Business case -Target sql

All the query is written on Bigquery

Question 1.C

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

Answer - Query how we can get the Data Type of all columns in the “Customers” table is AS →

```
SELECT
    COLUMN_NAME,
    DATA_TYPE
FROM
    business_case.INFORMATION_SCHEMA.COLUMNS
WHERE
    TABLE_NAME = 'customers';
```

Output of the above query I got is →

JOB INFORMATION | RESULTS | CHART | PREVIEW | JSON

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

Job history

INSIGHTS →

1. **Column Information:**
    - The query retrieves the COLUMN_NAME and DATA_TYPE for each column in the customers table. This information is for understanding the structure of the table.
2. **Data Type Understanding:**
    - The DATA_TYPE field provides information about the type of data stored in each column (e.g., VARCHAR, INTEGER, DATE). Understanding data types is important for data manipulation and analysis.
3. **Schema and Table Context:**
    - The query is specific to the customers table within the business_case schema. This suggests a focus on exploring or documenting the structure of this particular table.

Recommendations →

1. **Data Formatting:**
    - Formatting the output to make it more readable, especially if this information is presented to end-users.

Question 1.B

Get the time range between which the orders were placed.

Answer → The Query to get the time range between which the order was placed can be retrieved from 'order' Table AS→

```
-- Q Get the time range between which the orders were placed

SELECT
MIN(order_purchase_timestamp) AS Start_Time,
MAX(order_purchase_timestamp) AS End_Time
FROM `business_case.orders`
```

Output of this query is As→

JOB INFORMATION | RESULTS | CHART | PREVIEW

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

Insights →

1. **Human-Readable Date Format:**
    - The query uses the DATE_FORMAT function to present the order_purchase_timestamp column in a more readable format. This can be useful when presenting the results to end-users or for reporting purposes.
2. **Earliest and Latest Timestamps:**
    - The MIN and MAX functions are applied to the order_purchase_timestamp column to find the earliest and latest timestamps, respectively. This provides insights into the temporal range of the orders.

Recommendations →

1. **Performance Optimization:**
    - If the order_purchase_timestamp column is indexed, consider avoiding the use of the DATE_FORMAT function directly in the MIN and MAX functions. Instead, apply the formatting during presentation to potentially leverage the index for faster performance.
2. **Time zone Consideration:**
    - Ensure that the time zone of the timestamps is well understood. If the timestamps are stored in different time zones or if there are users across various time zones, consider handling time zone conversions as needed.

Question 1.C

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

Answer – Query of how I counted cities and states of each customer who ordered during the given period is AS →

```sql
SELECT
  COUNT(DISTINCT c.customer_city) AS No_Of_City,
  COUNT(DISTINCT c.customer_state) AS No_Of_State
FROM
  `business_case.customers` AS c
JOIN
  `business_case.orders` AS o
ON
  c.customer_id = o.customer_id
WHERE
  c.customer_city IN (
    SELECT
      c.customer_city
    FROM
      `business_case.orders` o
    JOIN
      `business_case.customers` c
    ON
      c.customer_id = o.customer_id
    WHERE
      o.order_purchase_timestamp BETWEEN
        (SELECT MIN(order_purchase_timestamp) FROM `business_case.orders`) AND
        (SELECT MAX(order_purchase_timestamp) FROM `business_case.orders`)
  )
AND c.customer_state IN (
    SELECT
      c.customer_state
    FROM
      `business_case.orders` o
    JOIN
      `business_case.customers` c
    ON
      c.customer_id = o.customer_id
    WHERE
      o.order_purchase_timestamp BETWEEN
        (SELECT MIN(order_purchase_timestamp) FROM `business_case.orders`) AND
        (SELECT MAX(order_purchase_timestamp) FROM `business_case.orders`)
  );
```

Output Of the above Query is AS below →

JOB INFORMATION | RESULTS | CHART | PREVIEW

Row	No_Of_City	No_Of_State
1	4119	27

Insights are →

1. **Number of Unique Cities (No_Of_City):**
   - The **COUNT(DISTINCT c.customer_city)** expression calculates the number of unique cities among customers who have placed orders within the given time period.
   - This could provide insight into the geographical distribution of customers during that time period.

2. **Filtering Customers based on Order Timestamp:**
    - The WHERE clause filters customers based on their city, considering only those whose city is in the list of cities where orders were placed within the given time period.
3. **Usage of Joins:**
    - The code uses INNER JOIN to link the **business_case.customers** table with the **business_case.orders** table based on the **customer_id** field.
    - This implies that only customers who have placed orders are considered in the counts.

Recommendations can be like →

1. **Geographical Targeting:**
    - Use the information on unique cities (**No_Of_City**) and states (**No_Of_State**) to identify regions with high customer engagement.
    - Tailor marketing strategies or promotions to target specific cities or states with a concentration of customers.
2. **Customer Segmentation:**
    - Analyse the distribution of customers across cities and states to identify patterns or clusters.
    - Segment customers based on location to create targeted marketing campaigns or personalized communication for different regions.
3. **Regional Preferences:**
    - Investigate product or service preferences based on the regions with higher customer concentration.
4. **Time-Based Insights:**
    - Identify peak periods or seasonality in customer activity and adjust inventory, staffing, or marketing efforts accordingly.
5. **Customer Engagement Strategies:**
    - Consider localized events, promotions, or partnerships to enhance customer engagement in specific regions.

