TARGET BUSINESS CASESTUDY

About Target:

- 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.

Brief About the Dataset:

- This dataset 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.

Problem Statement:

Analyze the dataset and find valuable insights into Target's operations in Brazil and provide Actionable Insights & Recommendations.

Objective:

- Our objective is to analyze this dataset on various aspects of the business, such as order processing, pricing strategies, payment and shipping efficiency, customer demographics, product characteristics, and customer satisfaction levels.
- We need to provide the Actionable Insights & Recommendations based on our Analysis.

TARGET

Details about the Dataset:

The data is available in 8 csv files:

1. customers.csv
2. sellers.csv
3. order_items.csv
4. geolocation.csv
5. payments.csv
6. reviews.csv
7. orders.csv
8. products.csv

The column description for these csv files is given below.

The **customers.csv** contain following features:

- **customer_id** ID of the consumer who made the purchase
- **customer_unique_id** Unique ID of the consumer
- **customer_zip_code_prefix** Zip Code of consumer's location
- **customer_city** Name of the City from where order is made
- **customer_state** State Code from where order is made (Eg. são paulo - SP)

The **sellers.csv** contains following features:

- **seller_id** Unique ID of the seller registered
- **seller_zip_code_prefix** Zip Code of the seller's location
- **seller_city** Name of the City of the seller
- **seller_state** State Code (Eg. são paulo - SP)

The **order_items.csv** contain following features:

- **order_id** A Unique ID of order made by the consumers
- **order_item_id** A Unique ID given to each item ordered in the order
- **product_id** A Unique ID given to each product available on the site
- **seller_id** Unique ID of the seller registered in Target
- **shipping_limit_date** The date before which the ordered product must be shipped
- **price** Actual price of the products ordered
- **freight_value** Price rate at which a product is delivered from one point to another

The **geolocations.csv** contain following features:

- **geolocation_zip_code_prefix** First 5 digits of Zip Code
- **geolocation_lat** Latitude
- **geolocation_lng** Longitude
- **geolocation_city** City
- **geolocation_state** State

The **payments.csv** contain following features:

- **order_id** A Unique ID of order made by the consumers
- **payment_sequential** Sequences of the payments made in case of EMI
- **payment_type** Mode of payment used (Eg. Credit Card)
- **payment_installments** Number of instalments in case of EMI purchase
- **payment_value** Total amount paid for the purchase order

The **orders.csv** contain following features:

- **order_id** — A Unique ID of order made by the consumers
- **customer_id** — ID of the consumer who made the purchase
- **order_status** — Status of the order made i.e. delivered, shipped, etc.
- **order_purchase_timestamp** — Timestamp of the purchase
- **order_delivered_carrier_date** — Delivery date at which carrier made the delivery
- **order_delivered_customer_date** — Date at which customer got the product
- **order_estimated_delivery_date** — Estimated delivery date of the products

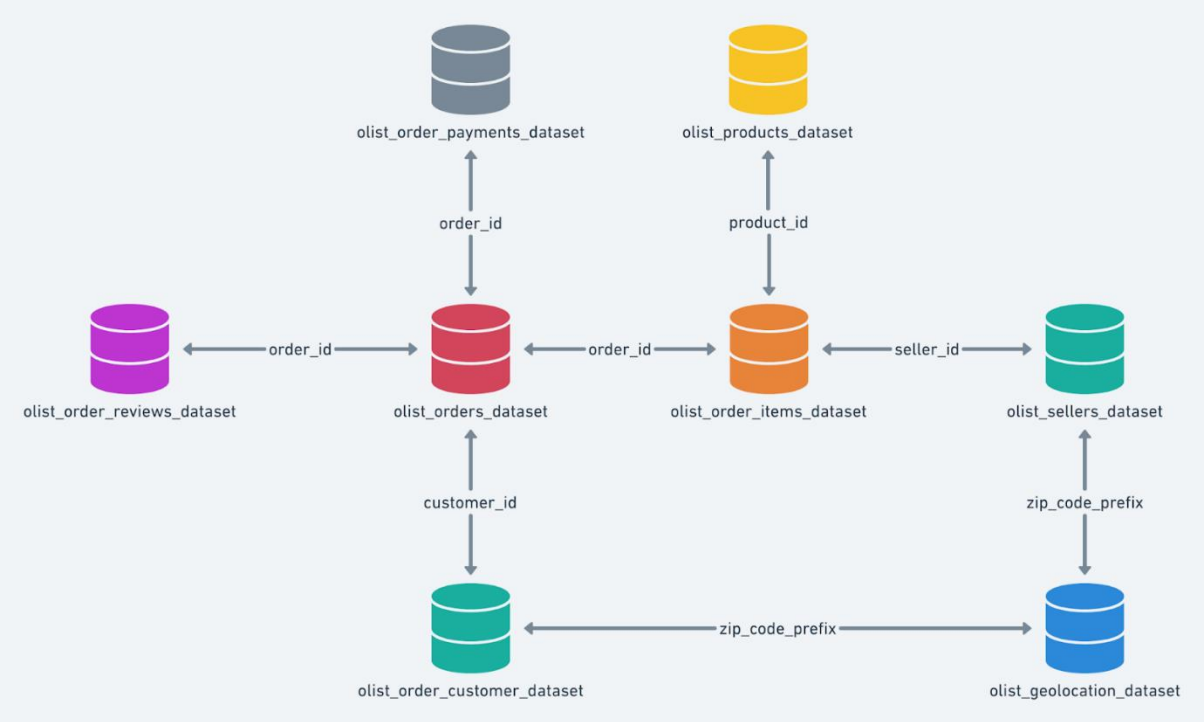
The **reviews.csv** contain following features:

- **review_id** — ID of the review given on the product ordered by the order id
- **order_id** — A Unique ID of order made by the consumers
- **review_score** — Review score given by the customer for each order on a scale of 1-5
- **review_comment_title** — Title of the review
- **review_comment_message** — Review comments posted by the consumer for each order
- **review_creation_date** — Timestamp of the review when it is created
- **review_answer_timestamp** — Timestamp of the review answered

The **products.csv** contain following features:

- **product_id** — A Unique identifier for the proposed project.
- **product_category_name** — Name of the product category
- **product_description_lenght** — Length of the description written for each product ordered on the site
- **product_photos_qty** — Number of photos of each product ordered available on the shopping portal
- **product_weight_g** — Weight of the products ordered in grams
- **product_length_cm** — Length of the products ordered in centimeters
- **product_height_cm** — Height of the products ordered in centimeters
- **product_width_cm** — Width of the product ordered in centimeters

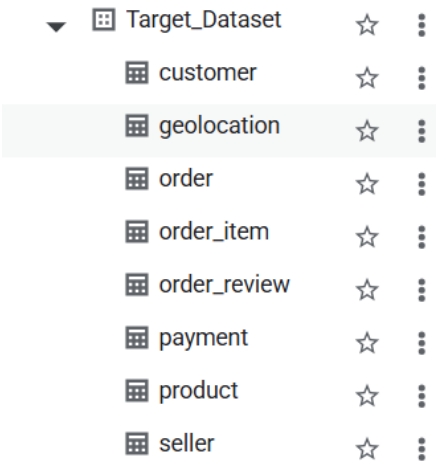
Database Schema:



We will be using BigQuery to Analyze this Dataset

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

We have created a new Dataset called **"Target_Dataset"** and uploaded all the 8 CSV files in this dataset and named according.



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

Now let us check the number of Datatype of each column, number of rows and columns of each table.

We will be using below query to check the datatype of each column of a table

- Let us find the information for "customer" table

Details about the function and clause used in below query

information_schema.columns - The Information Schema COLUMNS table provides information about columns in each table on the server. It contains the following columns: Column. Description. TABLE_CATALOG.

*In a SQL statement, the **WHERE clause** specifies criteria that field values must meet for the records that contain the values to be included in the query results.*

Query:

```
SELECT column_name, data_type
FROM dsml-sql-454708.Target_Dataset.INFORMATION_SCHEMA.COLUMNS
WHERE table_name = 'customer';
```

Output:

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

Now let us check the total number of records present this table "customer"

Details about the function used in below query

COUNT(*) counts all rows in a table, including rows with NULL values.
COUNT(column_name) counts only the rows where the specified column is not NULL.

