

Target SQL Case Study

Q1 :- Import the dataset and do usual exploratory analysis steps like checking the structure & characteristics of the dataset:

Q1.1 Data type of all columns in the "customers" table.

Query:-

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

Screenshot:-

```
1 SELECT column_name, data_type
2 FROM `target.INFORMATION_SCHEMA.COLUMNS`
3 WHERE table_name = 'customers'
```

Query results

JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS
Row	column_name	data_type		
1	customer_id	STRING		
2	customer_unique_id	STRING		
3	customer_zip_code_prefix	INT64		
4	customer_city	STRING		
5	customer_state	STRING		

Q1.2 Get the time range between which the orders were placed

Query:-

```
SELECT
MIN(order_purchase_timestamp) AS first_purchase_timestamp,
MAX(order_purchase_timestamp) AS last_purchase_timestamp
FROM `target.orders`;
```

Screenshot:-

```
1 SELECT MIN(order_purchase_timestamp) AS first_purchase_timestamp,
2 MAX(order_purchase_timestamp) AS last_purchase_timestamp
3 FROM `target.orders`
```

Query results

JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS	EXECUTION GRAPH
Row	first_purchase_timestamp	last_purchase_timestamp			
1	2016-09-04 21:15:19 UTC	2018-10-17 17:30:18 UTC			

Q1.3 Count the number of Cities and States in our dataset

Query:-

```
SELECT
COUNT(DISTINCT customer_city) AS num_cities,
COUNT(DISTINCT customer_state) AS num_states
FROM `target.customers`;
```

Screenshot:-

🔍

Untitled 4

▶ RUN

💾 SAVE ▾

👤 SHARE ▾

🕒 SCHEDULE ▾

⚙️ MORE ▾

1

SELECT COUNT(DISTINCT customer_city) AS num_cities,

2

||| COUNT(DISTINCT customer_state) AS num_states

3

FROM `target.customers`;

Query results

JOB INFORMATION

RESULTS

JSON

EXECUTION DETAILS

EXECUTION GRAPH

Row

num_cities ▾

num_states ▾

1

4119

27

Q2:- In-depth Exploration:

Q2.1 :- Is there a growing trend in the no. of orders placed over the past years?

Query:-

```
SELECT
EXTRACT(YEAR FROM order_purchase_timestamp) AS year,
EXTRACT(MONTH FROM order_purchase_timestamp) AS month,
COUNT(*) AS order_count
FROM `target.orders`
GROUP BY year, month
ORDER BY year, month
```

Screenshot:-

Untitled 5

RUN

SAVE

SHARE

SCHEDULE

```

1 SELECT
2 EXTRACT(YEAR FROM order_purchase_timestamp) AS year,
3 EXTRACT(MONTH FROM order_purchase_timestamp) AS month,
4 COUNT(*) AS order_count
5 FROM `target.orders`
6 GROUP BY year, month
7 ORDER BY year, month

```

Query results

JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS	EX
Row	year	month	order_count		
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		

Analysis:-

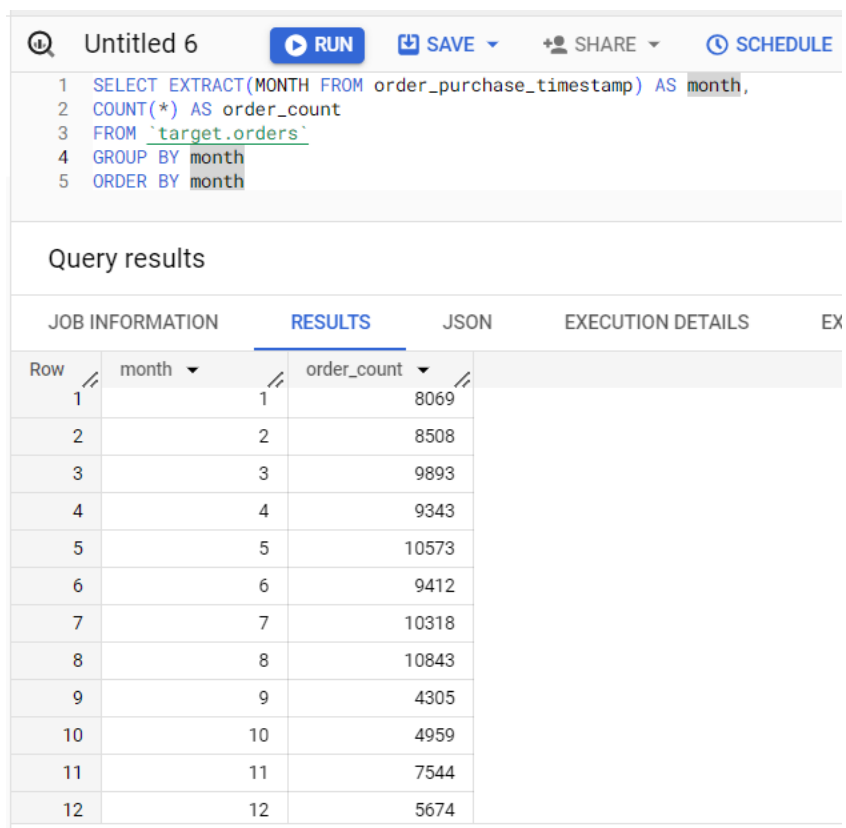
1. Max order placed in November 2017 i.e., 7544 orders.
2. Total order in
2016 = 329 , 2017 = 45101 , 2018 = 54011
3. We can conclude from the above data that no. of orders is increasing year over year

Q2.2 Can we see some kind of monthly seasonality in terms of the no. of orders being placed?

