Business Case: Target SQL

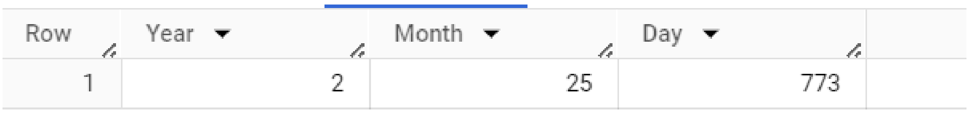
- What Does ‘good’ look like?

○ **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.
   - **customer_id = STRING**
   - **customer_unique_id = STRING**
   - **customer_zip_code_prefix = INTEGER**
   - **customer_city = STRING**
   - **customer_state = STRING**

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

```
SELECT
      DATE_DIFF(DATE (MAX(order_purchase_timestamp)),
                DATE (MIN(order_purchase_timestamp)),
                YEAR) as `Year`,
      DATE_DIFF(DATE (MAX(order_purchase_timestamp)),
                DATE (MIN(order_purchase_timestamp)),
                MONTH) as `Month`,
      DATE_DIFF(DATE (MAX(order_purchase_timestamp)),
                DATE (MIN(order_purchase_timestamp)),
                DAY) as `Day`
FROM
      `scalar-dsml-sql-411406.Target.orders`
```

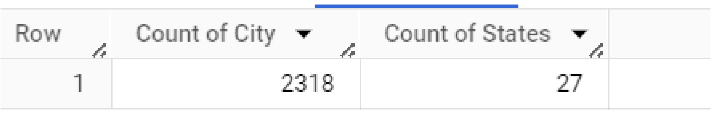


Row	Year	Month	Day
1	2	25	773

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

- Here I count the Cities and States of customers who ordered during Jan-March month of 2018.

```
SELECT
    COUNT(DISTINCT c.customer_city) as `Count of City`,
    COUNT(DISTINCT c.customer_state) as `Count of States`
FROM
    `scalar-dsml-sql-411406.Target.customers` c
JOIN
    `scalar-dsml-sql-411406.Target.orders` o ON c.customer_id =
o.customer_id
WHERE
    o.order_purchase_timestamp >= '2018-01-01' AND
    o.order_purchase_timestamp < '2018-04-01'
```

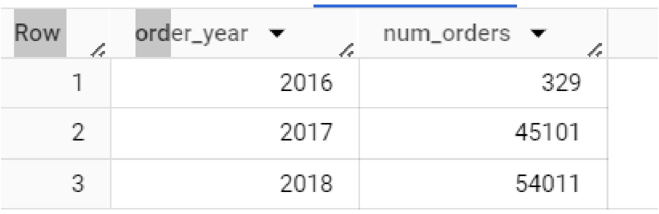


Row	Count of City	Count of States
1	2318	27

- **In-depth Exploration:**

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

```
SELECT
    EXTRACT(YEAR FROM order_purchase_timestamp) AS order_year,
    COUNT(*) AS num_orders
FROM
    `scalar-dsml-sql-411406.Target.orders`
GROUP BY
    order_year
ORDER BY
    order_year;
```



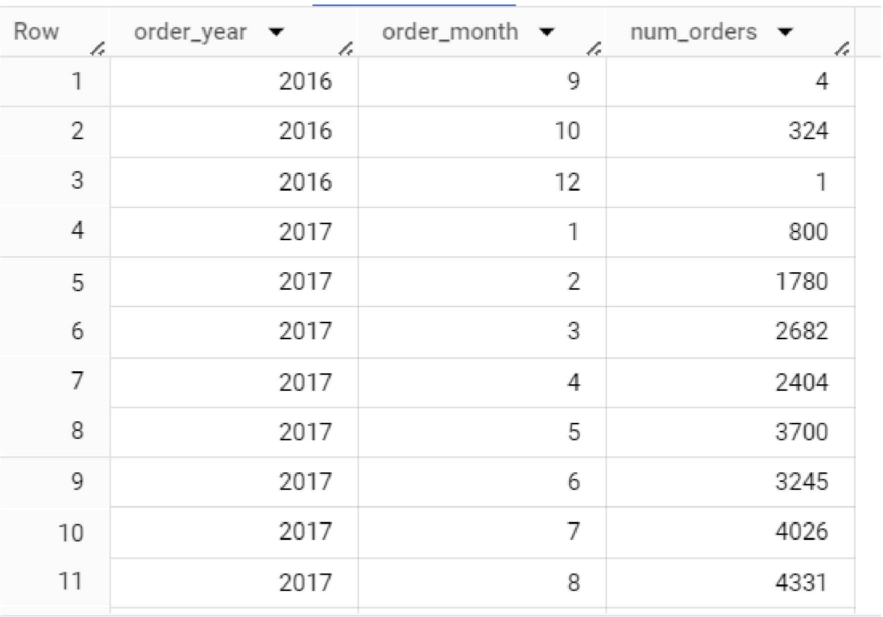
Row	order_year	num_orders
1	2016	329
2	2017	45101
3	2018	54011

- **Insight Information: We can see here there has been a significant increase in the number of orders from 2016 to 2017 and a further increase from 2017 to 2018.**
- **However, to ascertain a consistent upward trend, we should ideally consider more years of data. If the pattern of increase continues over additional years, it would indicate a consistent upward trend in the number of orders placed.**
- **In this specific case, based on the provided data alone, it seems there is a growing trend in the number of orders placed over the past years.**

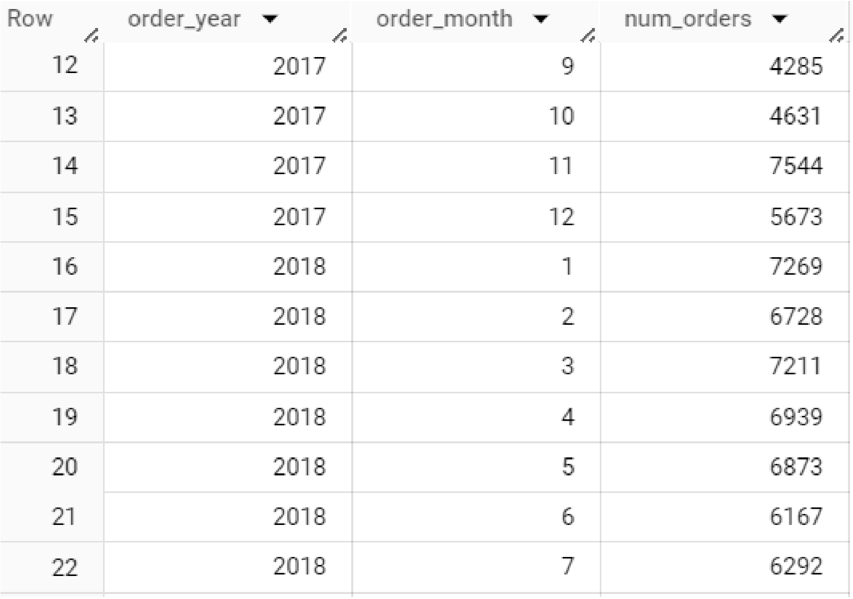
2. **Can we see some kind of monthly seasonality in terms of the no. of orders being placed?**

```
SELECT
    EXTRACT(YEAR FROM order_purchase_timestamp) AS order_year,
    EXTRACT(MONTH FROM order_purchase_timestamp) AS order_month,
    COUNT(*) AS num_orders
FROM
     `scalar-dsml-sql-411406.Target.orders`
GROUP BY
```

```
    order_year, order_month
ORDER BY
    order_year, order_month;
```