Question 2

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

Answer of the above question is AS →

The trend can be seen if the number of orders in each year has increased or not,

For that the query is As below :

```sql
SELECT
  EXTRACT(YEAR FROM order_purchase_timestamp) AS Order_Year,
  COUNT(*) AS Total_order

FROM `business_case.orders`
  GROUP BY Order_year
ORDER BY Order_year
```

The output of the above query is as below:

Row	Order_Year	Total_order
1	2016	329
2	2017	45101
3	2018	54011

Query aims to extract 'Year' from the order_purchase_timestamp column, and calculate the total COUNT(*) calculates the total number of order was placed in a year.

Insights :

1. **Yearly Order Details:**
   - The query provides a breakdown of the total number of orders for each year, allowing for a yearly analysis of order
2. **Counting Orders:**
   - The COUNT(*) function is utilized to count the number of orders for each extracted year
3. **Grouping by Year:**
   - The (GROUP BY) clause organizes the results by the extracted year, so that the Total_order is aggregated by the year separately.
4. **Ordering the output table:**
   - **The** (ORDER BY Order_year) clause sorts out then table in default order that is in ascending order of order_year.
   - By ascending the output data it is visually easy to compare the increase in orders over the year.

Recommendations :

1. **Check for unusual spikes:**
   - Identify any unusual spikes or drops in the order counts. These might indicate anomalies or errors in the data that require investigation.
2. **Good Planning:**
   - In the year 2016 we had 326 orders but when we compare the number of orders in the Year 2017 we have 45101 this could mean there was nice plan, product review was good So which helped in growing business.

Question 2

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

Answer → Yes we can see the number of orders being placed in month, for that the query is as below

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

SELECT
  FORMAT_DATETIME("%B", DATETIME(order_purchase_timestamp)) as month,
  EXTRACT(YEAR FROM order_purchase_timestamp) AS Year,
  COUNT(Order_id) AS Total_orders

FROM `business_case.orders`
GROUP BY month,Year
ORDER BY Year,month ASC
```

The output I got is as →

JOB INFORMATION | RESULTS | CHART PREVIEW | JSON

Row	month	Year	Total_orders
1	December	2016	1
2	October	2016	324
3	September	2016	4
4	April	2017	2404
5	August	2017	4331
6	December	2017	5673
7	February	2017	1780
8	January	2017	800
9	July	2017	4026
10	June	2017	3245
11	March	2017	2682
12	May	2017	3700
13	November	2017	7544
14	October	2017	4631
15	September	2017	4285
16	April	2018	6939
17	August	2018	6512
18	February	2018	6728
19	January	2018	7269
20	July	2018	6292
21	June	2018	6167
22	March	2018	7211
23	May	2018	6873
24	October	2018	4
25	September	2018	16

Insights →

1. **FORMAT_DATETIME:** Function (FORMAT_DATETIME('%B',DATETIME(table_Name) AS month ('%B',DATETIME(table_Name) AS month) providing the month name in words AS (January, February) Helping us to under separation of orders monthly.
2. **EXTRACT :** Using EXTARCT function to extract year from the order_purchase_timestamp column so that we can distinguish the months of different year I which orders were made!
3. **COUNT:** Using count to count the order placed during each month, which helps in distinguishing the monthly orders.
4. **FROM :** FROM function determines the table from which I am extracting the data for my output.

5. **GROUP BY :** GROUP BY determining the data to be distributed and created group of data which contains the separated order details of each month and year.

INSIGHTS →

1. **Seasonal Trends :** Seasonal Trends can be seen, E.g in month may in 2017 and 2018 Shows high spike which can indicates festive, peak season or some promotion events.
2. **Growth By Year:** Growth can be seen in year based too which can help in yearly planning of your business and can prepare for more better performance so that we can track the business overall.
3. **Identifying anomalies:** Sudden spikes or drops in orders can be indicating some anomalies or event effecting your business, needed to be check those so that we can track the reason of external factors which is effecting our business

Q-2

C- 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**

ANSWER- Orders can be seen based on Dawn, Morning, Afternoon, Night From in query as below :

