

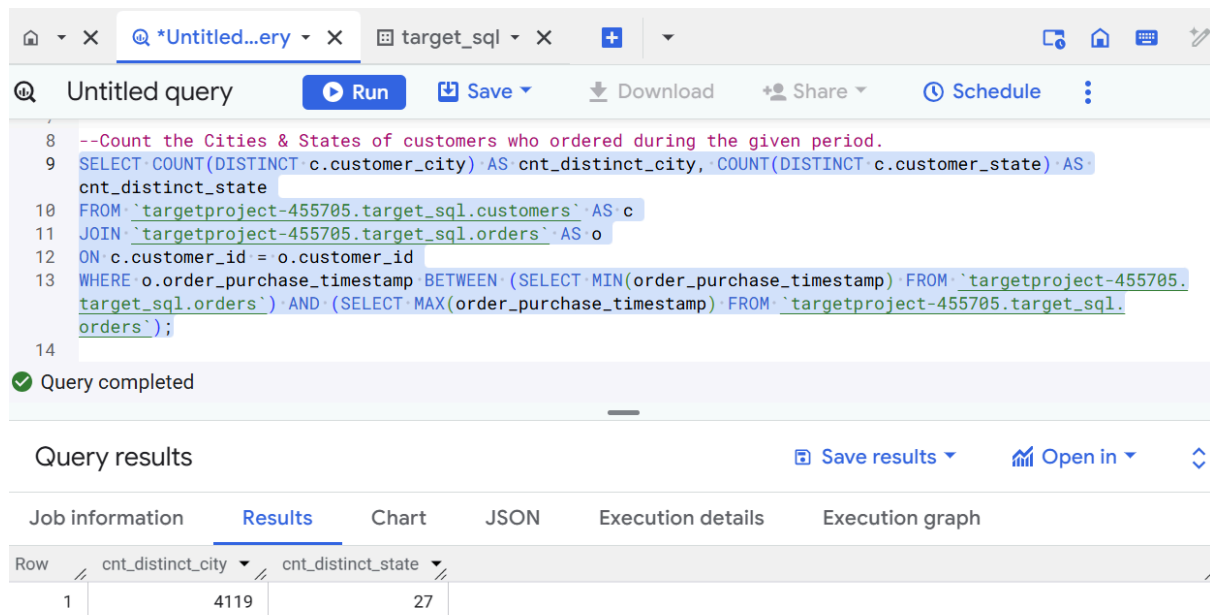
## Business Case: Target\_SQL:

Import the dataset and do usual exploratory analysis steps like checking the structure & characteristics of the dataset a. Data type of columns in a table b. Time period for which the data is given c. Cities and States of customers ordered during the given period

```
SELECT *  
FROM `targetproject-455705.target_sql.customers`;
```

```
SELECT *  
FROM `targetproject-455705.target_sql.orders`;
```

```
--Count the Cities & States of customers who ordered during the given period.  
SELECT COUNT(DISTINCT c.customer_city) AS cnt_distinct_city, COUNT(DISTINCT  
c.customer_state) AS cnt_distinct_state  
FROM `targetproject-455705.target_sql.customers` AS c  
JOIN `targetproject-455705.target_sql.orders` AS o  
ON c.customer_id = o.customer_id  
WHERE o.order_purchase_timestamp BETWEEN (SELECT MIN(order_purchase_timestamp) FROM  
`targetproject-455705.target_sql.orders`) AND (SELECT MAX(order_purchase_timestamp)  
FROM `targetproject-455705.target_sql.orders`);
```



The screenshot shows a SQL query editor interface. The query is as follows:

```
8 --Count the Cities & States of customers who ordered during the given period.  
9 SELECT COUNT(DISTINCT c.customer_city) AS cnt_distinct_city, COUNT(DISTINCT c.customer_state) AS  
cnt_distinct_state  
10 FROM `targetproject-455705.target_sql.customers` AS c  
11 JOIN `targetproject-455705.target_sql.orders` AS o  
12 ON c.customer_id = o.customer_id  
13 WHERE o.order_purchase_timestamp BETWEEN (SELECT MIN(order_purchase_timestamp) FROM `targetproject-455705.  
target_sql.orders`) AND (SELECT MAX(order_purchase_timestamp) FROM `targetproject-455705.target_sql.  
orders`);  
14
```

The query is completed, and the results are displayed in a table:

