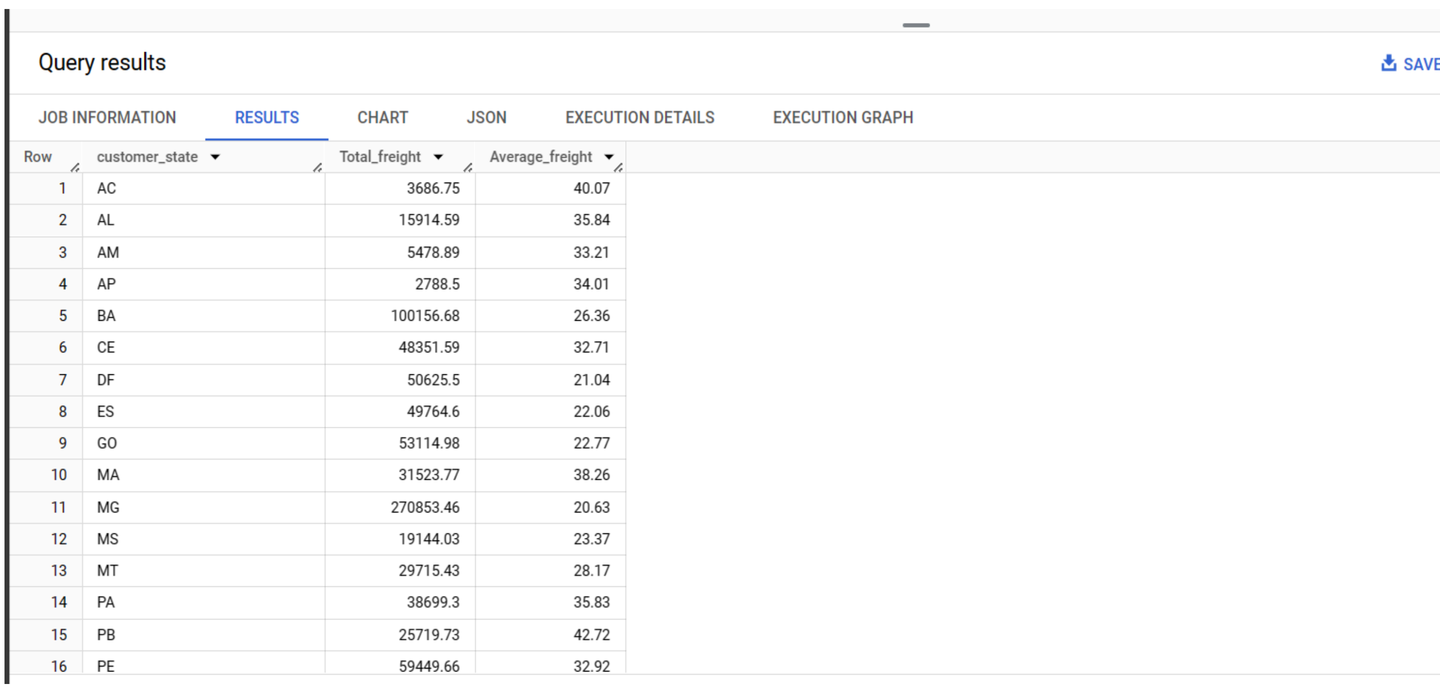
The analysis of order data reveals distinct patterns across different states. By calculating the average and total order prices for each state, we gain valuable insights into regional spending behavior.

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

Query:

```
select distinct customer_state,
       round(sum(freight_value) over(partition by customer_state),2) as
Total_freight_value,
       round(avg(freight_value) over(partition by customer_state),2) as
Average_freight_value
from `Target_abhi.order_items` i join `Target_abhi.orders` o on o.order_id=i.order_id
join `Target_abhi.customers` c using(customer_id)
order by 1
```

Result:



Query results

SAVE

JOB INFORMATION RESULTS CHART JSON EXECUTION DETAILS EXECUTION GRAPH

Row	customer_state	Total_freight	Average_freight
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
16	PE	59449.66	32.92

Inference:

We retrieve distinct customer_state values.
The SUM(freight_value) OVER (PARTITION BY customer_state) calculates the cumulative sum of freight costs for each state.
Similarly, AVG(freight_value) OVER (PARTITION BY customer_state) computes the average freight cost for each state.
The query joins the Order_Items, Orders, and Customers tables based on common columns (order_id and customer_id).

