

# Business Case: Target SQL PAGE DISTRIBUTION

## 1) Solutions and solution approach 2) Actionable Insights & Recommendations

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

Code :

```
SELECT
data_type
FROM usecase_business.INFORMATION_SCHEMA.COLUMNS
WHERE table_name= 'customers_usecase'
```

Screenshot :

The screenshot shows a data analysis interface with the following details:

- Explorer View:** Shows workspace resources, including a project named "dsmi-beginner-23-shirshendu" and a saved query named "Untitled 10".
- Query Editor:** A tab titled "Untitled 10" contains the following SQL code:

```
1 SELECT
2 data_type
3 FROM usecase_business.INFORMATION_SCHEMA.COLUMNS
4 WHERE table_name= 'customers_usecase'
5
6
7
```
- Results View:** The "RESULTS" tab is selected, displaying the query results in a table format. The table has one column, "data\_type", with five rows:

Row	data_type
1	STRING
2	STRING
3	INT64
4	STRING
5	STRING
- Toolbar:** Includes buttons for RUN, SAVE, SHARE, SCHEDULE, and MORE.
- Bottom Navigation:** Shows PERSONAL HISTORY and PROJECT HISTORY tabs, along with a system tray with various icons.

**Explanation:** From this INFORMATION\_SCHEMA.COLUMNS query we can identify the data type of customer table for all the columns.

**Insights: datatype of each column in customer table .can help us to proceed further dig down the analysis**

### 1.2: Get the time range between which the orders were placed.

Answer-

Code - USING BIG – Query

```
SELECT
MIN(order_purchase_timestamp) AS in_time,
MAX(order_purchase_timestamp) AS end_time
FROM `usecase_business.orders_usecase`
```

Screenshot:

The screenshot shows a BigQuery interface with the following details:

- Explorer:** On the left, there's a sidebar for "Viewing workspace resources" and a list of "Saved queries (21)". One query named "2023-05-10 00:17:58" is expanded, showing its code.
- Untitled 16:** The main query editor window titled "Untitled 16". It contains the following SQL code:

```
1 SELECT
2 MIN(order_purchase_timestamp) AS in_time,
3 MAX(order_purchase_timestamp) AS end_time
4
5 FROM `usecase_business.orders_usecase`
```
- Query results:** Below the code, the results table is displayed with two columns: "in\_time" and "end\_time".

Row	in_time	end_time
1	2016-09-04 21:15:19 UTC	2018-10-17 17:30:18 UTC
- Toolbar:** At the bottom, there are various icons for file operations, sharing, scheduling, and more.

**Explanation:** We are just using MIN and MAX function to retrieve the time range between which the orders were placed

**Insight - Output is providing us an insight about period at which orders were placed by users .**

### 1.3 : Count the number of Cities and States in our dataset

#### Code - USING BIG – Query

```
select
COUNT (DISTINCT geolocation_city) AS Number_of_cities,
COUNT (DISTINCT geolocation_state) AS Number_of_state
from `usecase_business.geolocation_usecase`
```

#### Screenshot:

The screenshot shows a data exploration interface with the following details:

- Explorer:** On the left, there's a sidebar with a search bar and a list of saved queries, including "Project queries" and "usecase\_business".
- Untitled:** The main workspace is titled "Untitled" and contains the SQL query provided in the code block.
- Query results:** Below the query editor, a table displays the results of the executed query. The table has two columns: "Number\_of\_cities" and "Number\_of\_state". The data row shows values 8011 and 27 respectively.

**Explanation :** Using count function from the geolocation table we can fetch out the details of number of Cities and States in our dataset

**Insights :** Pulling out a total number would be great for in depth exploration . if we Look at the total number of state and city that would be great to go further exploration and it will be easy to analyse data further

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

### Code:

```
WITH CTE1 AS
(SELECT
EXTRACT(YEAR FROM order_purchase_timestamp) AS year_part,
COUNT(order_id) AS trend
FROM `usecase_business.orders_usecase`
GROUP BY
EXTRACT(YEAR FROM order_purchase_timestamp))

SELECT
year_part,
trend
FROM CTE1
ORDER BY year_part, trend;
```