Row	order_year	order_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
11	2017	8	4331



Row	order_year	order_month	num_orders
12	2017	9	4285
13	2017	10	4631
14	2017	11	7544
15	2017	12	5673
16	2018	1	7269
17	2018	2	6728
18	2018	3	7211
19	2018	4	6939
20	2018	5	6873
21	2018	6	6167
22	2018	7	6292

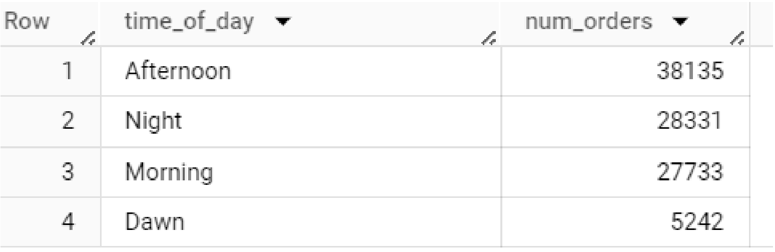
- **Analysis of Monthly Seasonality in Number of Orders**
- **Seasonal Peaks: There are noticeable peaks in certain months across the years, indicating potential seasonal trends in ordering behavior. For instance, March 2017, November 2017, and January 2018 exhibit significantly higher numbers of orders compared to other months.**

- ➔ **Consistent Patterns: While there are fluctuations month to month, there appears to be some consistency in the seasonal peaks, with certain months consistently showing higher order counts across different years.**

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 'Morning'
        WHEN EXTRACT(HOUR FROM order_purchase_timestamp) BETWEEN 13
AND 18 THEN 'Afternoon'
        ELSE 'Night'
    END AS time_of_day,
    COUNT(*) AS num_orders
FROM
   `scalar-dsml-sql-411406.Target.orders`

GROUP BY
    time_of_day
ORDER BY
    num_orders DESC;
```



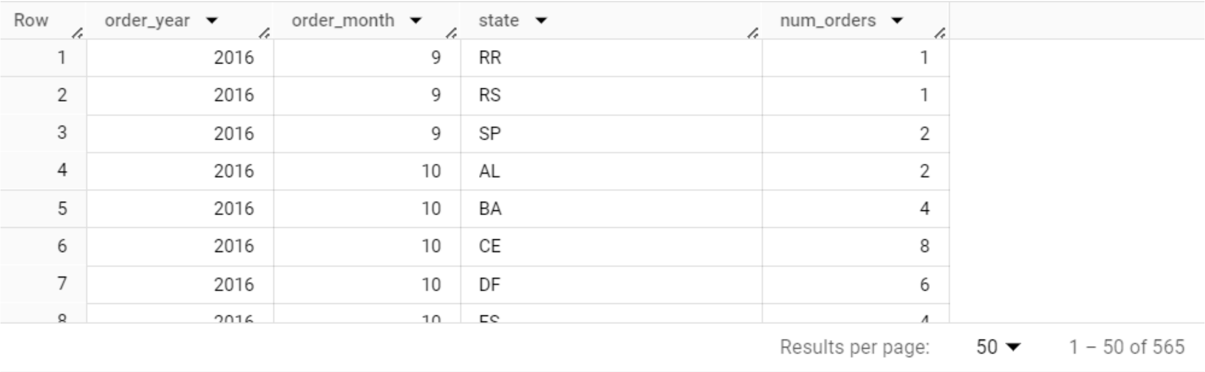
Row	time_of_day	num_orders
1	Afternoon	38135
2	Night	28331
3	Morning	27733
4	Dawn	5242

- ➔ **From this data, we can observe that Brazilian customers mostly place their orders during the afternoon, followed by the night, morning, and dawn. This suggests that the peak ordering hours for Brazilian customers are in the afternoon and early evening.**

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

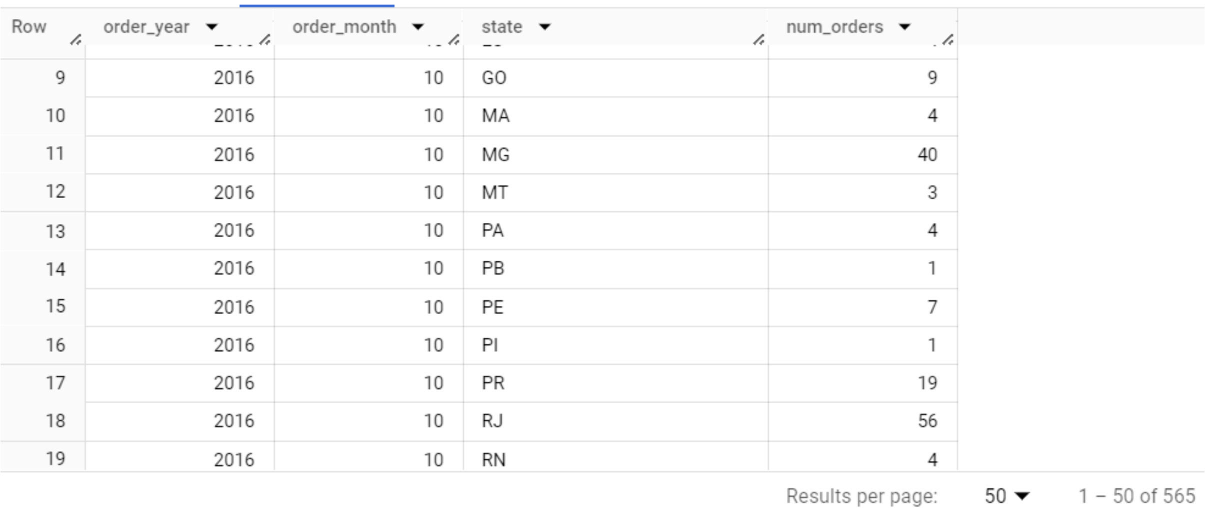
1. **Get the month on month no. of orders placed in each state.**

