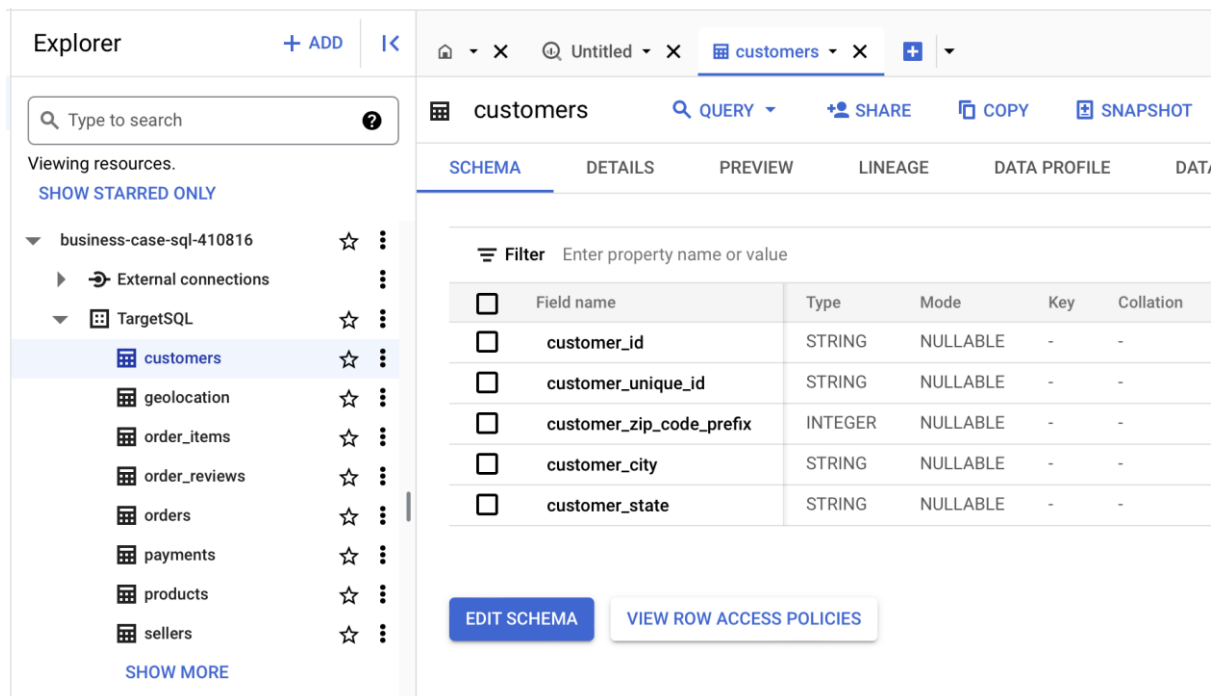


1) Import the dataset and do usual exploratory analysis steps like checking the structure & characteristics of the dataset:

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



The screenshot shows a data explorer interface. On the left, the 'Explorer' panel lists resources under 'business-case-sql-410816', including 'External connections', 'TargetSQL', and a list of tables: 'customers', 'geolocation', 'order_items', 'order_reviews', 'orders', 'payments', 'products', and 'sellers'. The 'customers' table is selected. The main panel shows the 'customers' table schema with columns: 'customer_id', 'customer_unique_id', 'customer_zip_code_prefix', 'customer_city', and 'customer_state'. The 'customer_zip_code_prefix' column is of type 'INTEGER', while the others are 'STRING'. All columns are 'NULLABLE' and have no key or collation.

Field name	Type	Mode	Key	Collation
customer_id	STRING	NULLABLE	-	-
customer_unique_id	STRING	NULLABLE	-	-
customer_zip_code_prefix	INTEGER	NULLABLE	-	-
customer_city	STRING	NULLABLE	-	-
customer_state	STRING	NULLABLE	-	-

Observation:

All columns are of data type "STRING" except 'customer_zip_code_prefix' which is of type INTEGER

2.2) Get the time range between which the orders were placed.

```
select
max(order_purchase_timestamp) as max_date,
min(order_purchase_timestamp) as min_date
from `TargetSQL.orders`
```

The screenshot shows a data analytics application with an Explorer panel on the left and a main workspace on the right. The Explorer panel displays a tree view of resources under 'business-case-sql-410816', including 'External connections', 'TargetSQL', and various tables like 'customers', 'geolocation', 'order_items', 'order_reviews', 'orders', 'payments', 'products', and 'sellers'. The main workspace shows a SQL query in a file named 'Untitled'. The query is designed to find the first and last order timestamps from the 'orders' table. Below the query editor, the 'Query results' section is visible, showing a table with two columns: 'first_order' and 'last_order'. The results show the first order on 2016-09-04 and the last order on 2018-10-17.

