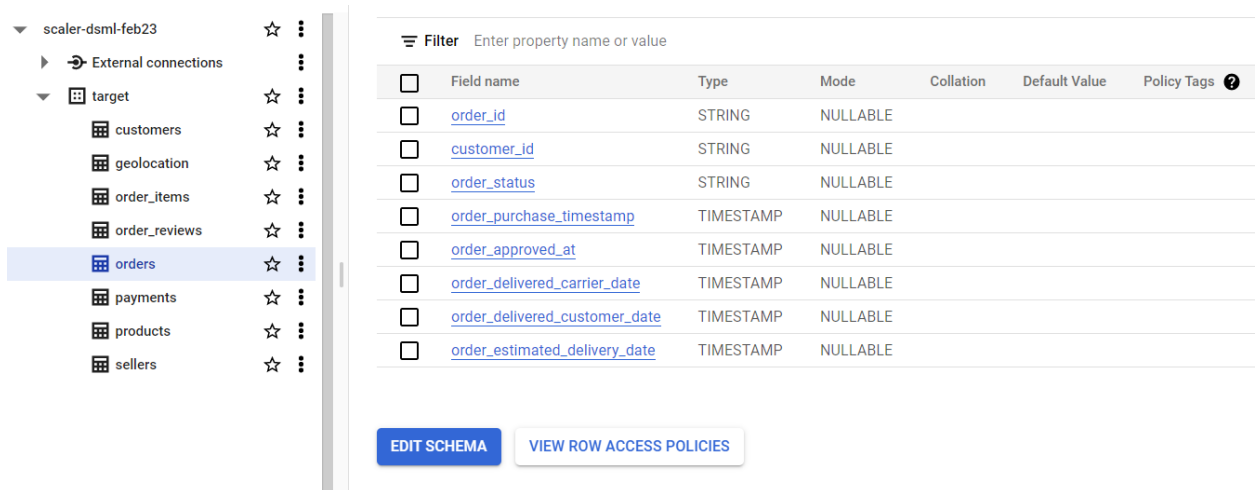


1. Import the dataset and do usual exploratory analysis steps like checking the structure & characteristics of the dataset

1. Data type of columns in a table



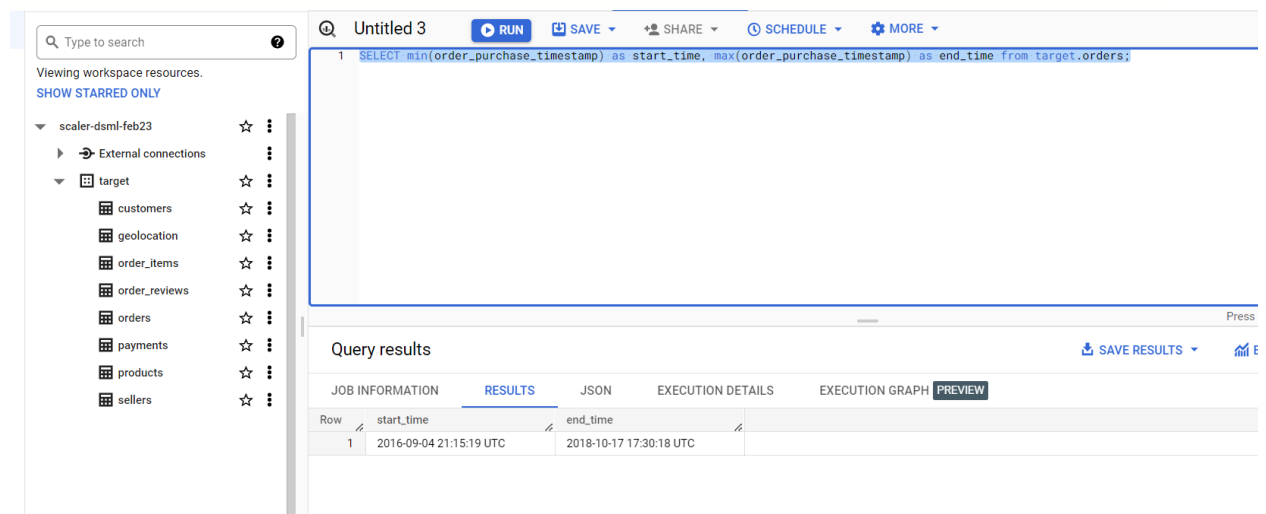
Field name	Type	Mode	Collation	Default Value	Policy Tags
order_id	STRING	NULLABLE			
customer_id	STRING	NULLABLE			
order_status	STRING	NULLABLE			
order_purchase_timestamp	TIMESTAMP	NULLABLE			
order_approved_at	TIMESTAMP	NULLABLE			
order_delivered_carrier_date	TIMESTAMP	NULLABLE			
order_delivered_customer_date	TIMESTAMP	NULLABLE			
order_estimated_delivery_date	TIMESTAMP	NULLABLE			