```sql

--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 h >= 0  AND h <= 6  THEN 'dawn'
    WHEN h >  6  AND h <= 12 THEN 'morning'
    WHEN h > 12  AND h <= 18 THEN 'evening'
    WHEN h > 18              THEN 'night'
      END AS moment
    , count(*) AS Total_orders
FROM  (
    SELECT extract(hour from order_purchase_timestamp) AS h
    FROM  `business_case.orders`
    ) Temp_
GROUP  BY moment
ORDER  BY count(*) DESC

```

The output of the above query is as below →

JOB INFORMATION | RESULTS | CHART | PREVIEW

Row	moment	Total_orders
1	evening	38135
2	night	28331
3	morning	27733
4	dawn	5242

Insights: Insights can be like

1. **CASE:** Using case function to determine the condition between the hours of 24 so that we can I can identify whether the people placed order during morning, evening, night or dawn.

2. **Count(*)** using the count function to count the number of order placed in different time of the day.
3. **FROM:** Using the FROM function to select the table name from where we can counting the orders and getting time at which the people placed orders.
4. **SUB_query:** Using the sub query to extract the hour from the order purchase timestamp!
5. **Group BY :** Grouping the data by moment so that we will get all the order placed In the given duration!

Recommendations :

1. **Tracking performance:**
   - After looking at the output we can tell that during evening sales are high so we need to keep eye on what makes customer order at this time of the day,
   - Maybe we can give some offers which might help in sales.
2. **Improving sales:**
   - Knowing some key factors will improve in improving the sales at different time zone too.
3. **Visualization:**
   - Consider creating visualizations (e.g., bar charts or pie charts) to represent the distribution of orders across different moments of the day

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

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

Answer: To get the total number of order placed by customers from each state in different month of the year can be As ➔

```sql

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

--A Q Get the month on month no. of orders placed in each state.

    SELECT
        FORMAT_DATETIME('%B', DATETIME(o.order_purchase_timestamp)) AS month,
        EXTRACT(YEAR FROM o.order_purchase_timestamp) AS Year,
        c.customer_state AS state,
        COUNT(*) AS Total_order
    FROM
        `business_case.orders` AS o
    JOIN
        `business_case.customers` AS c
    ON
        c.customer_id = o.customer_id
    GROUP BY
        month, state,Year
    ORDER BY
        Total_order DESC, month
            .
```

So, The output of the Query is like this ➔

Row	month	Year	state	Total_order
1	August	2018	SP	3253
2	May	2018	SP	3207
3	April	2018	SP	3059
4	January	2018	SP	3052
5	March	2018	SP	3037
6	November	2017	SP	3012
7	July	2018	SP	2777
8	June	2018	SP	2773
9	February	2018	SP	2703
10	December	2017	SP	2357
11	October	2017	SP	1793
12	August	2017	SP	1729
13	September	2017	SP	1638
14	July	2017	SP	1604

Insights ➔

1. **FORMAT_DATETIME:**
    - Using the format date time to get the month name in words so that its easily readable for everyone from order purchase timestamp column.
2. **EXTRACT:**
    - Here we are extracting the year from the order purchase timestamp so that we would know which year this month belongs to.
3. **JOIN:**
    - Joining the table 'customers' and 'orders' because state details belongs to customers table while the Data like month year and total number of orders can only be retrieved from orders table.
    - Condition of joining is based on the customer_id in orders table and customer_id in customer table.
4. **Grouping:**
    - Grouping the extracted data 1st by month so that the orders can be separated in month,
    - Then grouping data by state so data will be like orders were placed in which state in which month.
    - At last by year to distinguish the month from different year!
5. **Sorting:**
    - sorting the output table so that its not jumbling and easy to understand!

Recommendations ➔

1. **Performance :**
    - Tracking the number of orders from each state in every month throughout the year can be done so its easy to check which state has orders.
2. **Resource planning:**
    - We can plan how much product and which products would be required in next month after following the trend.
    - This way we can also save the resources, E.g in state 'TO' has significantly very low number of orders and needed to be audited and placed in some other state.
3. **More efficient Data:**
    - We can get more efficient data if we had also grouped the data by city so that we would know which city is preferring our business.
4. **Events and promotions:**
    - State which has low orders needs to launch some event to promote this business idea, or promotion of business which may help in increasing in sales.

Question 3

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

Answer ➔ Customer distributed across all the states or Total number of customers in each state can be obtained from table name 'customers' by using this query➔

```sql
--B Q How are the customers distributed across all the states?