Query:

```
SELECT COUNT(*) AS Total_Records FROM `dsml-sql-454708.Target_Dataset.customer`;
```

Output:

Row	Total_Records
1	99441

- Let us find the information for "**geolocation**" table

Query:

```
SELECT column_name, data_type
FROM dsml-sql-454708.Target_Dataset.INFORMATION_SCHEMA.COLUMNS
WHERE table_name = 'geolocation';
```

Output:

Row	column_name	data_type
1	geolocation_zip_code_prefix	INT64
2	geolocation_lat	FLOAT64
3	geolocation_lng	FLOAT64
4	geolocation_city	STRING
5	geolocation_state	STRING

Now let us check the total number of records present this table “geolocation”

Query:

```
SELECT COUNT(*) AS Total_Records FROM `dsml-sql-454708.Target_Dataset.geolocation`;
```

Output:

Row	Total_Records
1	1000163

- Let us find the information for “**order**” table

Query:

```
SELECT column_name, data_type
FROM dsml-sql-454708.Target_Dataset.INFORMATION_SCHEMA.COLUMNS
WHERE table_name = 'order';
```

Output:

Row	column_name	data_type
1	order_id	STRING
2	customer_id	STRING
3	order_status	STRING
4	order_purchase_timestamp	TIMESTAMP
5	order_approved_at	TIMESTAMP
6	order_delivered_carrier_date	TIMESTAMP
7	order_delivered_customer_date	TIMESTAMP
8	order_estimated_delivery_date	TIMESTAMP

Now let us check the total number of records present this table “order”

Query:

```
SELECT COUNT(*) AS Total_Records FROM `dsml-sql-454708.Target_Dataset.order`;
```

Output:

Row	Total_Records
1	99441

- Let us find the information for "**order_item**" table

Query:

```
SELECT column_name, data_type
FROM dsml-sql-454708.Target_Dataset.INFORMATION_SCHEMA.COLUMNS
WHERE table_name = 'order_item';
```

Output:

Row	column_name	data_type
1	order_id	STRING
2	order_item_id	INT64
3	product_id	STRING
4	seller_id	STRING
5	shipping_limit_date	TIMESTAMP
6	price	FLOAT64
7	freight_value	FLOAT64

Now let us check the total number of records present this table "order_item"

Query:

```
SELECT COUNT(*) AS Total_Records FROM `dsml-sql-454708.Target_Dataset.order_item`;
```

Output:

Row	Total_Records
1	112650

- Let us find the information for "**order_review**" table

Query:

```
SELECT column_name, data_type
FROM dsml-sql-454708.Target_Dataset.INFORMATION_SCHEMA.COLUMNS
WHERE table_name = 'order_review';
```

Output:

Row	column_name	data_type
1	review_id	STRING
2	order_id	STRING
3	review_score	INT64
4	review_comment_title	STRING
5	review_creation_date	TIMESTAMP
6	review_answer_timestamp	TIMESTAMP

Now let us check the total number of records present this table “order_review”

Query:

```
SELECT COUNT(*) AS Total_Records FROM `dsml-sql-
454708.Target_Dataset.order_review`;
```

Output:

Row	Total_Records
1	99224

- Let us find the information for “**payment**” table

Query:

```
SELECT column_name, data_type
FROM dsml-sql-454708.Target_Dataset.INFORMATION_SCHEMA.COLUMNS
WHERE table_name = 'payment';
```

Output:

Row	column_name	data_type
1	order_id	STRING
2	payment_sequential	INT64
3	payment_type	STRING
4	payment_installments	INT64
5	payment_value	FLOAT64

Now let us check the total number of records present this table “payment”

Query:

```
SELECT COUNT(*) AS Total_Records FROM `dsml-sql-454708.Target_Dataset.payment`;
```

Output:

Row	Total_Records
1	103886

- Let us find the information for “**product**” table

Query:

```
SELECT column_name, data_type
FROM dsml-sql-454708.Target_Dataset.INFORMATION_SCHEMA.COLUMNS
WHERE table_name = 'product';
```

Output:

Row	column_name	data_type
1	product_id	STRING
2	product category	STRING
3	product_name_length	INT64
4	product_description_length	INT64
5	product_photos_qty	INT64
6	product_weight_g	INT64
7	product_length_cm	INT64
8	product_height_cm	INT64
9	product_width_cm	INT64

Now let us check the total number of records present this table “product”

Query:

```
SELECT COUNT(*) AS Total_Records FROM `dsml-sql-454708.Target_Dataset.product`;
```

Output:

Row	Total_Records
1	32951

- Let us find the information for “**seller**” table

Query:

```
SELECT column_name, data_type
FROM dsml-sql-454708.Target_Dataset.INFORMATION_SCHEMA.COLUMNS
WHERE table_name = 'seller';
```

Output:

Row	column_name	data_type
1	seller_id	STRING
2	seller_zip_code_prefix	INT64
3	seller_city	STRING
4	seller_state	STRING

Now let us check the total number of records present this table “seller”

Query:

```
SELECT COUNT(*) AS Total_Records FROM `dsml-sql-454708.Target_Dataset.seller`;
```

Output:

Row	Total_Records
1	3095

Insights:

- Table **customer** consists of 5 columns where customer_id, customer_unique_id, customer_city, customer_state are of type **STRING** and customer_zip_code_prefix is of type **INTEGER**. There are **99441** records present on this table.
- Table **geoloaction** consists of 5 columns where geolocation_city, geolocation_state are of type **STRING** whereas geolocation_zip_code_prefix is of type **INTEGER** and geolocation_lat, geolocation_lng are of type **FLOAT**. There are **99441** records present on this table.
- Table **order** consists of 8 columns where order_id, customer_id, order_status are of type **STRING** and order_purchase_timestamp, order_approved_at, order_delivered_carrier_date, order_delivered_customer_date, order_estimated_delivery_date are of type **TIMESTAMP**. There are **99441** records present on this table.
- Table **order_item** consists of 7 columns where order_id, product_id, seller_id are of type **STRING** whereas order_item_id is of type **INTEGER** review_creation_date, shipping_limit_date is of type **TIMESTAMP** and price, freight_value are of type **FLOAT**. There are **112650** records present on this table.
- Table **order_review** consists of 6 columns where review_id, order_id, and review_comment_title are of type **STRING** whereas review_score is of type **INTEGER** and review_creation_date, review_answer_timestamp is of type **TIMESTAMP**. There are **99224** records present on this table.
- Table **payment** consists of 5 columns where order_id, payment_type are of type **STRING** whereas payment_sequential, payment_installments are of type **INTEGER** and payment_value is of type **FLOAT**. There are 103886 records present on this table.
- Table **product** consists of 9 columns where product_id, product category are of type **STRING** and product_name_length, product_description_length, product_photos_qty, product_weight_g, product_length_cm, product_height_cm, product_width_cm are of type **INTEGER**. There are **32915** records present on this table.
- Table **seller** consists of 4 columns where seller_id, seller_city , seller_state are of type **STRING** and seller_zip_code_prefix is of type **INTEGER**. There are **3095** records present in this table.