[EDIT SCHEMA](#) [VIEW ROW ACCESS POLICIES](#)

In orders table datatype are String and Timestamp. In this table only String and Timestamp data are present. Other type of data is not in this table.

2. Time period for which the data is given

```
SELECT
  MIN(order_purchase_timestamp) AS start_time,
  MAX(order_purchase_timestamp) AS end_time
FROM
  target.orders;
```



Query results

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

Time period for which the data is given is 2016-09-04 21:15:19 UTC to 2018-10-17 17:30:18 UTC

3. Cities and States of customers ordered during the given period

```
SELECT
    DISTINCT customers.customer_city,
    customers.customer_state
FROM
    target.customers AS customers
INNER JOIN
    `target.orders` AS orders
ON
    customers.customer_id = orders.customer_id;
```

The screenshot shows a data science IDE interface. On the left is an 'Explorer' panel with a search bar and a tree view of workspace resources. The tree shows a project named 'scaler-dsml-feb23' containing an 'External connections' folder and a 'target' folder. The 'target' folder contains several tables: 'customers', 'geolocation', 'order_items', 'order_reviews', 'orders', 'payments', 'products', and 'sellers'. The main editor area displays a SQL query in a file named 'Untitled 3'. The query is a SELECT statement that retrieves distinct customer cities and states from the 'target.customers' table, joined with the 'target.orders' table on the customer_id. Below the query editor, the 'Query results' section is visible, showing a table with two columns: 'customer_city' and 'customer_state'. The table contains 12 rows of data, with cities ranging from 'acu' to 'anta' and states ranging from 'RN' to 'RJ'. The interface also includes a top toolbar with buttons for 'RUN', 'SAVE', 'SHARE', 'SCHEDULE', and 'MORE'.

```
SELECT
    DISTINCT customers.customer_city,
    customers.customer_state
FROM
    target.customers AS customers
INNER JOIN
    `target.orders` AS orders
ON
    customers.customer_id = orders.customer_id;
```

Row	customer_city	customer_state
1	acu	RN
2	ico	CE
3	ipe	RS
4	ipu	CE
5	ita	SC
6	itu	SP
7	jau	SP
8	luz	MG
9	poa	SP
10	uba	MG
11	una	BA
12	anta	RJ

Orders are from multiple city and state. In this dataset customer base is huge and it is spread over multiple city and state.

2. In-depth Exploration:

1. Is there a growing trend on e-commerce in Brazil? How can we describe a complete scenario? Can we see some seasonality with peaks at specific months?

```
SELECT
EXTRACT(year
FROM
order_purchase_timestamp) AS year_,
EXTRACT(month
FROM
order_purchase_timestamp) AS month_,
COUNT(DISTINCT order_id) AS volume
FROM
target.orders
WHERE
order_status = 'delivered'
GROUP BY
year_,
month_
ORDER BY
year_,
month_;
```

Query results

JOB INFORMATION		RESULTS	JSON	EXEC
Row	year_	month_	volume	
1	2016	9	1	
2	2016	10	265	
3	2016	12	1	
4	2017	1	750	
5	2017	2	1653	
6	2017	3	2546	
7	2017	4	2303	
8	2017	5	3546	
9	2017	6	3135	
10	2017	7	3872	
11	2017	8	4193	
12	2017	9	4150	
13	2017	10	4478	
14	2017	11	7289	
15	2017	12	5513	
16	2018	1	7069	
17	2018	2	6555	

13	2017	10	4478
14	2017	11	7289
15	2017	12	5513
16	2018	1	7069
17	2018	2	6555
18	2018	3	7003
19	2018	4	6798
20	2018	5	6749
21	2018	6	6099
22	2018	7	6159
23	2018	8	6351

From 9/2016 to 11/2017 – There are up and down in business in month-to-month basis. But over all business are up trending.