```
SELECT
    EXTRACT(YEAR FROM o.order_purchase_timestamp) AS order_year,
    EXTRACT(MONTH FROM o.order_purchase_timestamp) AS order_month,
    c.customer_state AS state,
    COUNT(*) AS num_orders
FROM
    `scalar-dsml-sql-411406.Target.orders` o
JOIN
    `scalar-dsml-sql-411406.Target.customers` c ON o.customer_id = c.customer_id
GROUP BY
    order_year, order_month, state
ORDER BY
    order_year, order_month, state;
```



Row	order_year	order_month	state	num_orders
1	2016	9	RR	1
2	2016	9	RS	1
3	2016	9	SP	2
4	2016	10	AL	2
5	2016	10	BA	4
6	2016	10	CE	8
7	2016	10	DF	6
8	2016	10	ES	4

Results per page: 50 ▼ 1 – 50 of 565

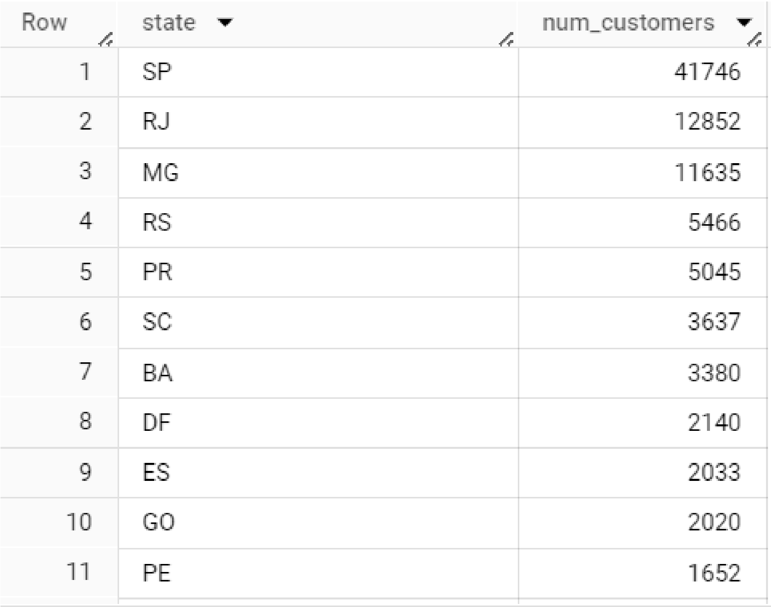


Row	order_year	order_month	state	num_orders
9	2016	10	GO	9
10	2016	10	MA	4
11	2016	10	MG	40
12	2016	10	MT	3
13	2016	10	PA	4
14	2016	10	PB	1
15	2016	10	PE	7
16	2016	10	PI	1
17	2016	10	PR	19
18	2016	10	RJ	56
19	2016	10	RN	4

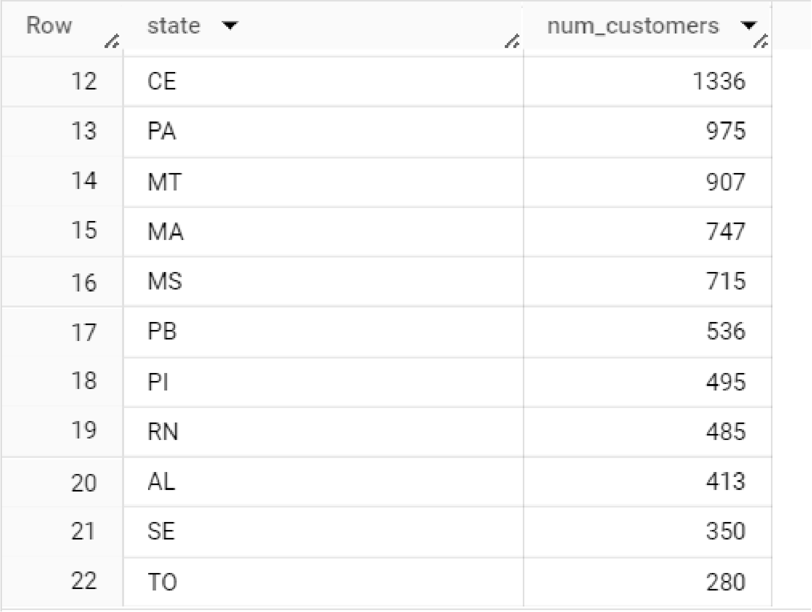
Results per page: 50 ▼ 1 – 50 of 565

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

```
SELECT
    customer_state AS state,
    COUNT(DISTINCT customer_id) AS num_customers
FROM
    `scalar-dsml-sql-411406.Target.customers`
GROUP BY
    state
ORDER BY
    num_customers DESC;
```



Row	state	num_customers
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

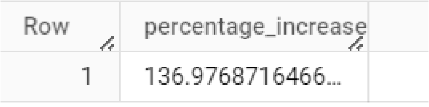


Row	state	num_customers
12	CE	1336
13	PA	975
14	MT	907
15	MA	747
16	MS	715
17	PB	536
18	PI	495
19	RN	485
20	AL	413
21	SE	350
22	TO	280

- **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.

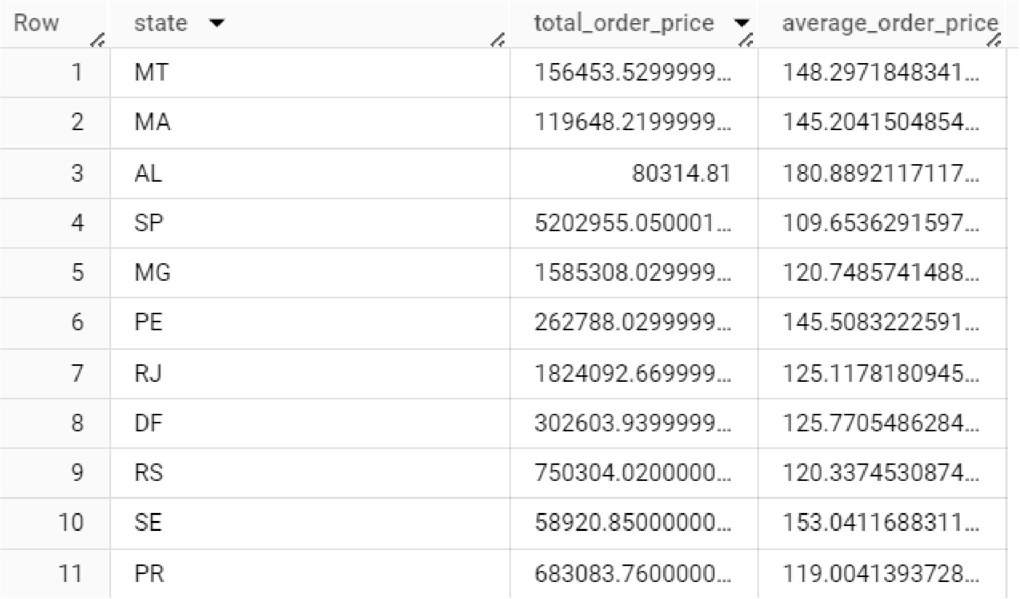
```
WITH YearlyPayments AS (
    SELECT
        EXTRACT(YEAR FROM o.order_purchase_timestamp) AS order_year,
        EXTRACT(MONTH FROM o.order_purchase_timestamp) AS order_month,
        SUM(p.payment_value) AS total_payment
    FROM
        `scalar-dsml-sql-411406.Target.orders` o
    JOIN
        `scalar-dsml-sql-411406.Target.payments` p ON o.order_id =
p.order_id
    WHERE
        EXTRACT(YEAR FROM o.order_purchase_timestamp) IN (2017, 2018) AND
        EXTRACT(MONTH FROM o.order_purchase_timestamp) BETWEEN 1 AND 8
    GROUP BY
        order_year, order_month
)
SELECT
    (year2018.total_payment - year2017.total_payment) /
year2017.total_payment * 100 AS percentage_increase
FROM
    (SELECT SUM(total_payment) AS total_payment FROM YearlyPayments WHERE
order_year = 2017) AS year2017,
    (SELECT SUM(total_payment) AS total_payment FROM YearlyPayments WHERE
order_year = 2018) AS year2018
```



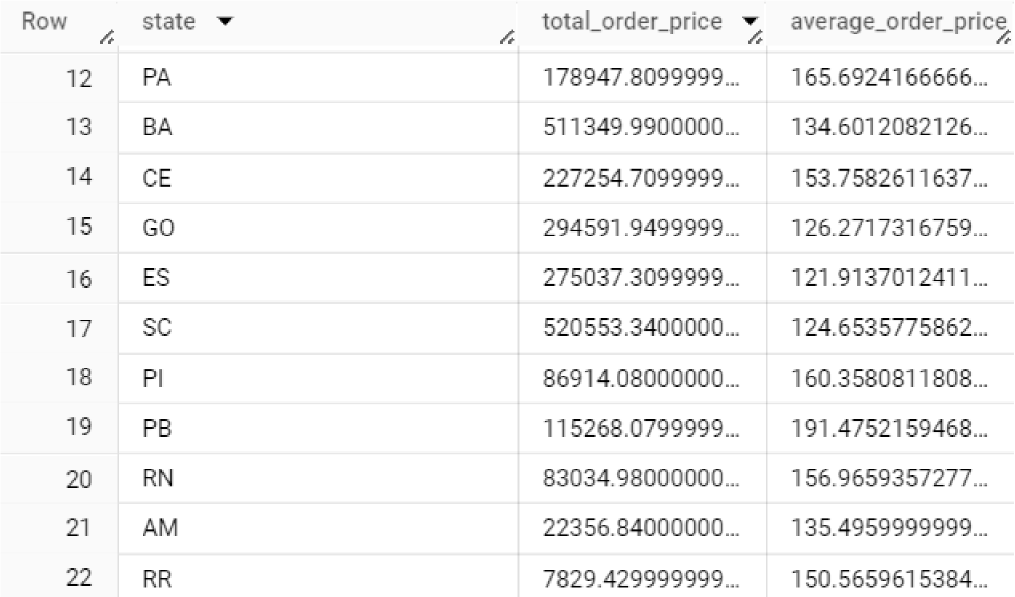
Row	percentage_increase
1	136.9768716466...

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