SELECT
    COUNT(DISTINCT(customer_id)) AS Total_customer,
    customer_state AS state_name
from
    `business_case.customers`
GROUP BY
    customer_state
ORDER BY
    Total_customer DESC;
```

and the output of this query is as →

JOB INFORMATION | RESULTS | CHART | PREVIEW

Row	Total_customer	state_name
1	41746	SP
2	12852	RJ
3	11635	MG
4	5466	RS
5	5045	PR
6	3637	SC
7	3380	BA
8	2140	DF
9	2033	ES
10	2020	GO
11	1652	PE
12	1336	CE

Insights :

1. **SELECT customer_state AS state_name:**
    - This part selects the customer_state column from the business_case.customers table and renames it as state_name in the result set.
2. **COUNT(DISTINCT customer_id) AS total_customers:**
    - This part counts the distinct customer_id values for each state. It uses the DISTINCT keyword to ensure that each customer is only counted once, even if they have multiple records in the table. The result is then given an alias total_customers for better readability.
3. **FROM business_case.customers:**
    - Specifies the table from which to retrieve the data, in this case, the business_case.customers table.
4. **GROUP BY customer_state:**
    - Groups the results by the customer_state column. This means that the count will be calculated separately for each unique state
5. **Sorting :**
    - ORDER BY is used to sort the table In decreasing order of customers.

Recommendations :

1. **Efficiency:**
    - Ensure that the customer_state and customer_id columns are appropriately indexed for better query performance, especially if the customers table is large.

2. **Regular Updates:**
    - If this analysis is part of regular reporting, consider automating the process and scheduling updates to ensure that insights are always based on the latest data

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

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

Answer:

The query to get the percent increase in cost of orders from 2017 to 2018 only between the month January to august can be written as

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

WITH YearlyCost AS (
   SELECT
    EXTRACT(YEAR FROM o.order_purchase_timestamp) AS year,
    SUM(p.payment_value) AS total_cost
FROM
    `business_case.orders` AS o
JOIN
    `business_case.payments` 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
GROUP BY
    year
)
SELECT
    (yc_2018.total_cost - yc_2017.total_cost) / yc_2017.total_cost * 100 AS percentage_increase
FROM
    YearlyCost AS yc_2017
JOIN
    YearlyCost AS yc_2018 ON yc_2017.year = 2017 AND yc_2018.year = 2018;
```

The output table is as:

Row	percentage_increase
1	136.97687164665652

The output table shows what percent has been the increased the cost:

Insights:

1. **Common Table Expression (CTE) - YearlyCost:**
    - From row number 63 to 75 of the query defines a Common Table Expression (CTE) named YearlyCost.
    - It extracts the year from the order_purchase_timestamp, sums up the payment_value for each year, and filters the data for the months between January and August.
    - The result is a table with two columns: year and total_cost.
2. **Percentage Increase Calculation:**
    - From row number 77 to 82 of the query calculates the percentage increase between the total costs of orders in 2017 and 2018.
    - It subtracts the total cost in 2017 (yc_2017.total_cost) from the total cost in 2018 (yc_2018.total_cost).
    - Then, it divides this difference by the total cost in 2017 and multiplies by 100 to get the percentage increase.
    - The result is a single column with the percentage increase.

Recommendations:

1. **Evaluate Pricing Strategy:**
   - Assess the impact of any changes in product prices on the total cost. If prices have increased, evaluate whether this has positively influenced revenue and profitability
2. **Promotions and Discounts:**
   - Review any promotions or discounts offered during this period. Assess the impact on sales and whether the associated costs are justified by increased revenue.

QUESTION

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

Answer : The query to derive The Total order price and Average price for each state is as:

```
67
68  -- 4.2  Calculate the Total & Average value of order price for each state
69  SELECT
70
71      c.customer_state AS state,
72      SUM(p.payment_value) AS Sum_value_of_state,
73      AVG(p.payment_value) AS Average_value_of_state
74
75  FROM
76      `business_case.payments` AS p
77  JOIN
78      `business_case.orders` AS o
79  ON   p.order_id=o.order_id
80  JOIN
81      `business_case.customers` AS C
82  ON   o.customer_id = C.customer_id
83
84  GROUP BY C.customer_state
85
86
```

And output of this query we got state in which we have total sum of payment value of each state and average of payment of each state.

JOB INFORMATION | RESULTS | CHART PREVIEW | JSON | EXECUTION DETA

Row	state	Sum_value_of_state	Average_value_of_state
1	BA	616645.8200000003	170.81601662049846
2	SP	5998226.95999983	137.50462977396745
3	RJ	2144379.689999993	158.5258882235531
4	MT	187029.29	195.228903966597
5	GO	350092.30999999994	165.76340435606042
6	ES	325967.55	154.70695301376369
7	RS	890898.5400000005	157.18040578687354
8	MG	1872257.2599999993	154.70643364733024
9	MA	152523.02	198.85661016949115
10	SC	623086.43000000052	165.97933670751212
11	PR	811156.38000000361	154.15362599771933

Insights:

1. **Sum**() :
   - At line number 72 "Sum(`p.payment_value`) `AS` `Sum_value_of_state`" sums the payment value As total value of a state.
2. **Average():**
   - At line 73 " `AVG`(`p.payment_value`) `AS` `Average_value_of_state`" we use AVG function to

get the average value of payment done in each state and saving the average of each state in “Average_value_of_state”

3. **Joining The Table:**
   - From line 75 to 82 We are joining the table payment, customer, and order with alias p, c, and o respectively so that we would we able to identify which column belongs to which table.
   - While adding the payment value in month wise we are calling the column name as “p.payment_value” Where “p” indicated the table and payment_value is the column name.
   - Joining the three tables with the common column in 2 tables Example as
     Joining the orders and payment table with the help of order_id column in both table and joining the third table customers with this joint table on customer_id which is present in order and customer table.
4. **Group By () :**
   - This is the most important part which helps in grouping the sum and average of payments done by customer in each State.

