



TARGET (SQL)

Introduction

- Target is a globally renowned brand and a prominent retailer in the United States. Target makes itself a preferred shopping destination by offering outstanding value, inspiration, innovation and an exceptional guest experience that no other retailer can deliver.
- This particular business case focuses on the operations of Target in Brazil and provides insightful information about 100,000 orders placed between 2016 and 2018. The dataset offers a comprehensive view of various dimensions including the order status, price, payment and freight performance, customer location, product attributes, and customer reviews.
- By analyzing this extensive dataset, it becomes possible to gain valuable insights into Target's operations in Brazil. The information can shed light on various aspects of the business, such as order processing, pricing strategies, payment and shipping efficiency.

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

A. Data type of all columns in the “customers” table.

Query: `SELECT`

`Column_name,`

`Data_type`

`FROM `Target.INFORMATION_SCHEMA.COLUMNS``

`WHERE TABLE_NAME = "customers"`

Output screenshot:

The screenshot shows the Google Cloud BigQuery interface. The query editor displays the following SQL query:

```
SELECT
  Column_name,
  Data_type
FROM `Target.INFORMATION_SCHEMA.COLUMNS`
WHERE TABLE_NAME = "customers"
```

The query results are displayed in a table with the following columns: Row, Column_name, and Data_type. The results show the data types for the columns in the 'customers' table.

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

Insights:

- All the data type is "STRING" except customer_zip_code_prefix it has "INT" data type.

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

Query:

SELECT

```
DATE_DIFF(last_order, a.first_order, DAY) AS Range_in_DAYS,
DATE_DIFF(last_order, a.first_order, MONTH) AS Range_in_MONTH,
DATE_DIFF(last_order, a.first_order, YEAR) AS Range_in_YEAR,
```

FROM

(SELECT

```
DATE(MIN(order_purchase_timestamp)) AS first_order,
DATE(MAX(order_purchase_timestamp)) AS last_order
```

FROM `Target.orders`) AS a

Output screenshot:

The screenshot shows the Google Cloud BigQuery console. On the left is the Explorer panel with a search bar and a list of resources. The main area displays a SQL query in a text editor titled 'Untitled 3'. The query is as follows:

```
SELECT
  DATE_DIFF(last_order, a.first_order, DAY) AS Range_in_DAYS,
  DATE_DIFF(last_order, a.first_order, MONTH) AS Range_in_MONTH,
  DATE_DIFF(last_order, a.first_order, YEAR) AS Range_in_YEAR,
FROM
  (SELECT
    DATE(MIN(order_purchase_timestamp)) AS first_order,
    DATE(MAX(order_purchase_timestamp)) AS last_order
  FROM `Target.orders` ) AS a
```

Below the query editor, the 'Query results' section is visible. It includes tabs for 'JOB INFORMATION', 'RESULTS', 'CHART', 'JSON', 'EXECUTION DETAILS', and 'EXECUTION GRAPH'. The 'RESULTS' tab is active, showing a table with the following data:

Row	Range_in_DAYS	Range_in_MONTH	Range_in_YEAR
1	773	25	2

Insights:

- We have data of 2 years and there are lots of transactions that were placed in the span of 2 years so we have lots of data to analyse.

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

Query:

SELECT

```
COUNT(DISTINCT customer_city) AS count_of_city,  
COUNT(DISTINCT customer_state) AS count_of_state
```

```
FROM `Target.customers`
```

Output screenshot:

The screenshot displays the Google Cloud BigQuery console. On the left, the Explorer pane shows the 'Target' dataset. The main editor area contains a SQL query titled 'Untitled 3'. The query is as follows:

```
SELECT  
  COUNT(DISTINCT customer_city) AS count_of_city,  
  COUNT(DISTINCT customer_state) AS count_of_state  
FROM `Target.customers`
```

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

Row	count_of_city	count_of_state
1	4119	27

Insights:

- There are **4,119 cities** and **27 states** from where orders are placed.

II. In-depth Exploration:

A. 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,

COUNT(DISTINCT order_id) AS count_of_order_id

FROM `Target.orders`

GROUP BY EXTRACT(YEAR FROM (order_purchase_timestamp))

ORDER BY year

Output screenshot:

The screenshot shows the Google Cloud BigQuery console interface. On the left is the Explorer pane with a search bar and a list of resources including 'target-414911' and 'Target'. The main area displays a SQL query in a text editor, which has been executed. Below the query editor, the 'Query results' section is visible, showing a table with 3 rows and 3 columns: 'Row', 'year', and 'count_of_order_id'. The results show an increasing trend in the number of orders over the years 2016, 2017, and 2018.