Job information	Results	Chart	JSON	Execution details	Execution graph
Row	cnt_distinct_city	cnt_distinct_state			
1	4119	27			

--In-depth Exploration:

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

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

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

```
FROM `targetproject-455705.target_sql.orders`
GROUP BY order_year, order_month
ORDER BY order_year, order_month;
```

The screenshot shows a Google Cloud BigQuery console interface. At the top, there's a tab for 'target\_sql' and a search bar. Below the search bar, the query editor shows a SQL query with comments. The query is:
 

```
--In-depth Exploration:
--Is there a growing trend in the no. of orders placed over the past years?
--Can we see some kind of monthly seasonality in terms of the no. of orders being placed?
SELECT EXTRACT(YEAR FROM order_purchase_timestamp) AS order_year,
EXTRACT(MONTH FROM order_purchase_timestamp) AS order_month, COUNT(order_id) AS total_orders
FROM `targetproject-455705.target_sql.orders`
GROUP BY order_year, order_month
ORDER BY order_year, order_month;
```

 Below the query editor, a status bar indicates 'This query will process 3.98 MB when run.' The 'Query results' section is active, showing a table with 3 rows and 4 columns: 'Row', 'order\_year', 'order\_month', and 'total\_orders'. The table data is:
 

Row	order_year	order_month	total_orders
1	2016	9	4
2	2016	10	324
3	2016	12	1

 At the bottom, there's a pagination bar showing 'Results per page: 50' and '1 - 25 of 25'.

Observations:

### 1. Initial phase (2016):

- Very few orders. (only 4, 324, and 1 in Sep, Oct, Dec 2016).
- Probably the system/platform was just launched or it was in early testing phase.

### 2. 2017 - Clear Growth:

- From Jan 2017 onwards, **orders start increasing sharply**.
- Month-over-month, there is a steady rise.
- **Peak months:**
  - November 2017 (7544 orders) - 🔥 highest in 2017.
  - December 2017 also strong (5673 orders).

- So you can **already sense seasonality – more orders towards year-end (Nov-Dec)**.

### 3. 2018 – Sustained High Activity:

- Jan 2018 starts strong (7269 orders) – even higher than Dec 2017.
- High orders continue in Feb, Mar, Apr, May, Aug.
- Again **dips slightly after mid-year** (Jun-Jul-Aug moderate).
- **Sept and Oct 2018 show very low numbers (16, 4) – this looks suspicious.**
  - Maybe the data collection stopped or something went wrong?

### 4. Seasonality Insight:

- **Orders peak between October – January** (festive seasons, holiday shopping).
- **Spring (Feb, Mar, Apr)** is also strong.
- **Summer months (Jun, Jul)** slightly lower but still healthy.
- **Sept-Oct of 2018** specifically looks like missing/broken data.

--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

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

'Morning'

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\_day,

COUNT(order\_id) AS total\_orders

```
FROM `target_sql.orders`
GROUP BY time_of_day
ORDER BY total_orders DESC;
```

Untitled query Run Save Download Share Schedule

```

30 -- 19-23 hrs : Night
31 SELECT
32 CASE
33 WHEN EXTRACT(HOUR FROM order_purchase_timestamp) BETWEEN 0 AND 6 THEN 'Dawn'
34 WHEN EXTRACT(HOUR FROM order_purchase_timestamp) BETWEEN 7 AND 12 THEN 'Morning'
35 WHEN EXTRACT(HOUR FROM order_purchase_timestamp) BETWEEN 13 AND 18 THEN 'Afternoon'
36 WHEN EXTRACT(HOUR FROM order_purchase_timestamp) BETWEEN 19 AND 23 THEN 'Night'
37 END AS time_of_day,
38 COUNT(order_id) AS total_orders

```

✓ This query will process 3.98 MB when run.

Query results Save results Open in

Job information **Results** Chart JSON Execution details Execution graph

Row	time_of_day	total_orders
1	Afternoon	38135
2	Night	28331
3	Morning	27733
4	Dawn	5242

Results per page: 50 1 – 4 of 4

Observations:

- Afternoon (13:00–18:00) is the peak time** for Brazilian customers:
  - Highest number of orders (38,135).
  - Brazilians seem most active in the afternoon for shopping or placing orders.
- Night (19:00–23:00) and Morning (7:00–12:00) are quite close:**
  - Night: 28,331 orders.
  - Morning: 27,733 orders.
  - Both are strong periods, but Night has a slight edge over Morning.
- Dawn (0:00–6:00) is the least active:**
  - Only 5,242 orders.
  - Expected – this is when most people are sleeping.

