TARGET SQL – Case Study

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.

```sql
SELECT
  column_name,
  data_type
FROM
  target_sql.INFORMATION_SCHEMA.COLUMNS
WHERE
  table_name='customers';
```

Query results

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

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

The above image gives the datatype information of all columns in the “customers” table

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

```sql
SELECT
  MIN(order_purchase_timestamp) AS first_order_timestamp,
  MAX(order_purchase_timestamp) AS last_order_timestamp
FROM
  target_sql.orders
```

Query results

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS

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

Seeing the data, the orders were placed between 4th Sep 2016 and 17th Oct 2018

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

```sql
SELECT
  COUNT(DISTINCT customer_city) AS num_of_cities,
  COUNT(DISTINCT customer_state) AS num_of_states
FROM
  `target_sql.customers`
WHERE
  customer_id IN (
  SELECT
    customer_id
  FROM
    `target_sql.orders`);
```

Query results

JOB INFORMATION | RESULTS | JSON | EXECUTIO

Row	num_of_cities	num_of_states
1	4119	27

Customers ordered during the given period belong to 4119 cities and 27 states

2. **In-depth Exploration:**

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

```sql
SELECT
  EXTRACT(year
  FROM
    order_purchase_timestamp) AS year,
  COUNT(order_id) AS num_of_orders
FROM
  `target_sql.orders`
GROUP BY
  year
ORDER BY
  year asc
```

Query results

JOB INFORMATION | RESULTS | JSON

Row	year	num_of_orders
1	2016	329
2	2017	45101
3	2018	54011

YES, there is a growing trend in the no. of orders placed over the past years. There was a huge increase in orders from year 2016 to 2017

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

```sql
SELECT
  EXTRACT(year
  FROM
    order_purchase_timestamp) AS year,
  EXTRACT(month
  FROM
    order_purchase_timestamp) AS month,
  COUNT(order_id) AS num_of_orders
FROM
  `target_sql.orders`
GROUP BY
  year,
  month
ORDER BY
  year,
  month ASC
```

Query results

SAVE RESULTS

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

Row	year	month	num_of_orders
1	2016	9	4
2	2016	10	324
3	2016	12	1
4	2017	1	800
5	2017	2	1780
6	2017	3	2682
7	2017	4	2404
8	2017	5	3700
9	2017	6	3245
10	2017	7	4026
11	2017	8	4331
12	2017	9	4285
13	2017	10	4631

Results per page: 50 1 – 25 of 25

- There is no monthly seasonality observed
- In the year 2017 we can observe an increasing trend in num_of_orders from the month Jan to Nov
- In the year 2016 and 2018(Q3-Q4) sales were comparatively very less

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

```sql
WITH
  CTE1 AS (
  SELECT
    order_id,
    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'
    ELSE
    'Night'
  END
    AS time_of_day
  FROM
    `target_sql.orders`)
SELECT
  time_of_day,
  COUNT(order_id) AS count
FROM
  CTE1
GROUP BY
  time_of_day
ORDER BY
  count DESC
LIMIT
  1;
```

Query results

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

Row	time_of_day	count
1	Afternoon	38135

Maximum number of orders were placed during the afternoon hours

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

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

```sql
SELECT
  c.customer_state AS state,
  EXTRACT(month
  FROM
    o.order_purchase_timestamp) AS month,
  COUNT(order_id) AS num_of_orders
FROM
  target_sql.orders o
INNER JOIN
  target_sql.customers c
ON
  o.customer_id = c.customer_id
GROUP BY
  state,
  month
ORDER BY
  state,
  month;
```

Query results

SAVE RESULTS

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

Row	state	month	num_of_orders
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

Results per page: 50 1 – 50 of 322

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

```
SELECT
  customer_state as state,
  COUNT(customer_id) AS num_of_cust
FROM
  `target_sql.customers`
GROUP BY
  state
ORDER BY
  num_of_cust DESC
```