```sql
SELECT
    c.customer_state AS state,
    SUM(oi.price) AS total_order_price,
    AVG(oi.price) AS average_order_price
FROM
    `scalar-dsml-sql-411406.Target.orders` o
JOIN
    `scalar-dsml-sql-411406.Target.customers` c ON o.customer_id = c.customer_id
JOIN
    `scalar-dsml-sql-411406.Target.order_items` oi ON o.order_id = oi.order_id
GROUP BY
    c.customer_state;
```



Row	state	total_order_price	average_order_price
1	MT	156453.5299999...	148.2971848341...
2	MA	119648.2199999...	145.2041504854...
3	AL	80314.81	180.8892117117...
4	SP	5202955.050001...	109.6536291597...
5	MG	1585308.029999...	120.7485741488...
6	PE	262788.0299999...	145.5083222591...
7	RJ	1824092.669999...	125.1178180945...
8	DF	302603.9399999...	125.7705486284...
9	RS	750304.0200000...	120.3374530874...
10	SE	58920.85000000...	153.0411688311...
11	PR	683083.7600000...	119.0041393728...



Row	state	total_order_price	average_order_price
12	PA	178947.8099999...	165.6924166666...
13	BA	511349.9900000...	134.6012082126...
14	CE	227254.7099999...	153.7582611637...
15	GO	294591.9499999...	126.2717316759...
16	ES	275037.3099999...	121.9137012411...
17	SC	520553.3400000...	124.6535775862...
18	PI	86914.08000000...	160.3580811808...
19	PB	115268.0799999...	191.4752159468...
20	RN	83034.98000000...	156.9659357277...
21	AM	22356.84000000...	135.4959999999...
22	RR	7829.429999999...	150.5659615384...

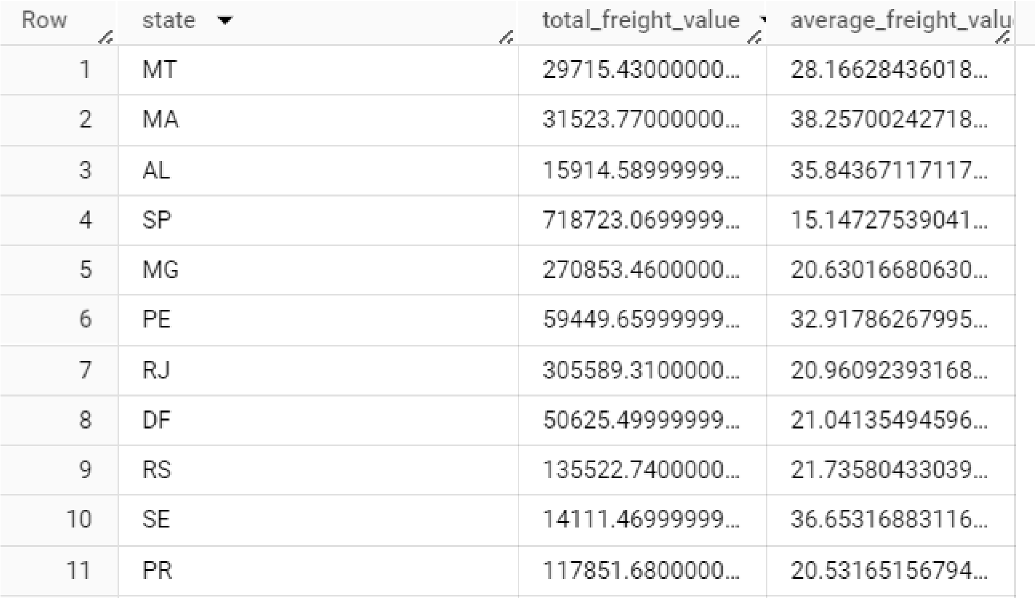
- ➔ **These values provide insights into the purchasing behavior and average spending habits of customers in each state. For example:**

- ➔ **São Paulo (SP) has the highest total order price, indicating a high volume of orders or higher-priced items being purchased.**

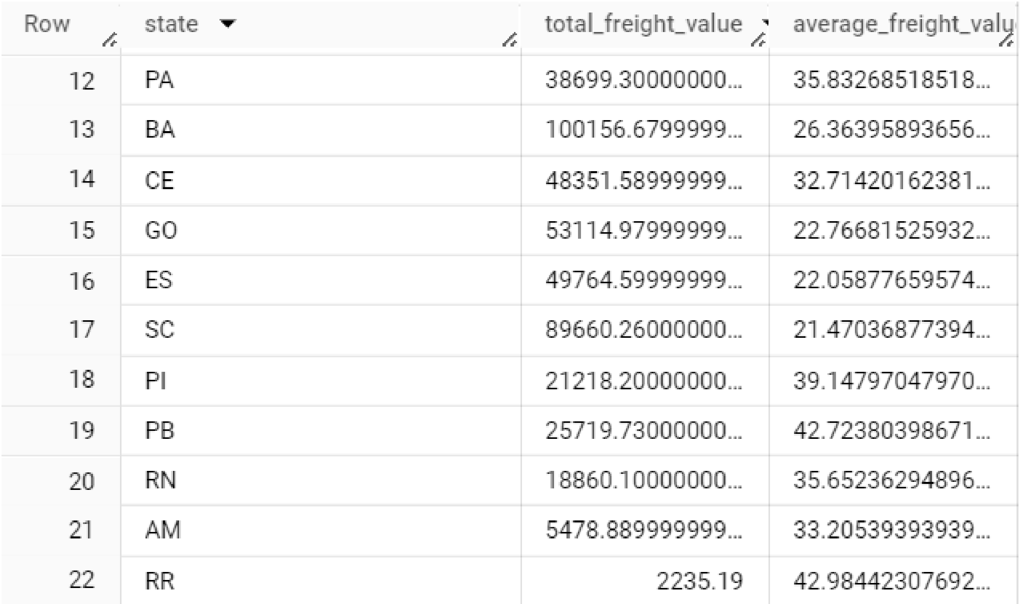
- ➔ **Paraíba (PB) has the highest average order price, suggesting that although the total order price might not be as high as in some other states, individual orders tend to be more expensive on average.**

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