### Screenshot:

The screenshot shows the BigQuery web interface. On the left, the 'Explorer' sidebar lists various datasets and tables, including 'usecase\_business' and its sub-tables like 'customers\_usecase', 'geolocation\_usecase', etc. In the center, a query editor window titled 'Target Business Use-case (B...et)' is open. The code entered is:

```
27 GROUP BY c.customer_city, c.customer_state
28
29 -----
30 WITH CTE1 AS
31 (SELECT
32 EXTRACT(YEAR FROM order_purchase_timestamp) AS year_part,
33 COUNT(customer_id) AS trend
34 FROM `usecase_business.orders_usecase`
35 GROUP BY
36 EXTRACT(YEAR FROM order_purchase_timestamp))
37
38 SELECT
39 year_part,
40 trend
41 FROM CTE1
42 ORDER BY year_part, trend;
43
```

Below the code, the 'RESULTS' tab is selected in the 'Query results' section, showing the following data:

Row	year_part	trend
1	2016	329
2	2017	45101
3	2018	54011

**Explanation :** Using Extract year function we are pulling out the years and then grouping them to get individual count of customer As trend who were in orders table .

If the count of customer As trend is greater than the previous year data we can clearly identify the trend

Using Common table expression CTE to get a better visibility on the code

**Insights:** Analysing the trend in the number of orders placed over the years. Look for consistent growth or decline in order volume. Identify any significant spikes or drops in order count As we can see that there is a clear trend of increasing order every year

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

Code :

```
WITH CTE1 AS
(SELECT
EXTRACT(YEAR FROM order_purchase_timestamp) AS year_part,
EXTRACT(MONTH FROM order_purchase_timestamp) AS month_part,
COUNT(customer_id) AS total_customers
FROM `usecase_business.orders_usecase`
GROUP BY
EXTRACT(YEAR FROM order_purchase_timestamp),
EXTRACT(MONTH FROM order_purchase_timestamp))

SELECT
year_part,
month_part,
total_customers
FROM CTE1
ORDER BY year_part, month_part, total_customers
```

Screenshot :

[Next page](#)

The screenshot shows the Google BigQuery web interface. On the left is the 'Explorer' sidebar with a search bar and a list of saved queries, including 'Target Business Use-case...'. The main area displays a query titled 'Target Business Use-case (B...et)' with the following SQL code:

```

30 -----
31 WITH CTE1 AS
32 (SELECT
33 EXTRACT(YEAR FROM order_purchase_timestamp) AS year_part,
34 EXTRACT(MONTH FROM order_purchase_timestamp) AS month_part,

```

The 'RESULTS' tab is selected, showing a table with three columns: 'year\_part', 'month\_part', and 'total\_customers'. The data is as follows:

Row	year_part	month_part	total_customers
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
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	6039

**Explanation :** Using Extract year and month function we are pulling out the years and month then grouping them to get individual count of customer As total customer per month data who were in orders table .

If the count of customer As total customers is greater than the previous month data we can clearly identify the z seasonality in terms of the no. of orders being placed.

Using Common table expression CTE to get a better visibility on the code

**Insights & recommendation :** Seasonal customer trends is basically Analyse the total number of customers for each year and month to identify seasonal patterns. Look for months or periods with higher customer counts and those with lower counts. This information can help you understand customer behaviour and plan marketing campaigns or promotions accordingly.

**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

## Code :

```
WITH CTE1 AS (
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(DISTINCT customer_id) AS customer_count
FROM `usecase_business.orders_usecase`
GROUP BY time_of_day
)

SELECT
time_of_day,
customer_count
```

## Screenshot :

### Next page

The screenshot shows the BigQuery web interface. On the left, there's an 'Explorer' sidebar with a search bar and a list of saved queries, including one named '2023-05-10 00:17:58'. The main area has tabs for 'RUN', 'SAVE', 'SHARE', 'SCHEDULE', and 'MORE'. A message at the top right says 'This query will process 3.98 MB when run.' Below the tabs, the query code is displayed:

```
1 WITH CTE1 AS (
2 SELECT
3 (CASE WHEN EXTRACT(HOUR FROM order_purchase_timestamp) BETWEEN 0 AND 6 THEN 'Dawn'
4 WHEN EXTRACT(HOUR FROM order_purchase_timestamp) BETWEEN 7 AND 12 THEN 'Morning'
5 WHEN EXTRACT(HOUR FROM order_purchase_timestamp) BETWEEN 13 AND 18 THEN 'Afternoon'
6 WHEN EXTRACT(HOUR FROM order_purchase_timestamp) BETWEEN 19 AND 23 THEN 'Night'
7 END) AS time_of_day,
8 COUNT(DISTINCT customer_id) AS customer_count
9 FROM `usecase_business.orders_usecase`
10 GROUP BY time_of_day
11 )
12
13 SELECT
14   time_of_day,
15   customer_count
```

