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.

```
select column_name, data_type
from Target_SQL.INFORMATION_SCHEMA.COLUMNS
where table_name = 'customers'
```

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS

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: Most columns are of string type.

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

```
select
min(order_purchase_timestamp) as earliest_order_time,
max(order_purchase_timestamp) as latest_order_time
from Target_SQL.orders
```

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS

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

Insights: This market in Brazil lasted for 2 years with starting in 2016 and ending in 2018.

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

```
SELECT
DISTINCT customer_city AS City,customer_state AS State,
COUNT(*) AS Count
FROM Target_SQL.customers c
JOIN Target_SQL.orders o ON c.customer_id = o.customer_id
WHERE o.order_purchase_timestamp BETWEEN '2016-01-01' AND
'2018-12-31'
GROUP BY customer_city, customer_state;
```

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

Row	City	State	Count
1	acu	RN	3
2	ico	CE	8
3	ipe	RS	2
4	ipu	CE	4
5	ita	SC	3
6	itu	SP	136
7	jau	SP	74
8	luz	MG	2
9	poa	SP	85
10	uba	MG	53

Insights: There are more than 4000 unique cities but screenshot is taken of just ten.

.....

2. **In-depth Exploration:**

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

```
SELECT
EXTRACT(YEAR FROM order_purchase_timestamp) AS
order_year,
EXTRACT(MONTH FROM order_purchase_timestamp) AS
order_month,
```

```
COUNT(*) AS order_count
FROM Target_SQL.orders
GROUP BY order_year, order_month
ORDER BY order_year, order_month;
```

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS

Row	order_year	order_month	order_count
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

Insights: We can see the growth in number of orders over the years. These are the 10 results out of 25.

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

```
SELECT
EXTRACT(MONTH FROM order_purchase_timestamp) AS
order_month,
COUNT(*) AS order_count
FROM Target_SQL.orders
GROUP BY order_month
ORDER BY order_count DESC;
```

JOB INFORMATION | RESULTS | JSON

Row	order_month	order_count
1	8	10843
2	5	10573
3	7	10318
4	3	9893
5	6	9412
6	4	9343
7	2	8508
8	1	8069
9	11	7544
10	12	5674
11	10	4959
12	9	4305

Insights: In the month of August, the number of orders is highest, i.e., 10843.

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

```
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(*) as order_cnt
from Target_SQL.orders
group by time_of_the_day
order by time_of_the_day;
```

JOB INFORMATION | RESULTS | JSON | EXECUTIC

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

Insights: Customers mostly place their orders in Afternoon.

.....

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

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

```
SELECT c.customer_state AS state,
EXTRACT(MONTH FROM o.order_purchase_timestamp) AS
order_month,
COUNT(*) AS order_count
FROM Target_SQL.orders o
JOIN Target_SQL.customers c
ON o.customer_id = c.customer_id
GROUP BY state, order_month
ORDER BY state, order_month;
```

Row	state	order_month	order_count
1	AC	1	8
2	AC	2	6
3	AC	3	4
4	AC	4	9
5	AC	5	10
6	AC	6	7
7	AC	7	9
8	AC	8	7
9	AC	9	5
10	AC	10	6
11	AC	11	5
12	AC	12	5
13	AL	1	39
14	AL	2	39

Insights： We can see the results of no of orders monthwise by states AC, AL and so on..

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

```
SELECT customer_state AS state,
COUNT(DISTINCT customer_id) AS unique_customer_count
FROM Target_SQL.customers
GROUP BY customer_state;
```

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS

Row	state	unique_customer_count
1	RN	485
2	CE	1336
3	RS	5466
4	SC	3637
5	SP	41746
6	MG	11635
7	BA	3380
8	RJ	12852
9	GO	2020
10	MA	747
11	PE	1652

Insights: Out of total 27 states, here are the results of 10 states in which state SP have the most unique customers.

.....