Analyzing freight costs by state provides valuable insights. By understanding the total and average freight expenses for each state, businesses can optimize logistics, negotiate better shipping rates, and enhance overall supply chain efficiency

5. 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.

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 DISTINCT
    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_abhi.orders`
WHERE lower(order_status)='delivered' and DATE_DIFF(order_delivered_customer_date,
order_purchase_timestamp, DAY) IS NOT NULL
    AND DATE_DIFF(order_estimated_delivery_date, order_delivered_customer_date, DAY)
IS NOT NULL
```

Result:

Query results

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

Row	order_id	time_to_deliver	diff_estimated_delive
1	635c894d068ac37e6e03dc54e...	30	1
2	3b97562c3aee8bdedcb5c2e45...	32	0
3	68f47f50f04c4cb6774570cfde...	29	1
4	276e9ec344d3bf029ff83a161c...	43	-4
5	54e1a3c2b97fb0809da548a59...	40	-4
6	fd04fa4105ee8045f6a0139ca5...	37	-1
7	302bb8109d097a9fc6e9cefc5...	33	-5
8	66057d37308e787052a32828...	38	-6
9	19135c945c554eebfd7576c73...	36	-2
10	4493e45e7ca1084efcd38ddeb...	34	0
11	70c77e51e0f179d75a64a6141...	42	-11
12	d7918e406132d7c81f1b84527...	35	-3
13	43f6604e77ce6433e7d68dd86...	32	-7
14	37073d851c3f30deebe598e5a...	31	-9
15	d064d4d070d914984df257750...	29	0
16	61d430273ff1e88f2944acb53e...	30	0
17	d2f8ef9dd1714fcac7de9f0aef1...	30	-8

Results per p

Inference:

The query utilizes the date_diff function to calculate the delivery time for each order. This function subtracts the order_purchase_timestamp from the order_delivered_customer_date, resulting in the number of days between purchase and delivery.

The same logic applies to calculating the difference between the estimated delivery date and the actual delivery date. However, it's worth noting that negative values for delivery time may occur. This indicates that the order arrived earlier than expected. Additionally, a value of zero signifies that the estimated and actual delivery dates coincided.

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

Query:

```
with cte as(
  select customer_state,
         round(avg(freight_value),2) as Average_freight_value
  from `Target_abhi.order_items` i join `Target_abhi.orders` o on
o.order_id=i.order_id join `Target_abhi.customers` c using(customer_id) WHERE
lower(order_status)='delivered' group by 1
)


(select  distinct customer_state, Average_freight_value, 'Top 5 States' as
Top_bottom_states  from cte order by Average_freight_value desc limit 5)