From 11/2017 to 8/2018 – There are downward trends in month-to-month basis. Some months are in up trends, but overall result are down trends.

In Brazil trend on e-commerce is not Growing. It is slightly downtrends.

Till 11/2017 market is in uptrends but after that it is downtrends.

In 11/2017 there is a pick but we can not come to a conclusion that it is a seasonal pick but it is a monthly pick for 2017.

But if we use payment table then result is different. Please refer Question 4.1

2. What time do Brazilian customers tend to buy (Dawn, Morning, Afternoon or Night)?

0-6 - Dawn

7-12 - Morning

13-18- Afternoon

19-23- Night

```

SELECT
SUM(CASE WHEN hour_ BETWEEN 0 AND 6 THEN volume ELSE 0 END) AS dawn,
SUM(CASE WHEN hour_ BETWEEN 7 AND 12 THEN volume ELSE 0 END) AS Morning,
SUM(CASE WHEN hour_ BETWEEN 13 AND 18 THEN volume ELSE 0 END) AS Afternoon,
SUM(CASE WHEN hour_ BETWEEN 19 AND 23 THEN volume ELSE 0 END) AS Night,
FROM (
SELECT
EXTRACT(hour
FROM
order_purchase_timestamp) AS hour_,
COUNT(DISTINCT order_id) AS volume
FROM
target.orders
WHERE
order_status='delivered'
GROUP BY
hour_);

```

The screenshot shows a data analytics interface with a sidebar on the left containing a search bar and a list of workspace resources. The main area displays a SQL query in a text editor, which has been executed. Below the query, the 'Query results' section shows a table with the following data:

Row	dawn	Morning	Afternoon	Night
1	5072	26919	36965	27522

Brazilian customers tend to buy in Afternoon. Second slot is Night, but Morning slot is also very close to night slot. But time zone is UTC. I am assuming here UTC = Brazilian time zone. Otherwise we need to calculate UTC – Brazilian time difference.

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

1. Get month on month orders by states

```

SELECT
    customers.customer_state,
    COUNT(orders.order_id) AS order_count,
    EXTRACT(MONTH
FROM
    orders.order_purchase_timestamp) AS month,
    EXTRACT(YEAR
FROM
    orders.order_purchase_timestamp) AS year,

FROM
    `target.customers` AS customers
    INNER JOIN
    `target.orders` AS orders
    ON
        customers.customer_id = orders.customer_id

GROUP BY
    customer_state,
    month,
    year
ORDER BY
    month,
    year;

```

The screenshot displays a data analytics platform interface. On the left, a sidebar shows a project named 'scaler-dsml-feb23' with a list of tables: customers, geolocation, order_items, order_reviews, orders, payments, products, and sellers. The main area is titled 'Untitled 3' and contains a SQL query. Below the query editor, the 'Query results' section is active, showing a table with 12 rows and 5 columns: customer_state, order_count, month, and year. The results show the distribution of orders by state in 2017. At the bottom right, it indicates 'Results per page: 50' and '1 - 50 of 565'.

Row	customer_state	order_count	month	year
1	SP	299	1	2017
2	MG	108	1	2017
3	RS	54	1	2017
4	RJ	97	1	2017
5	PR	65	1	2017
6	PA	12	1	2017
7	GO	18	1	2017
8	BA	25	1	2017
9	SC	31	1	2017
10	RN	5	1	2017
11	ES	12	1	2017
12	CE	9	1	2017

2. Distribution of customers across the states in Brazil

```

SELECT
    customer_state,
    COUNT(customer_id) AS customer_count
FROM
    `target.customers`
GROUP BY
    customer_state;

```

Explorer

+ ADD

IK

🔍

Type to search

?

Viewing workspace resources.