```
SELECT c.customer_state AS state,
       SUM(oi.freight_value) AS total_freight_value,
       AVG(oi.freight_value) AS average_freight_value
FROM `scalar-dsml-sql-411406.Target.orders` o
JOIN `scalar-dsml-sql-411406.Target.order_items` oi ON o.order_id =
oi.order_id
JOIN `scalar-dsml-sql-411406.Target.customers` c ON o.customer_id =
c.customer_id
GROUP BY c.customer_state;
```



Row	state	total_freight_value	average_freight_valu
1	MT	29715.43000000...	28.16628436018...
2	MA	31523.77000000...	38.25700242718...
3	AL	15914.58999999...	35.84367117117...
4	SP	718723.0699999...	15.14727539041...
5	MG	270853.4600000...	20.63016680630...
6	PE	59449.65999999...	32.91786267995...
7	RJ	305589.3100000...	20.96092393168...
8	DF	50625.49999999...	21.04135494596...
9	RS	135522.7400000...	21.73580433039...
10	SE	14111.46999999...	36.65316883116...
11	PR	117851.6800000...	20.53165156794...

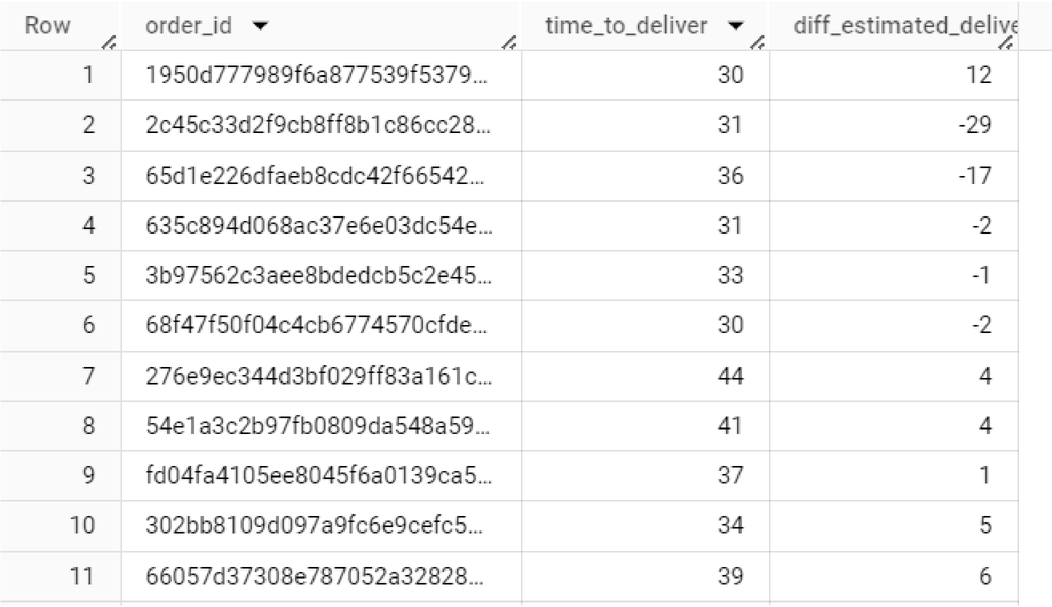


Row	state	total_freight_value	average_freight_valu
12	PA	38699.30000000...	35.83268518518...
13	BA	100156.6799999...	26.36395893656...
14	CE	48351.58999999...	32.71420162381...
15	GO	53114.97999999...	22.76681525932...
16	ES	49764.59999999...	22.05877659574...
17	SC	89660.26000000...	21.47036877394...
18	PI	21218.20000000...	39.14797047970...
19	PB	25719.73000000...	42.72380398671...
20	RN	18860.10000000...	35.65236294896...
21	AM	5478.889999999...	33.20539393939...
22	RR	2235.19	42.98442307692...

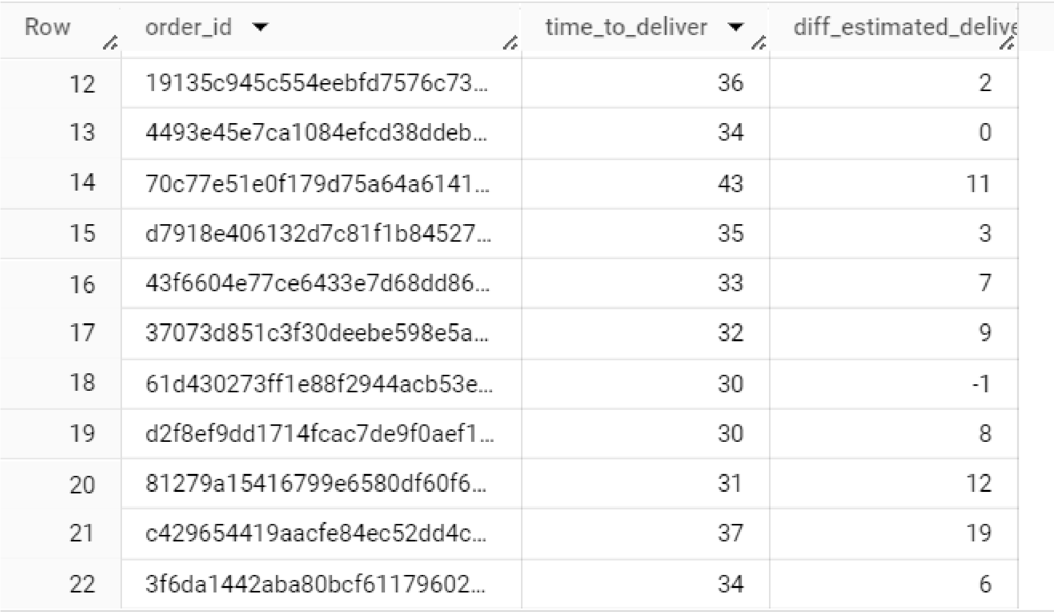
- **Analysis based on sales, freight and delivery time.**

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_delivered_customer_date - order_estimated_delivery_date