union distinct


(select  distinct customer_state, Average_freight_value, 'Bottom 5 States' as
Top_bottom_states from cte order by Average_freight_value limit 5)
```

Result:

Query results

SAVE RE

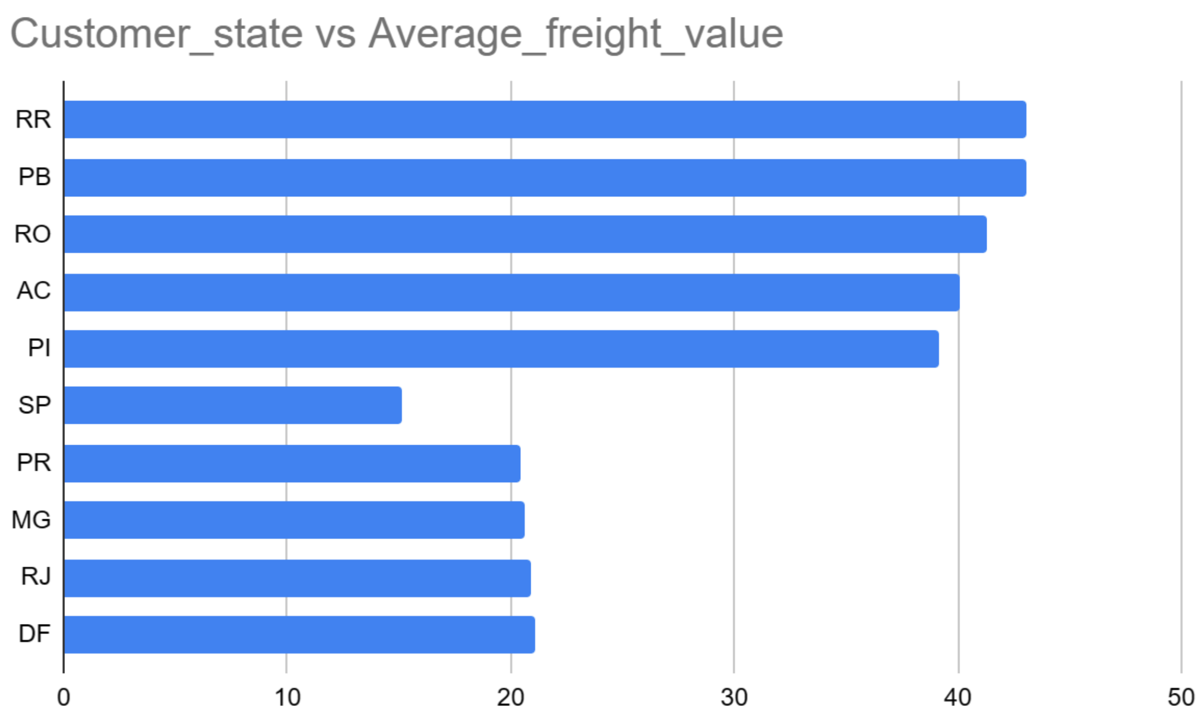
JOB INFORMATION | RESULTS | CHART | JSON | EXECUTION DETAILS | EXECUTION GRAPH

Row	customer_state	Average_freight_valu	Top_bottom_states
1	PB	43.09	Top 5 States
2	RR	43.09	Top 5 States
3	RO	41.33	Top 5 States
4	AC	40.05	Top 5 States
5	PI	39.12	Top 5 States
6	SP	15.12	Bottom 5 States
7	PR	20.47	Bottom 5 States
8	MG	20.63	Bottom 5 States
9	RJ	20.91	Bottom 5 States
10	DF	21.07	Bottom 5 States

Inference:

We created a Common Table Expression (CTE) named cte.
Within the CTE, it selects the customer_state and computes the average freight value using the avg(freight_value) function.
The group by clause ensures that the average is calculated separately for each state we filter based on order_status
-Top 5 States with Highest Average Freight Value:
The first part of the final result is obtained by querying the CTE.
It selects distinct customer_state, Average_freight_value, and labels it as ‘Top 5 States’.
The results are ordered by Average_freight_value in descending order and limited to 5 rows.
-Bottom 5 States with Lowest Average Freight Value:
The second part of the final result is obtained similarly.
It selects distinct customer_state, Average_freight_value, and labels it as ‘Bottom 5 States’.
The results are ordered by Average_freight_value in ascending order and limited to 5 rows.

This query will provide insights into the states with varying average freight values,


Customer_state vs Average_freight_value
RR
PB
RO
AC
PI
SP
PR
MG
RJ
DF
0
10
20
30
40
50

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

Query:

```
with cte as(
  select customer_state,
         round(avg(DATE_DIFF(order_delivered_customer_date, order_purchase_timestamp,
DAY)),2) AS time_to_deliver
  from `Target_abhi.orders` o join `Target_abhi.customers` c  using(customer_id) WHERE
lower(order_status)='delivered' group by 1
)