Below the code, the 'Query results' section shows a table with four rows:

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

**Explanation : Using Extract function we are pulling out the Hour details to get extract time if it is morning afternoon or night etc .**

**Then putting this on case statement and ending the case AS time \_of the day**

**Now we are counting distinct customer ID from the orders table who placed the order**

**Then grouping the time of the day part them to get individual data where customers placed order**

**And finally using Order by in Descending order to get during what time of the day, do the Brazilian customers mostly place their orders?**

**Insights & recommendation :** analysing the customer count based on the time of day can reveal patterns in order placement behaviour. By categorizing orders into different time slots (Dawn, Morning, Afternoon, Night), you can identify when customers are most active in making purchases. For example in our case the customers mostly active at the Afternoon time and less active would be Dawn time of the day

**Q-3- 1 Get the month on month no. of orders placed in each state.**

**Code :**

```
SELECT
customer_state,
year,
month,
number_of_orders
FROM (SELECT
c.customer_state,
EXTRACT(YEAR FROM order_purchase_timestamp) AS year,
EXTRACT(MONTH FROM order_purchase_timestamp) AS month,
COUNT(o.customer_id) AS number_of_orders
FROM `usecase_business.customers_usecase` AS c
LEFT JOIN `usecase_business.orders_usecase` AS o
ON c.customer_id = o.customer_id
GROUP BY
c.customer_state,
EXTRACT(YEAR FROM order_purchase_timestamp),
EXTRACT(MONTH FROM order_purchase_timestamp))
ORDER BY
customer_state,
year,
month,
number_of_orders
```

**Screenshot :**

The screenshot shows the BigQuery interface with a query results table. The table has the following structure:

Row	customer_state	year	month	number_of_orders
1	AC	2017	1	2
2	AC	2017	2	3
3	AC	2017	3	2
4	AC	2017	4	5
5	AC	2017	5	8
6	AC	2017	6	4
7	AC	2017	7	5
8	AC	2017	8	4
9	AC	2017	9	5
10	AC	2017	10	6
11	AC	2017	11	5
12	AC	2017	12	5
13	AC	2018	1	6
14	AC	2018	2	3
15	AC	2018	3	2
16	AC	2018	4	4
17	AC	2018	5	2
18	AC	2018	6	3
19	AC	2018	7	4
20	AC	2018	8	3
21	AI	2018	10	2

## Explanation :

We are extracting month on subsequently year wise data for each customer state

In order to do that first we are joining 2 tables :

- 1) Customers and Orders
- 2) From customer table we can get the state
- 3) From orders table we can get the customers who ordered
- 4) Now counting the customers who was there in the order table
- 5) GROUPING the data for each state and month
- 6) Using subquery only pulling out the relevant details
- 7) We are getting each year each state individual month by month orders data

**Insights & recommendation :** By analysing the number of orders placed month-on-month for each state, you can identify the order trends and patterns specific to each region. This information can help you understand the demand fluctuations in different states and make data-driven decisions accordingly.

Example : if we look at this data clearly we can see how each year corresponded to month by month data is signifying if there is any growth for a particular state or not

## Q3-2 How are the customers distributed across all the states?

### Code :

```

SELECT
customer_state,
number_of_customers
FROM (SELECT
c.customer_state,
COUNT(o.customer_id) AS number_of_customers
FROM `usecase_business.customers_usecase` AS c
LEFT JOIN `usecase_business.orders_usecase` AS o
ON c.customer_id = o.customer_id
GROUP BY
c.customer_state )
ORDER BY
customer_state,
number_of_customers

```

### Screenshot :

(Next Page )

The screenshot shows the BigQuery web interface. The top navigation bar includes 'Sandbox' (with a note to upgrade), 'Untitled 2' (the current query tab), and other tabs like 'test' and 'newtest cass'. Below the navigation is a search bar and a sidebar for workspace resources. The main area displays the query code and its results.