SHOW STARRED ONLY

▼ scaler-dsml-feb23

▶ 🔗 External connections

▼ 🗃 target

🗃 customers

🗃 geolocation

🗃 order_items

🗃 order_reviews

🗃 orders

🗃 payments

🗃 products

🗃 sellers

Untitled 4

🔍

🏠

✕

🔍

orders

✕

🔍

*Untitled

▶ RUN

📄 SAVE

👤 SHAF

```

1 SELECT
2   customer_state,
3   COUNT(customer_id) AS customer_count
4 FROM
5   `target.customers`
6 GROUP BY
7   customer_state;

```

Query results

JOB INFORMATION

RESULTS

JSON

EXECUTE

Row	customer_state	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
12	PB	536
13	ES	2033
14	PR	5045

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

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

```
WITH
base AS (
SELECT
    EXTRACT(year
FROM
    orders.order_purchase_timestamp) AS year_,
    SUM(payments.payment_value) AS revenue
FROM
    target.orders AS orders
INNER JOIN
    target.payments AS payments
ON
    orders.order_id=payments.order_id
WHERE
    EXTRACT(month
FROM
    orders.order_purchase_timestamp) BETWEEN 0
AND 8
GROUP BY
    year_),
base2 AS(
SELECT
    *,
    LAG(revenue) OVER(ORDER BY year_ ASC) AS prev_revenue
FROM
    base)
SELECT
    *,
    (revenue-prev_revenue)/prev_revenue*100 AS per_INC
FROM
    base2;
```

Explorer

Type to search

Viewing workspace resources.

SHOW STARRED ONLY

scaler-dsml-feb23

External connections

target

customers

geolocation

order_items

order_reviews

orders

payments

products

sellers

Untitled 4

RUN

SAVE

SHARE

SCHE

```

1 WITH
2   base AS (
3     SELECT
4       EXTRACT(year
5         FROM
6           orders.order_purchase_timestamp) AS year_,
7       SUM(payments.payment_value) AS revenue
8     FROM
9       target.orders AS orders
10    INNER JOIN
11      target.payments AS payments
12    ON
13      orders.order_id=payments.order_id
14   WHERE
15     EXTRACT(month
16       FROM
17         orders.order_purchase_timestamp) BETWEEN 0
18     AND 8
19   GROUP BY
20     year_),
21   base2 AS(
22     SELECT
23       *,
24       LAG(revenue) OVER(ORDER BY year_ ASC) AS prev_revenue
25     FROM
26       base)
27   SELECT

```

Query results

JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS	
Row	year_	revenue	prev_revenue	per_INC	
1	2017	3669022.12...	null	null	
2	2018	8694733.83...	3669022.12...	136.976871...	

136.976871% increase in cost of orders from 2017 to 2018(include months between Jan to Aug only).

2. Mean & Sum of price and freight value by customer state

```

SELECT
  c.customer_state,
  AVG(oi.price) AS mean_price,
  SUM(oi.price) AS sum_price,
  AVG(oi.freight_value) AS mean_freight,
  SUM(oi.freight_value) AS sum_freight
FROM
  `target.customers` AS c
JOIN

```

```

`target.orders` AS o
ON
  c.customer_id = o.customer_id
JOIN
  `target.order_items` AS oi
ON
  o.order_id = oi.order_id
GROUP BY
  c.customer_state;

```

Viewing workspace resources.

SHOW STARRED ONLY

- scaler-dsml-feb23
 - External connections
 - target
 - customers
 - geolocation
 - order_items
 - order_reviews
 - orders
 - payments
 - products
 - sellers

Untitled 4

RUN SAVE SHARE SCHEDULE MORE

```

1 SELECT
2   c.customer_state,
3   AVG(oi.price) AS mean_price,
4   SUM(oi.price) AS sum_price,
5   AVG(oi.freight_value) AS mean_freight,
6   SUM(oi.freight_value) AS sum_freight
7 FROM
8   `target.customers` AS c
9 JOIN
10  `target.orders` AS o
11 ON
12  c.customer_id = o.customer_id

```

Query results

JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS	EXECUTION GRAPH	PRE
Row	customer_state	mean_price	sum_price	mean_freight	sum_freight	
1	MT	148.297184...	156453.529...	28.1662843...	29715.4300...	
2	MA	145.204150...	119648.219...	38.2570024...	31523.7700...	
3	AL	180.889211...	80314.81	35.8436711...	15914.5899...	
4	SP	109.653629...	5202955.05...	15.1472753...	718723.069...	
5	MG	120.748574...	1585308.02...	20.6301668...	270853.460...	
6	PE	145.508322...	262788.029...	32.9178626...	59449.6599...	
7	RJ	125.117818...	1824092.66...	20.9609239...	305589.310...	
8	DF	125.770548...	302603.939...	21.0413549...	50625.4999...	
9	RS	120.337453...	750304.020...	21.7358043...	135522.740...	
10	SE	153.041168...	58920.8500...	36.6531688...	14111.4699...	