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

```
SELECT
  EXTRACT(YEAR FROM o.order_purchase_timestamp) AS order_year,
  EXTRACT(MONTH FROM o.order_purchase_timestamp) AS order_month,
  c.customer_state,
  COUNT(o.order_id) AS total_orders
FROM `targetproject-455705.target_sql.customers` AS c
JOIN `targetproject-455705.target_sql.orders` AS o
ON c.customer_id = o.customer_id
GROUP BY order_year, order_month, customer_state
ORDER BY order_year, order_month, customer_state;
```

The screenshot shows a SQL query editor interface. The query is pasted into the editor and highlighted with a blue border. Below the editor, a status bar indicates the query will process 10.81 MB. The 'Query results' section is expanded, showing a table with 5 columns: Row, order\_year, order\_month, customer\_state, and total\_orders. The table contains 4 rows of data for the year 2016, corresponding to states RR, RS, SP, and AL.

Query results

Job information Results Chart JSON Execution details Execution graph

Row	order_year	order_month	customer_state	total_orders
1	2016	9	RR	1
2	2016	9	RS	1
3	2016	9	SP	2
4	2016	10	AL	2

Results per page: 50 1 - 50 of 565

--How are the customers distributed across all the states?

```
SELECT customer_state,
  COUNT(DISTINCT customer_id) AS unique_customers
FROM `targetproject-455705.target_sql.customers`
GROUP BY customer_state
ORDER BY unique_customers;
```

Untitled query Run Save Download Share Schedule

```

54 ORDER BY order_year, order_month, customer_state;
55 ORDER BY order_year, order_month, customer_state;
56
57 --How are the customers distributed across all the states?
58 SELECT customer_state,
59 COUNT(DISTINCT customer_id) AS unique_customers
60 FROM `targetproject-455705.target_sql.customers`
61 GROUP BY customer_state
62 ORDER BY unique_customers;
63

```

✓ This query will process 3.6 MB when run.

Query results Save results Open in

Job information **Results** Chart JSON Execution details Execution graph

Row	customer_state	unique_customers
24	RS	5466
25	MG	11635
26	RJ	12852
27	SP	41746

Results per page: 50 1 - 27 of 27

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

```

SELECT *
FROM `targetproject-455705.target_sql.payments`;

```

```

SELECT *
FROM `targetproject-455705.target_sql.orders`;

```

```

WITH payment_summary AS (
SELECT EXTRACT(YEAR FROM o.order_purchase_timestamp) AS year,
       EXTRACT(MONTH FROM o.order_purchase_timestamp) AS month,
       SUM(p.payment_value) AS total_payment
FROM `targetproject-455705.target_sql.orders` o
JOIN `targetproject-455705.target_sql.payments` p
ON o.order_id = p.order_id
WHERE EXTRACT(MONTH FROM o.order_purchase_timestamp) BETWEEN 1 AND 8
AND EXTRACT(YEAR FROM o.order_purchase_timestamp) IN (2017,2018)
GROUP BY year, month)
SELECT round(((sum_2018 - sum_2017)/sum_2017) *100,2) AS percentage_increase
FROM (
SELECT
SUM(CASE WHEN year=2017 THEN total_payment ELSE 0 END ) AS sum_2017,
SUM(CASE WHEN year=2018 THEN total_payment ELSE 0 END ) AS sum_2018
FROM payment_summary);

```

2025-04-27 0... Run Save query (classic) Share Schedule

```

74 WITH payment_summary AS (
75 SELECT EXTRACT(YEAR FROM o.order_purchase_timestamp) AS year,
76        EXTRACT(MONTH FROM o.order_purchase_timestamp) AS month,
77        SUM(p.payment_value) AS total_payment
78 FROM `targetproject-455705.target_sql.orders` o
79 JOIN `targetproject-455705.target_sql.payments` p
80 ON o.order_id = p.order_id
81 WHERE EXTRACT(MONTH FROM o.order_purchase_timestamp) BETWEEN 1 AND 8
82 AND EXTRACT(YEAR FROM o.order_purchase_timestamp) IN (2017,2018)
83 GROUP BY year, month)
84 SELECT round((sum_2018 - sum_2017) / (sum_2017) * 100, 2) AS percentage_increase