Details about the function used in below query

IFNULL() *Return the specified value IF the expression is NULL, otherwise return the expression.*

IS NULL *is a logical operator in SQL that allows you to exclude rows with missing data from your results. Some tables contain null values - cells with no data in them at all.*

UNION ALL *Operator is used to combine the results of two or more SELECT statements into a single result set. Unlike the* ***UNION*** *operator, which eliminates duplicate records and UNION ALL includes all duplicates. This makes UNION ALL it faster and more efficient when we don't need to remove duplicates.*

Let us check how many **NULL** values are present for orders table in each column.

```
SELECT "order_id" AS column_name, IFNULL(COUNT(*),0) AS null_count
FROM `dsml-sql-454708.Target_Dataset.order`
WHERE order_id IS NULL
UNION ALL
SELECT "customer_id" AS column_name, IFNULL(COUNT(*),0) AS null_count
FROM `dsml-sql-454708.Target_Dataset.order`
WHERE customer_id IS NULL
UNION ALL
SELECT "order_status" AS column_name, IFNULL(COUNT(*),0) AS null_count
FROM `dsml-sql-454708.Target_Dataset.order`
WHERE order_status IS NULL
UNION ALL
SELECT "order_purchase_timestamp" AS column_name, IFNULL(COUNT(*),0) AS null_count
FROM `dsml-sql-454708.Target_Dataset.order`
WHERE order_purchase_timestamp IS NULL
UNION ALL
SELECT "order_approved_at" AS column_name, IFNULL(COUNT(*),0) AS null_count
FROM `dsml-sql-454708.Target_Dataset.order`
WHERE order_approved_at IS NULL
UNION ALL
SELECT "order_delivered_carrier_date" AS column_name, IFNULL(COUNT(*),0) AS
null_count
FROM `dsml-sql-454708.Target_Dataset.order`
WHERE order_delivered_carrier_date IS NULL
UNION ALL
SELECT "order_delivered_customer_date" AS column_name, IFNULL(COUNT(*),0) AS
null_count
FROM `dsml-sql-454708.Target_Dataset.order`
WHERE order_delivered_customer_date IS NULL
UNION ALL
SELECT "order_estimated_delivery_date" AS column_name, IFNULL(COUNT(*),0) AS
null_count
FROM `dsml-sql-454708.Target_Dataset.order`
WHERE order_estimated_delivery_date IS NULL
```

Output:

Row	column_name	null_count
1	order_id	0
2	customer_id	0
3	order_approved_at	160
4	order_purchase_timestamp	0
5	order_delivered_customer_date	2965
6	order_delivered_carrier_date	1783
7	order_estimated_delivery_date	0
8	order_status	0

Observations:

- We can see there are NULL values present the column order_approved_at, order_delivered_customer_date and order_delivered_carrier_date while the other attributes don't have NULL values.
- We can use IFNULL(column_name,value/expression) function to fill the NULL values present the table.

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

Details about the function used in below query

*The **MIN()** function returns the smallest value of the selected column.*

*The **MAX()** function returns the largest value of the selected column.*

***FORMAT_DATETIME()**: This is the function used in BigQuery to format a datetime value according to a specified format string. **format_string**: This is a parameter of the FORMAT_DATETIME function. It is a string literal that specifies the format in which the datetime value should be displayed. `"%H-%M-%S"` is used to display the hour-minutes-seconds in 24-hour format.*

Let us check the timespan of the order table

Query:

```
SELECT
MIN(order_purchase_timestamp) AS min_date,
MAX(order_purchase_timestamp) AS max_date
FROM `dsml-sql-454708.Target_Dataset.order`
```

Output:

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

Query:

```
SELECT
MIN(FORMAT_DATETIME("%H-%M-%S",order_purchase_timestamp)) AS min_time,
MAX(FORMAT_DATETIME("%H-%M-%S",order_purchase_timestamp)) AS max_time
FROM `dsml-sql-454708.Target_Dataset.order`
```

Output:

Row	min_time	max_time
1	00-00-00	23-59-59

- **Order table has the records from 2016-09-04 21:15:19 till 2018-10-17 17:30:18.**
- From the above output we can conclude that orders were made throughout the day.

Additionally let us also split the day into 6 parts as follows and check at what part of the day most of the orders are made:

- 12 to 4 AM -> Midnight
- 4 to 8 AM -> Early morning
- 8 to 12 PM -> Morning
- 12 to 4 PM -> Afternoon
- 4 to 8 PM -> Evening
- 8 to 12 AM -> Night

Details about the expression used in below query

*A **Common Table Expression (CTE)** in SQL is a temporary result set that can be referenced within a SELECT, INSERT, UPDATE, or DELETE statement. CTEs are defined using the WITH keyword and allow you to create a named, reusable subquery within your SQL statement.*

*The **CASE** expression goes through conditions and returns a value when the first condition is met (**like an if-then-else statement**). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause. If there is no ELSE part and no conditions are true, it returns NULL.*

*The **GROUP BY** statement groups rows that have the same values into summary rows, like "find the number of customers in each country". The GROUP BY statement is often used with aggregate functions **(COUNT(), MAX(), MIN(), SUM(), AVG())** to group the result-set by one or more columns.*

*The **ORDER BY** keyword is used to sort the result-set in ascending or descending order.*

Query:

```
WITH split_timestamp AS (
  SELECT customer_id,
  FORMAT_DATETIME("%H:%M",order_purchase_timestamp) AS hour_min
  FROM `dsml-sql-454708.Target_Dataset.order`
  WHERE order_status = "delivered"
),

catagory AS (SELECT *,
CASE
  WHEN hour_min BETWEEN "00:00" AND "03:59" THEN "midnight 12-4 AM"
  WHEN hour_min BETWEEN "04:00" AND "07:59" THEN "early morning 4-8 AM"
  WHEN hour_min BETWEEN "08:00" AND "11:59" THEN "morning 8-12 PM"
  WHEN hour_min BETWEEN "12:00" AND "15:59" THEN "afternoon 12-16 PM"
  WHEN hour_min BETWEEN "16:00" AND "19:59" THEN "evening 16-20 PM"
  ELSE "night 20-12 AM"
END AS catagory_type
FROM split_timestamp)

SELECT catagory_type,
COUNT(catagory_type) AS order_count
FROM catagory
GROUP BY catagory_type
ORDER BY COUNT(catagory_type) DESC
```

Output:

Row	catagory_type	order_count
1	afternoon 12-16 PM	24744
2	evening 16-20 PM	23825
3	night 20-12 AM	21721
4	morning 8-12 PM	19917
5	midnight 12-4 AM	4209
6	early morning 4-8 AM	2062

Insights:

- Most of the orders are placed between 12 noon to 4 PM.
- Between 4 PM till midnight there are also decent number of orders.
- As expected least number of orders are placed between 12 AM till 8 AM as customers are not active that time.

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

Details about the expression used in below query

*A **JOIN clause** is used to combine rows from two or more tables, based on a related column between them. The **LEFT JOIN** keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records from the right side, if there is no match.*

City wise count

Query:

```
WITH city_state AS (SELECT c.customer_id, customer_city, customer_state,
  FROM `dsml-sql-454708.Target_Dataset.order` o
  LEFT JOIN `dsml-sql-454708.Target_Dataset.customer` c ON o.customer_id =
c.customer_id
  WHERE order_status = "delivered")

  SELECT customer_city,
  COUNT(customer_city) AS order_count
  FROM city_state
  GROUP BY customer_city
  ORDER BY order_count DESC
```

Output:

Row	customer_city	order_count
1	sao paulo	15045
2	rio de janeiro	6601
3	belo horizonte	2697
4	brasilia	2071
5	curitiba	1489
6	campinas	1406
7	porto alegre	1342
8	salvador	1188
9	guarulhos	1144
10	sao bernardo do campo	911

State wise count

Query:

```
WITH city_state AS (SELECT c.customer_id, customer_city, customer_state,
  FROM `dsml-sql-454708.Target_Dataset.order` o
  LEFT JOIN `dsml-sql-454708.Target_Dataset.customer` c ON o.customer_id =
c.customer_id
  WHERE order_status = "delivered")

  SELECT customer_state,
  COUNT(customer_state) AS order_count
  FROM city_state
  GROUP BY customer_state
ORDER BY order_count DESC
```

Output:

Row	customer_state	order_count
1	SP	40501
2	RJ	12350
3	MG	11354
4	RS	5345
5	PR	4923
6	SC	3546
7	BA	3256
8	DF	2080
9	ES	1995
10	GO	1957

*Let us take the time period of **12 noon till 4 PM** and do the above Analysis.*