5. Analysis on sales, freight and delivery time

1. Calculate days between purchasing, delivering and estimated delivery

```

SELECT
    DATE_DIFF(order_delivered_carrier_date, order_purchase_timestamp, DAY) AS purchasing_delivering,
    DATE_DIFF(order_estimated_delivery_date, order_purchase_timestamp, DAY) AS purchasing_estimated,
    DATE_DIFF(order_estimated_delivery_date, order_delivered_carrier_date, DAY) AS delivering_estimated,
FROM
    `target.orders`;

```

Untitled 3

RUN

SAVE

SHARE

SCHEDULE

MORE

```

1 SELECT
2   DATE_DIFF(order_delivered_carrier_date, order_purchase_timestamp, DAY) AS purchasing_delivering,
3   DATE_DIFF(order_estimated_delivery_date, order_purchase_timestamp, DAY) AS purchasing_estimated,
4   DATE_DIFF(order_estimated_delivery_date, order_delivered_carrier_date, DAY) AS delivering_estimated,
5 FROM
6   `target.orders`;

```

Query results

JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS	EXECUTION GRAPH	PREVIEW
Row		purchasing_deliv	purchasing_estim	delivering_estim		
1		9	50	41		
2		2	6	3		
3		6	44	37		
4		22	54	31		
5		33	56	23		
6		18	54	35		
7		39	56	16		
8		1	41	40		
9		1	3	2		
10		0	3	2		
11		0	47	46		
12		1	44	42		

Load more

2. Find time_to_delivery & diff_estimated_delivery. Formula for the same given below:
 - a. $\text{time_to_delivery} = \text{order_purchase_timestamp} - \text{order_delivered_customer_date}$
 - b. $\text{diff_estimated_delivery} = \text{order_estimated_delivery_date} - \text{order_delivered_customer_date}$

```

SELECT
    DATE_DIFF(order_delivered_customer_date, order_purchase_timestamp, DAY) AS time_
to_delivery,
    DATE_DIFF(order_estimated_delivery_date, order_delivered_customer_date, DAY) AS
diff_estimated_delivery
FROM
    `target.orders`;

```

Untitled 3 ▶ RUN 📄 SAVE 👤 SHARE 🕒 SCHEDULE ⚙️ MORE

```

1 SELECT
2     DATE_DIFF(order_delivered_customer_date, order_purchase_timestamp, DAY) AS time_to_delivery,
3     DATE_DIFF(order_estimated_delivery_date, order_delivered_customer_date, DAY) AS diff_estimated_delivery
4 FROM
5     `target.orders`;

```

Query results 📄

JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS	EXECUTION GRAPH	PREVIEW
Row	time_to_delivery	diff_estimated_delivery				
1	30	-12				
2	30	28				
3	35	16				
4	30	1				
5	32	0				
6	29	1				
7	43	-4				
8	40	-4				
9	37	-1				
10	33	-5				
11	38	-6				
12	36	-2				
13	34	0				
14	42	-11				

Results per page

- Group data by state, take mean of freight_value, time_to_delivery, diff_estimated_delivery

```

SELECT

c.customer_state,

```

```

    DATE_DIFF(o.order_delivered_customer_date, o.order_purchase_timestamp, DAY) AS time_t
o_delivery,
    DATE_DIFF(o.order_estimated_delivery_date, o.order_delivered_customer_date, DAY) AS d
iff_estimated_delivery,
    AVG(oi.freight_value) AS mean_freight
FROM
    `target.customers` AS c
JOIN
    `target.orders` AS o
ON
    c.customer_id = o.customer_id
JOIN
    `target.order_items` AS oi
ON
    o.order_id = oi.order_id
GROUP BY
    customer_state,
    order_delivered_customer_date,
    order_purchase_timestamp,
    order_estimated_delivery_date;