```

Query completed

Query results Save results Open in

Job information **Results** Chart JSON Execution details Execution graph

Row	percentage_incre...
1	136.98

Results per page: 50 1 - 1 of 1

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

```

SELECT *
FROM `targetproject-455705.target_sql.orders`;

```

```

SELECT *
FROM `targetproject-455705.target_sql.customers`;

```

```

SELECT c.customer_state, ROUND(SUM(p.payment_value),2) AS
total_order_price_per_state,
       ROUND(AVG(p.payment_value),2) AS average_order_price_per_state
FROM
  `targetproject-455705.target_sql.orders` o
JOIN
  `targetproject-455705.target_sql.customers` c
ON
  o.customer_id = c.customer_id
JOIN
  `targetproject-455705.target_sql.payments` p
ON
  o.order_id = p.order_id
GROUP BY
  c.customer_state
ORDER BY total_order_price_per_state DESC;

```

2025-04-27 0... [Run](#) [Save query \(classic\)](#) [Share](#) [Schedule](#)

```

98
99 SELECT c.customer_state, ROUND(SUM(p.payment_value),2) AS total_order_price_per_state,
100 ROUND(AVG(p.payment_value),2) AS average_order_price_per_state
101 FROM
102 `targetproject-455705.target_sql.orders` o
103 JOIN
104 `targetproject-455705.target_sql.customers` c
105 ON
106 o.customer_id = c.customer_id
107 JOIN
108 `targetproject-455705.target_sql.payments` p

```

✓ Query completed

Query results [Save results](#) [Open in](#)

Job information **Results** Chart JSON Execution details Execution graph

Row	customer_state	total_order_price...	average_order_pri...
1	SP	5998226.96	137.5
2	RJ	2144379.69	158.53
3	MG	1872257.26	154.71

Results per page: 50 1 - 27 of 27

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

```

SELECT *
FROM `targetproject-455705.target_sql.order_items`;

SELECT c.customer_state, ROUND(SUM(oi.freight_value),2) AS total_freight_per_state,
ROUND(AVG(oi.freight_value),2) AS Average_freight_per_state
FROM
`targetproject-455705.target_sql.customers` c
JOIN
`targetproject-455705.target_sql.orders` o
ON
c.customer_id = o.customer_id
JOIN
`targetproject-455705.target_sql.order_items` oi
ON
o.order_id = oi.order_id
GROUP BY c.customer_state
ORDER BY total_freight_per_state DESC;

```



2025-04-27 0... [Run](#) [Save query \(classic\)](#) [Share](#) [Schedule](#)

```

120 SELECT c.customer_state, ROUND(SUM(oi.freight_value),2) AS total_freight_per_state,
121        ROUND(AVG(oi.freight_value),2) AS Average_freight_per_state
122 FROM
123        `targetproject-455705.target_sql.customers` c
124 JOIN
125        `targetproject-455705.target_sql.orders` o
126 ON
127        c.customer_id = o.customer_id
128 JOIN
129        `targetproject-455705.target_sql.order_items` oi

```

✓ This script will process 28.96 MB when run.

Query results [Save results](#) [Open in](#)

Job information **Results** Chart JSON Execution details Execution graph

Row	customer_state	total_freight_per...	Average_freight...
25	AC	3686.75	40.07
26	AP	2788.5	34.01
27	RR	2235.19	42.98

Results per page: 50 1 - 27 of 27

--Analysis based on sales, freight and delivery time.  
 --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 in a single query.

```

SELECT o.order_id, o.customer_id,
       DATE_DIFF(o.order_delivered_customer_date, o.order_purchase_timestamp, DAY) AS
time_to_deliver,
       DATE_DIFF(o.order_estimated_delivery_date, o.order_delivered_customer_date, DAY)
AS diff_estimated_delivery,
       SUM(oi.price) as total_sales_value,
       SUM(oi.freight_value) as total_freight_value
FROM
  `targetproject-455705.target_sql.orders` o
JOIN
  `targetproject-455705.target_sql.order_items` oi
ON
  o.order_id = oi.order_id
WHERE
  o.order_delivered_customer_date IS NOT NULL
GROUP BY
  o.order_id, o.customer_id, o.order_purchase_timestamp,
  o.order_delivered_customer_date, o.order_estimated_delivery_date
ORDER BY
  time_to_deliver;