SQL Query:

```

1 # 1.1) Data type of all columns in the "customers" table.
2
3 # 1.2) Get the time range between which the orders were placed.
4 select
5 min(order_purchase_timestamp) as first_order,
6 max(order_purchase_timestamp) as last_order
7 from
8 `TargetSQL.orders`
9

```

Query results

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

Observations:

The first order was placed on 04th September 2016 and the last order was placed on 17th October 2018

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

```

select
count(distinct customer_city),
count(distinct customer_state)
from `TargetSQL.customers`

```

Explorer

+ ADD

⏮

Type to search

?

Viewing resources.

SHOW STARRED ONLY

business-case-sql-410816

External connections

TargetSQL

customers

geolocation

order_items

order_reviews

orders

payments

products

sellers

⏮

Untitled

RUN

SAVE

DOWNLOAD

1.3) Count the Cities & States of customers who ord

select

count(distinct customer_city) as city,

count(distinct customer_state) as state

from TargetSQL.customers

Query results

JOB INFORMATION

RESULTS

CHART

PREVIEW

Row	city	state
1	4119	27

Observations

There are 4119 cities and 27 states, where customers who ordered during the given period (2016-09-04 and 2018-10-17)

2) In-depth Exploration:

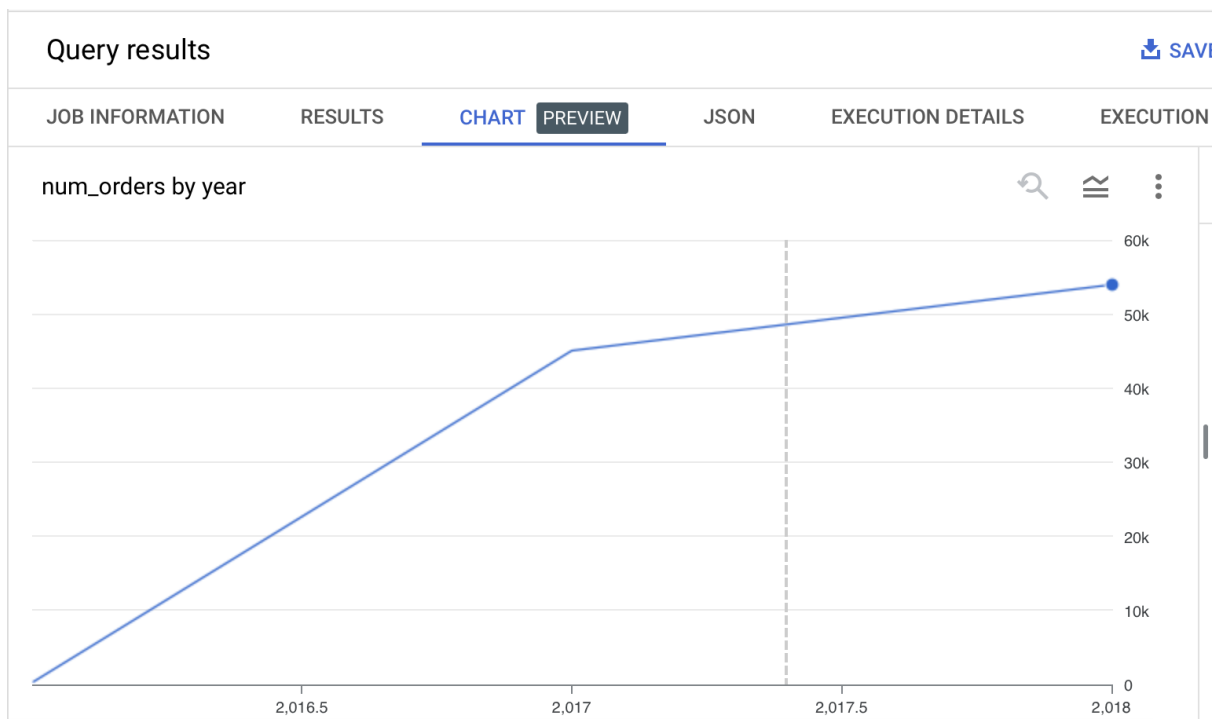
2.1) Is there a growing trend in the no. of orders placed over the past years?

```
select
extract (year from order_purchase_timestamp) as year,
count (distinct order_id) as num_orders
from TargetSQL.orders`
group by 1
order by 1
```

```
17 # 2.1) Is there a growing trend in the no. of orders placed over the past years?
18
19 select
20   extract(year from order_purchase_timestamp) as year,
21   count(distinct order_id) as num_orders
22 from `TargetSQL.orders`
23 group by 1
24 order by 1
25
26
27
```

Query results

JOB INFORMATION		RESULTS	CHART	PREVIEW	JSON	EXECUTION DETAILS	EX
Row	year	num_orders					
1	2016	329					
2	2017	45101					
3	2018	54011					



Observations:

In the initial year 2016 the number of orders placed was very minimum but in 2017 the number of orders were show some rapid increase and in 2018, the trend continues

2.2) 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 year_,
extract (month from order_purchase_timestamp) as month_,
count(1) as num_orders
from `TargetSQL.orders`
group by year_, month_
order by year_, month_
```

The screenshot shows a SQL query editor with a toolbar at the top containing icons for home, close, search, and tabs. The main area displays the SQL query for monthly order counts. The query is as follows:

```
# 2.2) 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 year_,
extract (month from order_purchase_timestamp) as month_,
count(1) as num_orders
from `TargetSQL.orders`
group by year_, month_
order by year_, month_
```

Query results

[SAVE RESULTS](#)

JOB INFORMATION		RESULTS	CHART	PREVIEW	JSON	EXECUTION DETAILS	EXECUTION GRAPH
Row	year_	month_	num_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				

The last months of 2016 the number of orders were placed very less but in 2017 it showed a gradual increase up to October and on November it spikes to a very large value (7544). Then it sustained around that value till 2018 August but in the last two months the value declined dramatically.

2.3) 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

```
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_day,
count(distinct order_id) as num_orders
from `TargetSQL.orders`
group by 1
order by 2 desc
```

The screenshot shows a SQL IDE interface with a query editor and a results table. The query editor contains a SQL query that categorizes orders by time of day and counts them. The results table shows the output of this query.

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_day,
count(distinct order_id) as num_orders
from `TargetSQL.orders`
group by 1
order by 2 desc
```

Query results:

Row	time_of_day	num_orders
1	Afternoon	38135
2	Night	28331
3	Mornings	27733
4	Dawn	5242

Observations:

During Afternoon the Brazilian customers mostly place their orders, also the least placed order was on Dawn time

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

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

```
select c.customer_state,
```

```

extract (year from o.order_purchase_timestamp) as year_,
extract (month from o.order_purchase_timestamp) as month_,
count(distinct o.order_id) as num_order
from `TargetSQL.orders` o
inner join `TargetSQL.customers` c
on o.customer_id = c.customer_id
group by 1, 2, 3
order by 4 desc

```

🏠 ✕ 🔍 *Untitled ✕ 📊 orders ✕ 📊 customers ✕ + ▼

🔍 Untitled ▶ RUN 💾 SAVE ⬇️ DOWNLOAD 👤 SHARE 🕒 SCHEDULE ⚙️

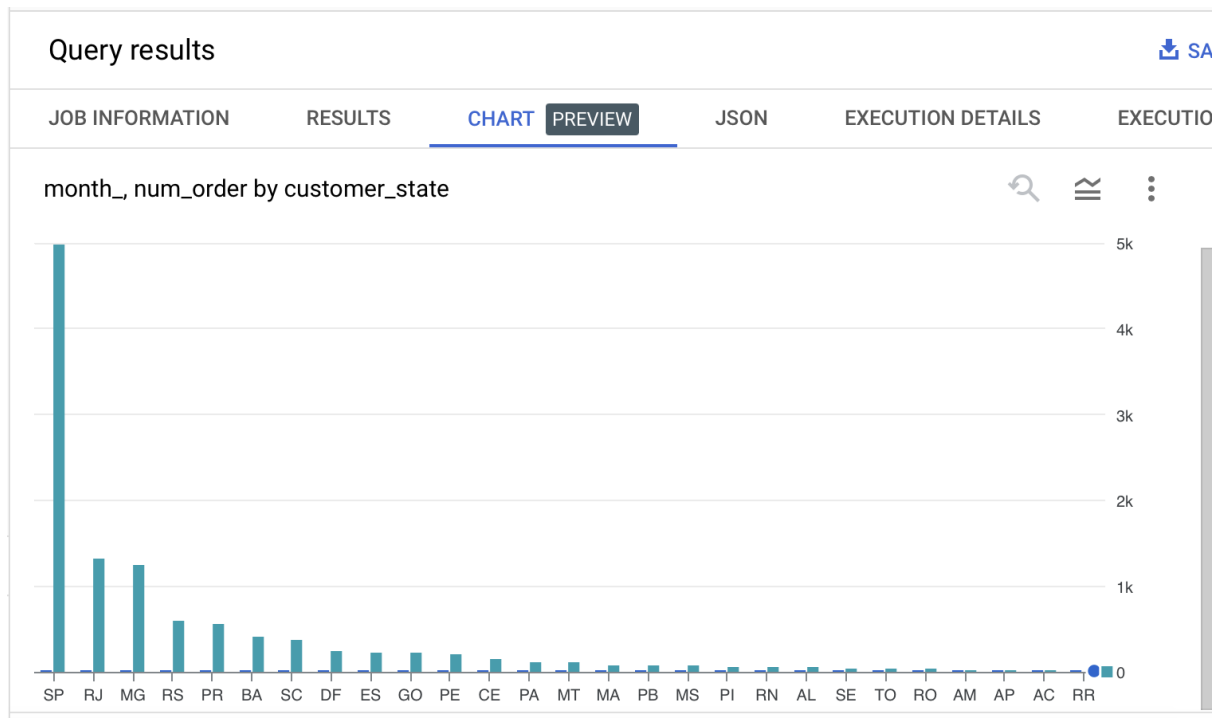
```

51 # 3.1) Get the month on month no. of orders placed in each state.
52 select c.customer_state,
53 extract (year from o.order_purchase_timestamp) as year_,
54 extract (month from o.order_purchase_timestamp) as month_,
55 count(distinct o.order_id) as num_order
56 from `TargetSQL.orders` o
57 inner join `TargetSQL.customers` c
58 on o.customer_id = c.customer_id
59 group by 1, 2, 3
60 order by 4 desc