Query results

SAVE RESULTS

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

Row	state	num_of_cust
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
12	CE	1336
13	PA	975
14	MT	907
15	MA	747
16	MS	715
17	PB	536

Results per page: 50 1 – 27 of 27

State Sao Paulo (SP) has the most number of customers compared to other states

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.

```
with
  orders_2017 as
  (select
    extract(year from order_purchase_timestamp) as Year_2017,
    extract(month from order_purchase_timestamp) as month_ordered,
    round(sum(payment_value),2) as total_payment_value_2017
  from
    `target_sql.payments` p
  inner join
    `target_sql.orders` o
  on
    o.order_id = p.order_id
  where
    extract(month from order_purchase_timestamp) between 1 and 8
    and extract(year from order_purchase_timestamp) = 2017
  group by Year_2017,month_ordered
  order by month_ordered,Year_2017),
  orders_2018 as
    (select
      extract(year from order_purchase_timestamp) as Year_2018,
      extract(month from order_purchase_timestamp) as month_ordered,
      round(sum(payment_value),2) as total_payment_value_2018
    from
      `target_sql.payments` p
    inner join
      `target_sql.orders` o
    on
      o.order_id = p.order_id
    where
      extract(month from order_purchase_timestamp) between 1 and 8
      and extract(year from order_purchase_timestamp) = 2018
    group by Year_2018,month_ordered
    order by month_ordered,Year_2018)
select o17.month_ordered,Year_2017,total_payment_value_2017,Year_2018,total_payment_value_2018,
      round(((total_payment_value_2018-total_payment_value_2017)/total_payment_value_2017)*100,2) as percentage_increase from orders_2017 o17
inner join
  orders_2018 o18
on o17.month_ordered = o18.month_ordered
order by o17.month_ordered;
```

Query results

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

Row	month_ordered	Year_2017	total_payment_value	Year_2018	total_payment_value	percentage_increase
1	1	2017	138488.04	2018	1115004.18	705.13
2	2	2017	291908.01	2018	992463.34	239.99
3	3	2017	449863.6	2018	1159652.12	157.78
4	4	2017	417788.03	2018	1160785.48	177.84
5	5	2017	592918.82	2018	1153982.15	94.63
6	6	2017	511276.38	2018	1023880.5	100.26
7	7	2017	592382.92	2018	1066540.75	80.04
8	8	2017	674396.32	2018	1022425.32	51.61

The above details give the month wise % increase in the cost of orders for the years 2017 and 2018. Looking at the stats above, we can see a positive increase in the cost of orders in every month of 2018 compared with 2017.

```sql
WITH orders_2017 AS (
  SELECT
    1 AS temp,
    EXTRACT(year FROM order_purchase_timestamp) AS Year_2017,
    ROUND(SUM(payment_value), 2) AS total_payment_value_2017
  FROM
    `target_sql.payments` p
  INNER JOIN
    `target_sql.orders` o ON o.order_id = p.order_id
  WHERE
    EXTRACT(month FROM order_purchase_timestamp) BETWEEN 1 AND 8
    AND EXTRACT(year FROM order_purchase_timestamp) = 2017
  GROUP BY
    Year_2017
  ORDER BY
    Year_2017),
orders_2018 AS (
  SELECT
    1 AS temp,
    EXTRACT(year FROM order_purchase_timestamp) AS Year_2018,
    ROUND(SUM(payment_value), 2) AS total_payment_value_2018
  FROM
    `target_sql.payments` p
  INNER JOIN
    `target_sql.orders` o ON o.order_id = p.order_id
  WHERE
    EXTRACT(month FROM order_purchase_timestamp) BETWEEN 1 AND 8
    AND EXTRACT(year FROM order_purchase_timestamp) = 2018
  GROUP BY
    Year_2018
  ORDER BY
    Year_2018)
SELECT
  Year_2017,total_payment_value_2017,Year_2018,total_payment_value_2018,
  ROUND(((total_payment_value_2018 - total_payment_value_2017) / total_payment_value_2017) * 100, 2) AS percentage_increase
FROM
  orders_2017 o17
INNER JOIN
  orders_2018 o18 ON o17.temp = o18.temp;
```

Query results

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

Row	Year_2017	total_payment_value	Year_2018	total_payment_value	percentage_increase
1	2017	3669022.12	2018	8694733.84	136.98

The above details give the % increase in the cost of orders for the years 2017 and 2018. Looking the stats above, we can a positive increase in the cost of orders in 2018 compared with 2017.