First let us check the city wise order count

Query:

```
WITH split_timestamp AS (
  SELECT c.customer_id, customer_city, customer_state,
  FORMAT_DATETIME("%H:%M",order_purchase_timestamp) AS hour_min
  FROM `dsml-sql-454708.Target_Dataset.order` o
  LEFT JOIN `dsml-sql-454708.Target_Dataset.customer` c ON o.customer_id =
c.customer_id
  WHERE order_status = "delivered"
),

catagory AS (SELECT *,
CASE
  WHEN hour_min BETWEEN "00:00" AND "03:59" THEN "midnight 12-4 AM"
  WHEN hour_min BETWEEN "04:00" AND "07:59" THEN "early morning 4-8 AM"
  WHEN hour_min BETWEEN "08:00" AND "11:59" THEN "morning 8-12 PM"
  WHEN hour_min BETWEEN "12:00" AND "15:59" THEN "afternoon 12-16 PM"
  WHEN hour_min BETWEEN "16:00" AND "19:59" THEN "evening 16-20 PM"
  ELSE "night 20-12 AM"
END AS catagory_type
FROM split_timestamp)

SELECT customer_city,
COUNT(customer_city) AS city_count
FROM catagory
WHERE catagory_type = "afternoon 12-16 PM"
GROUP BY customer_city
ORDER BY COUNT(customer_city) DESC
```

Output:

Row	customer_city	city_count
1	sao paulo	3963
2	rio de janeiro	1744
3	belo horizonte	685
4	brasilia	535
5	curitiba	403
6	campinas	369
7	porto alegre	365
8	guarulhos	287
9	salvador	275
10	sao bernardo do campo	237

Row	customer_city	city_count
2491	capitolio	1
2492	itatim	1
2493	lauro muller	1
2494	vereda	1
2495	dario meira	1
2496	santo antonio do aracangua	1
2497	moncao	1
2498	astorga	1
2499	urucurituba	1
2500	salvador do sul	1

Now let us check the state wise order count

Query:

```
WITH split_timestamp AS (
  SELECT c.customer_id, customer_city, customer_state,
  FORMAT_DATETIME("%H:%M",order_purchase_timestamp) AS hour_min
  FROM `dsml-sql-454708.Target_Dataset.order` o
  LEFT JOIN `dsml-sql-454708.Target_Dataset.customer` c ON o.customer_id =
c.customer_id
  WHERE order_status = "delivered"),
catagory AS (SELECT *,
CASE
  WHEN hour_min BETWEEN "00:00" AND "03:59" THEN "midnight 12-4 AM"
  WHEN hour_min BETWEEN "04:00" AND "07:59" THEN "early morning 4-8 AM"
  WHEN hour_min BETWEEN "08:00" AND "11:59" THEN "morning 8-12 PM"
  WHEN hour_min BETWEEN "12:00" AND "15:59" THEN "afternoon 12-16 PM"
  WHEN hour_min BETWEEN "16:00" AND "19:59" THEN "evening 16-20 PM"
  ELSE "night 20-12 AM"
END AS catagory_type
```

```
FROM split_timestamp)

SELECT customer_state,
COUNT(customer_state) AS state_count
FROM catagory
WHERE catagory_type = "afternoon 12-16 PM"
GROUP BY customer_state
ORDER BY COUNT(customer_state) DESC
```

Output:

Row	customer_state	state_count
1	SP	10571
2	RJ	3147
3	MG	2844
4	RS	1340
5	PR	1281
6	SC	922
7	BA	777
8	DF	537
9	ES	493
10	GO	489

Row	customer_state	state_count
18	PI	114
19	RN	108
20	AL	105
21	TO	84
22	SE	65
23	RO	64
24	AM	41
25	AC	22
26	RR	14
27	AP	12

Insights:

- **Top 3 cities and states leading based on orders count are sao Paulo, rio de Janeiro, belo horizonte and SP, RJ, MG when we considered the entire dataset.**
- Top cities from where most of the orders are placed between the timestamp 12 noon till 4 PM are sao paulo, rio de janeiro and belo horizonte and bottom cities are AC, RR and urucurituba, salvador do sul, crateus.
- Top states from where most of the orders are placed between the timestamp 12 noon till 4 PM are SP, RJ and MG and bottom states are AC, RR and AP.

2. In-depth Exploration:

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

We can see there are various order_status in the order table.

Query:

```
SELECT order_status, COUNT(order_status) AS count
FROM `dsml-sql-454708.Target_Dataset.order`
GROUP BY order_status
order by count desc
```

Output:

Row	order_status	count
1	delivered	96478
2	shipped	1107
3	canceled	625
4	unavailable	609
5	invoiced	314
6	processing	301
7	created	5
8	approved	2

Details about the functions used in below query

*The **ROUND** function is a fundamental tool that adjusts the precision of numerical data in SQL. It rounds values to a specified number of decimal places and simplifies data for analysis*

***LEAD()** and **LAG()** are time-series window functions used to access data from rows that come after, or before the current row within a result set based on a specific column order. LEAD() is the function that lets us peek into the future , and LAG() as a way to glance into the past .*

*The SQL **EXTRACT** function is used to extract specific date and time components (e.g., year, month, day, hour, minute) from a date or timestamp value. It allows you to access and work with individual parts of a date or time for various date and time operations in SQL.*

Let us only consider the orders where status is “**delivered**”

Query:

```
SELECT *,
IFNULL(ROUND((LEAD(total_order,1) OVER (ORDER BY order_year) - total_order)/
(total_order) * 100,2),0) AS percentage_growth
FROM (SELECT order_year,
COUNT(order_year) AS total_order
FROM (SELECT EXTRACT(year from order_purchase_timestamp) AS order_year,
FROM `dsml-sql-454708.Target_Dataset.order`
WHERE order_status = "delivered") t1
GROUP BY order_year) t2
ORDER BY Order_year
```

Output:

Row	order_year	total_order	percentage_growth
1	2016	267	16165.17
2	2017	43428	21.54
3	2018	52783	0.0

Details about the functions used in below query

*The **DATE_DIFF()** function compares two dates and returns the difference. The DATE_DIFF() function is specifically used to measure the difference between two dates in years, months, weeks, and so on.*

Let us check the how many days records are present for each year.

Query:

```
SELECT order_year, MAX(order_purchase_timestamp) AS max_order_date,
MIN(order_purchase_timestamp) AS min_order_date,
DATE_DIFF(MAX(order_purchase_timestamp),MIN(order_purchase_timestamp),day) AS
total_record_days
FROM (
  SELECT *,EXTRACT(year from order_purchase_timestamp) AS order_year,
FROM `dsml-sql-454708.Target_Dataset.order`
)
GROUP BY order_year
ORDER BY order_year
```

Output:

Row	order_year	max_order_date	min_order_date	total_record_days
1	2016	2016-12-23 23:16:47 UTC	2016-09-04 21:15:19 UTC	110
2	2017	2017-12-31 23:29:31 UTC	2017-01-05 11:56:06 UTC	360
3	2018	2018-10-17 17:30:18 UTC	2018-01-01 02:48:41 UTC	289

Insights:

- There is a significant growth in the number of orders from year to year.
- There is **16165%** growth from 2016 to 2017 whereas there is **21.54%** growth from 2017 to 2018.
- There are only 110 days of records available for the year 2016.
- Since there are few records available for 2016 so there is abnormality in the growth seen from the year 2016 to 2017.
- **Overall, we can see there is an increase in the number of orders year by year.**

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

Query:

```
SELECT date_and_month, COUNT(*) AS order_count
FROM (SELECT *, FORMAT_DATE("%Y-%m", order_purchase_timestamp) AS date_and_month
FROM `dsml-sql-454708.Target_Dataset.order`
WHERE order_status = "delivered") t1
GROUP BY date_and_month
ORDER BY order_count DESC
```

Output:

Row	date_and_month	order_count
1	2017-11	7289
2	2018-01	7069
3	2018-03	7003
4	2018-04	6798
5	2018-05	6749
6	2018-02	6555
7	2018-08	6351
8	2018-07	6159
9	2018-06	6099
10	2017-12	5513
11	2017-10	4478

Insights:

- Maximum number of orders placed in November 2017.
- Very few orders were placed in the 1st quarter of 2017.
- Most of the orders are placed in 1st two quarters of 2018 whereas for 2017 the spike is observed in the last 2 quarters.
- **Overall, from the ending of 2017 and the beginning of the 2018 there is maximum number of orders placed.**

2.3 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

For this analysis let us take all type of order_status as we have already done the analysis using the "delivered" order_status category and see if there is difference in output.