```

<div> Untitled orders *Untitled 3 *Untitled 4 custom </div>					
<div> Untitled 3 RUN SAVE SHARE SCHEDULE MORE </div>					
<pre> 1 SELECT 2 c.customer_state, 3 o.time_to_delivery, 4 o.diff_estimated_cost, 5 o.mean_freight 6 FROM customer c 7 JOIN orders o 8 ON c.customer_id = o.customer_id 9 </pre>					
Query results					
<div> JOB INFORMATION RESULTS JSON EXECUTION DETAILS EXECUTION GRAPH PR </div>					
Row	customer_state	time_to_delivery	diff_estimated_cost	mean_freight	
1	CE	46	-14	27.59	
2	CE	48	-16	27.1	
3	CE	37	-6	38.14	
4	CE	29	19	68.99	
5	SP	47	-28	12.89	
6	SP	39	20	17.29	
7	SP	34	0	8.31	
8	MG	30	-9	25.11	
9	MG	29	0	32.08	
10	MG	29	-6	14.43	
11	MG	44	-17	17.24	
12	MG	30	-2	17.61	
13	MG	37	-15	15.19	
14	BA	39	-13	22.82	
15	RS	34	-8	16.29	
16	RS	30	0	15.1	
17	RS	36	-8	18.23	

- Sort the data to get the following:
- Top 5 states with highest/lowest average freight value - sort in desc/asc limit 5

Highest –

```
SELECT
    c.customer_state,
    AVG(oi.freight_value) AS mean_freight
FROM
    `target.customers` AS c
JOIN
    `target.orders` AS o
ON
    c.customer_id = o.customer_id
JOIN
    `target.order_items` AS oi
ON
    o.order_id = oi.order_id
GROUP BY
    customer_state
ORDER BY
    mean_freight DESC
LIMIT
    5;
```


Untitled 3

RUNSAVESHARE

```
1 SELECT
2   c.customer_state,
3   AVG(oi.freight_value) AS mean_freight
4 FROM
5   `target.customers` AS c
6 JOIN
7   `target.orders` AS o
8 ON
9   c.customer_id = o.customer_id
10 JOIN
11   `target.order_items` AS oi
12 ON
13   o.order_id = oi.order_id
14 GROUP BY
15   customer_state
16 ORDER BY
17   mean_freight DESC
18 LIMIT
19   5;
```

Query results

JOB INFORMATION		RESULTS	JSON	EXECUTION
Row	customer_state	mean_freight		
1	RR	42.9844230...		
2	PB	42.7238039...		
3	RO	41.0697122...		
4	AC	40.0733695...		
5	PI	39.1479704...		

Lowest-

```
SELECT
    c.customer_state,
    AVG(oi.freight_value) AS mean_freight
FROM
    `target.customers` AS c
JOIN
    `target.orders` AS o
ON
    c.customer_id = o.customer_id
JOIN
    `target.order_items` AS oi
ON
    o.order_id = oi.order_id
GROUP BY
    customer_state
ORDER BY
    mean_freight
LIMIT
    5;
```

```

1 SELECT
2   c.customer_state,
3   AVG(oi.freight_value) AS mean_freight
4 FROM
5   `target.customers` AS c
6 JOIN
7   `target.orders` AS o
8 ON
9   c.customer_id = o.customer_id
10 JOIN
11   `target.order_items` AS oi
12 ON
13   o.order_id = oi.order_id
14 GROUP BY
15   customer_state
16 ORDER BY
17   mean_freight
18 LIMIT
19   5;

```

Query results

JOB INFORMATION		RESULTS	JSON	EX
Row	customer_state	mean_freight		
1	SP	15.1472753...		
2	PR	20.5316515...		
3	MG	20.6301668...		
4	RJ	20.9609239...		
5	DF	21.0413549...		

6. Top 5 states with highest/lowest average time to delivery

Highest –

```
SELECT
    c.customer_state,
    AVG(DATE_DIFF(o.order_delivered_customer_date, o.order_purchase_timestamp, DAY
    )) AS avg_time_to_delivery
FROM
    `target.customers` AS c
JOIN
    `target.orders` AS o
ON
    c.customer_id = o.customer_id
JOIN
    `target.order_items` AS oi
ON
    o.order_id = oi.order_id
GROUP BY
    customer_state
ORDER BY
    avg_time_to_delivery desc
LIMIT
    5;
```

