# Introduction

Sds

dfsdfds

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

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

Query:

SELECT

       COLUMN\_NAME, DATA\_TYPE

FROM

       targetsql.INFORMATION\_SCHEMA.COLUMNS

WHERE

       TABLE\_NAME = 'customers';

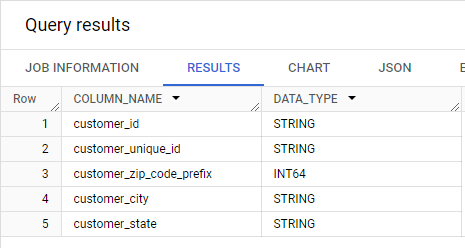


Figure 2—1: Data types of customers’ table

Similarly exploring other tables of the dataset data types

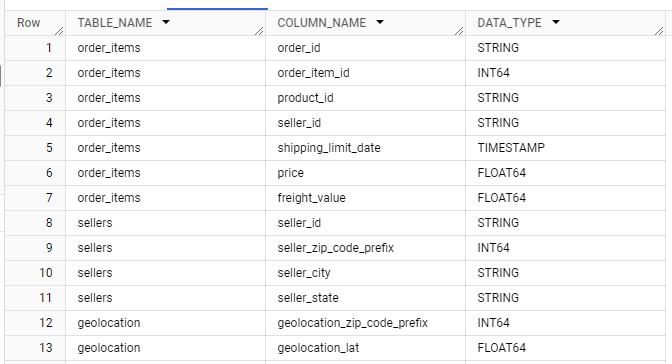
Query:

SELECT

       TABLE\_NAME, COLUMN\_NAME, DATA\_TYPE

FROM

       targetsql.INFORMATION\_SCHEMA.COLUMNS



With the datatypes thus obtained the detailed SCHEMA of the TARGET DATA SET

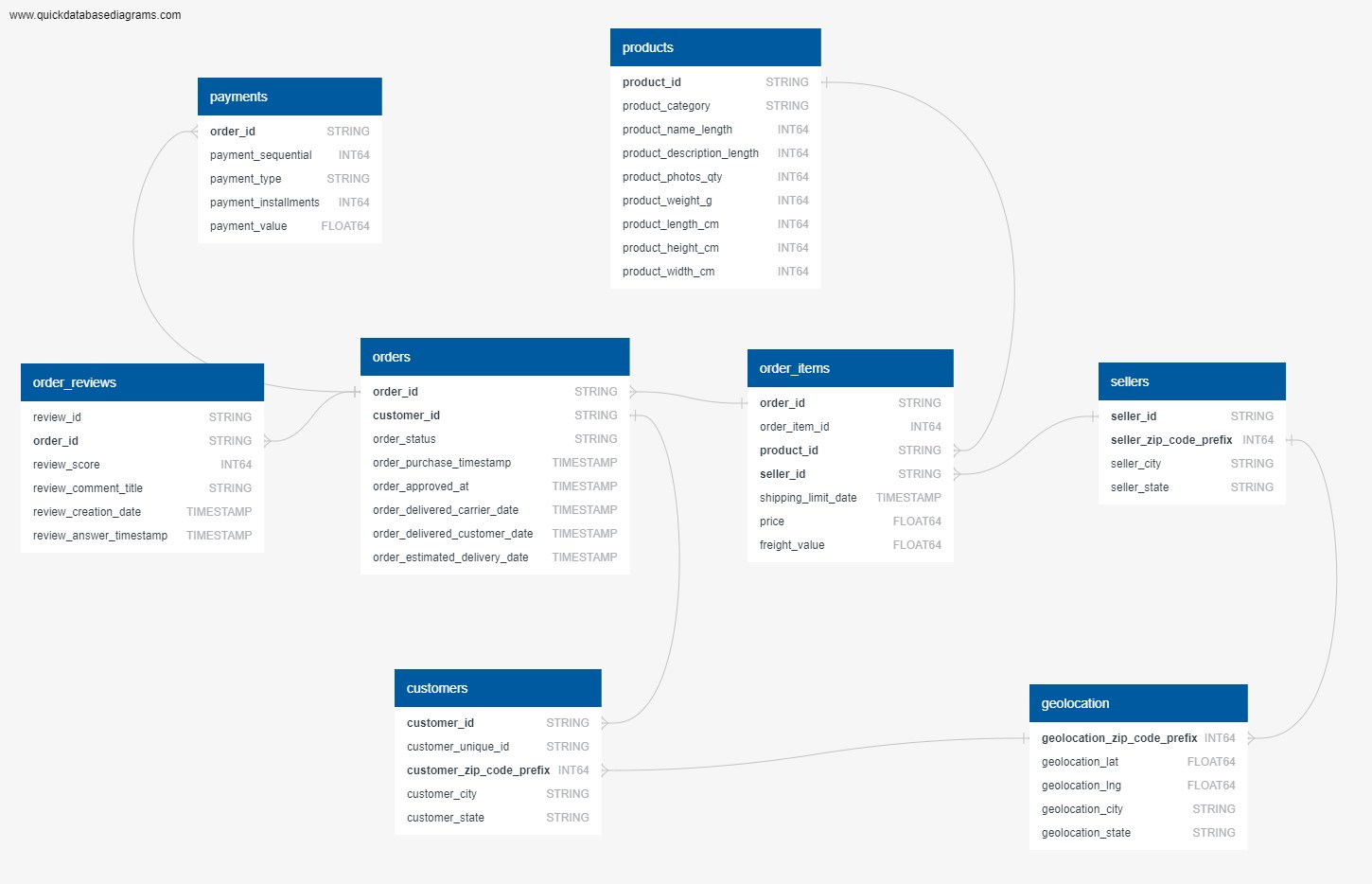


Figure 2—2: Schema of the dataset

INSIGHTS/RECOMMENDATIONS:

1. From the data it could be derived that the customer id, city, and state are all in STRING format.
2. The customer zip code is in INT i.e.; integer format.
3. Also all other data that contains text as input in is STRING format, numbers are in INT64 format and decimals are in FLOAT64 format.

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

Query:

SELECT MIN(order\_purchase\_timestamp) AS start\_date,

       MAX(order\_purchase\_timestamp) AS end\_date,

       DATE\_DIFF(MAX(order\_purchase\_timestamp), MIN(order\_purchase\_timestamp), DAY) AS time\_range

FROM

`targetsql.orders`;



Figure 2—3: Time range of the orders placed

INSIGHTS/RECOMMENDATIONS:

1. The start date of the dataset is 2016 9th of April and the end date is 2018 17th of October.
2. The dataset is over a period of 772 days.

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

Query:

-- Getting a list of all cities and countries of customers who placed orders during the period of 772 days.

SELECT

       distinct c.customer\_city, c.customer\_state

FROM

       `targetsql.customers` c

JOIN

`targetsql.orders` o ON c.customer\_id = o.customer\_id

order by

       c.customer\_state;

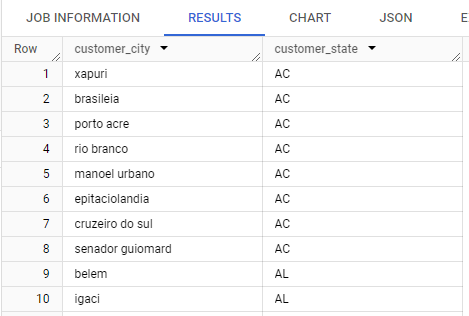


Figure 2—4:List of all cities and states

--Getting the count of cities and countries

SELECT count(distinct c.customer\_city) as city\_count,

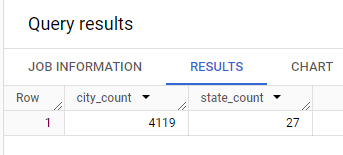
       count(distinct c.customer\_state) as state\_count

FROM

       `targetsql.customers` c

JOIN

       `targetsql.orders` o ON c.customer\_id = o.customer\_id;



INSIGHTS/RECOMMENDATIONS:

1. Customers from 4119 distinct cities and 27 states had ordered from the Target store.
2. This gives us the information of the areas where the orders are placed, giving us the scope to increase the e-commerce from the areas where there are few or no orders.

# ****In-depth Exploration:****

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

### Growing trend in the no. of orders placed over the past years (all orders)

Query:

SELECT EXTRACT(YEAR FROM order\_purchase\_timestamp) as years,

       EXTRACT(MONTH FROM order\_purchase\_timestamp) as months,

       count(\*) as num\_of\_orders

FROM

       `targetsql.orders`

GROUP BY

       1,2

ORDER BY

       years, months;

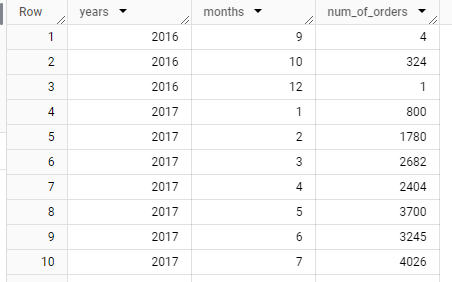


Figure 3—1: Trend in the orders placed over the years

### The trend of the only successfully delivered orders

Query:

SELECT EXTRACT(YEAR FROM order\_purchase\_timestamp) as years,

       EXTRACT(MONTH FROM order\_purchase\_timestamp) as months,

       count(\*) as num\_of\_orders

FROM

       `targetsql.orders`

WHERE

       lower(order\_status) = 'delivered'

GROUP BY

       1,2

ORDER BY

       1,2;

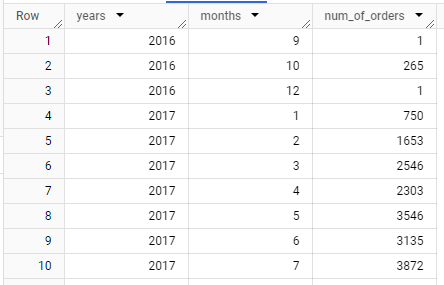


Figure 3—2: Month on Month of num of orders delivered

Figure 3—3: Month on month graph of orders delivered

INSIGHTS:

1. The numbers of orders per month is nothing but the sales that Target has made in that particular month
2. From the results it can be observed that the maximum sales happened in the year 2017 in the month of November, the next highest in the year 2018 Jan probably due to New Year offers.
3. It can be observed that the sales have steadily increased from 2017 January to December.
4. A similar case is noticed in the 2018 year.