Recommendation:

1. **Descriptive Alias Names:**
   - Choose clear and descriptive alias names for better readability.
2. **Data update:**
   - Data needs to be updated daily to keep track of exact figures.
3. **Data Accuracy:**
   - Making sure that the data filled or updated is accurate.

Question 4.3

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

Answer : The query to get that Total and Average value of order freight for each state is as:

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

SELECT

    c.customer_state AS state,
    SUM(oi.freight_value) AS Sum_value_of_state,
    AVG(oi.freight_value) AS Average_value_of_state

FROM
    `business_case.orders` AS o
JOIN
    `business_case.order_items` AS oi

ON   oi.order_id=o.order_id
JOIN
    `business_case.customers` AS C
ON   o.customer_id = C.customer_id

GROUP BY C.customer_state
```

The output of this looks like this :

JOB INFORMATION | RESULTS | CHART PREVIEW | JSON | EXECU

Row	state	Sum_value_of_state	Average_value_of_state
1	SP	718723.06999999378	15.147275390419132
2	RJ	305589.31000000431	20.960923931682483
3	PR	117851.68000000058	20.531651567944269
4	SC	89660.260000000053	21.470368773946323
5	DF	50625.499999999418	21.041354945968422
6	MG	270853.4600000073	20.630166806306651
7	PA	38699.300000000047	35.832685185185213
8	BA	100156.67999999922	26.36395893656228
9	GO	53114.979999999705	22.766815259322772
10	RS	135522.74000000197	21.735804330392952
11	TO	11732.679999999998	37.246603174603166
12	AM	5478.8900000000012	33.205393939393922

In this output table we have state and each state represents the sum of freight value and average of freight value done in each state.

Insights:

1. **SUM():**
   - At line number 93 "Sum(`oi.freight_value`) `AS Sum_value_of_state`" `which adds the freight value` As total value of a state.
2. **Average():**
   - At line 94 " `AVG(oi.freight_value) AS Average_value_of_state` we use AVG function to get the average value of freight_value done in each state and saving the average of each state in "Average_value_of_state"
3. **Joining The Table:**
   - From line 96 to 104 We are joining the table order_items, customer, and order with alias oi, c, and o respectively so that we would we able to identify which column belongs to which table.
   - While adding the freight_value in month wise we are calling the column name as "oi.freight_value" Where "oi" indicated the table and freight_value is the column name.
   - Joining the three tables with the common column in 2 tables Example as Joining the orders and order_items table with the help of order_id column in both table and joining the third table customers with this joint table on customer_id which is present in order and customer table.
4. **Group By () :**
   - GROUP BY Which helps in grouping the Total sum and average of freight_value customer in each state.

Recommendation:

1. **Descriptive Alias Names:**
   - Choose clear and descriptive alias names for better readability.
2. **Data update:**
   - Data needs to be updated daily to keep track of exact figures.
3. **Data Accuracy:**
   - Making sure that the data filled or updated is accurate.

Question

5.1 Find the number 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.

Answer:

Query to get the time taken to deliver the order and the difference between the estimated delivery and actual delivery time is as below:

```
--Question 5.1
--Find the no. of days taken to deliver each order from the order's purchase date as delivery time.
--Also, calculate the difference (in days) between the estimated & actual delivery date of an order.
--Do this in a single query.


SELECT
      order_id,
      ROUND(DATE_DIFF(order_delivered_carrier_date,order_purchase_timestamp,Second)/86400,2) AS Delivery_time,
      ROUND(DATE_DIFF(order_estimated_delivery_date,order_delivered_carrier_date,SECOND)/86400,2) AS Diff_OF_actual_estimated_deliveryTime

FROM
      `business_case.orders`
WHERE
       (DATE_DIFF(order_delivered_carrier_date,order_purchase_timestamp,Second)/86400) IS NOT NULL
AND
      (DATE_DIFF(order_estimated_delivery_date,order_delivered_carrier_date,SECOND)/86400) IS NOT NULL
```

And the output of the query is as below, In this table we have the all order id with its time taken to deliver the product and the difference of estimated and actual delivery date!

JOB INFORMATION | RESULTS | CHART | PREVIEW | JSON | EXECUTION DET

Row	order_id	Delivery_time	Diff_OF_actual_estimated_deliveryTime
1	f… Open sort menu …ccc5cb0e3…	3.95	23.52
2	1df2775799eecdf9dd8502425…	5.3	26.24
3	6190a94657e1012983a274b8…	2.18	31.25
4	58ce513a55c740a3a81e8c8b7…	1.94	13.3
5	088683f795a3d30bfd61152c4f…	7.42	24.16
6	aa380313c19905dd1651bd21e…	1.08	24.39
7	2e03cb2541b48c78aebca2dbf…	2.89	22.34
8	d1b7637acd3a7a42101faf906…	3.5	12.71
9	556b640fe8c3c509b624e3d5e…	0.73	32.39
10	c160599d4ea4eefa0e420db0a…	3.36	10.1
11	a34db9935aad747acfecadbba…	4.01	27.34
12	2d1cfcc5ed3232215b908e86ff…	9.96	5.13
13	7b9906348a4044ff73ce3034e…	0.82	27.28
14	5dfcc26420fd2e456e613e8dfb…	2.02	21.33
15	f247ddbea2a3d9e88b688d08f…	1.67	19.27

Insights:

1. **SELECTING:**
   - Select Function is being used to display the desired column from line number 116 to 118 we are displaying the desired output while at the same time getting the difference,
2. **DateDIff:**
   - Using the DateDiff to calculate the difference between the order purchase time stamp and order delivered time in seconds and then diving that with 86400(second in 24 hour) to get the exact time taken.
   - E.g 3.95 means it took 3 days and 9 hour and approx. 50 min to deliver the product and so the other orders.
3. **Filtering:**
   - I noticed some products were missing missing date so in have filtered out all the null value from both of the column.

Recommendations:

1. **Readability and Formatting:**
   - Consider using proper indentation for better readability.
   - Use aliases for table names to improve code readability. For example, instead of business_case.orders you can use ‘o’ as an alias.

2. **Continuous Improvement:**
   - Embrace a culture of continuous improvement. Regularly review and update your processes based on the insights derived from delivery time data

3. **Predictive Analytics:**
   - Explore the use of predictive analytics to forecast delivery times more accurately. Machine learning models can leverage historical data to make more precise estimates based on various factors.

Question 5.2

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

Answer: The query to get the top 5 state with highest & lowest average freight value can be written,

1st top 5 highest average freight value state is as :