Untitled
orders
*Untitled 3
*Untitled 4
customers

Untitled 3
RUN
SAVE
SHARE
SCHEDULE
MORE

```

1 SELECT
2   c.customer_state,
3   AVG(DATE_DIFF(o.order_delivered_customer_date, o.order_purchase_timestamp, DAY)) AS avg_time_to_delivery
4 FROM
5   `target.customers` AS c
6 JOIN
7   `target.orders` AS o
8 ON
9   c.customer_id = o.customer_id
10 JOIN
11   `target.order_items` AS oi
12 ON
13   o.order_id = oi.order_id
14 GROUP BY
15   customer_state
16 ORDER BY
17   avg_time_to_delivery desc
18 LIMIT
19   5;

```

Query results

JOB INFORMATION
RESULTS
JSON
EXECUTION DETAILS
EXECUTION GRAPH
PREVIEW

Row	customer_state	avg_time_to_delivery
1	RR	27.826086956521738
2	AP	27.753086419753075
3	AM	25.963190184049076
4	AL	23.992974238875881
5	PA	23.301707779886126

Lowest –

```

SELECT
    c.customer_state,
    AVG(DATE_DIFF(o.order_delivered_customer_date, o.order_purchase_timestamp, DAY
)) AS avg_time_to_delivery
FROM
    `target.customers` AS c
JOIN
    `target.orders` AS o
ON
    c.customer_id = o.customer_id
JOIN
    `target.order_items` AS oi
ON
    o.order_id = oi.order_id
GROUP BY

```

```

customer_state
ORDER BY
avg_time_to_delivery
LIMIT
5;

```

Untitled
orders
*Untitled 3
*Untitled 4
customers

Untitled 3
RUN
SAVE
SHARE
SCHEDULE
MORE

```

1 SELECT
2   c.customer_state,
3   AVG(DATE_DIFF(o.order_delivered_customer_date, o.order_purchase_timestamp, DAY)) AS avg_time_to_delivery
4 FROM
5   `target.customers` AS c
6 JOIN
7   `target.orders` AS o
8 ON
9   c.customer_id = o.customer_id
10 JOIN
11   `target.order_items` AS oi
12 ON
13   o.order_id = oi.order_id
14 GROUP BY
15   customer_state
16 ORDER BY
17   avg_time_to_delivery
18 LIMIT
19   5;

```

Query results

JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS	EXECUTION GRAPH	PREVIEW
Row	customer_state	avg_time_to_delivery				
1	SP	8.25960855241909				
2	PR	11.480793060718735				
3	MG	11.515522180072811				
4	DF	12.501486199575384				
5	SC	14.520985846754517				

- Top 5 states where delivery is really fast/ not so fast compared to estimated date

Really Fast compared to estimated date –

```

SELECT
  c.customer_state,
  AVG(DATE_DIFF(o.order_estimated_delivery_date, o.order_delivered_customer_date,
DAY)) AS diff_estimated_delivery
FROM

```

```

`target.customers` AS c
JOIN
`target.orders` AS o
ON
c.customer_id = o.customer_id
JOIN
`target.order_items` AS oi
ON
o.order_id = oi.order_id
GROUP BY
customer_state
ORDER BY
diff_estimated_delivery desc
LIMIT
5;

```

Untitled 3
RUN
SAVE
SHARE
SCHEDULE
MORE

```

1 SELECT
2   c.customer_state,
3   AVG(DATE_DIFF(o.order_estimated_delivery_date, o.order_delivered_customer_date, DAY)) AS diff_estimated_delivery
4 FROM
5   `target.customers` AS c
6 JOIN
7   `target.orders` AS o
8 ON
9   c.customer_id = o.customer_id
10 JOIN
11   `target.order_items` AS oi
12 ON
13   o.order_id = oi.order_id
14 GROUP BY
15   customer_state
16 ORDER BY
17   diff_estimated_delivery desc
18 LIMIT
19   5;

```

Query results
SAVE RESU

JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS	EXECUTION GRAPH	PREVIEW
Row	customer_state	diff_estimated_delivery				
1	AC	20.010989010989011				
2	RO	19.080586080586091				
3	AM	18.975460122699378				
4	AP	17.444444444444446				
5	RR	17.434782608695649				