Row	year	count_of_order_id
1	2016	329
2	2017	45101
3	2018	54011

Insights:

- There is **increase in trend**
- The **order is increasing** over the past years
- There is **positive business growth**

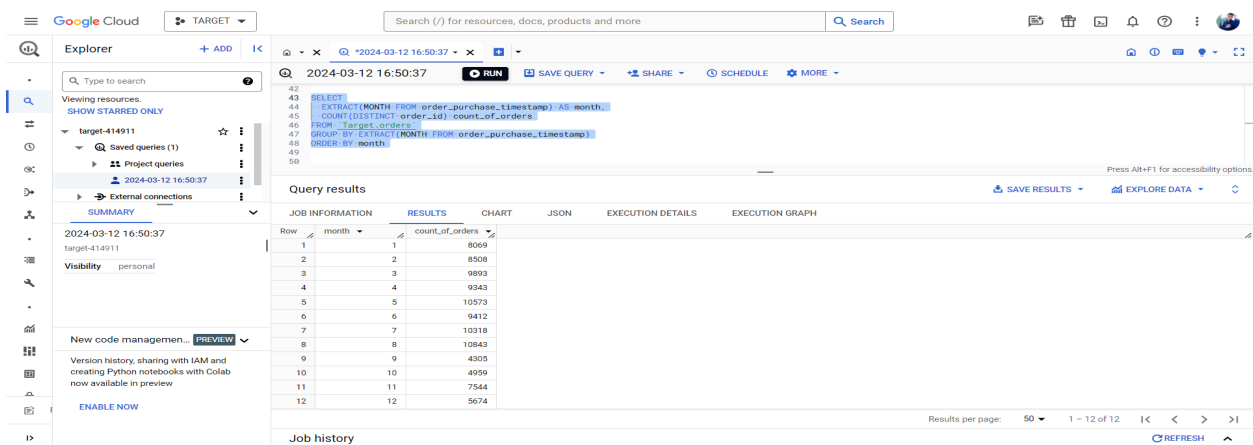
B. Can we see some kind of monthly seasonality in terms of the no. of orders being placed?

Query:

SELECT

```
EXTRACT(MONTH FROM order_purchase_timestamp) AS month,
COUNT(DISTINCT order_id) count_of_orders
FROM `Target.orders`
GROUP BY EXTRACT(MONTH FROM order_purchase_timestamp)
ORDER BY month
```

Output screenshot:



Insights:

- In the month of May, July and August there is highest sales.
- In the month of Sep and Oct there is lowest sales

Recommendation :

- In the month of Sep and Oct Company should give offers to their customers so it can increase its sales.

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

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 "Mornings"
    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_Interval,
COUNT(DISTINCT order_id) AS Count_of_order
FROM `Target.orders`
GROUP BY Time_Interval
ORDER BY Count_of_order DESC
```

Output screenshot:

The screenshot displays the Google Cloud BigQuery console. On the left, the Explorer pane shows a project named 'target-414911' with a saved query from '2024-03-12 16:50:37'. The main editor shows the SQL query used to categorize orders by time of day. The 'Query results' pane at the bottom shows the output of the query.

Row	Time_Interval	Count_of_order
1	Afternoon	38135
2	Night	28331
3	Mornings	27733
4	Dawn	5242

Insights:

- Afternoon is the peak period when Brazilian customers are most active in making purchases.
- Afternoon orders may indicate that Brazilian customers prefer to engage in shopping activities during their leisure time, perhaps during breaks from work or other daily responsibilities. This behavior may be influenced by factors such as convenience and availability.

Recommendation:

- At Afternoon time company should adjust their operational processes to better accommodate fluctuations in demand. For instance, staffing levels, inventory management, and logistics arrangements can be optimized to ensure efficient order processing and delivery during high-demand periods.
- Company can tailor their services based on the identified order placement trends. For example, offering special promotions or customer support services during Afternoon can enhance the overall shopping experience and drive customer satisfaction.

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

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

Query:

SELECT