Query :-

```
SELECT EXTRACT(MONTH FROM order_purchase_timestamp) AS month,  
COUNT(*) AS order_count  
FROM `target.orders`  
GROUP BY month  
ORDER BY month
```

Screenshot:-



The screenshot shows a SQL query editor interface. At the top, there's a toolbar with icons for search, run, save, share, and schedule. Below the toolbar, the query is displayed in a monospace font. The query is: `SELECT EXTRACT(MONTH FROM order_purchase_timestamp) AS month, COUNT(*) AS order_count FROM `target.orders` GROUP BY month ORDER BY month`. Below the query, there's a section titled "Query results" which contains a table with two columns: "month" and "order_count". The table has 12 rows, numbered 1 to 12, showing the order count for each month. The order counts are: 8069, 8508, 9893, 9343, 10573, 9412, 10318, 10843, 4305, 4959, 7544, and 5674.

Row	month	order_count
1	1	8069
2	2	8508
3	3	9893
4	4	9343
5	5	10573
6	6	9412
7	7	10318
8	8	10843
9	9	4305
10	10	4959
11	11	7544
12	12	5674

Analysis :-

1. Max order placed in August
2. From the above data we can conclude that the number of orders increases as we reach near the year mid (May, June, July, August), then we can see a decline in the orders during the year end.
3. The above data forms a bell curve having peak in mid year

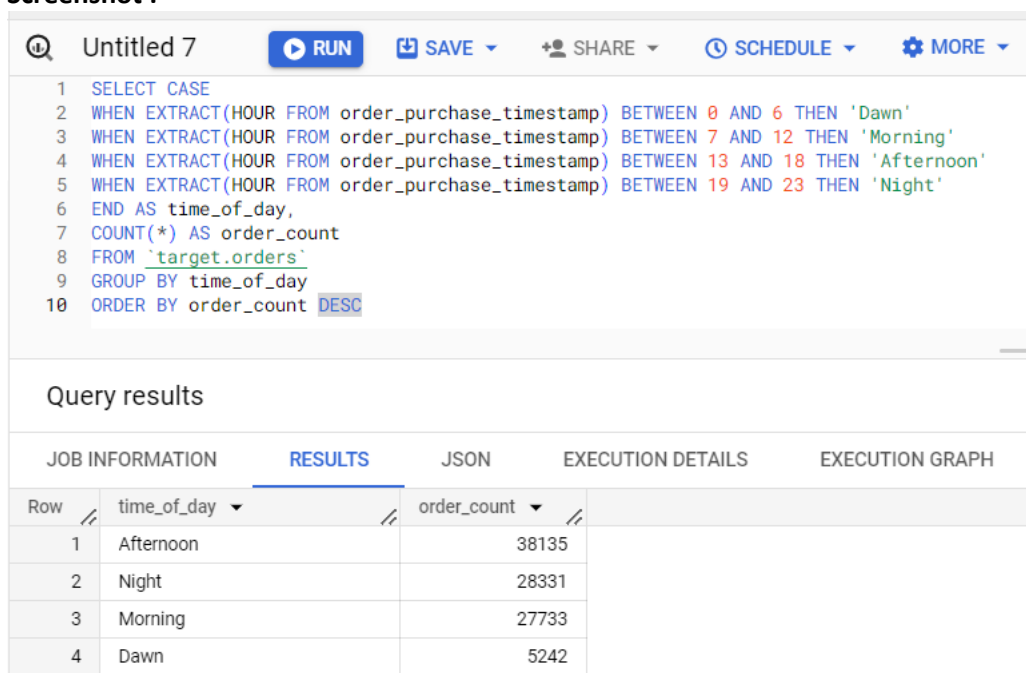
Q2.3 During what time of the day, do the Brazilian customers mostly place their orders? (Dawn, Morning, Afternoon or Night)

- 0-6 hrs : Dawn
- 7-12 hrs : Mornings
- 13-18 hrs : Afternoon
- 19-23 hrs : Night