```

Query results

JOB INFORMATION		RESULTS	CHART	PREVIEW	JSON	EXECUTION DETAILS
Row	customer_state	year_	month_	num_order		
1	SP	2018	8	3253		
2	SP	2018	5	3207		
3	SP	2018	4	3059		
4	SP	2018	1	3052		
5	SP	2018	3	3037		
6	SP	2017	11	3012		
7	SP	2018	7	2777		
8	SP	2018	6	2773		
9	SP	2018	2	2703		
10	SP	2017	12	2357		



A huge number of orders placed in SP and other states placed very less number of orders.

3.2) How are the customers distributed across all the states?

```
select customer_state,  
count(distinct customer_id) customer_num  
from `TargetSQL.customers`  
group by 1  
order by 2 desc
```


*Untitled
orders
customers

Untitled

RUN

SAVE

DOWNLOAD

SHARE

SCHEDULE

```

61
62 # 3.2) How are the customers distributed across all the states?
63 select customer_state,
64 count(distinct customer_id) customer_num
65 from `TargetSQL.customers`
66 group by 1
67 order by 2 desc
68
69
70

```

Query results

JOB INFORMATION		RESULTS	CHART	PREVIEW	JSON	EXECUTION DETAILS
Row	customer_state	customer_num				
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				

Most of the customers are of the state SP and states RJ & MG have customers above 10K while, rest of the states have very less number of customers

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

4.1) 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 base_1 as
(
select
*
from `TargetSQL.payments` p
inner join `TargetSQL.orders` o
```

```

on p.order_id = o.order_id
where extract (year from o.order_purchase_timestamp) between 2017 and 2018
and extract (month from o.order_purchase_timestamp) between 1 and 8
),
base_2 as
(
select
    extract (year from order_purchase_timestamp) as year,
    sum (payment_value) as cost
from base_1
group by 1
order by 1
),
base_3 as
(
    select *,
    lead (cost) over (order by year) as next_year
    from base_2
)
select *,
round((next_year - cost) / cost * 100, 2) as percent_increase
from base_3

```


Untitled
 RUN
 SAVE
 DOWNLOAD
 SHARE
 SCHEDULE
 MORE

```

70 # You can use the "payment_value" column in the payments table to get the cost of orders.
71 with base_1 as
72 (
73 select
74 *
75 from TargetSQL.payments p
76 inner join TargetSQL.orders o
77 on p.order_id = o.order_id
78 where extract (year from o.order_purchase_timestamp) between 2017 and 2018
79 and extract (month from o.order_purchase_timestamp) between 1 and 8
80 ),
81 base_2 as
82 (
83 select
84     extract (year from order_purchase_timestamp) as year,
85     sum (payment_value) as cost
86 from base_1
87 group by 1
88 order by 1
89 ),
90 base_3 as
91 (
92     select *,
93     lead (cost) over (order by year) as next_year
94     from base_2
95 )
96 select *,
97 round((next_year - cost) / cost * 100, 2) as percent_increase
98 from base_3
99

```

Query results

JOB INFORMATION		RESULTS	CHART	PREVIEW	JSON	EXECUTION DET
Row	year ▼	cost ▼	next_year ▼	percent_increase ▼		
1	2018	8694733.839999...	null	null		
2	2017	3669022.120000...	8694733.839999...	136.98		

The percentage increase from 2017 to 2018 (include months between Jan to Aug) is 136.98

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