(select  distinct customer_state, time_to_deliver, 'Top 5 States' as Top_bottom_states
from cte order by 2 desc limit 5)

union distinct

(select  distinct customer_state, time_to_deliver, 'Bottom 5 States' as
Top_bottom_states from cte order by 2 limit 5)
```

Result:

Query results

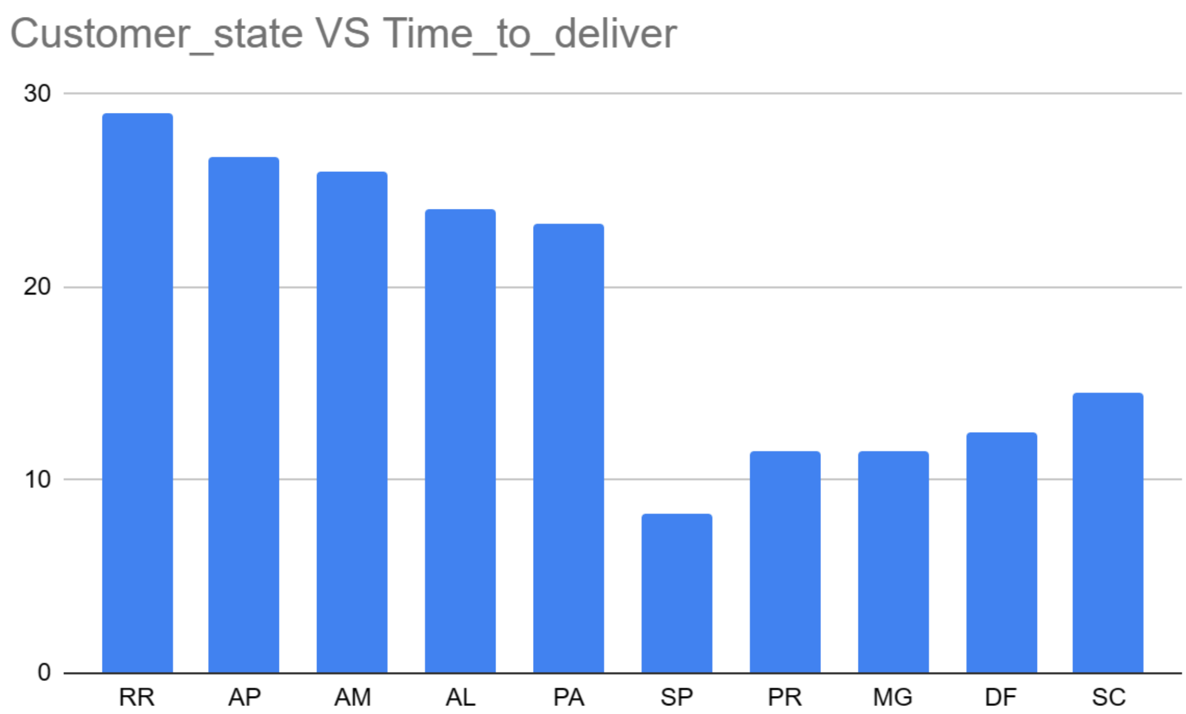
JOB INFORMATION RESULTS CHART JSON EXECUTION DETAILS EXECUTION GRAPH

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

Inference:
I started calculating the average delivery time for each state using the avg(), datediff(), and group by functions.
The average delivery time is computed in days by finding the difference between the order_delivered_customer_date and order_purchase_timestamp.

-To determine the top 5 states with the highest average delivery time:
I ordered the results by average_delivery_time in descending (DESC) order.
Then, I limited the output to the top 5 states.

-For the bottom 5 states with the lowest average delivery time:
I ordered the results by average_delivery_time in ascending (ASC) order.
Again, I limited the output to the bottom 5 states.
In summary, this query provides insights into the 5 top and bottom states with varying average delivery times


Customer_state VS Time_to_deliver
30
20
10
0
RR
AP
AM
AL
PA
SP
PR
MG
DF
SC

(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.

Query:

```
select c.customer_state,

round(avg(DATE_DIFF(o.order_delivered_customer_date,o.order_estimated_delivery_date,
DAY)),2) AS diff_estimated_delivery
  from `Target_abhi.orders` o join `Target_abhi.customers` c  using(customer_id) WHERE
lower(order_status)='delivered' group by 1 order by 2 asc limit 5
```

Result:

Query results SAVE RESULTS EXPLO

JOB INFORMATION RESULTS CHART JSON EXECUTION DETAILS EXECUTION GRAPH

Row	customer_state	diff_estimated_delive
1	AC	-19.76
2	RO	-19.13
3	AP	-18.73
4	AM	-18.61
5	RR	-16.41

Job history

Inference:

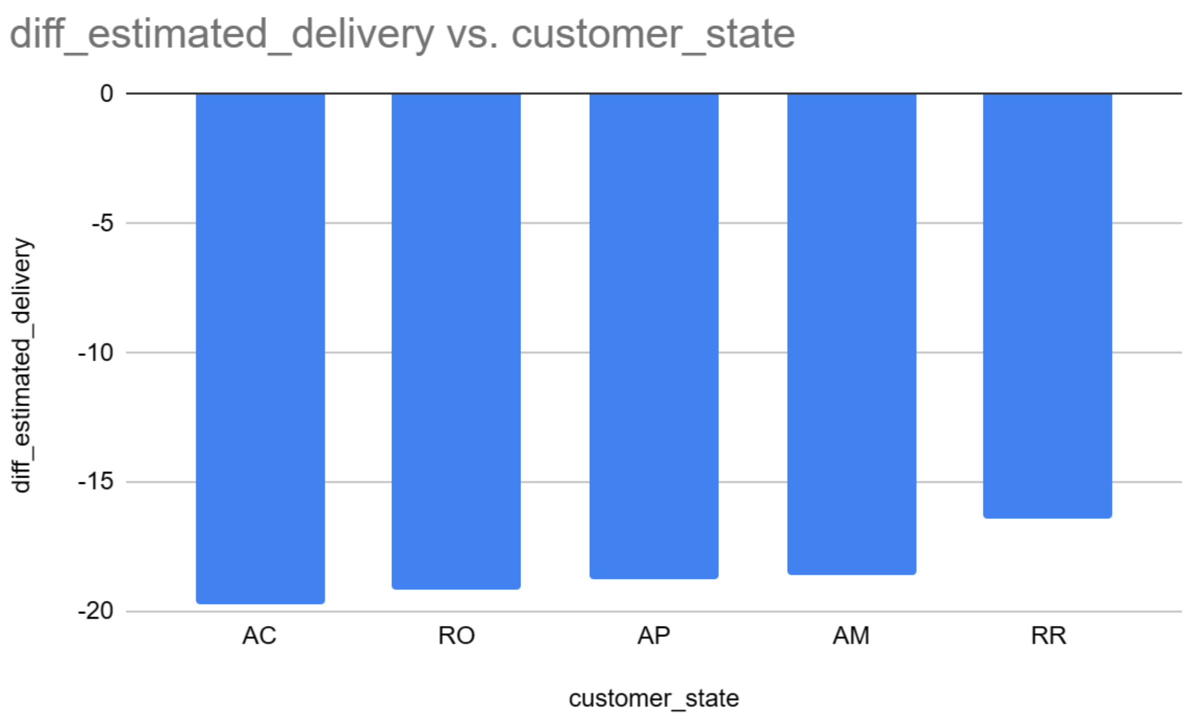
We are selecting two columns:
c.customer_state
round(avg(DATE_DIFF(o.order_delivered_customer_date, o.order_estimated_delivery_date, DAY)), 2) (aliased as diff_estimated_delivery)
The first column represents the state of the customer.
The second column calculates the average difference in days between the actual delivery date (order_delivered_customer_date) and the estimated delivery date (order_estimated_delivery_date). The result is rounded to two decimal places.

Then we are joining two tables:
Target_abhi.orders (aliased as o)
Target_abhi.customers (aliased as c)
The join condition is using(customer_id), which means we are joining the tables based on the common column customer_id.

We filter the rows where the order_status (converted to lowercase) is equal to 'delivered' and we group the results by the customer_state.

The results are sorted in ascending order based on the calculated average delivery time (diff_estimated_delivery).
LIMIT Clause:
Finally, we limit the output to the top 5 rows.

The query aims to identify the top 5 states where the actual delivery time is faster than the estimated delivery time.
By calculating the average difference between actual and estimated delivery dates, we can determine which states have the most efficient delivery performance.
The negative values for diff_estimated_delivery indicate that, on average, orders in those states are delivered earlier than expected.
The ORDER BY clause in ascending order ensures that the states with the fastest delivery times appear at the top of the result set i.e. AC in this case


diff_estimated_delivery vs. customer_state
0
-5
-10
-15
-20
diff_estimated_delivery
AC
RO
AP
AM
RR
customer_state

6. Analysis based on the payments:

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

Query:

```
WITH cte AS (
  SELECT
    P.payment_type,
    EXTRACT(YEAR FROM O.order_purchase_timestamp) AS Year,
    EXTRACT(MONTH FROM O.order_purchase_timestamp) AS Month,
    COUNT(*) AS payment_type_count
  FROM
    `Target_abhi.payments` P
  JOIN
     `Target_abhi.orders`  O
  ON P.order_id = O.order_id
  GROUP BY P.payment_type, Year, Month
),
cte2 AS (
  SELECT
    payment_type,
    Year,
    Month,
    payment_type_count,
    LAG(payment_type_count) OVER(PARTITION BY payment_type ORDER BY Year, Month) AS
prev_count
  FROM cte
)
SELECT
  *,
  round(((payment_type_count - prev_count) / prev_count) * 100) AS
growth_rate_in_count
FROM cte2 order by cte2.payment_type;
```

Result:

Query results

JOB INFORMATION RESULTS CHART JSON EXECUTION DETAILS EXECUTION GRAPH

Row	payment_type	Year	Month	payment_type_count	prev_count	growth_rate_in_coun
1	UPI	2016	10	63	null	null
2	UPI	2017	1	197	63	213.0
3	UPI	2017	2	398	197	102.0
4	UPI	2017	3	590	398	48.0
5	UPI	2017	4	496	590	-16.0
6	UPI	2017	5	772	496	56.0
7	UPI	2017	6	707	772	-8.0
8	UPI	2017	7	845	707	20.0
9	UPI	2017	8	938	845	11.0
10	UPI	2017	9	903	938	-4.0
11	UPI	2017	10	993	903	10.0
12	UPI	2017	11	1509	993	52.0
13	UPI	2017	12	1160	1509	-23.0
14	UPI	2018	1	1518	1160	31.0
15	UPI	2018	2	1325	1518	-13.0
16	UPI	2018	3	1352	1325	2.0
17	UPI	2018	4	1287	1352	-5.0

Results

Inference:

We defined two CTEs: cte and cte2.
cte calculates the count of each payment type for delivered orders, grouped by year and month.
cte2 builds upon cte and adds a column with the previous month's payment type count using the LAG() window function.

The main query selects all columns from cte2.
It also computes the growth rate in payment type count by comparing the current count with the previous count.
I calculated the growth rate as:
growth_rate_in_count=(payment_type_count−prev_count)×100/prev_count

The result is ordered by payment type.

The query aims to analyze the growth rate of payment types over time.
By comparing the current count with the previous count, it provides insights into how payment types are changing month by month

(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 orders_count
from `Target_abhi.payments` where payment_installments>=1
group by 1 order by 2 desc
```

Result:

Query results

JOB INFORMATION RESULTS CHART JSON EXECUTION DETAILS EXECUTION GRAPH

Row	payment_installment	orders_count
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

Inference:

This query retrieves the number of orders for each value of payment_installments where the installment count is greater than or equal to 1.
It groups the results by the payment_installments value and sorts them in descending order based on the orders_count.

The count(order_id) function calculates the number of orders for each payment_installments value.
The group by 1 clause groups the results by the first column (i.e., payment_installments).
The order by 2 desc clause sorts the results based on the second column (i.e., orders_count) in descending order.

The result of this query provides insights into the distribution of orders based on the number of payment installments