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 '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(*) AS order_count
FROM `target.orders`
GROUP BY time_of_day
ORDER BY order_count DESC
```

Screenshot :-



The screenshot shows a SQL query editor interface. At the top, there's a toolbar with buttons for 'RUN', 'SAVE', 'SHARE', 'SCHEDULE', and 'MORE'. Below the toolbar, the query is displayed in a text area. The query is a SQL statement that categorizes orders by time of day and counts them. Below the query, there's a section titled 'Query results' which contains a table with the results of the query. The table has two columns: 'time_of_day' and 'order_count'. The results are sorted by 'order_count' in descending order.

JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS	EXECUTION GRAPH
Row	time_of_day	order_count			
1	Afternoon	38135			
2	Night	28331			
3	Morning	27733			
4	Dawn	5242			

Analysis :-

1. From the above results we can conclude the Maximum orders are places in Afternoon
2. Followed by Night and Morning with a marginal difference of 1k orders
3. Least orders are placed in Dawn

Note:- Time zone of the given data set is not converted and used as it is. Incase we have to convert the time zone result may differ.

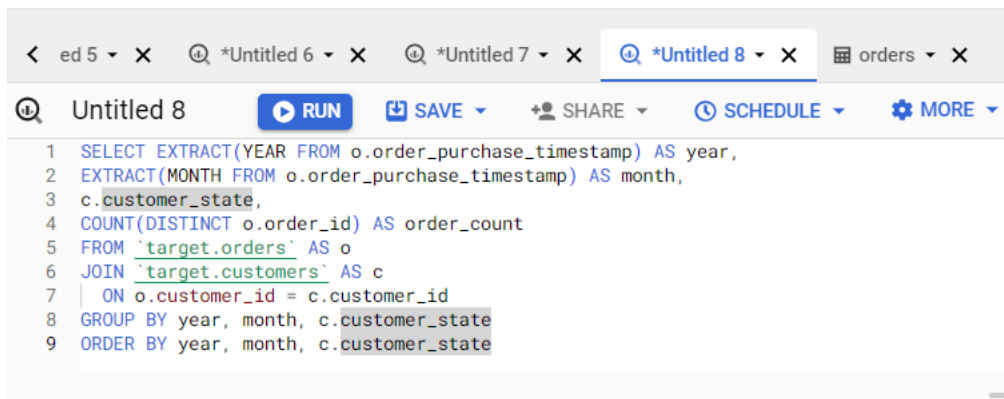
Q3. Evolution of E-commerce orders in the Brazil region:

Q3.1 Get the month on month no. of orders placed in each state.

Query :-

```
SELECT EXTRACT(YEAR FROM o.order_purchase_timestamp) AS year,
EXTRACT(MONTH FROM o.order_purchase_timestamp) AS month,
c.customer_state,
COUNT(DISTINCT o.order_id) AS order_count
FROM `target.orders` AS o
JOIN `target.customers` AS c
ON o.customer_id = c.customer_id
GROUP BY year, month, c.customer_state
ORDER BY year, month, c.customer_state
```

Screenshot:-



Query results

JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS	EXECUTION GRAPH
Row	year	month	customer_state	order_count	
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	
9	2016	10	GO	0	

PERSONAL HISTORY

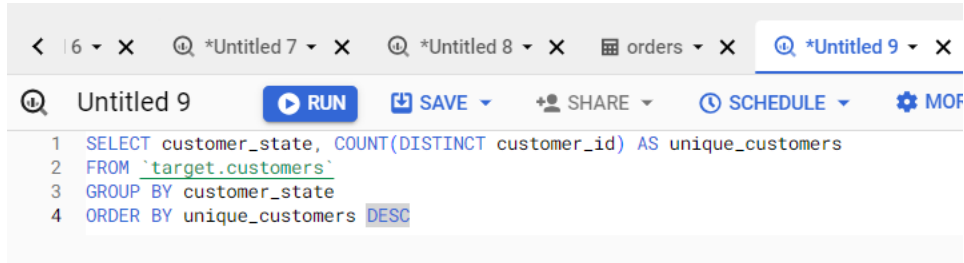
PROJECT HISTORY

Q3.2 How are the customers distributed across all the states?

Query:-

```
SELECT customer_state, COUNT(DISTINCT customer_id) AS unique_customers
FROM `target.customers`
GROUP BY customer_state
ORDER BY unique_customers DESC
```

Screenshot:-



Query results

JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS	EXECUTION GRA
Row	customer_state	unique_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			
12	CE	1336			

Analysis:-

1. Top three state in terms of customers SP, RJ and MG
2. Bottom three state in terms of customers are RR, AP and AC

Q4. Impact on Economy: Analyze the money movement by e-commerce by looking at order prices, freight and others.

Q4.1 Get the % increase in the cost of orders from year 2017 to 2018 (include months between Jan to Aug only).

Query :-

```
WITH orders_2017 AS (
SELECT
    o.order_id,
    o.order_purchase_timestamp,    p.payment_value
FROM `target.orders` o
JOIN `target.payments` p ON o.order_id = p.order_id
WHERE
EXTRACT(YEAR FROM o.order_purchase_timestamp) = 2017
AND EXTRACT(MONTH FROM o.order_purchase_timestamp) BETWEEN 1 AND 8),

orders_2018 AS (
SELECT
    o.order_id,
    o.order_purchase_timestamp,
    p.payment_value
FROM `target.orders` o
JOIN `target.payments` p ON o.order_id = p.order_id
WHERE
EXTRACT(YEAR FROM o.order_purchase_timestamp) = 2018
AND EXTRACT(MONTH FROM o.order_purchase_timestamp) BETWEEN 1 AND 8 )
SELECT
    (SUM(orders_2018.payment_value) - SUM(orders_2017.payment_value)) /
    SUM(orders_2017.payment_value) * 100 AS percentage_increase
FROM orders_2017
FULL OUTER JOIN
orders_2018 ON orders_2017.order_id = orders_2018.order_id;
```

screenshot

The screenshot displays a SQL query editor interface with a toolbar at the top containing icons for search, run, save, share, schedule, and more. The query is as follows:

```
15 SELECT
16     o.order_id,
17     o.order_purchase_timestamp,
18     p.payment_value
19 FROM
20     `target.orders` o
21 JOIN
22     `target.payments` p ON o.order_id = p.order_id
23 WHERE
24     EXTRACT(YEAR FROM o.order_purchase_timestamp) = 2018
25     AND EXTRACT(MONTH FROM o.order_purchase_timestamp) BETWEEN 1 AND 8
26 )
27 SELECT
28     (SUM(orders_2018.payment_value) - SUM(orders_2017.payment_value)) /
29     SUM(orders_2017.payment_value) * 100 AS percentage_increase
30 FROM
31     orders_2017
32 FULL OUTER JOIN
33     orders_2018 ON orders_2017.order_id = orders_2018.order_id;
```

Below the editor, the 'Query results' section is visible, featuring a 'SAVE RE' button. The results are presented in a table with the following structure:

JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS	EXECUTION GRAPH
Row	percentage_increase				
1	136.97687164666095				

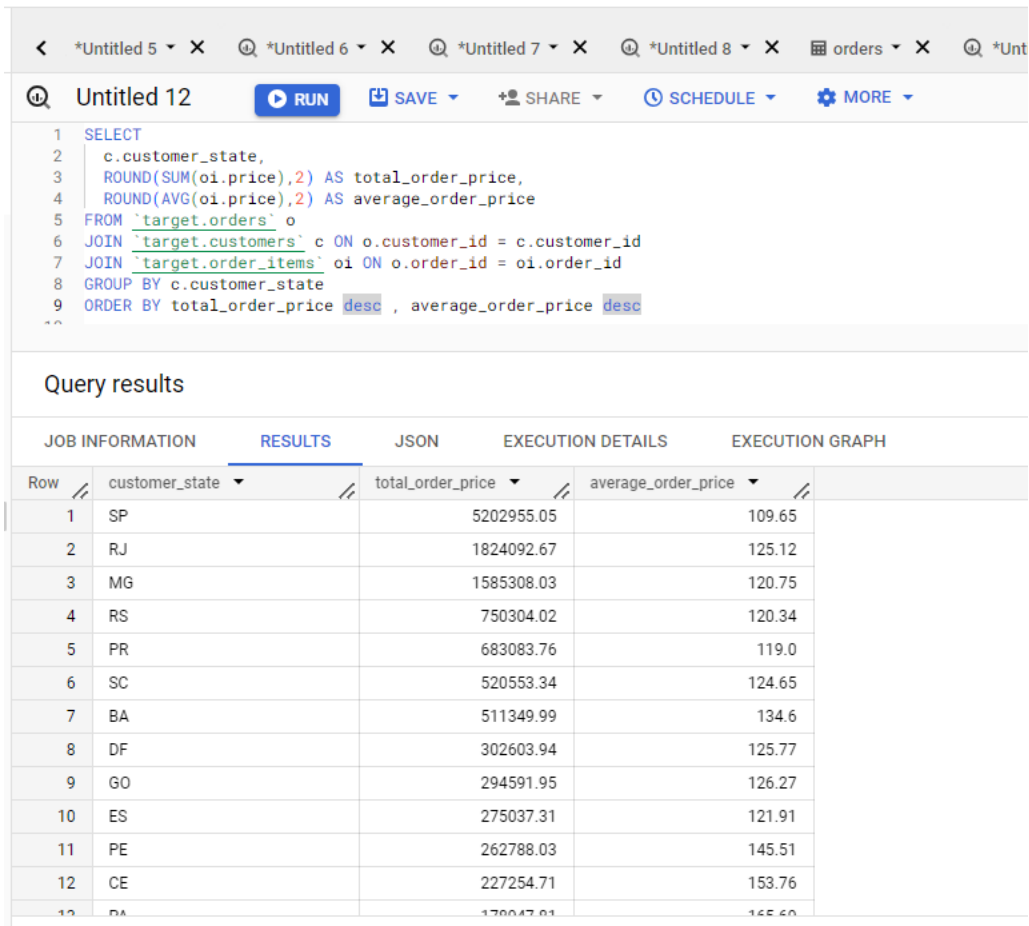
Q4.2. Calculate the Total & Average value of order price for each state.

Query:-

SELECT