```

2025-04-27 0... [Run](#) [Save query \(classic\)](#) [Share](#) [Schedule](#)

```

138 single query.
139 SELECT o.order_id, o.customer_id,
140       DATE_DIFF(o.order_delivered_customer_date, o.order_purchase_timestamp, DAY) AS time_to_deliver,
141       DATE_DIFF(o.order_estimated_delivery_date, o.order_delivered_customer_date, DAY) AS
diff_estimated_delivery,
142       SUM(oi.price) AS total_sales_value,
143       SUM(oi.freight_value) AS total_freight_value
144 FROM
145       `targetproject-455705.target_sql.orders` o
146       JOIN

```

✓ This query will process 14.07 MB when run.

Query results [Save results](#) [Open in](#)

Job information **Results** Chart JSON Execution details Execution graph

	customer_id	time_to_deliver	diff_estimated_d...	total_sales_value	total_freight_value
85190ba88580d6f82d...	344423c2e26d47d2b6d3dd363...	0	7	129.0	8.16
665bc7a1087d31751a...	225aed9e773953084b09cf496...	0	11	49.9	8.72
97d1a65fc65358a632b...	922a46283625e9c096bfd9989...	0	19	50.9	11.86

Results per page: 50 1 - 50 of 96476

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

```

WITH state_freight AS (
SELECT c.customer_state AS state,
       AVG(oi.freight_value) AS avg_freight_value
FROM
       `targetproject-455705.target_sql.customers` c
JOIN
       `targetproject-455705.target_sql.orders` o
ON
       c.customer_id = o.customer_id
JOIN
       `targetproject-455705.target_sql.order_items` oi
ON
       o.order_id = oi.order_id
GROUP BY
       c.customer_state)

SELECT state,
       avg_freight_value
FROM (

SELECT
       state,
       avg_freight_value

```

```

FROM
    state_freight
ORDER BY
    avg_freight_value ASC
LIMIT 5
)
UNION ALL

SELECT state,
    avg_freight_value
FROM (
SELECT
    state,
    avg_freight_value
FROM
    state_freight
ORDER BY
    avg_freight_value DESC
LIMIT 5)
;

```

2025-04-27 0...
Run
Save query (classic)
Share
Schedule

```

157
158 --Find out the top 5 states with the highest & lowest average freight value.
159
160 WITH state_freight AS (
161 SELECT c.customer_state AS state,
162        AVG(o.freight_value) AS avg_freight_value
163 FROM
164        `targetproject-455705.target_sql.customers` c
165 JOIN
166        `targetproject-455705.target_sql.orders` o

```

✓ This query will process 14.56 MB when run.

Query results

Save results
Open in

Job information
Results
Chart
JSON
Execution details
Execution graph

Row	state	avg_freight_value
1	RR	42.98442307692...
2	PB	42.72380398671...
3	RO	41.06971223021...

Results per page: 50
1 - 10 of 10

2025-04-27 0... Run Save query (classic) Share Schedule

```

157
158 --Find out the top 5 states with the highest & lowest average freight value.
159
160 WITH state_freight AS (
161 SELECT c.customer_state AS state,
162        AVG(oi.freight_value) AS avg_freight_value
163 FROM
164        `targetproject-455705.target_sql.customers` c
165 JOIN
166        `targetproject-455705.target_sql.orders` o

```

✓ This query will process 14.56 MB when run.

Query results Save results Open in

Job information **Results** Chart JSON Execution details Execution graph

Row	state	avg_freight_value
8	MG	20.63016680630...
9	RJ	20.96092393168...
10	DF	21.04135494596...

Results per page: 50 1 - 10 of 10

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

```

SELECT *
FROM `targetproject-455705.target_sql.orders`;

WITH state_delivery_time AS (
  SELECT
    c.customer_state AS state,
    AVG(DATE_DIFF(o.order_delivered_customer_date, o.order_purchase_timestamp,
DAY)) AS avg_delivery_time
  FROM
    `targetproject-455705.target_sql.orders` o
  JOIN
    `targetproject-455705.target_sql.customers` c
  ON
    o.customer_id = c.customer_id
  WHERE
    o.order_delivered_customer_date IS NOT NULL
  GROUP BY
    state
)

SELECT
  state,
  avg_delivery_time
FROM (