```
--Question 5.2 Find out the top 5 states with the highest & lowest average freight value.

Highest_5_AVG_freight
SELECT
      c.customer_state,
      AVG(oi.freight_value) AS AvgFreightValue
FROM
      `business_case.order_items` AS oi
JOIN
      `business_case.orders` AS o ON o.order_id = oi.order_id
JOIN
      `business_case.customers` AS c ON c.customer_id = o.customer_id
GROUP BY
      c.customer_state
ORDER BY
       AvgFreightValue DESC
LIMIT 5
```

And the output is as :

JOB INFORMATION | RESULTS | CHART | PREVIEW

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

Insights:

1. **Select():**
   - Since we only need the customer state and average freight value of each state so we are selecting only two column in select clause.
   - While we are selecting the customer state from customer table but we are generating average of freight value while selecting that average value.
2. **FROM & JOIN:**
   - We are required 3 tables to get the customer state from customer table and freight value column from order_items table.
   - We basically need only 2 tables but there is no common column in customer and order_item we need another table to help us in joining.
   - Since there is 3 table we need to join them with alias order_items table as 'oi' with order table on order_id and thencustomer table as 'c' on customer_id.
3. **Group By:**
   - To get the desired average we need to group the data by customer state respectively so that all the sum and average will occur on state wise.
4. **Sorting and limit():**
   - Since we need only top 5 highest that's why we are sorting the average freight in descending order and limiting the output for only 5.

Query to get the state which has top 5 lowest average freight value is as:

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

The output is as:

JOB INFORMATION | RESULTS | CHART | PREVIEW

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

Insights

1. **Select():**
   - Since we only need the customer state and average freight value of each state so we are selecting only two column in select clause.
   - While we are selecting the customer state from customer table but we are generating average of freight value while selecting that average value.
2. **FROM & JOIN:**
   - We are required 3 tables to get the customer state from customer table and freight value column from order_items table.
   - We basically need only 2 tables but there is no common column in customer and order_item we need another table to help us in joining.
   - Since there is 3 table we need to join them with alias order_items table as 'oi' with order table on order_id and thencustomer table as 'c' on customer_id.
3. **Group By:**
   - To get the desired average we need to group the data by customer state respectively so that all the sum and average will occur on state wise.
4. **Sorting and limit():**
   - Since we need only top 5 lowest that's why we are sorting the average freight in ascending order and limiting the output for only 5.

In Highest and lowest there in only sorting difference and the query can be same for both!

Recommendations on Both:

1. **Regional Freight:**
   - Analyse the top 5 customer states with the highest average freight values. Look for patterns or regional disparities in shipping costs. This information can be valuable for optimizing logistics and pricing strategies based on region
2. **Competitive Pricing:**
   - Evaluate how your shipping costs compare to competitors in the states with the highest average freight values. If your shipping costs are significantly higher, consider optimizing logistics or negotiating better rates with shipping partners to maintain competition.
3. **Customer Satisfaction:**
   - Assess whether customers in states with lowe average freight values are satisfied with the shipping experience. If there are concerns or complaints related to shipping, communication, or delivery times, address them to enhance overall customer satisfaction and loyalty.

Question 5.3

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

Answer : The query to get the top 5 states with highest and lowest average delivery time can be written as:

For highest time taken to deliver the order is as:

```
--Q5.3 Find out the top 5 states with the highest & lowest average delivery time.