**Query results:**

Row	customer_state	number_of_customers
1	AC	81
2	AL	413
3	AM	148
4	AP	68
5	BA	3380
6	CE	1336
7	DF	2140
8	ES	2033
9	GO	2020
10	MA	747
11	MG	11635
12	MS	715
13	MT	907
14	PA	975
15	PB	536
16	PE	1652

### Explanation. :

- 1) JOINING Customers and Orders using LEFT join
- 2) From customer table we can get the state
- 3) From orders table we can get the customers who ordered
- 4) Now counting the customers who was there in the order table
- 5) GROUPING the data for each state
- 6) Using subquery only pulling out the relevant details
- 7) We are getting each state customer count who has placed order

**Insights :** The number of customers in each state, you can identify states with higher customer concentrations. This information can help you allocate marketing resources and prioritize customer engagement activities in states where you have a larger customer base, and also it will help you grow your business

**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.

**Code :**

```
SELECT
ROUND((
    SUM(IF(EXTRACT(YEAR FROM o.order_purchase_timestamp) = 2018, p.payment_value,
0)) -
    SUM(IF(EXTRACT(YEAR FROM o.order_purchase_timestamp) = 2017, p.payment_value,
0))
) / SUM(IF(EXTRACT(YEAR FROM o.order_purchase_timestamp) = 2017,
p.payment_value, 0)) * 100, 2) AS percentage_increase
FROM
`usecase_business.orders_usecase` AS o
LEFT JOIN `usecase_business.payments_usecase` AS 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
```

**Screenshot :**

The screenshot shows a data analysis interface with a sidebar titled 'Explorer' containing a tree view of workspace resources. The main area displays a query editor with the following code:

```

1 SELECT
2   ROUND(

```

Below the code, the 'RESULTS' tab is selected in the 'Query results' section, showing a single row of data:

Row	percentage_increase
1	136.98

### Explanation :

**Cost of orders:** The query utilizes the "payment value" column in the payments table to calculate the cost of orders.

**Time range:** The calculation is specific to the years 2017 and 2018, considering only the months between January and August. This allows for a focused analysis within a specific period.

**Percentage increase:** The query calculates the percentage increase in the cost of orders from 2017 to 2018. It compares the sum of payment values for 2018 orders with the sum of payment values for 2017 orders and calculates the percentage change.

**Rounded result:** The final percentage increase value is rounded to two decimal places for a more readable and concise presentation.

**Insights :** analyse the corresponding impact on profitability due to the increase in order costs. Evaluate whether the revenue generated from the higher payment values justifies the associated expenses As we can clearly see there is a percentage increase from our data

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

#### Code :

```

SELECT
c.customer_state,
ROUND(SUM(p.payment_value),2) AS total_order_price,
ROUND(AVG(p.payment_value),2) AS average_order_price
FROM `usecase_business.customers_usecase` AS c
LEFT JOIN `usecase_business.orders_usecase` AS o
ON c.customer_id = o.customer_id

```

```

LEFT JOIN `usecase_business.payments_usecase` AS p
ON o.order_id = p.order_id
GROUP BY c.customer_state

```

## Screenshot :

The screenshot shows a database query editor interface. On the left is an 'Explorer' sidebar with a tree view of workspace resources, including 'Book problem', 'DATE & TIME', 'Target Business Use-c...', 'Test case 3', 'Week over week analy...', 'facebook interview Qu...', 'facebook interview qu...', 'Impact on Economy: A...', 'Impact on Economy: A...', 'new test for use case', 'newtest casse', 'test', 'test\_q\_for\_business...', 'year', 'Farmars\_market', 'usecase\_business', and various sub-tables like 'customers\_usecase', 'geolocation\_use...', 'order\_items', 'orders-reviews\_u...', 'orders\_usecase', 'payments\_usecase', 'products\_usecase', and 'sellers'. The main area contains a query editor with the following SQL code:

```

30
31   SELECT
32     c.customer_state,
33     ROUND(SUM(p.payment_value),2) AS total_order_price,
34     ROUND(AVG(p.payment_value),2) AS average_order_price
35   FROM `usecase_business.customers_usecase` AS c
36   LEFT JOIN `usecase_business.orders_usecase` AS o
37   ON c.customer_id = o.customer_id
38   LEFT JOIN `usecase_business.payments_usecase` AS p
39   ON o.order_id = p.order_id
40   GROUP BY c.customer_state

```

Below the code is a 'Query results' table with the following data:

customer_state	total_order_price	average_order_price
RN	102718.13	196.78
CE	279464.03	199.9
RS	890098.54	157.18
SC	623086.43	165.98
SP	5998226.96	137.5
MG	1872257.26	154.71
BA	616645.82	170.82
RJ	2144379.69	158.53
GO	350092.31	165.76
MA	152523.02	198.86
PE	324850.44	187.99
PB	141545.72	248.33
ES	325967.55	154.71
PR	811156.38	154.15
RO	60866.2	233.2