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

```sql
SELECT
  c.customer_state AS state,
  ROUND(SUM(i.price),2) AS total_price_value,
  COUNT(DISTINCT o.order_id) AS num_of_orders,
  ROUND(SUM(i.price)/COUNT(DISTINCT o.order_id),2) AS avg_price_value
FROM
  `target_sql.orders` o
INNER JOIN
  `target_sql.customers` c
ON
  o.customer_id = c.customer_id
INNER JOIN
  `target_sql.order_items` i
ON
  o.order_id = i.order_id
GROUP BY
  state
ORDER BY
  avg_price_value desc;
```

Query results

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

Row	state	total_price_value	num_of_orders	avg_price_value
17	BA	511349.99	3358	152.28
18	AM	22356.84	147	152.09
19	GO	294591.95	2007	146.78
20	SC	520553.34	3612	144.12
21	RJ	1824092.67	12762	142.93
22	DF	302603.94	2125	142.4
23	RS	750304.02	5432	138.13
24	MG	1585308.03	11544	137.33
25	PR	683083.76	4998	136.67
26	ES	275037.31	2025	135.82
27	SP	5202955.05	41375	125.75

For the state SP, even though the average price value is less, there seems to be a huge market in that state, which led to a higher number of orders and so higher price value

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

```sql
SELECT
  c.customer_state AS state,
  ROUND(SUM(i.freight_value),2) AS total_freight_value,
  COUNT(DISTINCT o.order_id) AS num_of_orders,
  ROUND(SUM(i.freight_value)/COUNT(DISTINCT o.order_id),2) AS avg_freight_value
FROM
  `target_sql.orders` o
INNER JOIN
  `target_sql.customers` c
ON
  o.customer_id = c.customer_id
INNER JOIN
  `target_sql.order_items` i
ON
  o.order_id = i.order_id
GROUP BY
  state
ORDER BY
  avg_freight_value desc;
```

Query results

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

Row	state	total_freight_value	num_of_orders	avg_freight_value
17	BA	100156.68	3358	29.83
18	MS	19144.03	709	27.0
19	GO	53114.98	2007	26.46
20	RS	135522.74	5432	24.95
21	SC	89660.26	3612	24.82
22	ES	49764.6	2025	24.58
23	RJ	305589.31	12762	23.95
24	DF	50625.5	2125	23.82
25	PR	117851.68	4998	23.58
26	MG	270853.46	11544	23.46
27	SP	718723.07	41375	17.37

Same case as above, for the state SP, even though the average freight value is less, there seems to be a huge market in that state, which led to a higher number of orders and so higher freight value

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_estimated_delivery_date - order_delivered_customer_date

```sql
SELECT
  distinct order_id,
  DATE_DIFF(order_delivered_carrier_date, order_purchase_timestamp,day) AS time_to_deliver,
  DATE_DIFF(order_estimated_delivery_date, order_delivered_carrier_date,day) AS diff_estimated_delivery
FROM
  `target_sql.orders`
WHERE
  order_delivered_carrier_date IS NOT NULL
order by time_to_deliver desc
```