SELECT
      c.customer_state,
      AVG(DATE_DIFF(o.order_delivered_carrier_date,o.order_purchase_timestamp,Second))/3600 AS AVG_highest_delivery_time


FROM
      `business_case.orders` AS o
JOIN
      `business_case.customers` AS c
ON
      o.customer_id=c.customer_id
GROUP BY c.customer_state
ORDER BY AVG_highest_delivery_time DESC
LIMIT 5
```

And the output of this query is as:

JOB INFORMATION | RESULTS | CHART | PREVIEW

Row	customer_state	AVG_highest_delivery_time
1	RR	109.15188888888891
2	SE	87.1727365956072
3	MA	86.6835547996977
4	RN	86.24499768999766
5	AP	84.517056384742943

These are the top 5 state which has highest average delivery time.

And for lowest top 5 is as:

```
SELECT
      c.customer_state,
      AVG(DATE_DIFF(o.order_delivered_carrier_date,o.order_purchase_timestamp,Second))/3600 AS AVG_Lowest_delivery_time


FROM
      `business_case.orders` AS o
JOIN
      `business_case.customers` AS c
ON
      o.customer_id=c.customer_id
GROUP BY c.customer_state
ORDER BY AVG_Lowest_delivery_time ASC
LIMIT 5
```

The output I got is as:

JOB INFORMATION | RESULTS | CHART | PREVIEW

Row	customer_state	AVG_Lowest_delivery
1	RO	68.36794467306...
2	AM	69.61633219954...
3	SP	75.95866546603...
4	MS	76.02440893308...
5	GO	76.24028434165...

Since the query for both highest and lowest is almost same so the insights are also similar.

Insights:

5. **Select():**
   - Since we only need the customer state and average Delivery time of each state so we are selecting only two column in select clause.
   - While we are selecting the customer state from customer table but we are also generating average of delivery time while selecting that value.

- To get the delivery time taken we have to use DATE_DIFF function which provides difference in purchase time and delivered time which is out time taken.

6. **FROM & JOIN:**
   - We are required 2 tables to get the customer state from customer table and average value of delivery date column from order table.
   - Since there is 2 table we need to join them with alias order table as 'o' and customer table as 'c' on customer_id.
7. **Group By:**
   - To get the desired average we need to group the data by customer state respectively so that all average will occur on state wise.
8. **Sorting and limiting():**
   - Since we need only top 5 highest and lowest that's why we are sorting the delivery date in descending and ascending order at last limiting the output for only 5.

Recommendations:

1. **Variations over region:**
   - The analysis reveals variations in average delivery times across different customer states. Some states experience quicker deliveries, while others may have longer delivery times.
   - Understanding these regional differences is crucial for optimizing logistics and meeting customer expectations

2. **Efficient Operations:**

   - States with lower average delivery times may indicate efficient operational processes, well-located distribution centres, or optimized logistics.
3. **Customer Satisfaction Considerations:**
   - Customer satisfaction is closely tied to delivery times.
   - Quicker deliveries may contribute to higher satisfaction, while longer delivery times may lead to customer dissatisfaction.
   - Managing customer expectations and addressing potential concerns in regions with longer delivery times is crucial.

Question 5.4

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

Answer:

The query to get the top 5 state where order delivery is really fast as compared to the estimated date of delivery is as:

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

SELECT
      c.customer_state,
FROM
      `business_case.orders` AS o
JOIN
      `business_case.customers` AS c
ON
      o.customer_id=c.customer_id
WHERE
      ROUND(DATE_DIFF(order_delivered_carrier_date,order_purchase_timestamp,Second)/86400,2) <
      ROUND(DATE_DIFF(order_estimated_delivery_date,order_delivered_carrier_date,SECOND)/86400,2)
ORDER BY (
     ROUND(DATE_DIFF(order_delivered_carrier_date,order_purchase_timestamp,Second)/86400,2) <
      ROUND(DATE_DIFF(order_estimated_delivery_date,order_delivered_carrier_date,SECOND)/86400,2)
) DESC
```

And the output is as:

JOB INFORMATION | RESULTS

Row	customer_state
1	SP
2	MT
3	MA
4	MT
5	AL

Output table shows the top 5 state where delivery is really fast.

Insights:

1. **SELECT:**
   - Since we only need state so we are only selecting the state column.
2. **FROM:**
   - Selecting the table from where we will extract the desired output that table is customers and orders table.

3. **JOIN:**
   - Since there is two table we need to perform join on them with alias and condition on customer_id.
4. **WHERE():**
   - Here we are filtering out those data using the DATE_DIFF() to get the delivery time taken then comparing with obtained time which were expected will take to deliver.
   - To get the delivery time taken we got the difference in seconds using DATE_DIFF function the delivery time and purchase time. Similarly calculated the time which was supposed to take.
   - Compare condition is like if the delivery time is less then expected time then it is really fast.
5. **Sorting:**
   - Sorting the output table with ORDER BY function in descending order.
6. **Limiting:**
   - Limiting the output so that we will only have top 5 output.