## Explanation. :

- 1) **customers\_usecase, orders\_usecase, and payments\_usecase tables to retrieve the necessary information. It calculates the total order price by summing up the payment values**
- 2) **rounds the result to two decimal places using the ROUND function for our better visibility**
- 3) **In the same way it is calculates the average order price by averaging the payment values and also rounds the result to two decimal places.**
- 4) **The GROUP BY clause groups the results by the customer state**

## Insights :

This query provides insights into the total and average order prices for each state. Analyse states with low total order prices we can Identify states where the total order price is relatively lower than others. Investigate the reasons behind this lower average and then accordingly put some marketing campaign to increase the sal

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

### Code

```

SELECT
c.customer_state,
ROUND(SUM(ot.freight_value), 2) AS total_freight_value,
ROUND(AVG(ot.freight_value), 2) AS Avg_freight_value
FROM `usecase_business.customers_usecase` AS c
LEFT JOIN `usecase_business.orders_usecase` AS o
ON c.customer_id = o.customer_id
LEFT JOIN `usecase_business.order_items` AS ot
ON o.order_id = ot.order_id
GROUP BY c.customer_state;

```

## Screenshot :

The screenshot shows a data analysis interface with the following details:

- Explorer:** On the left, showing a tree view of workspace resources under "usecase\_business".
- Query Editor:** A tab labeled "Untitled 5" containing the SQL query provided above.
- Results View:** A table titled "Query results" showing the output of the query. The table has columns: Row, customer\_state, total\_freight\_value, and Avg\_freight\_value.

Row	customer_state	total_freight_value	Avg_freight_value
1	RN	18860.1	35.65
2	CE	48351.59	32.71
3	RS	135522.74	21.74
4	SC	89660.26	21.47
5	SP	718723.07	15.15
6	MG	270853.46	20.63
7	BA	100156.68	26.36
8	RJ	305589.31	20.96
9	GO	53114.98	22.77
10	MA	31523.77	38.26

## Explanation :

- 1) JOINING customers\_usecase table with the orders\_usecase table on the customer\_id column and then joins the resulting table with the order\_items
- 2) The SUM function is used to calculate the total freight value, and the AVG function is used to calculate the average freight value.
- 3) Then results are rounded to two decimal places
- 4) The GROUP BY clause groups the results by customer\_state.

**Insights :** Identify states where the total freight value is relatively lower than others accordingly we can start identifying what needs to be our next step to increase that amount

**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.**

**Do this in a single query.**

**Code :**

```
select
order_id,
timestamp_diff(order_delivered_customer_date , order_purchase_timestamp, DAY) AS
time_to_deliver,
timestamp_diff(order_estimated_delivery_date ,order_delivered_customer_date, DAY) AS
diff_estimated_delivery
from `usecase_business.orders_usecase`
```

**Screenshot :**

The screenshot shows a data analysis interface with a sidebar containing a tree view of workspace resources. The main area displays a table titled 'Query results' with two tabs: 'RESULTS' (selected) and 'PREVIEW'. The table has columns: Row, order\_id, time\_to\_deliver, and diff\_estimated\_delivery. The data consists of 21 rows of order IDs and their respective delivery times and differences. The bottom of the screen shows a Mac OS X dock with various application icons.

Row	order_id	time_to_deliver	diff_estimated_delivery
1	1950d77789f6a877539f5379...	30	-12
2	2c45c33d2f9cb8fb8fb1c86cc28...	30	28
3	65df1e22edfaeb8cd042f6542...	35	16
4	635c894d68ab3c76e03dc54e...	30	1
5	3b97562c3aae8bedeb5c2e45...	32	0
6	68f4750f04c4c8b774570cde...	29	1
7	2769e9e344d3bf0229f83a161c...	43	-4
8	54e1a3c2b97fb0809da548a59...	40	-4
9	fdd4fa4105ee043f6d0139a5...	37	-1
10	302bb8109d097a9fc6e9cef5...	33	-5
11	66057637308e787052a32828...	38	-6
12	19135c945c554ebfd7576c73...	36	-2
13	4493e45e7ca1084efcd38dddb...	34	0
14	70c77e510f179d75a646141...	42	-11
15	d7918e406132d7c81f1b84527...	35	-3
16	43f606477ce6433e7d68dd86...	32	-7
17	37073d851c3f30eeb6598e5a...	31	-9
18	d0644d4d07d014984d257750...	29	0
19	61d430273ff1e887294ac5b5e...	30	0
20	d2fe9effd1714fcac7de990aef1...	30	-8
21	81279a15416799e6580d6f60f...	31	-12