```

```
SELECT
    state,
    avg_delivery_time
FROM
    state_delivery_time
ORDER BY
    avg_delivery_time ASC
LIMIT 5
)

UNION ALL

SELECT
    state,
    avg_delivery_time
FROM (
    SELECT
        state,
        avg_delivery_time
    FROM
        state_delivery_time
    ORDER BY
        avg_delivery_time DESC
    LIMIT 5
)
ORDER BY
    avg_delivery_time ASC
;
```

2025-04-27 0... [Run](#) [Save query \(classic\)](#) [Share](#) [Schedule](#)

```

210 WITH state_delivery_time AS (
211     SELECT
212         c.customer_state AS state,
213         AVG(DATE_DIFF(o.order_delivered_customer_date, o.order_purchase_timestamp, DAY)) AS avg_delivery_time
214     FROM
215         `targetproject-455705.target_sql.orders` o
216     JOIN
217         `targetproject-455705.target_sql.customers` c
218     ON
219         o.customer_id = c.customer_id

```

✓ This script will process 188.78 MB when run.

Query results [Save results](#) [Open in](#)

Job information **Results** Chart JSON Execution details Execution graph

Row	state	avg_delivery_time
1	SP	8.298061489072...
2	PR	11.52671135486...
3	MG	11.54381329810...

Results per page: 50 1 – 10 of 10

2025-04-27 0... [Run](#) [Save query \(classic\)](#) [Share](#) [Schedule](#)

```

210 WITH state_delivery_time AS (
211     SELECT
212         c.customer_state AS state,
213         AVG(DATE_DIFF(o.order_delivered_customer_date, o.order_purchase_timestamp, DAY)) AS avg_delivery_time
214     FROM
215         `targetproject-455705.target_sql.orders` o
216     JOIN
217         `targetproject-455705.target_sql.customers` c
218     ON
219         o.customer_id = c.customer_id

```

✓ This script will process 188.78 MB when run.

Query results [Save results](#) [Open in](#)

Job information **Results** Chart JSON Execution details Execution graph

Row	state	avg_delivery_time
8	AM	25.98620689655...
9	AP	26.73134328358...
10	RR	28.97560975609...

Results per page: 50 1 – 10 of 10

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

```

WITH state_delivery_diff AS (
SELECT
    c.customer_state AS state,
    AVG(DATE_DIFF(o.order_estimated_delivery_date, o.order_delivered_customer_date,
DAY)) AS avg_days_early
FROM
    `targetproject-455705.target_sql.customers` c
JOIN

```

```

        `targetproject-455705.target_sql.orders` o
ON
    c.customer_id = o.customer_id
WHERE
    o.order_delivered_customer_date IS NOT NULL
GROUP BY
    state)

SELECT
    state,
    avg_days_early
FROM
    state_delivery_diff
ORDER BY
    avg_days_early DESC
LIMIT 5;

```

2025-04-27 0... [Run](#) [Save query \(classic\)](#) [Share](#) [Schedule](#)

```

260 --the estimated date of delivery.
261 WITH state_delivery_diff AS (
262     SELECT
263         c.customer_state AS state,
264         AVG(DATE_DIFF(o.order_estimated_delivery_date, o.order_delivered_customer_date, DAY)) AS avg_days_early
265     FROM
266         `targetproject-455705.target_sql.customers` c
267     JOIN

```

✓ This query will process 8.32 MB when run.

Query results [Save results](#) [Open in](#)

Job information **Results** Chart JSON Execution details Execution graph

Row	state	avg_days_early
1	AC	19.7625
2	RO	19.13168724279...
3	AP	18.73134328358...
4	AM	18.60689655172...
5	RR	16.41463414634...

Results per page: 50 1 - 5 of 5

--Based on how much earlier or later a product is usually delivered each state as "Early", "On Time", or "Late"

```

WITH state_delivery_diff AS (
    SELECT
        c.customer_state AS state,
        AVG(DATE_DIFF(o.order_estimated_delivery_date, o.order_delivered_customer_date,
DAY)) AS avg_days_early
    FROM
        `targetproject-455705.target_sql.orders` o
    JOIN
        `targetproject-455705.target_sql.customers` c