Query results

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

Row	order_id	time_to_deliver	diff_estimated_delive
1	da81fbc27b55e0f3d2813cf207…	125	-99
2	97f48024fcc76f1898e397ad69…	107	-80
3	8b7fd198ad184563c23165367…	104	-90
4	866314550f6d7a55c82917d9b…	66	-39
5	5cc475c7c03290048eb2e742c…	62	-12
6	a4a57f1ffa25b90dea9f150fee8…	61	-10
7	2805499c211b52dfc1e64a134…	55	-31
8	2631dba338efbcea9c3ace77c…	55	39
9	7d86c4aa9e59504b23f16c7ca…	54	-11
10	bfbd0f9bdef84302105ad712d…	53	-34
11	5d6e9993ecc20a59e637ce711…	52	-14
12	bc9c2fb123725ad0d99fb7688…	49	-9
13	7141a7eee8944cb711fa1fd4f7…	49	-12
14	d8734ba226623cf1c86b3cce8…	49	-16
15	840d7cc10efae8ba460cc8ea8…	49	-19
16	abbbf52551bc34cd52a7851c0…	49	-3
17	e4d88b96fb1801eb436496008…	48	-13
18	8ab08a832f16966f270a9bbffe…	47	-18

On looking at the data, we found that there are multiple orders delivered on the same day. The most delayed order took 125 days to deliver which was 99 days later than the estimate delivery date.

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

Seller info highest

```sql
WITH
  sellers_info AS (
  SELECT
    s.seller_state AS state,
    ROUND(AVG(i.freight_value),2) AS avg_value
  FROM
    `target_sql.order_items` i
  INNER JOIN
    `target_sql.orders`o
  ON
    i.order_id = o.order_id
  INNER JOIN
    `target_sql.sellers` s
  ON
    i.seller_id = s.seller_id
  GROUP BY
    state)
SELECT
  state,
  avg_value
FROM
  sellers_info
ORDER BY
  avg_value DESC
LIMIT
  5
```

Query results

JOB INFORMATION | RESULTS | JSON | EXEC

Row	state	avg_value
1	RO	50.91
2	CE	46.38
3	PB	39.19
4	PI	36.94
5	AC	32.84

Seller info lowest

```sql
WITH
  sellers_info AS (
  SELECT
    s.seller_state AS state,
    ROUND(AVG(i.freight_value),2) AS avg_value
  FROM
    `target_sql.order_items` i
  INNER JOIN
    `target_sql.orders`o
  ON
    i.order_id = o.order_id
  INNER JOIN
    `target_sql.sellers` s
  ON
    i.seller_id = s.seller_id
  GROUP BY
    state)
SELECT
  state,
  avg_value
FROM
  sellers_info
ORDER BY
  avg_value ASC
LIMIT
  5
```

Query results

JOB INFORMATION | RESULTS | JSON | EXEC

Row	state	avg_value
1	SP	18.45
2	PA	19.39
3	RJ	19.47
4	DF	20.57
5	PR	22.72

Customer info highest

```sql
WITH
  customer_info AS (
  SELECT
    c.customer_state AS state,
    ROUND(AVG(i.freight_value),2) AS avg_value
  FROM
    `target_sql.order_items` i
  INNER JOIN
    `target_sql.orders`o
  ON
    i.order_id = o.order_id
  INNER JOIN
    `target_sql.customers` c
  ON
    o.customer_id = c.customer_id
  GROUP BY
    state)
SELECT
  state,
  avg_value
FROM
  customer_info
ORDER BY
  avg_value DESC
LIMIT
  5
```

Query results

JOB INFORMATION | RESULTS | JSON | EXEC

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

Customer info lowest

```sql
WITH
  customer_info AS (
  SELECT
    c.customer_state AS state,
    ROUND(AVG(i.freight_value),2) AS avg_value
  FROM
    `target_sql.order_items` i
  INNER JOIN
    `target_sql.orders`o
  ON
    i.order_id = o.order_id
  INNER JOIN
    `target_sql.customers` c
  ON
    o.customer_id = c.customer_id
  GROUP BY
    state)
SELECT
  state,
  avg_value
FROM
  customer_info
ORDER BY
  avg_value ASC
LIMIT
  5
```

Query results

JOB INFORMATION | RESULTS | JSON | EXEC

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

As the question didn't give any brief about whether we have to consider the seller state or customer state to calculate the freight value, so we have executed both ways.