Query:

```
WITH split_timestamp AS (
  SELECT c.customer_id, customer_city, customer_state,
  FORMAT_DATETIME("%H:%M",order_purchase_timestamp) AS hour_min
  FROM `dsml-sql-454708.Target_Dataset.order` o
  LEFT JOIN `dsml-sql-454708.Target_Dataset.customer` c ON o.customer_id =
c.customer_id
  WHERE order_status = "delivered"
),

catagory AS (SELECT *,
CASE
  WHEN hour_min BETWEEN "00:00" AND "06:59" THEN "Dawn"
  WHEN hour_min BETWEEN "07:00" AND "12:59" THEN "Mornings"
  WHEN hour_min BETWEEN "13:00" AND "18:59" THEN "Afternoon"
  ELSE "Night"
END AS catagory_type
FROM split_timestamp)

SELECT catagory_type,
COUNT(catagory_type) AS order_count
FROM catagory
GROUP BY catagory_type
ORDER BY COUNT(catagory_type) DESC
```

Output:

Row	catagory_type	order_count
1	Afternoon	36965
2	Night	27522
3	Mornings	26919
4	Dawn	5072

Insights:

- Most of the orders are placed at Afternoon between (13-18 hours).
- Very few numbers of orders are placed at Dawn between (0-6 hours).

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

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

Query:

```
WITH state_month_order AS (SELECT customer_state,
FORMAT_DATETIME("%Y-%m", order_purchase_timestamp) AS year_month
FROM `dsml-sql-454708.Target_Dataset.order` o
LEFT JOIN `dsml-sql-454708.Target_Dataset.customer` c ON o.customer_id =
c.customer_id)

SELECT *,
COUNT(customer_state) AS total_orders
FROM state_month_order
GROUP BY customer_state, year_month
ORDER BY COUNT(customer_state) DESC
```

Output:

Row	customer_state	year_month	total_orders
1	SP	2018-08	3253
2	SP	2018-05	3207
3	SP	2018-04	3059
4	SP	2018-01	3052
5	SP	2018-03	3037
6	SP	2017-11	3012
7	SP	2018-07	2777
8	SP	2018-06	2773
9	SP	2018-02	2703
10	SP	2017-12	2357

Row	customer_state	year_month	total_orders
41	RJ	2017-09	609
42	RJ	2017-07	571
43	RJ	2017-08	562
44	MG	2017-10	560
45	MG	2017-09	507
46	RJ	2017-05	488
47	MG	2017-08	469
48	MG	2017-07	453
49	MG	2017-05	428
50	RS	2017-11	422

Row	customer_state	year_month	total_orders
558	MS	2017-01	1
559	AM	2017-06	1
560	AP	2017-07	1
561	TO	2017-07	1
562	RR	2017-07	1
563	RR	2017-09	1
564	RR	2018-05	1
565	RR	2016-09	1

Let us also find out which state has the greatest number of orders

Details about the functions used in below query

SUM() is a SQL aggregate function. that totals the values in a given column. Unlike COUNT, we can only use SUM on columns containing numerical values.

*In SQL, a **window function** performs calculations across a set of rows related to the current row, without grouping them into a single output row. It's like a sliding window that analyses data in a defined scope or window, allowing for aggregate calculations, ranking, and more, while maintaining the individual rows. Unlike aggregate functions, which summarize data into a single value per group, window functions produce a result for each row, providing detailed row-level analysis.*

Query:

```
WITH state_month_order AS (SELECT customer_state,
FORMAT_DATETIME("%Y-%m", order_purchase_timestamp) AS year_month
FROM `dsml-sql-454708.Target_Dataset.order` o
LEFT JOIN `dsml-sql-454708.Target_Dataset.customer` c ON o.customer_id =
c.customer_id),

total_order_state AS (
  SELECT customer_state,
COUNT(customer_state) AS total_state_order
FROM state_month_order
GROUP BY customer_state
ORDER BY total_state_order DESC)
SELECT *,
ROUND((SUM(total_state_order) OVER (PARTITION BY total_state_order)/
SUM(total_state_order) OVER ())*100,2) AS Total
FROM total_order_state
ORDER BY total_state_order DESC
```

Output:

Row	customer_state	total_state_order	Total
1	SP	41746	41.98
2	RJ	12852	12.92
3	MG	11635	11.7
4	RS	5466	5.5
5	PR	5045	5.07
6	SC	3637	3.66
7	BA	3380	3.4
8	DF	2140	2.15
9	ES	2033	2.04
10	GO	2020	2.03

Insights:

- From SP state most of the orders were placed on monthly basis.
- Almost there is no orders placed from the states AP, RR, TO, MS etc.
- **Almost 42% of the total orders are placed from the state SP followed by RJ 13%.**

3.2 How are the customers distributed across all the states?

From the output we can conclude that:

Insights:

- **SP and RJ state alone contribute 50% of the total order place at Target.**
- **These states might have large population than the other states or there is more Target outlet as compared to other states.**
- AP, RR, TO, MS are very weak contributed to the order counts.

4. Impact on Economy: Analyse 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).
You can use the "payment_value" column in the payments table to get the cost of orders.

Query:

```
WITH order_value AS (
  SELECT o.order_id,
  price,
  FORMAT_DATETIME("%Y-%m",order_purchase_timestamp) AS year_month
  FROM `dsml-sql-454708.Target_Dataset.order` o
```

```
  LEFT JOIN `dsml-sql-454708.Target_Dataset.order_item` oi ON o.order_id =
oi.order_id
  WHERE price IS NULL
)

SELECT * FROM order_value
```

Output:

Row	order_id	price	year_month
1	809a282bbd5dbcabb6f2f724fc...	null	2016-09
2	e5215415bb6f76fe3b7cb6810...	null	2016-10
3	92d731517f17c26f16182d454...	null	2016-10
4	ddaec6fff982b13e7e048b627a...	null	2016-10
5	c549f0d88f33bfe43351b893f1...	null	2016-10
6	9bd5312d12f48b04a460f702b...	null	2016-10
7	032cf17f7a713287e46999fdde...	null	2016-10
8	5226f1a731e8a1215da5e1bea...	null	2016-10
9	7fd4b0e047195ca197c366077...	null	2016-10
10	123e27a1a4d0b2481d8618ac3...	null	2016-10
11	1aecadf4362edaca7fa033e882...	null	2016-10

Results per page: 50 1 – 50 of 775

*So, there are **775 NULL values** in the price column as we have considered all the order_status.*

*So let us only consider the order_status as **"delivered"***

Query:

```
WITH order_value AS (
  SELECT o.*,
  payment_value,
  FORMAT_DATETIME("%Y-%m",order_purchase_timestamp) AS year_month
  FROM `dsml-sql-454708.Target_Dataset.order` o
  LEFT JOIN `dsml-sql-454708.Target_Dataset.payment` oi ON o.order_id = oi.order_id
  WHERE order_status = "delivered"),
sales_for_2017 AS (SELECT
FORMAT_DATETIME("%B", order_purchase_timestamp) AS month,
ROUND(SUM(payment_value),2) AS total_price17
FROM order_value
WHERE year_month BETWEEN "2017-01" AND "2017-08"
GROUP BY FORMAT_DATETIME("%B", order_purchase_timestamp)),

sales_for_2018 AS (SELECT
FORMAT_DATETIME("%B", order_purchase_timestamp) AS month,
ROUND(SUM(payment_value),2) AS total_price18
FROM order_value
WHERE year_month BETWEEN "2018-01" AND "2018-08"
GROUP BY FORMAT_DATETIME("%B", order_purchase_timestamp)),

sales_17_18 AS (
  SELECT s1.month,
  total_price17 AS cost_2017, total_price18 AS cost_2018
```

```
  FROM sales_for_2017 s1
  JOIN sales_for_2018 s2 ON s1.month = s2.month
)

SELECT *,
ROUND(((cost_2018 - cost_2017)/ cost_2017) * 100,2) AS percent_growth
FROM sales_17_18
```

Output:

Row	month	cost_2017	cost_2018	percent_growth
1	January	127545.67	1078606.86	745.66
2	February	271298.65	966510.88	256.25
3	March	414369.39	1120678.0	170.45
4	April	390952.18	1132933.95	189.79
5	May	567066.73	1128836.69	99.07
6	June	490225.6	1012090.68	106.45
7	July	566403.93	1027903.86	81.48
8	August	646000.61	985414.28	52.54

In addition to the above query if we add the below query, we can also get the total % cost of orders increases from 2017 to 2018 conserving the given months

Query:

```
SELECT
SUM(cost_2017) total_2017_cost,
SUM(cost_2018) total_2018_cost,
ROUND(((SUM(cost_2018) - SUM(cost_2017))/ SUM(cost_2017)) * 100,2) AS
growth_percent
FROM sales_17_18
```

Output:

Row	total_2017_cost	total_2018_cost	growth_percent
1	3473862.76	8452975.200000...	143.33

Let us also find the order amount Growth %.