```

```

ON
    o.customer_id = c.customer_id
WHERE
    o.order_delivered_customer_date IS NOT NULL
GROUP BY
    state
)

SELECT
    state,
    avg_days_early,
    CASE
        WHEN avg_days_early > 2 THEN 'Early'
        WHEN avg_days_early BETWEEN -2 AND 2 THEN 'On Time'
        ELSE 'Late'
    END AS delivery_performance
FROM
    state_delivery_diff
ORDER BY
    avg_days_early DESC
LIMIT 5
;

```

2025-04-27 0... [Run](#) [Save query \(classic\)](#) [Share](#) [Schedule](#)

300 )  
301  
302 SELECT  
303 state,  
304 avg\_days\_early,  
305 CASE  
306 WHEN avg\_days\_early > 2 THEN 'Early'  
307 WHEN avg\_days\_early BETWEEN -2 AND 2 THEN 'On Time'

✓ This script will process 205.42 MB when run.

### Query results

[Save results](#) [Open in](#)

Job information	Results	Chart	JSON	Execution details	Execution graph
Row	state	avg_days_early	delivery_performance		
1	AC	19.76249999999...	Early		
2	RO	19.13168724279...	Early		
3	AP	18.73134328358...	Early		
4	AM	18.60689655172...	Early		
5	RR	16.41463414634...	Early		

Results per page: 50 1 - 5 of 5

--Analysis based on the payments

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

```

SELECT *
FROM `targetproject-455705.target_sql.payments`;

```



```
SELECT *
FROM `targetproject-455705.target_sql.orders`;
```

```
SELECT
  EXTRACT(YEAR FROM o.order_purchase_timestamp) AS year,
  EXTRACT(MONTH FROM o.order_purchase_timestamp) AS month,
  p.payment_type,
  COUNT(DISTINCT o.order_id) AS number_of_orders
FROM
  `targetproject-455705.target_sql.orders` o
JOIN
  `targetproject-455705.target_sql.payments` p
ON
  o.order_id = p.order_id
WHERE
  o.order_status != 'canceled'
GROUP BY
  year,
  month,
  payment_type
ORDER BY
  year,
  month,
  payment_type;
```

2025-04-27 0...
Run
Save query (classic)
Share
Schedule

```

328 SELECT
329   EXTRACT(YEAR FROM o.order_purchase_timestamp) AS year,
330   EXTRACT(MONTH FROM o.order_purchase_timestamp) AS month,
331   p.payment_type,
332   COUNT(DISTINCT o.order_id) AS number_of_orders
333 FROM

```

✓ This script will process 27.61 MB when run.

**Query results**
Save results
Open in

Job information
Results
Chart
JSON
Execution details
Execution graph

Row	year	month	payment_type	number_of_orders
1	2016	9	credit_card	1
2	2016	10	UPI	60
3	2016	10	credit_card	233
4	2016	10	debit_card	2
5	2016	10	voucher	10
6	2016	12	credit_card	1

Results per page: 50
1 – 50 of 87

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

```
SELECT *  
FROM `targetproject-455705.target_sql.payments`;
```

```
SELECT *  
FROM `targetproject-455705.target_sql.orders`;
```

```
SELECT  
  p.payment_installments AS number_of_installements,  
  COUNT(DISTINCT o.order_id) AS number_of_orders  
FROM  
  `targetproject-455705.target_sql.orders` o  
JOIN  
  `targetproject-455705.target_sql.payments` p  
ON  
  o.order_id = p.order_id  
WHERE  
  p.payment_installments > 0 and p.payment_value > 0  
GROUP BY  
  p.payment_installments  
ORDER BY  
  p.payment_installments ASC;
```

🔍 2025-04-27 0...	▶ Run	💾 Save query (classic) ▾	👤 Share ▾	🕒 Schedule	⋮
359					
360	SELECT				
361	p.payment_installments AS number_of_installements,				
362	COUNT(DISTINCT o.order_id) AS number_of_orders				
363	FROM				
364	`targetproject-455705.target_sql.orders` o				
✔ This script will process 26.28 MB when run.					
Query results <span>💾 Save results ▾</span> <span>📊 Open in ▾</span> <span>⌵</span>					
Job information <b>Results</b> Chart   JSON   Execution details   Execution graph					
Row	number_of_instal...	number_of_orders			
1	1	49057			
2	2	12389			
3	3	10443			
4	4	7088			
5	5	5234			
6	6	3916			
Results per page: 50 ▾   1 – 23 of 23   ⏪ ⏩ ⏴ ⏵					