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

Highest

```sql
SELECT
  c.customer_state AS state,
  ROUND(AVG(DATE_DIFF(o.order_delivered_customer_date, o.order_purchase_timestamp, day)),2) AS avg_time
FROM
  `target_sql.orders` o
INNER JOIN
  `target_sql.customers`c
ON
  o.customer_id = c.customer_id
GROUP BY
  state
ORDER BY
  avg_time DESC
LIMIT
  5
```

Query results

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

Row	state	avg_time
1	RR	28.98
2	AP	26.73
3	AM	25.99
4	AL	24.04
5	PA	23.32

Lowest

```sql
SELECT
  c.customer_state AS state,
  ROUND(AVG(DATE_DIFF(o.order_delivered_customer_date, o.order_purchase_timestamp, day)),2) AS avg_time
FROM
  `target_sql.orders` o
INNER JOIN
  `target_sql.customers`c
ON
  o.customer_id = c.customer_id
GROUP BY
  state
ORDER BY
  avg_time ASC
LIMIT
  5
```

Query results

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

Row	state	avg_time
1	SP	8.3
2	PR	11.53
3	MG	11.54
4	DF	12.51
5	SC	14.48

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.

```sql
SELECT
  c.customer_state AS state,
  ROUND(AVG(DATE_DIFF(order_estimated_delivery_date, order_delivered_customer_date, day)),2) AS early_delivery
FROM
  `target_sql.orders` o
INNER JOIN
  `target_sql.customers` c
ON
  o.customer_id = c.customer_id
GROUP BY
  state
ORDER BY
  early_delivery ASC
LIMIT
  5
```

Query results

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

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

Above were the top 5 states where the orders were delivered before the estimated delivery date

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

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

```sql
SELECT
  EXTRACT(month
  FROM
    o.order_purchase_timestamp) AS month_of_order,
  p.payment_type,
  COUNT(p.order_id) AS num_of_orders
FROM
  `target_sql.payments` p
INNER JOIN
  `target_sql.orders` o
ON
  o.order_id = p.order_id
GROUP BY
  payment_type,
  month_of_order
ORDER BY
  payment_type,
  month_of_order ASC
```

Query results

SAVE RESULTS

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

Row	month_of_order	payment_type	num_of_orders
7	7	UPI	2074
8	8	UPI	2077
9	9	UPI	903
10	10	UPI	1056
11	11	UPI	1509
12	12	UPI	1160
13	1	credit_card	6103
14	2	credit_card	6609
15	3	credit_card	7707
16	4	credit_card	7301
17	5	credit_card	8350

Results per page: 200 1 – 50 of 50

Looking at the data most people were using credit cards for purchasing and very few were using debit cards.

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

```
SELECT
  payment_installments,
  COUNT(order_id) AS num_of_orders
FROM
  `target_sql.payments`
WHERE
  payment_installments != 0
GROUP BY
  payment_installments
ORDER BY
  num_of_orders DESC
```

Query results

SAVE RESULTS

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

Row	payment_installment	num_of_orders
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
16	20	17

Results per page: 50 1 – 23 of 23

By looking at the data we can say that most people preferred to pay in a single instalment.