```
SELECT
      order_id,
      DATE_DIFF(DATE(order_delivered_customer_date),
  DATE(order_purchase_timestamp), DAY) AS time_to_deliver,
      DATE_DIFF(DATE(order_delivered_customer_date),
  DATE(order_estimated_delivery_date), DAY) AS diff_estimated_delivery
  FROM
      `scalar-dsml-sql-411406.Target.orders`
  WHERE
      order_delivered_customer_date IS NOT NULL
      AND order_purchase_timestamp IS NOT NULL
      AND order_estimated_delivery_date IS NOT NULL;
```



Row	order_id	time_to_deliver	diff_estimated_delive
1	1950d777989f6a877539f5379...	30	12
2	2c45c33d2f9cb8ff8b1c86cc28...	31	-29
3	65d1e226dfaeb8cdc42f66542...	36	-17
4	635c894d068ac37e6e03dc54e...	31	-2
5	3b97562c3aee8bdedcb5c2e45...	33	-1
6	68f47f50f04c4cb6774570cfde...	30	-2
7	276e9ec344d3bf029ff83a161c...	44	4
8	54e1a3c2b97fb0809da548a59...	41	4
9	fd04fa4105ee8045f6a0139ca5...	37	1
10	302bb8109d097a9fc6e9cefc5...	34	5
11	66057d37308e787052a32828...	39	6



Row	order_id	time_to_deliver	diff_estimated_delive
12	19135c945c554eebfd7576c73...	36	2
13	4493e45e7ca1084efcd38ddeb...	34	0
14	70c77e51e0f179d75a64a6141...	43	11
15	d7918e406132d7c81f1b84527...	35	3
16	43f6604e77ce6433e7d68dd86...	33	7
17	37073d851c3f30deebe598e5a...	32	9
18	61d430273ff1e88f2944acb53e...	30	-1
19	d2f8ef9dd1714fcac7de9f0aef1...	30	8
20	81279a15416799e6580df60f6...	31	12
21	c429654419aacfe84ec52dd4c...	37	19
22	3f6da1442aba80bcf61179602...	34	6

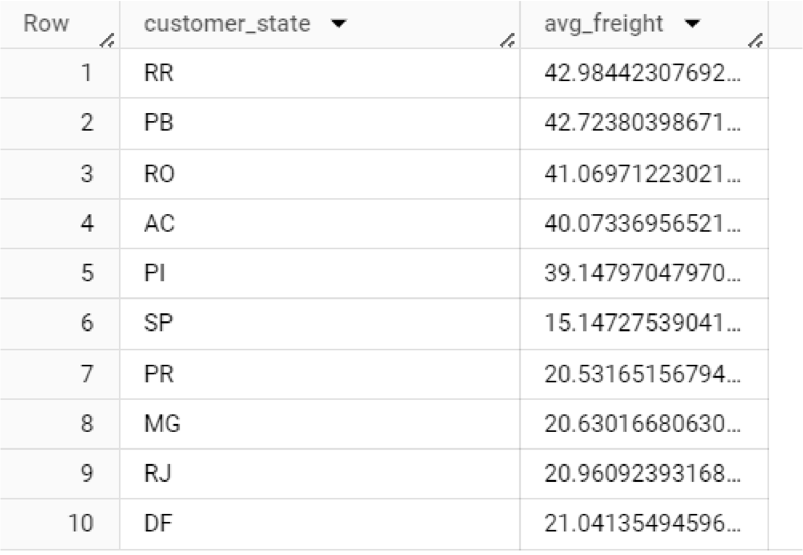
2. Find out the top 5 states with the highest & lowest average freight value.

```
WITH AvgFreight AS (
    SELECT
        c.customer_state,
        AVG(oi.freight_value) AS avg_freight
    FROM
        `scalar-dsml-sql-411406.Target.customers` c
    INNER JOIN
        `scalar-dsml-sql-411406.Target.orders` o ON c.customer_id =
o.customer_id
    INNER JOIN
        `scalar-dsml-sql-411406.Target.order_items` oi ON o.order_id =
oi.order_id
    GROUP BY
        c.customer_state
)

SELECT
    customer_state,
    avg_freight
FROM
    (
        SELECT
            customer_state,
            avg_freight
        FROM
            AvgFreight
        ORDER BY
            avg_freight DESC
        LIMIT 5
    )

UNION ALL

SELECT
    customer_state,
    avg_freight
FROM
    (
        SELECT
            customer_state,
            avg_freight
        FROM
            AvgFreight
        ORDER BY
            avg_freight ASC
        LIMIT 5
    );
```



Row	customer_state	avg_freight
1	RR	42.98442307692...
2	PB	42.72380398671...
3	RO	41.06971223021...
4	AC	40.07336956521...
5	PI	39.14797047970...
6	SP	15.14727539041...
7	PR	20.53165156794...
8	MG	20.63016680630...
9	RJ	20.96092393168...
10	DF	21.04135494596...

➔ **Above First 5 rows for the highest average fright and last 5 rows shows lowest average fright.**

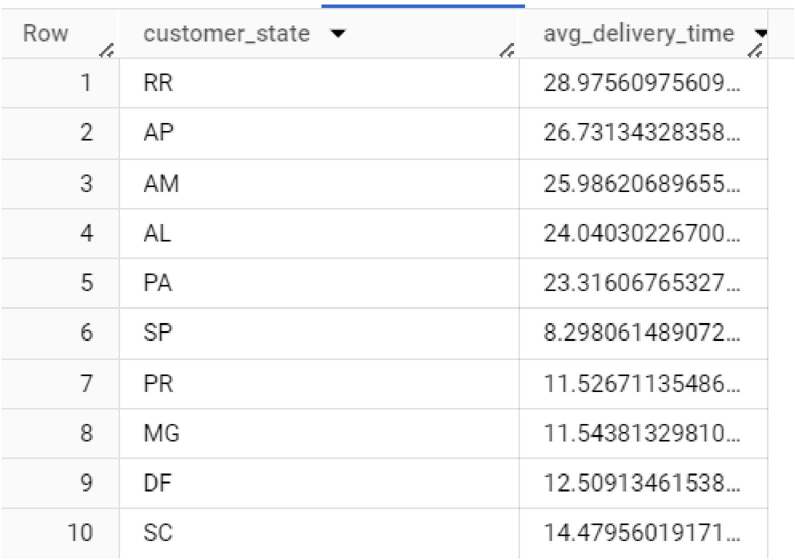
3. Find out the top 5 states with the highest & lowest average delivery time

```sql
WITH AvgDeliveryTime AS (
    SELECT
        c.customer_state,
        AVG(DATE_DIFF(o.order_delivered_customer_date,
o.order_purchase_timestamp, DAY)) AS avg_delivery_time
    FROM
        `scalar-dsml-sql-411406.Target.orders` o
    JOIN
        `scalar-dsml-sql-411406.Target.customers` c ON o.customer_id =
c.customer_id
    WHERE
        o.order_delivered_customer_date IS NOT NULL
        AND o.order_purchase_timestamp IS NOT NULL
    GROUP BY
        c.customer_state
)

SELECT
    customer_state,
    avg_delivery_time
FROM
    (
        SELECT
            customer_state,
            avg_delivery_time
        FROM
            AvgDeliveryTime
        ORDER BY
            avg_delivery_time DESC
        LIMIT 5
    )