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

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

```
SELECT c.customer_state AS state,
SUM(p.payment_value) AS total_order_value,
AVG(p.payment_value) AS average_order_value
FROM Target_SQL.customers c
JOIN Target_SQL.orders o ON c.customer_id = o.customer_id
```

```
JOIN Target_SQL.payments p ON o.order_id = p.order_id
GROUP BY c.customer_state
ORDER BY c.customer_state;
```

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS

Row	state	total_order_value	average_order_value
1	AC	19680.61999999...	234.2930952380...
2	AL	96962.05999999...	227.0774238875...
3	AM	27966.93	181.6034415584...
4	AP	16262.79999999...	232.3257142857...
5	BA	616645.8200000...	170.8160166204...
6	CE	279464.0299999...	199.9027396280...
7	DF	355141.0800000...	161.1347912885...
8	ES	325967.55	154.7069530137...
9	GO	350092.3100000...	165.7634043560...
10	MA	152523.0200000...	198.8566101694...
11	MG	1872257.260000...	154.7064336473...
12	MS	137534.8399999...	186.8679891304...
13	MT	187029.2900000...	195.2289039665...

Insights: 13 results shown out of 27 in which states are arranged in ascending order.

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

```
SELECT
c.customer_state AS state,
SUM(oi.freight_value) AS total_freight_value,
AVG(oi.freight_value) AS average_freight_value
FROM Target_SQL.order_items oi
JOIN Target_SQL.orders o
ON oi.order_id = o.order_id
JOIN Target_SQL.customers c
ON o.customer_id = c.customer_id
GROUP BY c.customer_state
ORDER BY c.customer_state;
```

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS

Row	state	total_freight_value	average_freight_valu
1	AC	3686.750000000...	40.07336956521...
2	AL	15914.58999999...	35.84367117117...
3	AM	5478.890000000...	33.20539393939...
4	AP	2788.500000000...	34.00609756097...
5	BA	100156.6799999...	26.36395893656...
6	CE	48351.58999999...	32.71420162381...
7	DF	50625.49999999...	21.04135494596...
8	ES	49764.59999999...	22.05877659574...
9	GO	53114.97999999...	22.76681525932...
10	MA	31523.77000000...	38.25700242718...

Insights: Freight values are shown above according to respective states arranged in ascending order.

.....

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:
        a. **time_to_deliver** = order_delivered_customer_date - order_purchase_timestamp
        b. **diff_estimated_delivery** = order_estimated_delivery_date - order_delivered_customer_date

```
SELECT order_id,
DATE_DIFF(order_delivered_customer_date,
order_purchase_timestamp, DAY) AS time_to_deliver,
DATE_DIFF(order_estimated_delivery_date,
order_delivered_customer_date, DAY) AS
diff_estimated_delivery
```

```
FROM Target_SQL.orders;
```

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS

Row	order_id	time_to_deliver	diff_estimated_delive
1	1950d777989f6a877539f5379...	30	-12
2	2c45c33d2f9cb8ff8b1c86cc28...	30	28
3	65d1e226dfaeb8cdc42f66542...	35	16
4	635c894d068ac37e6e03dc54e...	30	1
5	3b97562c3aee8bdedcb5c2e45...	32	0
6	68f47f50f04c4cb6774570cfde...	29	1
7	276e9ec344d3bf029ff83a161c...	43	-4
8	54e1a3c2b97fb0809da548a59...	40	-4
9	fd04fa4105ee8045f6a0139ca5...	37	-1
10	302bb8109d097a9fc6e9cefc5...	33	-5

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

   For highest average freight value:

```
SELECT c.customer_state AS state,
ROUND(AVG(freight_value), 2) AS
high_average_freight_value
FROM Target_SQL.customers c
JOIN Target_SQL.orders o using(customer_id)
JOIN Target_SQL.order_items oi using(order_id)
GROUP BY customer_state
ORDER BY high_average_freight_value DESC
LIMIT 5;
```

JOB INFORMATION | RESULTS | JSON | EXECUT

Row	state	high_average_freight
1	RR	42.98
2	PB	42.72
3	RO	41.07
4	AC	40.07
5	PI	39.15

Insights: Out of five, RR state has the highest average freight value.