7. **Insights from this case study :**

```
SELECT
  DATE_DIFF(order_approved_at, order_purchase_timestamp, day) AS approval_time_in_days,
  COUNT(order_id) AS count
FROM
  `target_sql.orders`
GROUP BY
  approval_time_in_days
ORDER BY
  approval_time_in_days DESC;
```

Query results

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

Row	approval_time_in_da	count
6	23	3
7	16	1
8	13	2
9	12	15
10	11	5
11	10	14
12	9	3
13	8	13
14	7	17
15	6	31
16	5	262
17	4	397
18	3	1491
19	2	2904
20	1	12254
21	0	81861

Most of the orders were approved on the same day of order, but there are still many orders that are taking time for approval. This should be taken care.

```
SELECT
  customer_state as state,
  COUNT(customer_id) AS num_of_cust
FROM
  `target_sql.customers`
GROUP BY
  state
ORDER BY
  num_of_cust ASC
```

Query results

JOB INFORMATION | RESULTS | JSON | EXECUTION DETAILS

Row	state	num_of_cust
1	RR	46
2	AP	68
3	AC	81
4	AM	148
5	RO	253
6	TO	280
7	SE	350
8	AL	413
9	RN	485
10	PI	495

State RR has the least num of customers, where the company might have to consider marketing more.

```sql
SELECT
  review_score,
  COUNT(review_id) AS count
FROM
  `target_sql.order_reviews`
GROUP BY
  review_score
ORDER BY
  1 ASC
```

Query results

JOB INFORMATION | RESULTS | JSON | EXECUTION DETA

Row	review_score	count
1	1	11424
2	2	3151
3	3	8179
4	4	19142
5	5	57328

Most of the customers were satisfied with their orders which led to more 5 star ratings.

```sql
SELECT
  review_comment_title,
  COUNT(review_id) AS count
FROM
  `target_sql.order_reviews`
WHERE
  review_score = 1
  AND review_comment_title IS NOT NULL
GROUP BY
  review_comment_title
ORDER BY
  count DESC
```

Query results

JOB INFORMATION | RESULTS | JSON | EXEC

Row	review_comment_title	count
1	I recommend	94
2	I didn't receive the product	52
3	Product not delivered	45
4	Bad	43
5	Wrong product	40
6	PÃ © ssimo	32
7	Defective product	30
8	I didn't receive	22
9	Delivery delay	15
10	I do not recommend	15
11	No recommend	14
12	Incomplete delivery	13

Above were some of the review comments given by customers who gave a 1-star rating. Most people complained as they are not receiving the product. These need to be taken care.

```sql
SELECT
  product_category,
  COUNT(product_id) AS count
FROM
  `target_sql.products`
GROUP BY
  product_category
ORDER BY
  count DESC
```

Query results

JOB INFORMATION | RESULTS | JSON | EXECUT

Row	product_category	count
1	bed table bath	3029
2	sport leisure	2867
3	Furniture Decoration	2657
4	HEALTH BEAUTY	2444
5	housewares	2335
6	automotive	1900
7	computer accessories	1639
8	toys	1411

Among all the products, the product category “bed table bath” has more no of products and “cds music dvds” has less no of products.

```sql
SELECT
  COUNT(product_id) AS products_with_no_desc
FROM
  `target_sql.products`
WHERE
  product_description_length IS null
```

Query results

JOB INFORMATION | RESULTS | JSON | EXECU

Row	products_with_no_desc
1	610

There were 610 products available with no descriptions

```sql
SELECT
  seller_state,
  COUNT(seller_id) AS count
FROM
  `target_sql.sellers`
GROUP BY
  seller_state
ORDER BY
  count DESC;
```

Query results

JOB INFORMATION | RESULTS | JSON | EXECUTION DET

Row	seller_state	count
1	SP	1849
2	PR	349
3	MG	244
4	SC	190
5	RJ	171
6	RS	129
7	GO	40
8	DF	30
9	ES	23
10	BA	19
11	CE	13

State Sao Paulo (SP) has the most number of sellers compared to other states same as the customers

```sql

SELECT
  DATE_DIFF(order_delivered_carrier_date, order_purchase_timestamp, day) AS delivery_days,
  COUNT(order_id) AS count
FROM
  `target_sql.orders`
GROUP BY
  delivery_days
ORDER BY
  1
```

Query results

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

Row	delivery_days	count
2	-171	1
3	-4	1

I observed some data errors where 2 records showed that the delivery date was earlier than the purchase date.