UNION ALL

SELECT
    customer_state,
    avg_delivery_time
FROM
    (
        SELECT
            customer_state,
            avg_delivery_time
        FROM
            AvgDeliveryTime
        ORDER BY
            avg_delivery_time ASC
        LIMIT 5
    );
```



Row	customer_state	avg_delivery_time
1	RR	28.97560975609...
2	AP	26.73134328358...
3	AM	25.98620689655...
4	AL	24.04030226700...
5	PA	23.31606765327...
6	SP	8.298061489072...
7	PR	11.52671135486...
8	MG	11.54381329810...
9	DF	12.50913461538...
10	SC	14.47956019171...

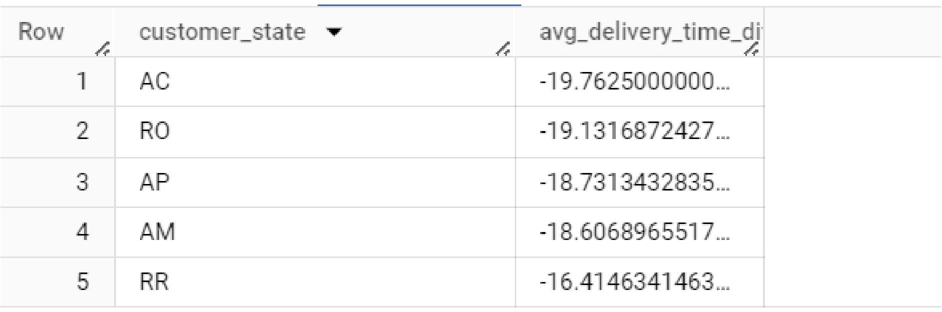
- ➔ **Above First 5 rows for the highest average delivery time and last 5 rows shows lowest average delivery time.**

4. Find out the top 5 states where the order delivery is really fast as compared to the estimated date of delivery.
You can use the difference between the averages of actual & estimated delivery date to figure out how fast the delivery was for each state.

```
WITH DeliveryTimeDiff AS (
    SELECT
        c.customer_state,
        AVG(DATE_DIFF(o.order_delivered_customer_date,
o.order_estimated_delivery_date, DAY)) AS avg_delivery_time_diff
    FROM
        `scalar-dsml-sql-411406.Target.orders` o
    JOIN
        `scalar-dsml-sql-411406.Target.customers` c ON o.customer_id =
c.customer_id
    WHERE
        o.order_delivered_customer_date IS NOT NULL
        AND o.order_estimated_delivery_date IS NOT NULL
    GROUP BY
        c.customer_state
)

SELECT
    customer_state,
    avg_delivery_time_diff
```

```
FROM
    (
        SELECT
            customer_state,
            avg_delivery_time_diff
        FROM
            DeliveryTimeDiff
        ORDER BY
            avg_delivery_time_diff ASC
        LIMIT 5
    );
```



Row	customer_state	avg_delivery_time_di
1	AC	-19.7625000000...
2	RO	-19.1316872427...
3	AP	-18.7313432835...
4	AM	-18.6068965517...
5	RR	-16.4146341463...

➔ **The output shows the top 5 states where the order delivery is faster compared to the estimated date of delivery, based on the average difference between the actual delivery date and the estimated delivery date:**

➔ **AC (Acre): Average delivery time difference of approximately -19.76 days (deliveries are on average almost 20 days earlier than estimated).**

➔ **RO (Rondônia): Average delivery time difference of approximately -19.13 days.**

➔ **AP (Amapá): Average delivery time difference of approximately -18.73 days.**

➔ **AM (Amazonas): Average delivery time difference of approximately -18.61 days.**

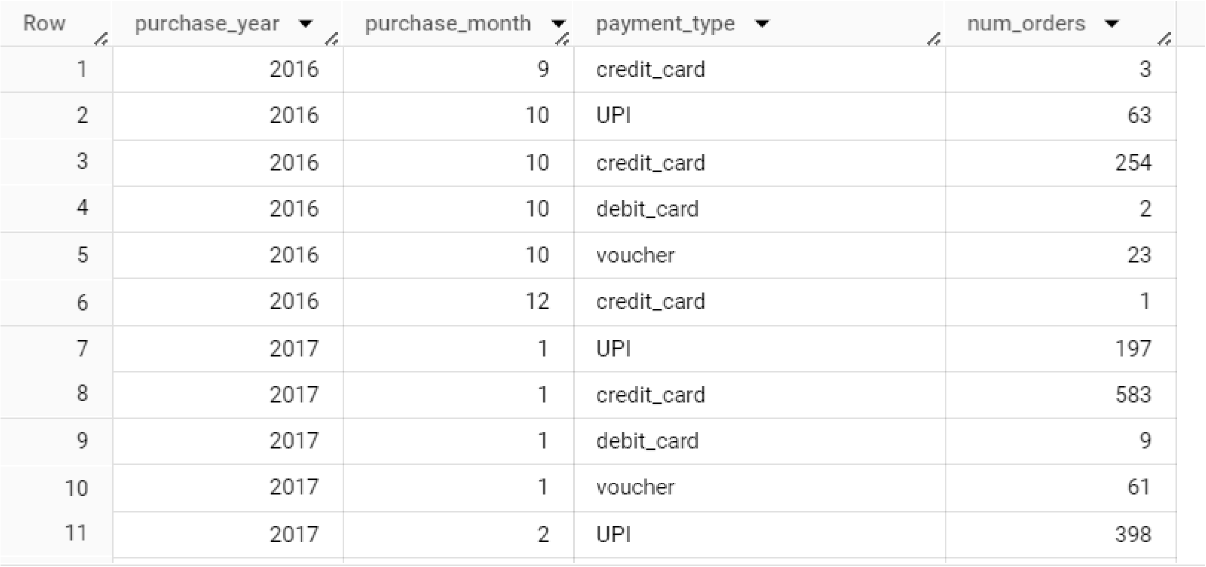
➔ **RR (Roraima): Average delivery time difference of approximately -16.41 days.**

- **Analysis based on the payments:**

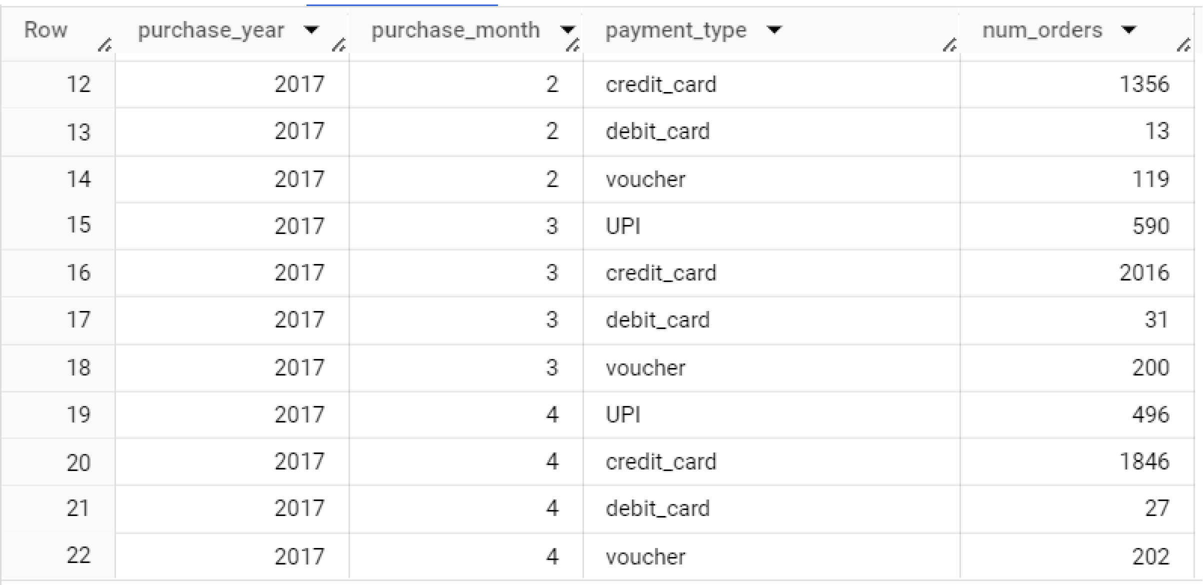
1. **Find the month on month no. of orders placed using different payment types.**