For lowest average freight value:

```
SELECT c.customer_state AS state,
ROUND(AVG(freight_value), 2) AS low_average_freight_value
FROM Target_SQL.customers c
JOIN Target_SQL.orders o using(customer_id)
JOIN Target_SQL.order_items oi using(order_id)
GROUP BY customer_state
ORDER BY low_average_freight_value ASC
LIMIT 5;
```

JOB INFORMATION | RESULTS | JSON | EXEC

Row	state	low_average_freight_
1	SP	15.15
2	PR	20.53
3	MG	20.63
4	RJ	20.96
5	DF	21.04

Insights: Out of five, SP state has the lowest average freight value.

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

For highest average delivery time:

```
SELECT c.customer_state AS state,
AVG(DATE_DIFF(order_delivered_customer_date,
order_purchase_timestamp, day)) AS avg_delivery_time
FROM Target_SQL.customers c
JOIN Target_SQL.orders o using(customer_id)
JOIN Target_SQL.order_items oi using(order_id)
GROUP BY c.customer_state
ORDER BY avg_delivery_time DESC
LIMIT 5;
```

JOB INFORMATION | RESULTS | JSON | EXE

Row	state	avg_delivery_time
1	RR	27.82608695652...
2	AP	27.75308641975...
3	AM	25.96319018404...
4	AL	23.99297423887...
5	PA	23.30170777988...

Insights: Out of five, RR state has the highest average delivery time.

For lowest average delivery time:

```
SELECT c.customer_state AS state,
AVG(DATE_DIFF(order_delivered_customer_date,
order_purchase_timestamp, day)) AS avg_delivery_time
FROM Target_SQL.customers c
JOIN Target_SQL.orders o using(customer_id)
JOIN Target_SQL.order_items oi using(order_id)
GROUP BY c.customer_state
ORDER BY avg_delivery_time
LIMIT 5;
```

JOB INFORMATION | RESULTS | JSON | EXE

Row	state	avg_delivery_time
1	SP	8.259608552419...
2	PR	11.48079306071...
3	MG	11.51552218007...
4	DF	12.50148619957...
5	SC	14.52098584675...

Insights: Out of five, SP state has the 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.
   You can use the difference between the averages of actual & estimated delivery date to figure out how fast the delivery was for each state.

```
SELECT c.customer_state AS state,
AVG(DATE_DIFF(o.order_estimated_delivery_date,
o.order_delivered_customer_date, day)) AS
avg_delivery_time_difference
FROM Target_SQL.customers c
JOIN Target_SQL.orders o ON c.customer_id = o.customer_id
GROUP BY c.customer_state
ORDER BY avg_delivery_time_difference ASC
LIMIT 5;
```

JOB INFORMATION | RESULTS | JSON | EXE

Row	state	avg_delivery_time_di
1	AL	7.9471032745592
2	MA	8.768479776847...
3	SE	9.173134328358...
4	ES	9.618546365914...
5	BA	9.934889434889...

Insights: These are the top 5 states where the order delivery is fast as compared to the estimated date of delivery.

6. **Analysis based on the payments:**

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

```
SELECT p.payment_installments,
COUNT(o.order_id) AS num_orders
FROM Target_SQL.payments p
JOIN Target_SQL.orders o ON p.order_id = o.order_id
GROUP BY p.payment_installments
ORDER BY p.payment_installments;
```

JOB INFORMATION | RESULTS | JSON

Row	payment_installment	num_orders
1	0	2
2	1	52546
3	2	12413
4	3	10461
5	4	7098
6	5	5239
7	6	3920
8	7	1626
9	8	4268
10	9	644

Insights: Out of 27 results, 10 results shown above.