```
c.customer_state,
EXTRACT(MONTH FROM o.order_purchase_timestamp) AS MONTH,
COUNT(DISTINCT o.order_id) AS No_or_orders
FROM `Target.orders` AS o
LEFT JOIN `Target.payments` AS p
ON o.order_id = p.order_id
LEFT JOIN `Target.customers` AS c
ON o.customer_id = c.customer_id
GROUP BY MONTH,c.customer_state
ORDER BY c.customer_state,MONTH
```

Output screenshot:

Query results

Row	customer_state	MONTH	No_or_orders
1	AC	1	8
2	AC	2	6
3	AC	3	4
4	AC	4	9
5	AC	5	10
6	AC	6	7
7	AC	7	9
8	AC	8	7
9	AC	9	5
10	AC	10	6
11	AC	11	5
12	AC	12	5

Results per page: 50 1 - 50 of 322

Insights:

- SP has the lowest sales in the month of "Sep", "Nov " and "Dec ".
- GO has the lowest sales in "Sep".
- There is a reduction in sales at "MG" after "Aug".
- There is a reduction in sales at "RJ" after "Aug".

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

Query:

SELECT

```
customer_state,
COUNT(DISTINCT customer_id) AS No_of_customers
FROM `Target.customers`
GROUP BY customer_state
ORDER BY No_of_customers DESC
```

Output screenshot:

The screenshot shows the Google Cloud BigQuery interface. The query results are displayed in a table with the following data:

Row	customer_state	No_of_customers
1	SP	41746
2	RJ	12852
3	MG	11633
4	RS	5466
5	PR	5045
6	SC	3637
7	BA	3380
8	DF	2140
9	ES	2039
10	GO	2020
11	PE	1652
12	CE	1336
13	PA	979
14	MT	907
15	MA	747
16	MS	715
17	PB	556
18	PI	493
19	RN	483
20	AL	413
21	SE	350
22	TO	280
23	RO	253
24	AM	148
25	AC	81
26	AP	68
27	RR	46

Insights:

- "SP" has the highest number of customers.
- "RR" has the lowest number of customers.

Recommendation:

- Since "RR" has the lowest number of customers so there is a growth opportunity in this state, the company should focus on "RR" to expand its customer base.
- Company should allocate fewer resources as compared to other states.

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

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

Query:

```
WITH order_2017 AS (SELECT
    EXTRACT(MONTH FROM o.order_purchase_timestamp ) AS month,
    SUM(t.payment_value) AS Sum_1
FROM `Target.orders` AS o
JOIN `Target.payments` AS t
ON o.order_id = t.order_id
WHERE EXTRACT(YEAR FROM o.order_purchase_timestamp ) = 2017 AND
    EXTRACT(MONTH FROM o.order_purchase_timestamp ) BETWEEN 1 AND 8
GROUP BY month),
order_2018 AS (SELECT
    EXTRACT(MONTH FROM o.order_purchase_timestamp ) AS month,
    SUM(t.payment_value) AS Sum_2
FROM `Target.orders` AS o
JOIN `Target.payments` AS t
ON o.order_id = t.order_id
WHERE EXTRACT(YEAR FROM o.order_purchase_timestamp ) = 2018 AND
    EXTRACT(MONTH FROM o.order_purchase_timestamp ) BETWEEN 1 AND 8
GROUP BY month)
SELECT
    ROUND((SUM(o2.sum_2)-SUM(o1.sum_1))*100/SUM(o1.sum_1),2) AS percentage_change
FROM order_2017 AS o1
JOIN order_2018 AS o2
ON o1.month = o2.month
```

Output screenshot:

The screenshot displays the Google Cloud BigQuery console. The left sidebar shows the Explorer with a project named 'target-414911'. The main area shows a SQL query for the date '2024-03-12 16:50:37'. The query calculates the percentage change in sales from 2017 to 2018, filtered by month (January to August).