Recommendations:

1. **Efficient Operations:**

   - States with lower average delivery times may indicate efficient operational processes, well-located distribution centres, or optimized logistics.
2. **Customer Satisfaction Considerations:**
   - Customer satisfaction is closely tied to delivery times.
   - Quicker deliveries may contribute to higher satisfaction, while longer delivery times may lead to customer dissatisfaction.
   - Managing customer expectations and addressing potential concerns in regions with longer delivery times is crucial.

Question 6 Analysis based on the payments:

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

Answer:

Query To get the month on month no. of orders placed using different payment types is as:

```
--Q6.1 Find the month on month no. of orders placed using different payment types
SELECT
      CONCAT(FORMAT_DATETIME('%B', DATETIME(o.order_purchase_timestamp)), " ",EXTRACT(YEAR FROM o.order_purchase_timestamp))AS
month_of_year,
      p.payment_type AS payment_type,
      COUNT(p.payment_type) AS Number_of_payment
FROM
      `business_case.payments` AS p
JOIN
      `business_case.orders` AS o
ON
      p.order_id=o.order_id
GROUP BY
      p.payment_type,month_of_year
ORDER BY Number_of_payment DESC
```

The output table is like:

Row	month_of_year	payment_type	Number_of_payment
1	November 2017	credit_card	5897
2	March 2018	credit_card	5691
3	January 2018	credit_card	5520
4	May 2018	credit_card	5497
5	April 2018	credit_card	5455
6	February 2018	credit_card	5253
7	August 2018	credit_card	4985
8	June 2018	credit_card	4813
9	July 2018	credit_card	4755
10	December 2017	credit_card	4377
11	October 2017	credit_card	3524
12	August 2017	credit_card	3284
13	September 2017	credit_card	3283
14	July 2017	credit_card	3086

The output contains month belongs To which year and payment type Contains the all modes of payment While the number of payment Shows total times each mode was used to pay in different month.

Insights:

1. **SELECT():** In SELECT I need 3 type of data ,
   - 1st is “month of year” containing the month name of the corresponding years, for this I need to get the month name using FROMAT_DATETIME function and to get year I am using EXRACT function,
   - To get both together I am using CONCAT function.
   - Payment_type column from payment table and providing alias as Number_of_payment.
   - To get the data for how many times each mode was used I am using using function COUNT() so that it can count the number of time each mode was used to pay.
2. **FROM():**
   - Using the From function to get the table we are required.
   - In this case we are required only two tables those are payment and orders
3. **JOIN():**
   - Since there is two table I am joining them using order_id table which is present in both table and with the help of alias as o.order_id=p.order_id
4. **Group BY ():**
   - Since we need data grouped by month and different payment mode so we are grouping the data by month and Payment_type.

Recommendations:

1. **Payment Type Optimization:**
    - Identify payment types that consistently contribute to high payment counts. Consider optimizing processes related to those payment types, such as processing or negotiating favourable terms with payment providers.
2. **Payment Method Popularity:**
    - The query allows you to identify the popularity of different payment methods. By examining the number of payments for each payment type, you can gain insights into which methods are preferred by customers.
3. **User Experience Enhancement:**
    - Enhancing the user experience can be key factor in business growth.

Question 6.2

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

Answer: The Query To get the number of orders placed on the basis of the payment instalments that have been paid can be written as:

```
239
240 --Q6.2 Find the no. of orders placed on the basis of the payment installments that have been paid.
241 SELECT
242       COUNT(order_id) AS Number_of_order,
243       payment_installments AS Number_of_installment_paid
244 FROM
245       `business_case.payments`
246 WHERE
247       payment_installments IS NOT NULL
248 GROUP BY
249        payment_installments
250 ORDER BY
251        payment_installments DESC
252
253
254
```

The result table has Number of orders which contains the number of order and the column Number of installment shows how many installment have been paid of how many orders.

Example: Row no 1 has 18 number of order in which these 18 orders have paid 24 instalments.

Result table as ➔

JOB INFORMATION | RESULTS | CHART | PREVIEW

Row	Number_of_order	Number_of_installme
1	18	24
2	1	23
3	1	22
4	3	21
5	17	20
6	27	18
7	8	17
8	5	16
9	74	15
10	15	14
11	16	13
12	133	12
13	23	11
14	5328	10

Insights:

1. **SELECT():** In SELECT I need 2 column of data .
    - 1st is “Number of orders” containing the each order count based on number of instalment paid.
    - 2nd column Number of instalment containing the number of instalment paid.
2. **FROM():**
    - Using the From function to select the table we are required.

- In this case we are required only one tables that is payments table

3. **WHERE():**
    - Filtering and removing the null value rows from table by using IS NOT NULL in payment_instalments.
4. **Group BY ():**
    - Since we need data grouped by payment instalments so we are grouping them by Payment_instalments Fron payment table

Recommendations:

1. **Financial Planning:**
    - Use the insights to plan and forecast cash flows and revenue based on different instalment options. Understanding the distribution of instalment payments helps in predicting future financial performance.
2. **Technology and Infrastructure:**
    - Ensure that your technology infrastructure can support and efficiently manage instalment payments. Invest in systems that provide a seamless experience for both customers and internal operations