```
select
c.customer_state,
sum (oi.price) as total_value,
avg (oi.price) as avg_price
from
`TargetSQL.order_items` oi inner join `TargetSQL.orders` o
on oi.order_id = o.order_id
inner join `TargetSQL.customers` c
on c.customer_id = o.customer_id
group by customer_state
order by 2 desc
```

🏠

✕

*Untitled

✕

orders

✕

order_items

✕

custom

Untitled

RUN

SAVE

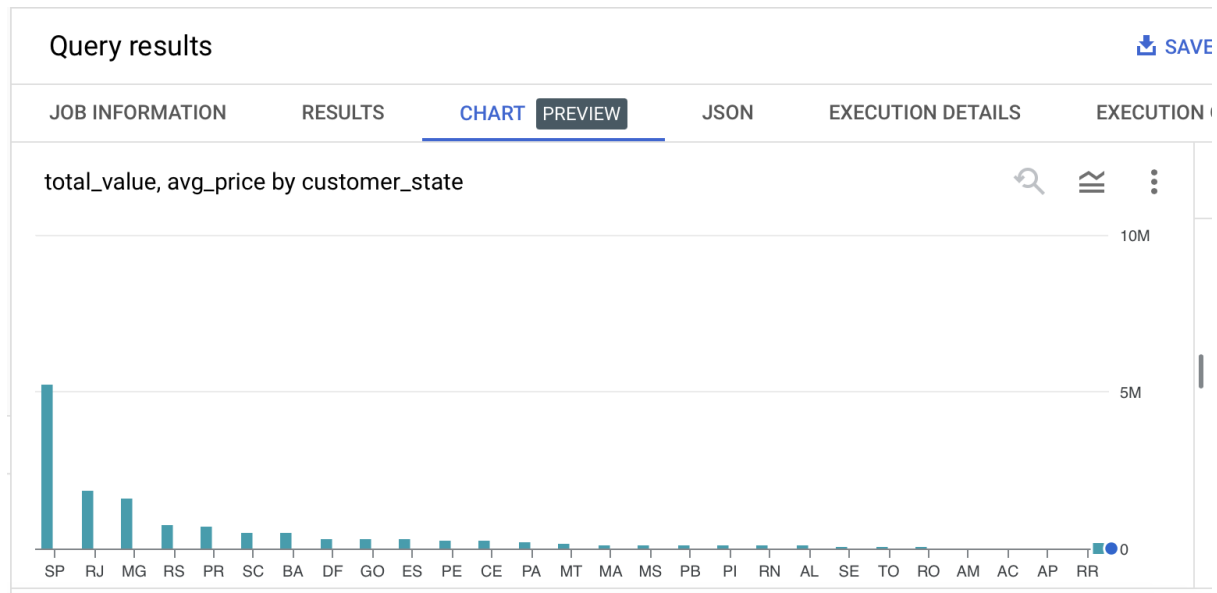
DOWNLOAD

SHARE

```
101 select
102 c.customer_state,
103 sum(oi.price) as total_value,
104 avg(oi.price) as avg_price
105 from
106 `TargetSQL.order_items` oi inner join `TargetSQL.orders` o
107 on oi.order_id = o.order_id
108 inner join `TargetSQL.customers` c
109 on c.customer_id = o.customer_id
110 group by customer_state
111 order by ? desc
```

Query results

JOB INFORMATION		RESULTS	CHART	PREVIEW	JSON	EX
Row	customer_state	total_value	avg_price			
1	SP	5202955.050001...	109.6536291597...			
2	RJ	1824092.669999...	125.1178180945...			
3	MG	1585308.029999...	120.7485741488...			
4	RS	750304.0200000...	120.3374530874...			
5	PR	683083.7600000...	119.0041393728...			
6	SC	520553.3400000...	124.6535775862...			
7	BA	511349.9900000...	134.6012082126...			
8	DF	302603.9399999...	125.7705486284...			
9	GO	294591.9499999...	126.2717316759...			
10	ES	275037.3099999...	121.9137012411...			



The total value is very high in the state SP while all other state are very less compared to the state SP, but the average price is around 100

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

```
select c.customer_state, sum(oi.freight_value) total_value, avg(oi.freight_value)
avg_price
from `TargetSQL.order_items` oi left join `TargetSQL.orders` o
on oi.order_id = o.order_id
left join `TargetSQL.customers` c
on o.customer_id = c.customer_id
group by 1
```

🏠

✕

🔍 *Untitled ✕

📊 order_items ✕

📊 orders ✕

📊 customers ✕

+

▼

🔍

Untitled

▶ RUN

💾 SAVE ▼

⬇️ DOWNLOAD

👤 SHARE ▼

🕒 SCHEDULE

⚙️ MORE ▼