**Explanation :**

- 1) The query retrieves the order\_id and calculates the time\_to\_deliver by subtracting the order\_purchase\_timestamp from the order\_delivered\_customer\_date and specifying the unit as DAY.
- 2) In the same way it calculates the diff\_estimated\_delivery by subtracting the order\_delivered\_customer\_date from the order\_estimated\_delivery\_date and specifying the unit as DAY
- 3) And we are getting our result

**Insights :** By analysing the delivery time, businesses can identify any delays in the delivery process. This can help optimize logistics and improve overall efficiency of the customer service process

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

**Code :**

```
WITH CTE1 AS

(SELECT
c.customer_state,
ROUND(AVG(ot.freight_value), 2) AS Avg_freight_value,
FROM`usecase_business.customers_usecase` AS c
LEFT JOIN `usecase_business.orders_usecase` AS o
ON c.customer_id =o.customer_id
LEFT JOIN`usecase_business.order_items` AS ot
ON o.order_id = ot.order_id
GROUP BY c.customer_state),CTE2 AS
(
SELECT
*,
DENSE_RANK() OVER (ORDER BY Avg_freight_value DESC) AS rank_highest,
DENSE_RANK() OVER (ORDER BY Avg_freight_value ASC) AS rank_lowest
FROM
CTE1)

SELECT
customer_state,
Avg_freight_value,
rank_highest AS Ranking,
'Highest' AS Rank_type
FROM
CTE2
WHERE
rank_highest <= 5

UNION ALL

SELECT
customer_state,
Avg_freight_value,
rank_lowest AS Ranking,
'Lowest' AS Rank_type
FROM
CTE2
```

```

WHERE
rank_lowest <= 5

ORDER BY Avg_freight_value DESC, ranking;

```

## Screenshot :

The screenshot shows a data analysis interface with a sidebar on the left containing various icons and a search bar. The main area has tabs for 'Untitled 2' through 'Untitled 8'. A query editor window titled 'Untitled 7' is open, displaying the following SQL code:

```

1 WITH CTE1 AS
2
3 (SELECT
4 c.customer_state,

```

Below the code, a 'Query results' table is shown with the following data:

Row	customer_state	Avg_freight_value	Ranking	Rank_type
1	RR	42.98	1	Highest
2	PB	42.72	2	Highest
3	RO	41.07	3	Highest
4	AC	40.07	4	Highest
5	PI	39.15	5	Highest
6	DF	21.04	5	Lowest
7	RJ	20.96	4	Lowest
8	MG	20.63	3	Lowest
9	PR	20.53	2	Lowest
10	SP	15.15	1	Lowest

## Explanation :

- 1) (CTE1 and CTE2) using the **WITH** clause. CTE1 calculates the average freight value for each customer state by joining the **customers\_usecase**, **orders\_usecase**, and **order\_items** tables. The result is grouped by the customer state, and the average freight value is rounded to 2 decimal places.
- 2) CTE2 builds on CTE1 and adds two additional columns **rank\_highest** and **rank\_lowest**. These columns are calculated using the **RANK()** function and assign a ranking to each state based on the average freight value. The highest average freight values receive lower ranks, while the lowest average freight values receive higher ranks.

**Insights :** Highest Average Freight Value: The top 5 states with the highest average freight  
 Lowest Average Freight Value: The top 5 states with the lowest average freight value are also presented,

where customers generally have lower shipping costs. These states may benefit . Through this information we can improve the channelisation of the transport and make better customer experience next time

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

**CODE:**

```
WITH CTE1 AS

(SELECT
c.customer_state,
ROUND(AVG(timestamp_diff(o.order_delivered_customer_date,
o.order_purchase_timestamp, DAY)),2) AS avg_time_to_deliver
FROM `usecase_business.orders_usecase` AS o
LEFT JOIN `usecase_business.customers_usecase` AS c
ON o.customer_id = c.customer_id
GROUP BY
c.customer_state), CTE2 AS

(
SELECT
*, 
DENSE_RANK() OVER ( ORDER BY avg_time_to_deliver DESC ) AS Highest_avarage,
DENSE_RANK() OVER ( ORDER BY avg_time_to_deliver ASC ) AS Lowest_avarage
FROM CTE1
)

SELECT
customer_state,
CTE2.avg_time_to_deliver,
Highest_avarage AS Highest_and_Lowest_avg,
'Highest' AS Rank_type
FROM CTE2
WHERE
Highest_avarage<= 5

UNION ALL

SELECT
customer_state,
CTE2.avg_time_to_deliver,
Lowest_avarage AS Highest_and_Lowest_avg,
'Lowest' AS Rank_type
FROM CTE2
WHERE
Lowest_avarage<= 5
ORDER BY avg_time_to_deliver DESC ,
Highest_and_Lowest_avg,
Rank_type
```