```
c.customer_state,  
ROUND(SUM(oi.price),2) AS total_order_price,  
ROUND(AVG(oi.price),2) AS average_order_price  
FROM `target.orders` o  
JOIN `target.customers` c ON o.customer_id = c.customer_id  
JOIN `target.order_items` oi ON o.order_id = oi.order_id  
GROUP BY c.customer_state  
ORDER BY total_order_price desc , average_order_price desc
```

Screenshot:-



The screenshot shows a SQL query editor with a toolbar at the top containing buttons for RUN, SAVE, SHARE, SCHEDULE, and MORE. The query is displayed in the editor area. Below the query, the 'Query results' section is visible, showing a table with columns for customer_state, total_order_price, and average_order_price. The table contains 12 rows of data, sorted by total_order_price in descending order.

Row	customer_state	total_order_price	average_order_price
1	SP	5202955.05	109.65
2	RJ	1824092.67	125.12
3	MG	1585308.03	120.75
4	RS	750304.02	120.34
5	PR	683083.76	119.0
6	SC	520553.34	124.65
7	BA	511349.99	134.6
8	DF	302603.94	125.77
9	GO	294591.95	126.27
10	ES	275037.31	121.91
11	PE	262788.03	145.51
12	CE	227254.71	153.76

Analysis :-

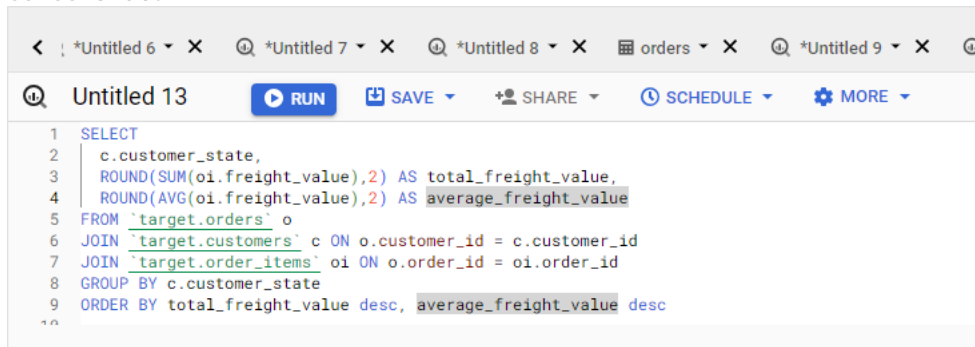
1. Top three state in terms of max total order price are SP, RJ, MG
2. Top three state in terms of avg order price are PB, AL, AC
3. Bottom three state in terms of total order price are RR, AP, AC
4. Bottom three state in terms of avg order price are SP, PR, RS

Q4.3 Calculate the Total & Average value of order freight for each state

Query :-

```
SELECT
    c.customer_state,
    ROUND(SUM(oi.freight_value),2) AS total_freight_value,
    ROUND(AVG(oi.freight_value),2) AS average_freight_value
FROM `target.orders` o
JOIN `target.customers` c ON o.customer_id = c.customer_id
JOIN `target.order_items` oi ON o.order_id = oi.order_id
GROUP BY c.customer_state
ORDER BY total_freight_value desc, average_freight_value desc
```

Screenshot :-



The screenshot shows a SQL query editor with a toolbar at the top containing icons for undo, redo, search, and other functions. The query is pasted into the editor and is ready to be executed. Below the editor, the 'Query results' section is visible, showing a table with the results of the query.

Row	customer_state	total_freight_value	average_freight_value
1	SP	718723.07	15.15
2	RJ	305589.31	20.96
3	MG	270853.46	20.63
4	RS	135522.74	21.74
5	PR	117851.68	20.53
6	BA	100156.68	26.36
7	SC	89660.26	21.47
8	PE	59449.66	32.92
9	GO	53114.98	22.77
10	DF	50625.5	21.04
11	ES	49764.6	22.06
12	CE	48351.59	32.71

Analysis :-

1. Top three state in terms of max total freight value are SP, RJ, MG
2. Top three state in terms of avg freight value are RR, PB, R0
3. Bottom three state in terms of total freight value are RR, AP, AC
4. Bottom three state in terms of avg freight value are SP, PR, MG

Q5. Analysis based on sales, freight and delivery time.

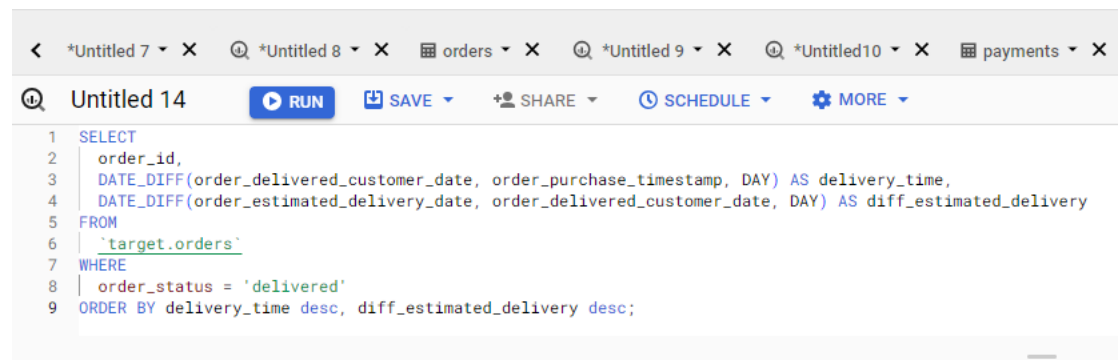
Q5.1 Find the no. of days taken to deliver each order from the order's purchase date as delivery time.

Also, calculate the difference (in days) between the estimated & actual delivery date of an order.

Query :-