```
1 # Calculate the Total & Average value of order freight for each state.
2
3 select c.customer_state, sum(oi.freight_value) total_value, avg(oi.freight_value) avg_price
4 from `TargetSQL.order_items` oi left join `TargetSQL.orders` o
5 on oi.order_id = o.order_id
6 left join `TargetSQL.customers` c
7 on o.customer_id = c.customer_id
8 group by
```

Query results [SAVE RES](#)

JOB INFORMATION		RESULTS	CHART	PREVIEW	JSON	EXECUTION DETAILS	EXECUTION GRAI
Row	customer_state	total_value	avg_price				
1	SP	718723.0699999...	15.14727539041...				
2	RJ	305589.3100000...	20.96092393168...				
3	PR	117851.6800000...	20.53165156794...				
4	SC	89660.26000000...	21.47036877394...				
5	DF	50625.49999999...	21.04135494596...				
6	MG	270853.4600000...	20.63016680630...				
7	PA	38699.30000000...	35.83268518518...				
8	BA	100156.6799999...	26.36395893656...				
9	GO	53114.97999999...	22.76681525932...				
10	RS	135522.7400000...	21.73580433039...				



The highest of total value is in SP (~718.72K) while all other states shows significantly very less values. The average price is around 0.02K for most of the states.

5.1) 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

```
select
timestamp_diff(order_delivered_customer_date, order_purchase_timestamp, day)
time_to_deliver,
timestamp_diff(order_delivered_customer_date, order_estimated_delivery_date, day)
diff_estimated_delivery
from `TargetSQL.orders`
where order_status = 'delivered'
```

more order status

🔍 ×

*Untitled ×

📊 orders ×

+

▼

🔍

Untitled

▶ RUN

💾 SAVE ▼

⬇️ DOWNLOAD

👤 SHARE ▼

🕒 SCHEDULE

⚙️ MORE ▼

15

-- time_to_deliver = order_delivered_customer_date - order_purchase_timestamp

16

-- diff_estimated_delivery = order_delivered_customer_date - order_estimated_delivery_date

17

18

select

19

timestamp_diff(order_delivered_customer_date, order_purchase_timestamp, 'day') time_to_deliver,

20

timestamp_diff(order_delivered_customer_date, order_estimated_delivery_date, 'day') diff_estimated_delivery

21

from `TargetSQL.orders`

22

where order_status = 'delivered'

23

24

Query results

⬇️ SAVE RESULTS ▼

JOB INFORMATION

RESULTS

CHART

PREVIEW

JSON

EXECUTION DETAILS

EXECUTION GRAPH

Row	time_to_deliver	diff_estimated_delivery
1	30	-1
2	32	0
3	29	-1
4	43	4
5	40	4
6	37	1
7	33	5
8	38	6
9	36	2
10	34	0

The time to deliver ranges around 30 to 50 but difference between the estimated & actual delivery shows a significant range difference.

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

Highest

```
select
  c.customer_state,
  avg(oi.freight_value) as avg_val
from `TargetSQL.order_items` oi inner join `TargetSQL.orders` o using (order_id)
inner join `TargetSQL.customers` c using (customer_id)
group by customer_state
order by 2 desc
limit 5;
```

🏠 X 🔍 *Untitled X 📊 order_items X 📊 customers X 📊 orders X + ▾

🔍 Untitled ▶ RUN 📄 SAVE ▾ ⬇️ DOWNLOAD 👤 SHARE ▾ 🕒 SCHEDULE ⚙️

24 # 5.2) Find out the top 5 states with the highest & lowest average freight value.

25

26 select

27 c.customer_state,

28 avg(oi.freight_value) as avg_val

29 from `TargetSQL.order_items` oi inner join `TargetSQL.orders` o using (order_id)

30 inner join `TargetSQL.customers` c using (customer_id)

31 group by customer_state

32 order by 2 desc

33 limit 5;

34

Query results

JOB INFORMATION

RESULTS

CHART

PREVIEW

JSON

EXECUTION DETAILS

Row	customer_state ▾	avg_val ▾	
1	RR	42.98442307692...	
2	PB	42.72380398671...	
3	RO	41.06971223021...	
4	AC	40.07336956521...	
5	PI	39.14797047970...	

The highest average freight value ranges around 40 and the top one state is RR

Lowest

```
select
  c.customer_state,
```



```

    avg(oi.freight_value) as avg_val
from `TargetSQL.order_items` oi inner join `TargetSQL.orders` o using (order_id)
inner join `TargetSQL.customers` c using (customer_id)
group by customer_state
order by 2
limit 5;

```

The screenshot shows a SQL query editor with a toolbar at the top containing icons for home, search, and tabs for 'Untitled', 'order_items', 'customers', and 'orders'. Below the toolbar, the query is displayed in a text area with line numbers 35 to 45. The query is a SQL statement that selects customer_state and the average freight value (avg_val) from TargetSQL.order_items, joined with TargetSQL.orders and TargetSQL.customers. The results are ordered by customer_state and limited to 5 rows.

Below the query editor, the 'Query results' section is visible. It contains a table with 3 columns: Row, customer_state, and avg_val. The table shows the top 5 states with the lowest average freight value.

Row	customer_state	avg_val
1	SP	15.14727539041...
2	PR	20.53165156794...
3	MG	20.63016680630...
4	RJ	20.96092393168...
5	DF	21.04135494596...

The lowest average freight value is ~15 in SP and the 5th lowest is in DF

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

Highest