Query:

```
WITH order_value AS (
  SELECT o.*,
  price,
  FORMAT_DATETIME("%Y-%m",order_purchase_timestamp) AS year_month
  FROM `dsml-sql-454708.Target_Dataset.order` o
  LEFT JOIN `dsml-sql-454708.Target_Dataset.order_item` oi ON o.order_id =
oi.order_id
  WHERE order_status = "delivered"
),
```

```
sales_for_2017 AS (SELECT
FORMAT_DATETIME("%B", order_purchase_timestamp) AS month,
ROUND(SUM(price),2) AS total_price17
FROM order_value
WHERE year_month BETWEEN "2017-01" AND "2017-08"
GROUP BY FORMAT_DATETIME("%B", order_purchase_timestamp)),

sales_for_2018 AS (SELECT
FORMAT_DATETIME("%B", order_purchase_timestamp) AS month,
ROUND(SUM(price),2) AS total_price18
FROM order_value
WHERE year_month BETWEEN "2018-01" AND "2018-08"
GROUP BY FORMAT_DATETIME("%B", order_purchase_timestamp)),

sales_17_18 AS (
  SELECT s1.month,
  total_price17 AS sales_2017, total_price18 AS sales_2018
  FROM sales_for_2017 s1
  JOIN sales_for_2018 s2 ON s1.month = s2.month
)

SELECT *,
ROUND(((sales_2018 - sales_2017)/ sales_2017) * 100,2) AS percent_growth
FROM sales_17_18
```

Output:

Row	month	sales_2017	sales_2018	percent_growth
1	January	111798.36	924645.0	727.06
2	February	234223.4	826437.13	252.84
3	March	359198.85	953356.25	165.41
4	April	340669.68	973534.09	185.77
5	May	489338.25	977544.69	99.77
6	June	421923.37	856077.86	102.9
7	July	481604.52	867953.46	80.22
8	August	554699.7	838576.64	51.18

In addition to the above query if we add the below query, we can also get the total % Growth from 2017 to 2018 conserving the given months

Query:

```
SELECT
SUM(sales_2017) total_2017_sales,
SUM(sales_2018) total_2018_sales,
ROUND(((SUM(sales_2018) - SUM(sales_2017))/ SUM(sales_2017)) * 100,2) AS
growth_percent
FROM sales_17_18
```

Output:

Row	total_2017_sales	total_2018_sales	growth_percent
1	2993456.13	7218125.12	141.13

Insights:

- There is significant increase in cost of order/sales in every month from 2017 to 2018.
- **Maximum growth was observed in the month January.**
- We can see mostly there is a decline in Growth % from January till August where April and June month being exception.
- **Overall, there is 141% of Growth in sales from 2017 to 2018 whereas there is 143% of growth in cost of order.**

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

Details about the functions used in below query

AVG () computes the average of a set of values by dividing the sum of those values by the count of non-null values. If the sum exceeds the maximum value for the data type of the return value, AVG() returns an error. AVG is a deterministic function when used without the OVER and ORDER BY clauses

Query:

```
WITH order_and_state AS (
  SELECT oi.order_id, oi.price, customer_state
  FROM `dsml-sql-454708.Target_Dataset.order_item` oi
  LEFT JOIN `dsml-sql-454708.Target_Dataset.order` o ON oi.order_id = o.order_id
  LEFT JOIN `dsml-sql-454708.Target_Dataset.customer` c ON o.customer_id =
c.customer_id
  WHERE order_status = "delivered"
)

SELECT customer_state,
ROUND(SUM(price),2) AS total_order_price,
ROUND(AVG(price),2) AS avg_order_price,
COUNT(customer_state) AS order_count
FROM order_and_state
GROUP BY customer_state
ORDER BY total_order_price DESC
```

Output:

Row	customer_state	total_order_price	avg_order_price	order_count
1	SP	5067633.16	109.1	46448
2	RJ	1759651.13	124.42	14143
3	MG	1552481.83	120.2	12916
4	RS	728897.47	118.83	6134
5	PR	666063.51	117.91	5649
6	SC	507012.13	123.75	4097
7	BA	493584.14	134.02	3683
8	DF	296498.41	125.9	2355
9	GO	282836.7	124.21	2277
10	ES	268643.45	120.74	2225

Also let us sort the output by avg_order_price in descending order.

Output:

Row	customer_state	total_order_price	avg_order_price	order_count
1	PB	112586.82	192.13	586
2	AL	78855.72	184.67	427
3	AC	15930.97	175.07	91
4	RO	45682.76	167.34	273
5	PA	174470.59	165.53	1054
6	AP	13374.81	165.12	81
7	PI	84721.0	161.99	523
8	RN	82105.66	157.59	521
9	TO	48402.51	156.14	310
10	CE	219757.38	154.11	1426

Insights:

- Top 3 states with respect to total order price and order count are SP, RJ and MG.
- **If we consider the average order price top states are PB, AL and AC.**
- Interestingly SP and MG states have low average order price.
- Though the average order price is high for the States **PB, AL and AC** but the order count is very less.

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

Query:

```
WITH freight_and_state AS (
  SELECT oi.order_id, oi.freight_value, customer_state
  FROM `dsml-sql-454708.Target_Dataset.order_item` oi
  LEFT JOIN `dsml-sql-454708.Target_Dataset.order` o ON oi.order_id = o.order_id
```

```
  LEFT JOIN `dsml-sql-454708.Target_Dataset.customer` c ON o.customer_id =
c.customer_id
  WHERE order_status = "delivered"
)

SELECT customer_state,
ROUND(SUM(freight_value),2) AS total_freight_price,
ROUND(AVG(freight_value),2) AS avg_freight_price,
COUNT(customer_state) AS order_count
FROM freight_and_state
GROUP BY customer_state
ORDER BY total_freight_price DESC
```

Output:

Row	customer_state	total_freight_price	avg_freight_price	order_count
1	SP	702069.99	15.12	46448
2	RJ	295750.44	20.91	14143
3	MG	266409.84	20.63	12916
4	RS	132575.32	21.61	6134
5	PR	115645.29	20.47	5649
6	BA	97553.67	26.49	3683
7	SC	88115.65	21.51	4097
8	PE	57082.56	32.69	1746
9	GO	51375.65	22.56	2277
10	DF	49624.94	21.07	2355

Let us also check the output of avg_freight_price in descending order

Output:

Row	customer_state	total_freight_price	avg_freight_price	order_count
1	PB	25251.73	43.09	586
2	RR	1982.05	43.09	46
3	RO	11283.24	41.33	273
4	AC	3644.36	40.05	91
5	PI	20457.19	39.12	523
6	MA	30794.17	38.49	800
7	TO	11604.86	37.44	310
8	SE	13714.94	36.57	375
9	AL	15316.77	35.87	427
10	RN	18609.12	35.72	521

Insights:

- SP, RJ and MG are the top States where the total freight price is large.
- Average freight price is hight for the states PB, RR and RO.
- **SP, PR and MG have the lowest average freight price.**
- Additionally, we can also see that average freight price is not much dependent on total order counts.

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

   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.