```
WITH MonthlyOrders AS (
    SELECT
        EXTRACT(YEAR FROM order_purchase_timestamp) AS purchase_year,
        EXTRACT(MONTH FROM order_purchase_timestamp) AS purchase_month,
        payment_type,
        COUNT(order_id) AS num_orders
    FROM
        `scalar-dsml-sql-411406.Target.payments`
    JOIN
        `scalar-dsml-sql-411406.Target.orders` USING(order_id)
    GROUP BY
        purchase_year, purchase_month, payment_type
)

SELECT
    purchase_year,
    purchase_month,
    payment_type,
    num_orders
FROM
    MonthlyOrders
ORDER BY
    purchase_year, purchase_month, payment_type;
```



Row	purchase_year	purchase_month	payment_type	num_orders
1	2016	9	credit_card	3
2	2016	10	UPI	63
3	2016	10	credit_card	254
4	2016	10	debit_card	2
5	2016	10	voucher	23
6	2016	12	credit_card	1
7	2017	1	UPI	197
8	2017	1	credit_card	583
9	2017	1	debit_card	9
10	2017	1	voucher	61
11	2017	2	UPI	398



Row	purchase_year	purchase_month	payment_type	num_orders
12	2017	2	credit_card	1356
13	2017	2	debit_card	13
14	2017	2	voucher	119
15	2017	3	UPI	590
16	2017	3	credit_card	2016
17	2017	3	debit_card	31
18	2017	3	voucher	200
19	2017	4	UPI	496
20	2017	4	credit_card	1846
21	2017	4	debit_card	27
22	2017	4	voucher	202

➔ **Based on the provided data, we can observe the following trends:**

➔ **Overall Growth: The number of orders placed seems to increase over time, as indicated by the increasing number of orders each month.**

➔ **Popular Payment Methods: Credit card and UPI appear to be the most popular payment methods, as they consistently have the highest number of orders across different months.**

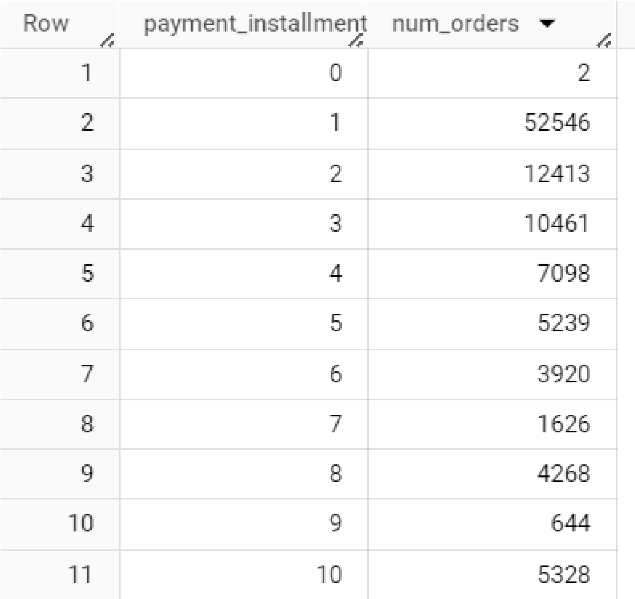
➔ **Seasonal Variations: There might be some seasonal variations in the number of orders, with certain months having higher order volumes compared to others.**

➔ **Emerging Payment Methods: While credit card and UPI dominate, other payment methods like debit card and vouchers also show consistent usage, albeit with lower frequencies.**

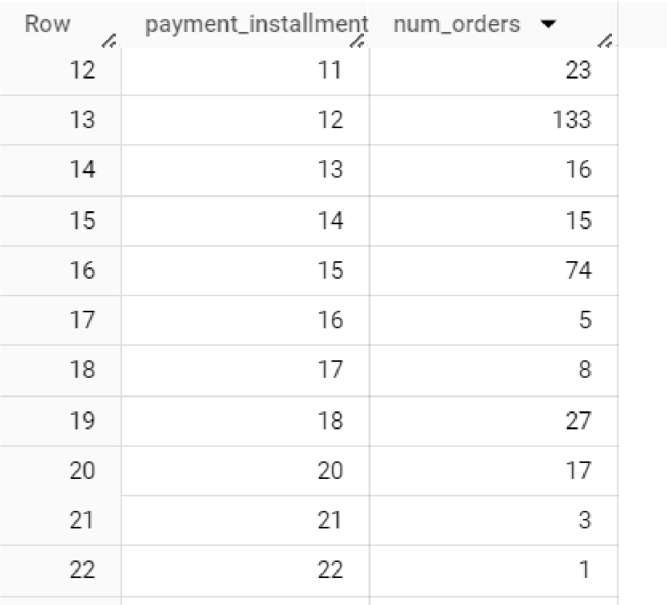
➔ **Trend Analysis: By analyzing month-on-month changes in the number of orders for each payment method, businesses can identify patterns, assess the effectiveness of promotions or marketing campaigns, and make informed decisions to optimize their payment processing strategies.**

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_orders
FROM
    `scalar-dsml-sql-411406.Target.payments`
GROUP BY
    payment_installments
ORDER BY
    payment_installments;
```



Row	payment_installment	num_orders ▾
1	0	2
2	1	52546
3	2	12413
4	3	10461
5	4	7098
6	5	5239
7	6	3920
8	7	1626
9	8	4268
10	9	644
11	10	5328



Row	payment_installment	num_orders ▾
12	11	23
13	12	133
14	13	16
15	14	15
16	15	74
17	16	5
18	17	8
19	18	27
20	20	17
21	21	3
22	22	1

➔ **Based on the data that we got , here is the breakdown of the number of orders placed based on the payment installments:**

0 installments: 2 orders

1 installment: 52,546 orders

2 installments: 12,413 orders

3 installments: 10,461 orders

4 installments: 7,098 orders

5 installments: 5,239 orders

6 installments: 3,920 orders

7 installments: 1,626 orders

8 installments: 4,268 orders

9 installments: 644 orders

10 installments: 5,328 orders

11 installments: 23 orders

12 installments: 133 orders

13 installments: 16 orders

14 installments: 15 orders

15 installments: 74 orders

16 installments: 5 orders

17 installments: 8 orders

18 installments: 27 orders

20 installments: 17 orders

21 installments: 3 orders

22 installments: 1 order

23 installments: 1 order

24 installments: 18 orders