```
SELECT order_id,
DATE_DIFF(order_delivered_customer_date, order_purchase_timestamp, DAY) AS
delivery_time,
DATE_DIFF(order_estimated_delivery_date, order_delivered_customer_date, DAY) AS
diff_estimated_delivery
FROM `target.orders`
WHERE order_status = 'delivered'
ORDER BY delivery_time desc, diff_estimated_delivery desc;
```

Screenshot :-



JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS	EXECUTION GRAPH
Row	order_id	delivery_time	diff_estimated_delivery		
1	ca07593549f1816d26a572e06...	209	-181		
2	1b3190b2dfa9d789e1f14c05b...	208	-188		
3	440d0d17af552815d15a9e41a...	195	-165		
4	2fb597c2f772eca01b1f5c561b...	194	-155		
5	0f4519c5f1c541ddec9f21b3bd...	194	-161		
6	285ab9426d6982034523a855f...	194	-166		
7	47b40429ed8cce3aee9199792...	191	-175		
8	2fe324feb907e3ea3f2aa9650...	189	-167		
9	2d7561026d542c8dbd8f0daea...	188	-159		
10	437222e3fd1b07396f1d9ba8c...	187	-144		
11	c27815f7e3dd0b926b5855262...	187	-162		
12	dfe5f68118c2576143240b8d7...	186	-153		

Analysis :-

1. Maximum delivery time = 209 days
2. Minimum delivery time = 0 days
3. Maximum diff estimated delivery = 146 days
4. Minimum diff estimated delivery = -188 days (delivered before time)

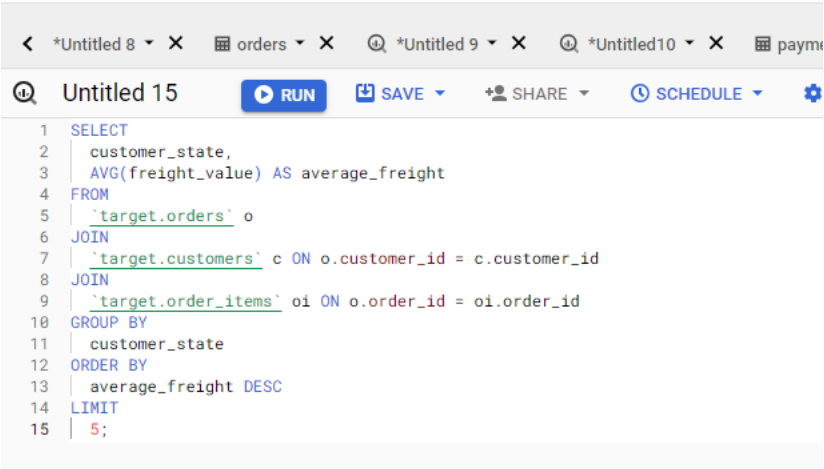
Q5.2 Find out the top 5 states with the highest & lowest average freight value.

A. Highest

Query

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

Screenshot



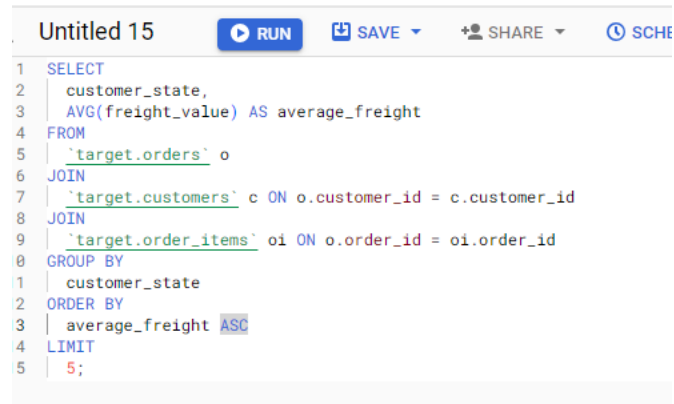
JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS	EXECUTION C
Row	customer_state	average_freight			
1	RR	42.984423076923093			
2	PB	42.723803986710941			
3	RO	41.069712230215842			
4	AC	40.073369565217405			
5	PI	39.147970479704767			

B. Lowest

Query :-

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

Screenshot:-



The screenshot shows a SQL query editor with a toolbar at the top containing buttons for 'RUN', 'SAVE', 'SHARE', and 'SCHE'. The query is as follows:

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

Query results

JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS
#	customer_state	average_freight		
1	SP	15.14727539041...		
2	PR	20.53165156794...		
3	MG	20.63016680630...		
4	RJ	20.96092393168...		
5	DF	21.04135494596...		

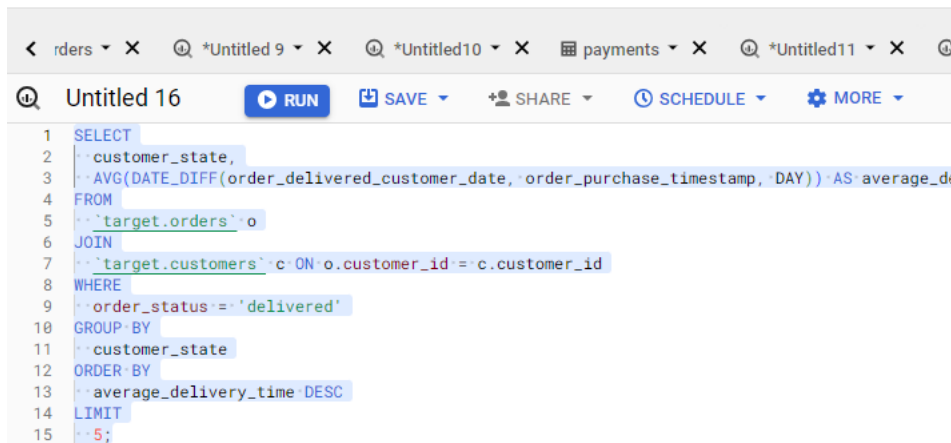
Q 5.3 Find out the top 5 states with the highest & lowest average delivery time.

A. highest

Query :-