**Screenshot:**

The screenshot shows a data analysis workspace with the following details:

- Explorer:** On the left, there's a sidebar with various project and connection items.
- Query Editor:** The main area displays a query titled "top 5 states with the hig...". The query code is as follows:
 

```

1 WITH CTE1 AS
2   SELECT
3     c.customer_state,
4     ROUND(AVG(timestamp_diff(o.order_delivered_customer_date, o.order_purchase_timestamp, DAY)),2) AS avg_time_to_deliver
5   FROM `usecase_business.orders_usecase` AS o
6   LEFT JOIN `usecase_business.customers_usecase` AS c
7   ON o.customer_id = c.customer_id
8   GROUP BY customer_state
9   ORDER BY avg_time_to_deliver DESC
10 LIMIT 5
      
```
- Query Results:** Below the editor, a table shows the results of the query. The columns are:
 

customer_state	avg_time_to_deliver	Highest_and_Lowest	Rank_type
RR	28.98	1	Highest
AP	26.73	2	Highest
AM	25.99	3	Highest
AL	24.04	4	Highest
PA	23.32	5	Highest
SC	14.48	5	Lowest
DF	12.51	4	Lowest
MG	11.54	3	Lowest
PR	11.53	2	Lowest
SP	8.3	1	Lowest
- Toolbar:** At the bottom, there are tabs for PERSONAL HISTORY and PROJECT HISTORY, along with a REFRESH button and a set of system icons.

### Explanation :

- 1) **CTE1:** This CTE calculates the average delivery time in days for each state. It uses the orders\_usecase table to calculate the time difference between the order purchase date and the delivery date. The average delivery time is rounded to two decimal places.
- 2) **CTE2:** This CTE assigns rankings to the average delivery time values. It uses the DENSE\_RANK() function to assign ranks based on the average delivery time, both in ascending and descending order.

**Insights :** insights into the top 5 states with the highest and lowest average delivery times. Businesses can use this information to identify regions where delivery times are longer or shorter on average. This can help in optimizing logistics operation

**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.**

**CODE :**

```
WITH CTE1 AS (
  SELECT
    c.customer_state,
    o.order_estimated_delivery_date,
    o.order_delivered_customer_date,
    ROUND(AVG(TIMESTAMP_DIFF(o.order_delivered_customer_date,
      o.order_estimated_delivery_date, DAY)), 2) AS delivery_time_difference
  FROM `usecase_business.orders_usecase` AS o
  LEFT JOIN `usecase_business.customers_usecase` AS c ON o.customer_id =
    c.customer_id
  WHERE
    o.order_estimated_delivery_date IS NOT NULL
    AND o.order_delivered_customer_date IS NOT NULL
  GROUP BY
    c.customer_state,
    o.order_estimated_delivery_date,
    o.order_delivered_customer_date
  ),
  CTE2 AS (
    SELECT
      *,
      DENSE_RANK() OVER (ORDER BY delivery_time_difference ASC) AS raank
    FROM
      CTE1
  )
  SELECT
    customer_state,
    order_estimated_delivery_date,
    order_delivered_customer_date,
    delivery_time_difference,
    raank AS Top_5
  FROM
    CTE2
  WHERE
    raank <= 5
  ORDER BY
    delivery_time_difference;
```

**Screenshot :**

The screenshot shows a data analysis interface with a sidebar containing a tree view of workspace resources, including a folder named 'dsmt-beginner-23-shirshendu' which contains 'Saved queries (18)'. One query titled '5 states where the order delivery is exceptionally fast' is selected and displayed in the main editor area. The code for this query is:

```

1 WITH CTE1 AS (
2     SELECT
3         c.customer_state,
4         o.order_estimated_delivery_date,
5         o.order_delivered_customer_date,
6         ROUND(AVG(TIMESTAMP_DIFF(o.order_delivered_customer_date, o.order_estimated_delivery_date, DAY)), 2) AS delivery_time_difference
7     FROM `usecase_business.orders_usecase` AS o
8     LEFT JOIN `usecase_business.customers_usecase` AS c ON o.customer_id = c.customer_id
9     WHERE
10        o.order_estimated_delivery_date IS NOT NULL
11    AND o.order_delivered_customer_date IS NOT NULL
12    GROUP BY
13        c.customer_state.