```

select
    c.customer_state,
    extract (time from order_delivered_customer_date) time_
from `TargetSQL.orders` oi inner join `TargetSQL.order_items` o using (order_id)
inner join `TargetSQL.customers` c using (customer_id)
group by 1, 2
order by 2 desc
limit 5

```

🏠
✕
*Untitled ✕
📊 order_items ✕
📊 customers ✕
📊 orders ✕
+

🔍 Untitled
▶ RUN
💾 SAVE
⬇️ DOWNLOAD
👤 SHARE
🕒 SCHEDULE
⚙️ MO

```

39 # 5.3) Find out the top 5 states with the highest & lowest average delivery time.
40
41 select
42   c.customer_state,
43   extract (time from order_delivered_customer_date) time_
44 from `TargetSQL.orders` oi inner join `TargetSQL.order_items` o using (order_id)
45 inner join `TargetSQL.customers` c using (customer_id)
46 group by 1, 2
47 order by 2 desc
48 limit 5
49
50
51
52
53
54

```

Query results

JOB INFORMATION		RESULTS	CHART	PREVIEW	JSON	EXECUTION DETAILS	EXE
Row	customer_state	time_					
1	MG	23:59:59					
2	BA	23:59:55					
3	RJ	23:59:52					
4	SP	23:59:52					
5	SP	23:59:49					

The highest average delivery time is in MG.

Lowest

```

select
  c.customer_state,
  extract (time from order_delivered_customer_date) time_
from `TargetSQL.orders` oi inner join `TargetSQL.order_items` o using (order_id)
inner join `TargetSQL.customers` c using (customer_id)
where extract (time from order_delivered_customer_date) is not null
group by 1, 2
order by 2
limit 5

```

Untitled 2
RUN
SAVE
DOWNLOAD
SHARE
SCHEDULE
MORE

```

1 select
2   c.customer_state,
3   extract(time from order_delivered_customer_date) time_
4 from `TargetSQL.orders` oi inner join `TargetSQL.order_items` o using (order_id)
5 inner join `TargetSQL.customers` c using (customer_id)
6 where extract(time from order_delivered_customer_date) is not null
7 group by 1, 2
8 order by 2
9 limit 5
10

```

Query results
SAVE

JOB INFORMATION		RESULTS	CHART	PREVIEW	JSON	EXECUTION DETAILS	EXECUTION C
Row	customer_state	time_					
1	RJ	00:01:18					
2	SP	00:02:12					
3	RS	00:02:13					
4	SP	00:02:23					
5	RJ	00:02:26					

The lowest of average delivery time is 00.01.18 in RJ

5.4) 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
customer_state AS state,
ROUND(SUM(TIMESTAMP_DIFF(order_delivered_customer_date, order_purchase_timestamp,
DAY))/COUNT(ORDER_ID), 2) AS average_time_for_del,
ROUND(SUM(TIMESTAMP_DIFF(order_estimated_delivery_date, order_purchase_timestamp,
DAY))/COUNT(ORDER_ID), 2) AS average_est_dil_time,
FROM `TargetSQL.orders` o
INNER JOIN `TargetSQL.customers` c
ON o.customer_id=c.customer_id
WHERE order_status='delivered'
GROUP BY customer_state
ORDER BY (average_time_for_del-average_est_dil_time)