   You can calculate the delivery time and the difference between the estimated & actual delivery date using the given formula:

   - **time_to_deliver** = order_delivered_customer_date - order_purchase_timestamp
   - **diff_estimated_delivery** = order_delivered_customer_date - order_estimated_delivery_date

Query:

```
SELECT order_id,
DATE_DIFF(order_delivered_customer_date, order_purchase_timestamp, day) AS
time_to_deliver_days,
DATE_DIFF(order_delivered_customer_date, order_estimated_delivery_date, day) AS
diff_estimated_delivery_days
FROM `dsml-sql-454708.Target_Dataset.order`
WHERE order_status = "delivered"
ORDER BY time_to_deliver_days DESC
```

Output:

Row	order_id	time_to_deliver_days	diff_estimated_delivery_days
1	ca07593549f1816d26a572e06...	209	181
2	1b3190b2dfa9d789e1f14c05b...	208	188
3	440d0d17af552815d15a9e41a...	195	165
4	285ab9426d6982034523a855f...	194	166
5	0f4519c5f1c541ddec9f21b3bd...	194	161
6	2fb597c2f772eca01b1f5c561b...	194	155
7	47b40429ed8cce3aee9199792...	191	175
8	2fe324febf907e3ea3f2aa9650...	189	167
9	2d7561026d542c8dbd8f0daea...	188	159
10	c27815f7e3dd0b926b5855262...	187	162

Insights:

- **There are 63 orders which took more than 100 days to get delivered.**
- **There are 39 orders which took more than 100 days than addition to the actual estimated delivery date.**
- Also, there are 6534 orders where Target was failed to meet the estimated delivery date.
- 89936 such orders are there where the order was delivered on or before the estimated delivery date.
- 33696 orders were delivered within 7 days of the order date.
- **Only 1585 orders were there where orders were delivered within 1 days of the order date.**

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

Details about the clause used in below query

*The **LIMIT** clause is used to specify the number of records to return. The LIMIT clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.*

Query:

Top 5 states with highest average freight value

```
WITH freight_and_state AS (
  SELECT oi.order_id, oi.freight_value, customer_state
  FROM `dsml-sql-454708.Target_Dataset.order_item` oi
  LEFT JOIN `dsml-sql-454708.Target_Dataset.order` o ON oi.order_id = o.order_id
  LEFT JOIN `dsml-sql-454708.Target_Dataset.customer` c ON o.customer_id =
c.customer_id
  WHERE order_status = "delivered")
SELECT customer_state,
ROUND(AVG(freight_value),2) AS avg_freight_price,
FROM freight_and_state
GROUP BY customer_state
ORDER BY avg_freight_price DESC
LIMIT 5
```

Output:

Row	customer_state	avg_freight_price
1	RR	43.09
2	PB	43.09
3	RO	41.33
4	AC	40.05
5	PI	39.12

Top 5 states with lowest average freight value

Query:

```
WITH freight_and_state AS (
  SELECT oi.order_id, oi.freight_value, customer_state
  FROM `dsml-sql-454708.Target_Dataset.order_item` oi
  LEFT JOIN `dsml-sql-454708.Target_Dataset.order` o ON oi.order_id = o.order_id
  LEFT JOIN `dsml-sql-454708.Target_Dataset.customer` c ON o.customer_id =
c.customer_id
  WHERE order_status = "delivered"
)

SELECT customer_state,
ROUND(AVG(freight_value),2) AS avg_freight_price,
FROM freight_and_state
GROUP BY customer_state
ORDER BY avg_freight_price
LIMIT 5
```

Output:

Row	customer_state	avg_freight_price
1	SP	15.12
2	PR	20.47
3	MG	20.63
4	RJ	20.91
5	DF	21.07

Insights:

- Top 5 states with high average freight values are RR, PB, RO, AC and PI.
- Bottom 5 states with low average freight values are SP, PR, MG, RJ and DF.

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

Top 5 states with highest average delivery time

Query:

```
WITH delivery_time AS (
  SELECT o.order_id, customer_state,
  DATE_DIFF(order_delivered_customer_date, order_purchase_timestamp, day) AS
time_to_deliver_days
  FROM `dsml-sql-454708.Target_Dataset.order` o
  LEFT JOIN `dsml-sql-454708.Target_Dataset.customer` c ON o.customer_id =
c.customer_id
  WHERE order_status = "delivered"
)
```

```sql
SELECT customer_state,
ROUND(AVG(time_to_deliver_days),2) AS avg_delivery_time
FROM delivery_time
GROUP BY customer_state
ORDER BY avg_delivery_time DESC
LIMIT 5
```

Output:

Row	customer_state	avg_delivery_time
1	RR	28.98
2	AP	26.73
3	AM	25.99
4	AL	24.04
5	PA	23.32

Top 5 states with lowest average delivery time

Query:

```sql
WITH delivery_time AS (
  SELECT o.order_id, customer_state,
  DATE_DIFF(order_delivered_customer_date, order_purchase_timestamp, day) AS
time_to_deliver_days
  FROM `dsml-sql-454708.Target_Dataset.order` o
  LEFT JOIN `dsml-sql-454708.Target_Dataset.customer` c ON o.customer_id =
c.customer_id
  WHERE order_status = "delivered")
SELECT customer_state,
ROUND(AVG(time_to_deliver_days),2) AS avg_delivery_time
FROM delivery_time
GROUP BY customer_state
ORDER BY avg_delivery_time
LIMIT 5
```

Output:

Row	customer_state	avg_delivery_time
1	SP	8.3
2	PR	11.53
3	MG	11.54
4	DF	12.51
5	SC	14.48

Insights:

- Top 5 states with highest average delivery time are RR, AB, AM, AL and PA.
- Top 5 states with lowest average delivery time are SP, PR, MG, DF and SC.

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.

Query:

```
WITH delivery_time AS (
  SELECT o.order_id, customer_state,
  DATE_DIFF(order_delivered_customer_date, order_purchase_timestamp, day) AS
time_to_deliver_days,
  DATE_DIFF(order_estimated_delivery_date, order_purchase_timestamp, day) AS
estimate_deliver_days
  FROM `dsml-sql-454708.Target_Dataset.order` o
  LEFT JOIN `dsml-sql-454708.Target_Dataset.customer` c ON o.customer_id =
c.customer_id
  WHERE order_status = "delivered"
),

avg_time AS (SELECT customer_state,
ROUND(AVG(time_to_deliver_days),2) AS avg_delivery_days,
ROUND(AVG(estimate_deliver_days),2) AS avg_estimate_days
FROM delivery_time
GROUP BY customer_state)

SELECT *,
ROUND((avg_estimate_days - avg_delivery_days),2) AS avg_diff
FROM avg_time
ORDER BY avg_diff DESC
```

Output:

Row	customer_state	avg_delivery_days	avg_estimate_days	avg_diff
1	AC	20.64	40.72	20.08
2	RO	18.91	38.39	19.48
3	AP	26.73	45.87	19.14
4	AM	25.99	44.92	18.93
5	RR	28.98	45.63	16.65

Insights:

- AC, RO, AP, AM and RR are the top 5 states where order is delivered much before than the average estimated delivery for those states.
- **In the state AC on average the order is delivered 20 days before the average estimated delivery date.**

6. Analysis based on the payments:

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

Query:

```
WITH payment_method AS (
  SELECT o.order_id,
  FORMAT_DATETIME("%B_%Y",order_purchase_timestamp) AS order_month,
  payment_type
  FROM `dsml-sql-454708.Target_Dataset.order` o
  LEFT JOIN `dsml-sql-454708.Target_Dataset.payment` p ON o.order_id = p.order_id
  WHERE order_status = "delivered" AND payment_type IS NOT NULL
)

SELECT order_month, payment_type,
COUNT(*) AS payment_count
FROM payment_method
GROUP BY order_month, payment_type
```

Output:

Row	order_month	payment_type	payment_count
1	October_2016	UPI	51
2	October_2016	credit_card	209
3	October_2016	debit_card	2
4	October_2016	voucher	20
5	December_2016	credit_card	1
6	January_2017	UPI	188
7	January_2017	credit_card	542
8	January_2017	debit_card	9
9	January_2017	voucher	60
10	February_2017	UPI	371