```

WITH order_2017 AS (SELECT
  EXTRACT(MONTH FROM o.order_purchase_timestamp) AS month,
  SUM(t.payment_value) AS Sum_1
FROM `Target.orders` AS o
JOIN `Target.payments` AS t
ON o.order_id = t.order_id
WHERE EXTRACT(YEAR FROM o.order_purchase_timestamp) = 2017 AND
EXTRACT(MONTH FROM o.order_purchase_timestamp) BETWEEN 1 AND 8
GROUP BY month),
order_2018 AS (SELECT
  EXTRACT(MONTH FROM o.order_purchase_timestamp) AS month,
  SUM(t.payment_value) AS Sum_2
FROM `Target.orders` AS o
JOIN `Target.payments` AS t
ON o.order_id = t.order_id
WHERE EXTRACT(YEAR FROM o.order_purchase_timestamp) = 2018 AND
EXTRACT(MONTH FROM o.order_purchase_timestamp) BETWEEN 1 AND 8
GROUP BY month)
SELECT
  ROUND((SUM(o2.sum_2)-SUM(o1.sum_1))*100/SUM(o1.sum_1),2) AS percentage_change
FROM order_2017 AS o1
JOIN order_2018 AS o2
ON o1.month = o2.month

```

The query results are shown in a table with the following data:

Row	percentage_change%
1	136.98

Below the results table, there is a 'Job history' section with a 'REFRESH' button.

Insights:

- There is a **136.98% increase** in sales **from 2017 to 2018**.
- We have taken the sales **data only from Jan to Aug** for both the years.
- There is an **increasing trend**.

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

Query:

SELECT

```
c.customer_state AS State,
ROUND(SUM(p.payment_value)) AS Total_sales,
ROUND(AVG(p.payment_value)) AS Avg_sales
FROM `Target.orders` AS o
JOIN `Target.payments` AS p
ON o.order_id = p.order_id
JOIN `Target.customers` AS c
ON o.customer_id = c.customer_id
GROUP BY c.customer_state
ORDER BY Total_sales DESC
```

Output screenshot:

The screenshot shows the Google Cloud BigQuery interface. The query editor displays the following SQL query:

```
SELECT
  c.customer_state AS State,
  ROUND(SUM(p.payment_value)) AS Total_sales,
  ROUND(AVG(p.payment_value)) AS Avg_sales
FROM `Target.orders` AS o
JOIN `Target.payments` AS p
ON o.order_id = p.order_id
JOIN `Target.customers` AS c
ON o.customer_id = c.customer_id
GROUP BY c.customer_state
ORDER BY Total_sales DESC
```

The query results are displayed in a table with the following columns: Row, State, Total_sales, and Avg_sales. The results are sorted by Total_sales in descending order.

Row	State	Total_sales	Avg_sales
1	SP	5998227.0	138.0
2	RJ	2144380.0	159.0
3	MG	1872257.0	155.0
4	RS	890899.0	157.0
5	PR	811156.0	154.0
6	SC	623086.0	166.0
7	BA	616646.0	171.0
8	DF	355141.0	161.0
9	GO	350092.0	166.0
10	ES	325968.0	155.0

The interface also shows a sidebar with project information, a search bar, and various tool icons. The bottom of the screen displays the job history and a refresh button.

Insights:

- **"SP"** has the **highest total sales** but **lowest average sales**. It means the number of customers are higher but they spend a low amount on their purchase.
- **"RR"** has the **lowest total sales**.
- **"PB"** has the **highest average sales**.

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

Query:

```
SELECT
  c.customer_state AS State,
  ROUND(SUM(oi.freight_value)) AS Total_freight,
  ROUND(AVG(oi.freight_value)) AS Avg_freight
FROM `Target.orders` AS o
JOIN `Target.order_items` AS oi
ON o.order_id = oi.order_id
JOIN `Target.customers` AS c
ON o.customer_id = c.customer_id
GROUP BY c.customer_state
ORDER BY Total_freight DESC
```

Output screenshot:

The screenshot displays the Google Cloud BigQuery console. The query editor shows the SQL query for calculating total and average freight by state. The query results are displayed in a table with columns: Row, State, Total_freight, and Avg_freight. The results are sorted by Total_freight in descending order.

Row	State	Total_freight	Avg_freight
1	SP	718723.0	15.0
2	RJ	305589.0	21.0
3	MG	270853.0	21.0
4	RS	135523.0	22.0
5	PR	117852.0	21.0
6	BA	100157.0	26.0
7	SC	89660.0	21.0
8	PE	59450.0	33.0
9	GO	53115.0	23.0
10	DF	50625.0	21.0

Insights:

- "SP" has the highest total freight and lowest average freight
- "RR" has the lowest total freight.
- "PB" has the highest average freight.

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

A. 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.

Query:

```
SELECT
  order_id,
  DATE_DIFF(order_delivered_customer_date, order_purchase_timestamp, DAY) AS time_to_deliver,
  DATE_DIFF(order_estimated_delivery_date, order_delivered_customer_date, DAY) AS diff_estimated_delivery
FROM `Target.orders`
```

Output screenshot:

The screenshot shows the Google Cloud BigQuery console. The query editor displays the following SQL query:

```
SELECT
  order_id,
  DATE_DIFF(order_delivered_customer_date, order_purchase_timestamp, DAY) AS time_to_deliver,
  DATE_DIFF(order_estimated_delivery_date, order_delivered_customer_date, DAY) AS diff_estimated_delivery
FROM `Target.orders`
```

The query results are displayed in a table with the following columns: Row, order_id, time_to_deliver, and diff_estimated_delivery. The results show 10 rows of data.

Row	order_id	time_to_deliver	diff_estimated_delivery
1	1950d777989f6a877539f5379...	30	-12
2	2c45c33d2f9cb8ff8b1c86cc28...	30	28
3	65d1e22edfaeb8cdc42f66542...	35	16
4	635c894d068ac37efe03dc54e...	30	1
5	3b97562c3aee8bdecb5c2e45...	32	0
6	68f47f50f04c4cb6774570cfe...	29	1
7	276e9ec344d3bf029ff83a161c...	43	-4
8	54e1a3c2b977b0809da548a59...	40	-4
9	fd04fa4105ee8045f6a0139ca5...	37	-1
10	302bb8109d097a9fc6e9cfc5...	33	-5

Insights:

- Maximum delivery time is 209 days.
- Minimum delivery day is the same day.
- Maximum delay in delivery over estimation is 188 days. It shows the inefficient logistics.

- Since there is too much of a gap between estimated date and actual delivery date, it means it does not give estimated date accurately and so on customers will not rely on their estimated delivery date.

Recommendation:

- If there are significant delays between the estimated and actual delivery dates for certain orders, it may indicate potential bottlenecks in the company's supply chain or logistics operations. Identifying these bottlenecks can help company to streamline processes and improve overall efficiency.

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

Query:

```
SELECT
  x.State AS High_State,
  x.Avg_freight AS High_Avg_Freight,
  y.State AS Low_State,
  y.Avg_freight AS Low_Avg_Freight
FROM
  (SELECT
    *,
    DENSE_RANK() OVER(ORDER BY Avg_freight DESC) AS rank_
  FROM
    (SELECT
      c.customer_state AS State,
      ROUND(AVG(oi.freight_value),2) AS Avg_freight,
    FROM `Target.orders` AS o
    JOIN `Target.order_items` AS oi
    ON o.order_id = oi.order_id
    JOIN `Target.customers` AS c
    ON o.customer_id = c.customer_id
    GROUP BY c.customer_state
    ORDER BY Avg_freight DESC) AS n
  ORDER BY rank_
  LIMIT 5) AS x
```



```

JOIN
(SELECT
  *,
  DENSE_RANK() OVER(ORDER BY Avg_freight ASC) AS rank_
FROM
(SELECT
  c.customer_state AS State,
  ROUND(AVG(oi.freight_value),2) AS Avg_freight,
FROM `Target.orders` AS o
JOIN `Target.order_items` AS oi
ON o.order_id = oi.order_id
JOIN `Target.customers` AS c
ON o.customer_id = c.customer_id
GROUP BY c.customer_state
ORDER BY Avg_freight ASC) AS n
ORDER BY rank_
LIMIT 5) AS y
ON x.rank_ = y.rank_

```

Output screenshot:

The screenshot displays the Google Cloud BigQuery console. The top navigation bar includes the Google Cloud logo, a 'TARGET' dropdown, and a search bar. The left sidebar shows the 'Explorer' view with a search bar and a list of resources, including a folder named '2024-03-12 16:50:37'. The main panel shows the query results for the query executed at 2024-03-12 16:50:37. The query is completed, and the results are displayed in a table with 5 rows and 5 columns. The columns are labeled 'High_State', 'High_Avg_Freight', 'Low_State', and 'Low_Avg_Freight'. The table also includes a 'Row' column for indexing.

Row	High_State	High_Avg_Freight	Low_State	Low_Avg_Freight
1	RR	42.98	SP	15.15
2	PB	42.72	PR	20.53
3	RO	41.07	MG	20.63
4	AC	40.07	RJ	20.96
5	PI	39.15	DF	21.04

The bottom of the interface shows a 'Job history' section with a 'REFRESH' button.

Insights:

- "RR", "PB", "RO", "AC" and "PI" are states with higher average freight. In these states shipping costs are generally higher. These States may face challenges in terms of accessibility or transportation infrastructure, leading to higher shipping costs.
- "SP", "PR", "MG", "RJ" and "DF" are states with lower average freight. In these states shipping costs are generally lower. These states may have better access to transportation routes or logistics hubs.

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

Query:

```
SELECT
    State,
    Avg_freight
FROM
    (SELECT
        *
    FROM
        (SELECT
            *,
            DENSE_RANK() OVER(ORDER BY Avg_freight DESC) AS rank_
        FROM
            (SELECT
                c.customer_state AS State,
                ROUND(AVG(oi.freight_value),2) AS Avg_freight,
            FROM `Target.orders` AS o
            JOIN `Target.order_items` AS oi
            ON o.order_id = oi.order_id
            JOIN `Target.customers` AS c
            ON o.customer_id = c.customer_id
            GROUP BY c.customer_state
            ORDER BY Avg_freight DESC) AS n
```

```

ORDER BY rank_
LIMIT 5)
UNION ALL
(SELECT
    *,
    DENSE_RANK() OVER(ORDER BY Avg_freight ASC) AS rank_
FROM
(SELECT
    c.customer_state AS State,
    ROUND(AVG(oi.freight_value),2) AS Avg_freight,
FROM `Target.orders` AS o
JOIN `Target.order_items` AS oi
ON o.order_id = oi.order_id
JOIN `Target.customers` AS c
ON o.customer_id = c.customer_id
GROUP BY c.customer_state
ORDER BY Avg_freight ASC) AS m
ORDER BY rank_
LIMIT 5)) AS k
ORDER BY k.Avg_freight

```

Output screenshot:

The screenshot displays the Google Cloud BigQuery console. The top navigation bar includes the Google Cloud logo, a dropdown menu for 'TARGET', a search bar, and various utility icons. The left sidebar contains the 'Explorer' panel with a search bar and a list of resources, including a folder for '2024-03-12 16:50:37' and a 'SUMMARY' section. The main panel shows a query execution interface with a query editor, a 'RUN' button, and a 'Query results' section. The query results are displayed in a table with columns 'Row', 'State', and 'Avg_freight'. The table contains 10 rows of data, sorted by 'Avg_freight' in ascending order. The bottom of the interface shows a 'Job history' section with a 'REFRESH' button.

Row	State	Avg_freight
1	SP	15.15
2	PR	20.53
3	MG	20.63
4	RJ	20.96
5	DF	21.04
6	PI	39.15
7	AC	40.07
8	RO	41.07
9	PB	42.72
10	RR	42.98

Insights:

- "RR", "PB", "RO","AC" and "PI" are states with higher average delivery time. Analysing average delivery times across different states can help identify areas where customers may be less satisfied due to longer delivery times. Addressing these issues could lead to improved customer retention and loyalty.
- "SP", "PR", "MG", "RJ" and "DF" are states with lowest average delivery time. Identifying the factors contributing to these shorter delivery times could provide insights into best practices that could be applied to other regions to improve overall efficiency.

D. 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.

Query:

```
SELECT
    c.customer_state,

ROUND(AVG(DATE_DIFF(o.order_estimated_delivery_date,o.order_delivered_customer_date,DAY)),2) AS Avg_fast_delivery
FROM `Target.orders` AS o
JOIN `Target.customers` AS c
ON o.customer_id = c.customer_id
GROUP BY c.customer_state
ORDER BY Avg_fast_delivery DESC
LIMIT 5
```

Output screenshot:

The screenshot displays the Google Cloud BigQuery console. The top navigation bar includes the Google Cloud logo, a project selector set to 'TARGET', and a search bar. The left sidebar shows the 'Explorer' view with a search bar and a list of resources, including a project named 'target-414911' with a saved query from '2024-03-12 16:50:37'. The main panel shows the SQL query editor with the following query:

```

270 SELECT
271   c.customer_state,
272   ROUND(AVG(DATE_DIFF(o.order_estimated_delivery_date,o.order_delivered_customer_date,DAY)),2) AS Avg_fast_delivery
273 FROM `Target.orders` AS o
274 JOIN `Target.customers` AS c
275 ON o.customer_id = c.customer_id
276 GROUP BY c.customer_state
277 ORDER BY Avg_fast_delivery DESC
278 LIMIT 5
  
```

Below the query editor, the 'Query results' section is visible, showing a table with 5 rows of data. The table has columns for 'customer_state' and 'Avg_fast_delivery'.

Row	customer_state	Avg_fast_delivery
1	AC	19.76
2	RO	19.13
3	AP	18.73
4	AM	18.61
5	RR	16.41

The bottom of the interface shows the 'Job history' section with a 'REFRESH' button.

Insights:

- "AC", "RO", "AP", "AM" and "RR" are the states where the order delivery is really fast as compared to the estimated date of delivery.
- These are the states where the actual delivery time is consistently faster than the estimated time indicating efficient logistical operations or better-than-expected delivery performance.
- The query provides a benchmark for comparing delivery performance across different states. This allows businesses to set targets and track improvements over time, fostering continuous optimization of delivery operations.

VI. Analysis based on the payments:

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

Query:

SELECT

```
EXTRACT(YEAR FROM order_purchase_timestamp) AS YEAR,
EXTRACT(MONTH FROM order_purchase_timestamp) AS Month,
p.payment_type,
COUNT(DISTINCT o.order_id) AS No_of_order
FROM `Target.orders` AS o
JOIN `Target.payments` AS p
ON o.order_id = p.order_id
GROUP BY YEAR, MONTH, p.payment_type
ORDER BY YEAR, MONTH
```

Output screenshot:

The screenshot displays the Google Cloud BigQuery console. The query editor on the right contains the following SQL query:

```
SELECT
  EXTRACT(YEAR FROM order_purchase_timestamp) AS YEAR,
  EXTRACT(MONTH FROM order_purchase_timestamp) AS Month,
  p.payment_type,
  COUNT(DISTINCT o.order_id) AS No_of_order
FROM `Target.orders` AS o
JOIN `Target.payments` AS p
ON o.order_id = p.order_id
GROUP BY YEAR, MONTH, p.payment_type
ORDER BY YEAR, MONTH
```

The query results are displayed in a table with the following columns: Row, YEAR, Month, payment_type, and No_of_order. The results show the number of orders for each payment type across different years and months.

Row	YEAR	Month	payment_type	No_of_order
1	2016	9	credit_card	3
2	2016	10	credit_card	253
3	2016	10	UPI	63
4	2016	10	voucher	11
5	2016	10	debit_card	2
6	2016	12	credit_card	1
7	2017	1	credit_card	582
8	2017	1	UPI	197
9	2017	1	voucher	33
10	2017	1	debit_card	9

The interface also shows the Explorer panel on the left with the project 'target-414911' and the query '2024-03-12 16:50:37' selected. The bottom of the screen shows the 'Job history' section with a 'REFRESH' button.

Insights:

- **Mostly** customers are using **"Debit card"** and **"Credit card"**.

Recommendation:

- **Company** **should give purchase through "Debit card" and "Credit card" options on all the products.**

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

Query:

SELECT

*

FROM

(SELECT

payment_installments,

COUNT(DISTINCT order_id) AS count_of_order

FROM `Target.payments`

GROUP BY payment_installments) AS n

WHERE n.payment_installments > 0

Output screenshot:

The screenshot displays the Google Cloud BigQuery console. On the left, the Explorer pane shows the project structure with a query named '2024-03-12 16:50:37' selected. The main editor shows the following SQL query:

```

SELECT
  *
FROM (
  SELECT
    payment_installments,
    COUNT(DISTINCT order_id) AS count_of_order
  FROM `Target.payments`
  GROUP BY payment_installments) AS n
WHERE n.payment_installments > 0
  
```

Below the query editor, the 'Query results' section is visible, showing a table with two columns: 'payment_installment' and 'count_of_order'. The results are as follows:

Row	payment_installment	count_of_order
1	1	49060
2	2	12389
3	3	10443
4	4	7088
5	5	5234
6	6	3916
7	7	1623
8	8	4253
9	9	644
10	10	5315
11	11	23

The interface also includes a search bar at the top, a sidebar with navigation options, and a bottom status bar indicating 'Results per page: 50' and '1 - 23 of 23'.



Insights:

- There are 49,060 orders who have paid their 1st installments but when the number of installments are increasing the number of orders who are paying there installments are decreasing.