```
SELECT
    customer_state,
    AVG(DATE_DIFF(order_delivered_customer_date, order_purchase_timestamp, DAY)) AS
average_delivery_time
FROM
    `target.orders` o
JOIN
    `target.customers` c ON o.customer_id = c.customer_id
WHERE
    order_status = 'delivered'
GROUP BY
    customer_state
ORDER BY
    average_delivery_time DESC
LIMIT
    5;
```

Screenshot:-



```
1 SELECT
2     customer_state,
3     AVG(DATE_DIFF(order_delivered_customer_date, order_purchase_timestamp, DAY)) AS average_de
4 FROM
5     `target.orders` o
6 JOIN
7     `target.customers` c ON o.customer_id = c.customer_id
8 WHERE
9     order_status = 'delivered'
10 GROUP BY
11     customer_state
12 ORDER BY
13     average_delivery_time DESC
14 LIMIT
15     5;
```

Query results

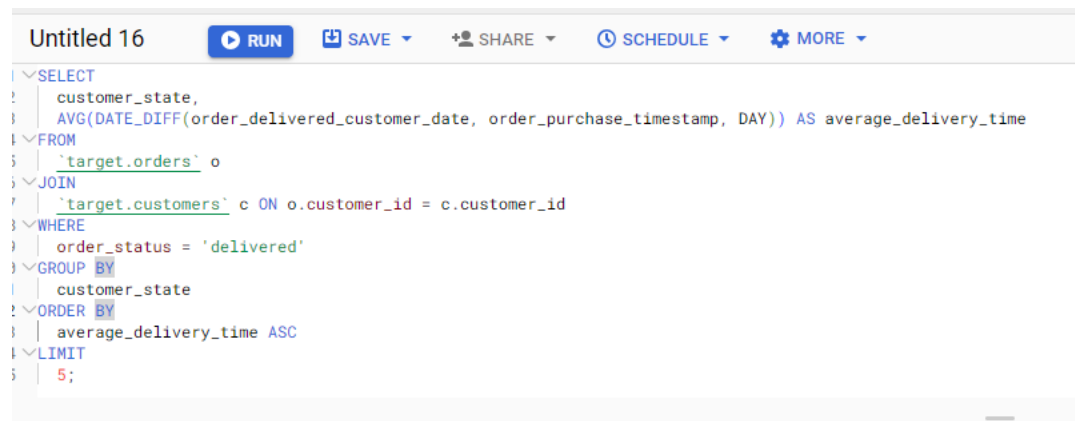
JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS	EXECUTION GRAPH
Row	customer_state	average_delivery_time			
1	RR	28.97560975609...			
2	AP	26.73134328358...			
3	AM	25.98620689655...			
4	AL	24.04030226700...			
5	PA	23.31606765327...			

B. Lowest

Query :-

```
SELECT
    customer_state,
    AVG(DATE_DIFF(order_delivered_customer_date, order_purchase_timestamp, DAY)) AS
average_delivery_time
FROM
    `target.orders` o
JOIN
    `target.customers` c ON o.customer_id = c.customer_id
WHERE
    order_status = 'delivered'
GROUP BY
    customer_state
ORDER BY
    average_delivery_time ASC
LIMIT
    5;
```

Screenshot:-



Query results

DB INFORMATION	RESULTS	JSON	EXECUTION DETAILS	EXECUTION GRAPH
	customer_state	average_delivery_time		
1	SP	8.298093544722...		
2	PR	11.52671135486...		
3	MG	11.54218777523...		
4	DF	12.50913461538...		
5	SC	14.47518330513...		

Q 5.4. Find out the top 5 states where the order delivery is really fast as compared to the estimated date of delivery.

Query :-

```
SELECT
    customer_state,
    AVG(DATE_DIFF(order_estimated_delivery_date, order_delivered_customer_date, DAY)) AS diff_average_delivery
FROM
    `target.orders` o
JOIN
    `target.customers` c ON o.customer_id = c.customer_id
WHERE
    order_status = 'delivered'
GROUP BY
    customer_state
HAVING
    diff_average_delivery > 0
ORDER BY
    diff_average_delivery ASC
LIMIT
    5;
```

Screenshot:-

```
SELECT
    customer_state,
    AVG(DATE_DIFF(order_estimated_delivery_date, order_delivered_customer_date, DAY)) AS diff_average_delivery
FROM
    `target.orders` o
JOIN
    `target.customers` c ON o.customer_id = c.customer_id
WHERE
    order_status = 'delivered'
GROUP BY
    customer_state
HAVING
    diff_average_delivery > 0
ORDER BY
    diff_average_delivery ASC
LIMIT
    5;
```

Query results

3 INFORMATION			RESULTS	JSON	EXECUTION DETAILS	EXECUTION GRAPH
			customer_state	diff_average_delivery		
1	AL			7.9471032745592		
2	MA			8.768479776847...		
3	SE			9.173134328358...		
4	ES			9.618546365914...		
5	BA			9.934889434889...		

Q6. Analysis based on the payments:

Q6.1 Find the month on month no. of orders placed using different payment types.

Query:-