```

The results table shows the top 5 states with their delivery time differences:

customer_state	order_estimated_delivery_date	order_delivered_customer_date	delivery_time_difference	Top_5
SP	2018-08-03 00:00:00 UTC	2018-03-09 23:36:47 UTC	-146.0	1
MA	2017-07-04 00:00:00 UTC	2017-02-14 14:27:45 UTC	-139.0	2
RS	2018-07-12 00:00:00 UTC	2018-02-27 16:35:43 UTC	-134.0	3
SP	2017-10-11 00:00:00 UTC	2017-06-09 13:35:54 UTC	-123.0	4
RJ	2018-01-30 00:00:00 UTC	2017-10-13 13:49:07 UTC	-108.0	5

## Explanation :

### Common Table Expressions (CTEs):

- 1) **CTE1:** This CTE calculates the delivery time difference in days between the estimated delivery date and the actual delivered date for each state. It uses the `orders_usecase` table to calculate the time difference and rounds it to two decimal places.
- 2) The CTE filters out records where either the estimated delivery date or the actual delivered date is null.
- 3) **CTE2:** This CTE assigns rankings based on the delivery time difference values. It uses the `DENSE_RANK()` function to assign ranks in ascending order.
- 4) **Note : Formula we have used :**

difference between the averages of actual & estimated delivery date to figure out how fast the delivery was for each state.

**Insight :** From this insights into the top 5 states where the order delivery is exceptionally fast compared to the estimated date of delivery. This information can help businesses identify regions where their logistics operations are performing efficiently and delivering orders ahead of schedule.

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

### CODE:

```
SELECT
EXTRACT(YEAR FROM o.order_purchase_timestamp) AS year,
EXTRACT(MONTH FROM o.order_purchase_timestamp) AS month,
p.payment_type,
COUNT(DISTINCT o.order_id) AS order_count
FROM `usecase_business.orders_usecase` AS o
LEFT JOIN `usecase_business.payments_usecase` AS p
ON o.order_id = p.order_id
GROUP BY
month,
payment_type,
year
ORDER BY
payment_type,
order_count,
year
```

### Screenshot :

The screenshot shows a database interface with the following details:

- Explorer:** Shows workspace resources, including a saved query named "Untitled 11".
- Query:** Untitled 11 (SELECT query)
- Results:** A table titled "Query results" showing the count of orders by month, payment type, and year.
- Data:** The table contains 14 rows of data:

Row	year	month	payment_type	order_count
1	2016	9	null	1
2	2016	10	UPI	63
3	2017	1	UPI	197
4	2017	2	UPI	398
5	2017	4	UPI	496
6	2017	3	UPI	590
7	2017	6	UPI	707
8	2017	5	UPI	772
9	2017	7	UPI	845
10	2017	9	UPI	903
11	2017	8	UPI	938
12	2017	10	UPI	993
13	2018	6	UPI	1100
14	2018	8	UPI	1139

**Insights :** The month-to-month variation in the number of orders placed using different payment types. By grouping the data based on payment types, you can identify trends and patterns and then understand the payment behaviour of the customer

**6.2 Find the no. of orders placed on the basis of the payment instalments that have been paid.**

**Code :**

```
SELECT
payment_installments,
COUNT(order_id) AS order_count
FROM
`usecase_business.payments_usecase`
GROUP BY
payment_installments
```

**Screenshot:**

The screenshot shows a data analysis interface with the following details:

- Explorer:** On the left, there's a tree view of workspace resources under "dsmi-beginner-23-shirshendu".
- Query Editor:** The main area displays a query named "Untitled 11" with the following code:

```
17
18
19
20 SELECT
21 payment_installments,
22 COUNT(order_id) AS order_count
23 FROM
24 `usecase_business.payments_usecase`
```
- Query Results:** Below the editor, the results are shown in a table with two columns: "payment\_installment" and "order\_count". The data is as follows:

payment_installment	order_count
1	0
2	1
3	2
4	3
5	4
6	5
7	6
8	7
9	8
10	9
11	10
12	11
13	12
14	13
15	14
16	15
17	16

**Explanation :**

**The main query selects the payment installments and counts the number of order IDs. It retrieves data from the payments\_usecase table.**

**The result is grouped by payment installments to calculate the count of order IDs for each installment count.**

**Insights :** The distribution of orders based on the number of payment instalments that have been paid. This information can be useful for understanding customer payment behaviour and preferences