```

```

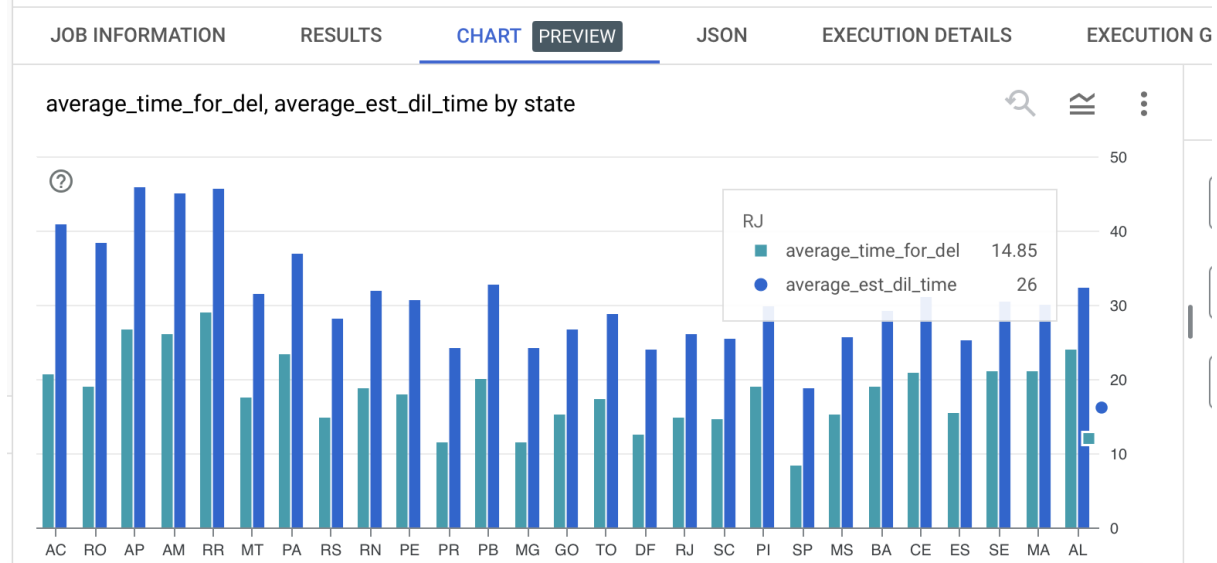
63 SELECT
64 customer_state AS state,
65 ROUND(SUM(TIMESTAMP_DIFF(order_delivered_customer_date, order_purchase_timestamp,
66 DAY))/COUNT(ORDER_ID), 2) AS average_time_for_del,
67 ROUND(SUM(TIMESTAMP_DIFF(order_estimated_delivery_date, order_purchase_timestamp,
68 DAY))/COUNT(ORDER_ID), 2) AS average_est_dil_time,
69 FROM `TargetSQL.orders` o
70 INNER JOIN `TargetSQL.customers` c
71 ON o.customer_id=c.customer_id
72 WHERE order_status='delivered'
73 GROUP BY customer_state
74 ORDER BY (average_time_for_del-average_est_dil_time)

```

Query results

JOB INFORMATION		RESULTS	CHART	PREVIEW	JSON	EXECUTION DETAILS	EXECU
Row	state	average_time_for_del	average_est_dil_time				
1	AC	20.64	40.73				
2	RO	18.91	38.39				
3	AP	26.73	45.87				
4	AM	25.99	44.92				
5	RR	28.98	45.63				
6	MT	17.59	31.37				
7	PA	23.32	36.79				
8	RS	14.82	28.16				
9	DN	19.92	31.97				

Query results



Average delivery time for the top 5 states is around 40.

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

```
select
p.payment_type,
extract(month from o.order_purchase_timestamp) as month_,
count(distinct p.order_id) as num_order
from `TargetSQL.payments` p inner join `TargetSQL.orders` o using(order_id)
group by 1, month_
order by month_, num_order desc
```

🏠 X 🔍 *Untitled X 📊 order_items X 📊 customers X 📊 orders X + ▾

🔍 Untitled ▶ RUN 💾 SAVE ▾ ⬇ DOWNLOAD 👤 SHARE ▾ 🕒 SCHEDULE ⚙ MORE ▾

75

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

77

78 select

79 p.payment_type,

80 extract(month from o.order_purchase_timestamp) as month_,

81 count(distinct p.order_id) as num_order

82 from `TargetSQL.payments` p inner join `TargetSQL.orders` o using(order_id)

83 group by 1, month_

84 order by month_, num_order desc

85

Query results 📄 SAV

JOB INFORMATION		RESULTS	CHART	PREVIEW	JSON	EXECUTION DETAILS	EXECUTION
Row	payment_type ▾	month_ ▾	num_order ▾				
1	credit_card	1	6093				
2	UPI	1	1715				
3	voucher	1	337				
4	debit_card	1	118				
5	credit_card	2	6582				
6	UPI	2	1723				
7	voucher	2	288				
8	debit_card	2	82				
9	credit_card	3	7682				
10	UPI	3	1942				

Results per page: !

Credit card is the highest used payment type whereas debit card is the least.

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

```
select
count(order_id) num_orders,
payment_installments
from `TargetSQL.payments`
```

```
where payment_installments >= 1
group by 2
```

🏠 🔍 *Untitled 🔍 📊 order_items 🔍 📊 customers 🔍 📊 orders 🔍 ➕

🔍 Untitled ▶ RUN 📄 SAVE ⬇️ DOWNLOAD 👤 SHARE 🕒 SCHEDULE ⚙️ MORE

86 # 6.2) Find the no. of orders placed on the basis of the payment installments that have been paid.

87

88 select

89 count(order_id) num_orders,

90 payment_installments

91 from `TargetSQL.payments`

92 where payment_installments >= 1

93 group by 2

94

95

Query results 📄 SAVE RESULT

JOB INFORMATION

RESULTS

CHART

PREVIEW

JSON

EXECUTION DETAILS

EXECUTION GRAPH

Row	num_orders	payment_installment
1	52546	1
2	12413	2
3	10461	3
4	7098	4
5	5239	5
6	3920	6
7	1626	7
8	4268	8
9	644	9
10	5328	10

Results per page 50

Majority of people use single payment installment.