Not so fast compared to estimated date –

```

SELECT
    c.customer_state,
    AVG(DATE_DIFF(o.order_estimated_delivery_date, o.order_delivered_customer_date, DAY)) AS diff_estimated_delivery
FROM
    `target.customers` AS c
JOIN
    `target.orders` AS o
ON
    c.customer_id = o.customer_id
JOIN
    `target.order_items` AS oi
ON
    o.order_id = oi.order_id
GROUP BY
    customer_state
ORDER BY
    diff_estimated_delivery
LIMIT
    5;

```

Untitled 3
RUN
SAVE
SHARE
SCHEDULE
MORE

```

1 SELECT
2   c.customer_state,
3   AVG(DATE_DIFF(o.order_estimated_delivery_date, o.order_delivered_customer_date, DAY)) AS diff_estimated_delivery
4 FROM
5   `target.customers` AS c
6 JOIN
7   `target.orders` AS o
8 ON
9   c.customer_id = o.customer_id
10 JOIN
11   `target.order_items` AS oi
12 ON
13   o.order_id = oi.order_id
14 GROUP BY
15   customer_state
16 ORDER BY
17   diff_estimated_delivery
18 LIMIT
19   5;

```

Query results
SAVE RES

JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS	EXECUTION GRAPH	PREVIEW
Row	customer_state	diff_estimated_delivery				
1	AL	7.9765807962529349				
2	MA	9.1099999999999923				
3	SE	9.1653333333333276				
4	ES	9.7685393258427116				
5	BA	10.119467825142538				

6. Payment type analysis:

1. Month over Month count of orders for different payment types

```
WITH
  base AS(
    SELECT
      payment_type,
      COUNT(payments.order_id) AS count_of_order,
      EXTRACT(month
    FROM
      orders.order_purchase_timestamp) AS month_,
      EXTRACT(year
    FROM
      orders.order_purchase_timestamp) AS year_
    FROM
      target.orders AS orders
    INNER JOIN
      target.payments AS payments
    ON
      orders.order_id=payments.order_id
    GROUP BY
      payment_type,
      month_,
      year_
    ORDER BY
      year_,
      month_)
  base2 AS(
    SELECT
      *,
      LAG(count_of_order) OVER(PARTITION BY payment_type ORDER BY year_, month_) AS prev_
    count_of_order
    FROM
      base
    ORDER BY
      year_,
      month_)
  SELECT
    *,
    (count_of_order-prev_count_of_order)AS Diff_count_of_order
  FROM
    base2
  ORDER BY
    year_,
    month_;
```

```
7 FROM
8 orders.order_purchase_timestamp) AS month_,
9 EXTRACT(year
10 FROM
```

Query results

JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS		EXECUTION GRAPH		PF
Row	payment_type	count_of_order	month_	year_	prev_count_of_order	Diff_count_of_order		
1	credit_card	3	9	2016	null	null		
2	voucher	23	10	2016	null	null		
3	debit_card	2	10	2016	null	null		
4	credit_card	254	10	2016	3	251		
5	UPI	63	10	2016	null	null		
6	credit_card	1	12	2016	254	-253		
7	voucher	61	1	2017	23	38		
8	debit_card	9	1	2017	2	7		
9	credit_card	583	1	2017	1	582		
10	UPI	197	1	2017	63	134		
11	voucher	119	2	2017	61	58		
12	debit_card	13	2	2017	9	4		
13	credit_card	1356	2	2017	583	773		
14	UPI	398	2	2017	197	201		
15	voucher	200	3	2017	119	81		
16	debit_card	31	3	2017	13	18		

2. Count of orders based on the no. of payment installments

```
SELECT
    COUNT(order_id),
    payment_installments
FROM
    target.payments
GROUP BY
    payment_installments;
```

<div> Untitled orders </div>			
<div> Untitled 6 <div> RUN </div> <div> SAVE </div> </div>			
<pre> 1 SELECT 2 COUNT(order_id), 3 payment_installments 4 FROM 5 target.payments 6 GROUP BY 7 payment_installments; </pre>			
Query results			
JOB INFORMATION		RESULTS	JSON
Row	f0_	payment_installments	
1	2	0	
2	52546	1	
3	12413	2	
4	10461	3	
5	7098	4	
6	5239	5	
7	3920	6	
8	1626	7	
9	4268	8	
10	644	9	
11	5328	10	
12	23	11	
13	133	12	
14	16	13	