Let us short the above query in various ways and see the outputs

Query:

```
WITH payment_method AS (
  SELECT o.order_id,
  FORMAT_DATETIME("%B_%Y",order_purchase_timestamp) AS order_month,
  payment_type
  FROM `dsml-sql-454708.Target_Dataset.order` o
  LEFT JOIN `dsml-sql-454708.Target_Dataset.payment` p ON o.order_id = p.order_id
  WHERE order_status = "delivered" AND payment_type IS NOT NULL
),

final_table AS (SELECT order_month, payment_type,
COUNT(*) AS payment_count
FROM payment_method
GROUP BY order_month, payment_type
ORDER BY payment_type DESC)

SELECT *
FROM final_table
ORDER BY payment_count DESC
```

Output:

Row	order_month	payment_type	payment_count
1	November_2017	credit_card	5716
2	March_2018	credit_card	5526
3	May_2018	credit_card	5398
4	January_2018	credit_card	5368
5	April_2018	credit_card	5341
6	February_2018	credit_card	5114
7	August_2018	credit_card	4904
8	June_2018	credit_card	4760
9	July_2018	credit_card	4660
10	December_2017	credit_card	4245

Query:

```
WITH payment_method AS (
  SELECT o.order_id,
  FORMAT_DATETIME("%B_%Y",order_purchase_timestamp) AS order_month,
  payment_type
  FROM `dsml-sql-454708.Target_Dataset.order` o
  LEFT JOIN `dsml-sql-454708.Target_Dataset.payment` p ON o.order_id = p.order_id
  WHERE order_status = "delivered" AND payment_type IS NOT NULL
),

final_table AS (SELECT order_month, payment_type,
COUNT(*) AS payment_count
FROM payment_method
GROUP BY order_month, payment_type
ORDER BY payment_type DESC)

SELECT payment_type,
SUM(payment_count) total_payment_count
```

```
FROM final_table
GROUP BY payment_type
ORDER BY total_payment_count DESC
```

Output:

Row	payment_type	total_payment_count
1	credit_card	74586
2	UPI	19191
3	voucher	5493
4	debit_card	1486

Insights:

- In November_2017 5716 payments were done using Credit card.
- **Majority of the payments are done using credit card followed by UPI.**

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

Query:

```
SELECT
payment_installments,
COUNT(order_id) AS order_count
FROM`dsml-sql-454708.Target_Dataset.payment`
GROUP BY payment_installments
ORDER BY order_count DESC
```

Output:

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

Insights:

- Maximum orders were placed where payment instalment is 1.
- Very few orders were placed have payment instalment greater than 12

Let us also investigate the customer review section

Query:

```
SELECT review_score,
COUNT(review_score) AS revire_count
FROM `dsml-sql-454708.Target_Dataset.order_review`
GROUP BY review_score
ORDER BY revire_count DESC
```

Output:

Row	review_score	revire_count
1	5	57328
2	4	19142
3	1	11424
4	3	8179
5	2	3151

Insights:

- **Most of the orders got good review i.e., 5 or 4 but there are many orders which got 1 and 2 rating.**
- 3 can be considered as a neutral rating where customer is neither satisfied nor happy.

7. Actionable Insights & Recommendations

Final Insights:

1. Order Growth and Seasonality:

- Orders grew significantly from 2016 to 2018, with a 16,165% increase from 2016 to 2017 (likely due to limited 2016 data) and 21.54% from 2017 to 2018.
- Peak order months were November 2017 and the first two quarters of 2018, indicating seasonality tied to holidays or promotions.
- Most orders are placed in the afternoon (13–18 hours), particularly between 12–4 PM, with minimal activity at dawn (0–6 hours).

2. Geographic Distribution:

- São Paulo (SP), Rio de Janeiro (RJ), and Minas Gerais (MG) dominate order volumes, contributing ~66% of total orders (SP alone accounts for 42%).
- Smaller states like Amapá (AP), Roraima (RR), and Tocantins (TO) have negligible order counts, suggesting limited market penetration.
- Urban centres like São Paulo, Rio de Janeiro, and Belo Horizonte drive orders, while rural areas (e.g., Urucurituba) have minimal activity.

3. Economic Impact:

- Sales grew 141% and order costs 143% from 2017 to 2018 (Jan–Aug), with January showing the highest growth (727% for sales).
- SP, RJ, and MG have the highest total order and freight values, but states like Paraíba (PB) and Alagoas (AL) have higher average order prices despite lower order counts.
- Freight costs are lowest in SP (avg. $15.12) and highest in PB/RR (~$43), indicating logistical challenges in remote areas.

4. Delivery Performance:

- Average delivery times are lowest in SP (8.3 days), PR (11.53 days), and MG (11.54 days), and highest in RR (28.98 days) and AP (26.73 days).
- States like Acre (AC) and Rondônia (RO) deliver significantly earlier than estimated (20 and 19 days, respectively), suggesting conservative estimates or efficient logistics in these regions.
- However, 6,534 orders missed estimated delivery dates, and 63 orders took over 100 days, highlighting occasional logistical bottlenecks.

5. **Payment Trends:**
   - Credit cards dominate payments (74,586 orders), followed by UPI (19,191) and vouchers (5,493).
   - Most orders (52,546) are paid in a single installment, with fewer opting for higher installments (e.g., 644 for 9 installments).
   - Payment preferences remain consistent across months, with credit card usage peaking in November 2017 (5,716 orders).
   - Most orders are completed with a single installment. Very few orders involve payment installments beyond 12 months.

6. **Customer Satisfaction:**
   - Most customers are highly satisfied, with 5-star (57,328) and 4-star (19,142) reviews dominating.
   - However, a notable portion are dissatisfied, with 1-star (11,424) and 2-star (3,151) reviews indicating issues.
   - The 3-star reviews (8,179) reflect a neutral sentiment, suggesting some customers are indifferent.

Recommendations:

1. **Enhance Market Penetration in Underserved States:**
   - Increase marketing efforts and establish more distribution centres in states like AP, RR, and TO, which have low order counts. These states contribute <1% of orders, likely due to limited awareness or logistical barriers. Localized campaigns and partnerships with regional sellers could boost demand. Target should also perform a targeted digital marketing campaign in RR and AP, leveraging social media platforms popular in Brazil (e.g., WhatsApp, Instagram).

2. **Optimize Freight Costs in High-Cost States:**
   - Negotiate bulk shipping contracts or partner with local logistics providers in states with high freight costs (e.g., PB, RR, RO). Average freight prices in PB ($43.09) and RR ($43.09) are nearly triple those in SP ($15.12), inflating costs for customers and reducing competitiveness. Target should Implement a freight subsidy program for orders above a certain value in high-cost states to encourage purchases.

3. **Improve Delivery Time Reliability:**
   - Investigate and address causes of extreme delivery delays (>100 days) and missed estimated delivery dates (6,534 orders). Long delivery times in states like RR (28.98 days) and AP (26.73 days) may deter customers, while early deliveries in AC and RO suggest overly conservative estimates. Target should use predictive analytics to refine estimated delivery dates and prioritize logistics improvements in RR and AP (e.g., regional warehouses).

4. **Leverage Peak Ordering Times:**
   - Schedule promotions and flash sales during afternoon hours (12–4 PM) and key months (November, January–March). Afternoon orders dominate (46,965), and November 2017 and Q1 2018 saw the highest order volumes, likely tied to holidays like Black Friday and Carnival. Target can Launch a "Midday Deals" campaign with time-limited discounts between 12–4 PM to capitalize on peak activity.

5. **Encourage Higher Installment Payments:**
   - Offer incentives (e.g., discounts or free shipping) for customers opting for 2–6 installment payments. Single-installment payments dominate (52,546 orders), but installments (2–6) are popular (30,091 orders), indicating affordability concerns. Target can partner with credit card providers to offer zero-interest installments for orders above a threshold (e.g., $100).

6. **Build high Customer Satisfaction level:**
   - Conduct a survey or detailed analyse feedback from 1-star and 2-star reviews to identify common issues (e.g., delivery delays, product quality). Target should develop a sentiment analysis model to categorize review comments and prioritize improvements in low-scoring areas (e.g., late deliveries).

xxxxxThis is the END of Case studyxxxxx

TARGET