```
SELECT
    EXTRACT(MONTH FROM o.order_purchase_timestamp) AS month,
    p.payment_type,
    COUNT(DISTINCT o.order_id) AS num_orders
FROM
    `target.orders` o
JOIN
    `target.payments` p ON o.order_id = p.order_id
GROUP BY
    month,
    p.payment_type
ORDER BY
    month;
```

Screenshot:-



The screenshot shows a SQL query editor interface. At the top, there's a toolbar with buttons for 'RUN', 'SAVE', 'SHARE', 'SCHEDULE', and 'MORE'. Below the toolbar, the SQL query is displayed in a monospace font. The query is a SELECT statement that extracts the month from the order purchase timestamp, joins the orders table with the payments table, and counts the number of distinct orders for each month and payment type. The results are ordered by month.

Query results

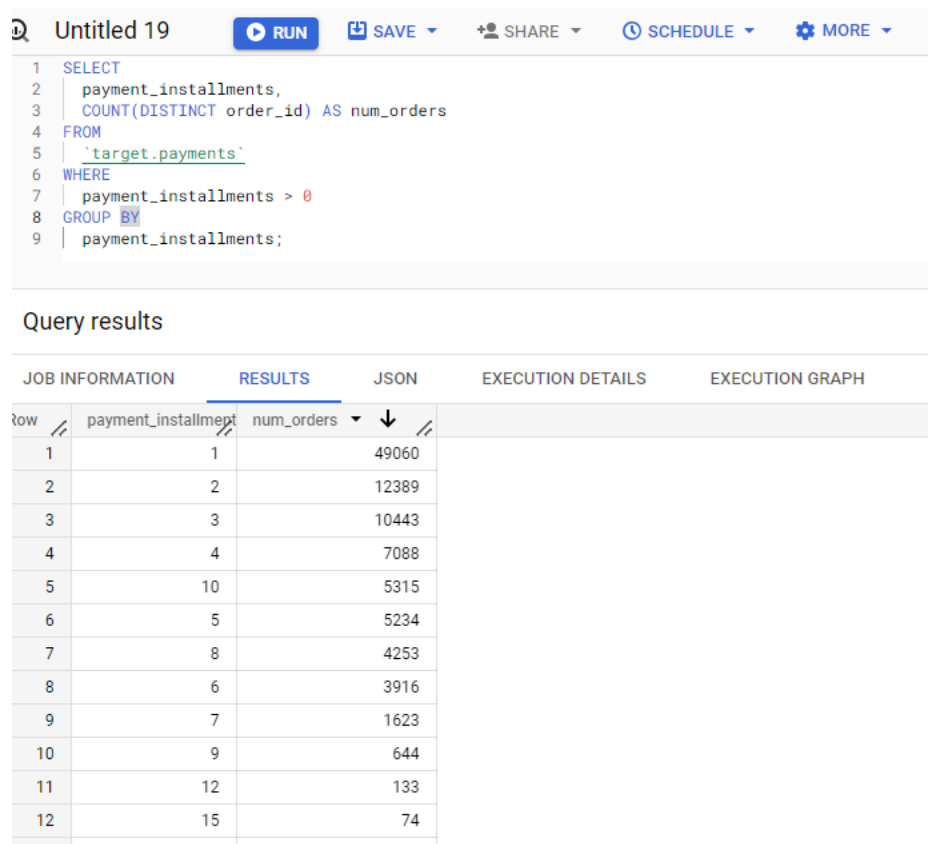
DB INFORMATION	RESULTS	JSON	EXECUTION DETAILS	EXECUTION GRAPH
month	payment_type	num_orders		
1	credit_card	6093		
2	UPI	1715		
3	voucher	337		
4	debit_card	118		
5	UPI	1723		
6	credit_card	6582		
7	voucher	288		
8	debit_card	82		
9	credit_card	7682		
10	UPI	1942		

Q6.2 Find the no. of orders placed on the basis of the payment installments that have been paid.

Query:-

```
SELECT
    payment_installments,
    COUNT(DISTINCT order_id) AS num_orders
FROM
    `target.payments`
WHERE
    payment_installments > 0
GROUP BY
    payment_installments;
```

Screenshot :-



The screenshot displays a SQL query editor interface with a toolbar at the top containing buttons for RUN, SAVE, SHARE, SCHEDULE, and MORE. The query is as follows:

```
1 SELECT
2     payment_installments,
3     COUNT(DISTINCT order_id) AS num_orders
4 FROM
5     `target.payments`
6 WHERE
7     payment_installments > 0
8 GROUP BY
9     payment_installments;
```

Below the editor, the 'Query results' section is visible, showing a table with the following data:

JOB INFORMATION	RESULTS	JSON	EXECUTION DETAILS	EXECUTION GRAPH
row	payment_installment	num_orders		
1	1	49060		
2	2	12389		
3	3	10443		
4	4	7088		
5	10	5315		
6	5	5234		
7	8	4253		
8	6	3916		
9	7	1623		
10	9	644		
11	12	133		
12	15	74		

Analysis :-

1. Maximum order places with single instalment i.e., full payment
2. Minimum orders are placed with 22 and 23 instalments (single order each)