RECOMMENDATIONS:

1. An increase in e-commerce has been noticed from the period of 2017-2018.
2. To keep the rate of sales at a steady rate, recommendations/suggestions for the orders regularly purchased can be shown to customers.

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

Query:

SELECT EXTRACT(YEAR FROM order\_purchase\_timestamp) as years,

       EXTRACT(MONTH FROM order\_purchase\_timestamp) as months,

       count(\*) as num\_orders

FROM

       `targetsql.orders`

GROUP BY

       1,2

ORDER BY

       3 DESC;



Figure 3—4: Peaks in the sales

--For orders the orders that are sucessfully delivered

SELECT EXTRACT(YEAR FROM order\_purchase\_timestamp) as years,

       EXTRACT(MONTH FROM order\_purchase\_timestamp) as months,

       count(\*) as num\_orders

FROM

       `targetsql.orders`

WHERE

       lower(order\_status) = 'delivered'

GROUP BY

       1,2

ORDER BY

       3 DESC;

Figure 3—5: Growth trend of orders delivered

INSIGHTS:

1. From the graph based on the result it can be observed that there has been increase in sales from 2016- 2018.
2. In 2016 the sales were low might be due to the limited data available, a few other reasons could be the recession, political reforms, or less reach.
3. Peaks in sales can be observed in months October, and December months wrt the given data.
4. Peak season is observed from September to February in the above graph.

RECOMMENDATIONS:

1. Offers: To increase sales in other seasons, offers such as combo, sales could bring up the number of sales
2. Seasonal Products: Including seasonal products in the recommendation of the regular customers and other outreach programs might also help increase the numbers in sales.

## 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:

### Creating a temporary table with new column that represents the part of the day based on the purchase time

--Adding a new column that gives info about the part of the day

select order\_id, order\_purchase\_timestamp, time(order\_purchase\_timestamp) as order\_time,

       CASE

       WHEN time(order\_purchase\_timestamp) between '00:00:00' and '06:59:59' then 'Dawn'

       WHEN time(order\_purchase\_timestamp) between '07:00:00' and '12:59:59' then 'Mornings'

       WHEN time(order\_purchase\_timestamp) between '13:00:00' and '18:59:59' then 'Afternoon'

       WHEN time(order\_purchase\_timestamp) between '19:00:00' and '23:59:59' then 'Night'

       end as time\_of\_day

from

       `targetsql.orders`

order by

       order\_purchase\_timestamp;

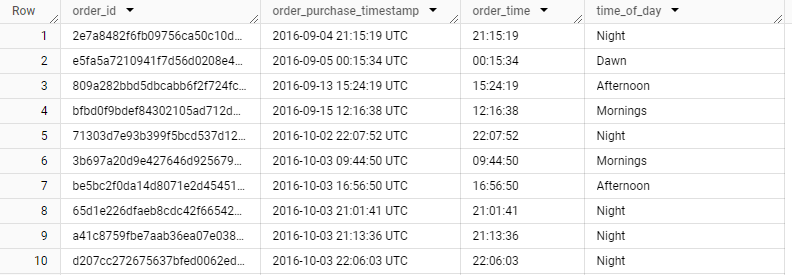


Figure 3—6:Orders table along with part of the day column

### Using CTE using the temporary table created and grouping the orders based on the part of the day

--count of orders during the part of the day

WITH orders\_time\_day as (select order\_id, order\_purchase\_timestamp, time(order\_purchase\_timestamp) as order\_time,

       CASE

       WHEN time(order\_purchase\_timestamp) between '00:00:00' and '06:59:59' then 'Dawn'

       WHEN time(order\_purchase\_timestamp) between '07:00:00' and '12:59:59' then 'Mornings'

       WHEN time(order\_purchase\_timestamp) between '13:00:00' and '18:59:59' then 'Afternoon'

       WHEN time(order\_purchase\_timestamp) between '19:00:00' and '23:59:59' then 'Night'

       end as time\_of\_day

from

       `targetsql.orders`

order by

       order\_purchase\_timestamp)

SELECT o.time\_of\_day,

       COUNT(\*) as num\_orders\_prt\_day

FROM

       orders\_time\_day o

GROUP BY

       o.time\_of\_day

ORDER BY

       num\_orders\_prt\_day DESC;

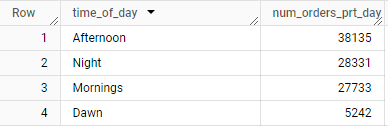


Figure 3—7: Number of orders based on part of the day

INSIGHTS:

1. From the result it is observed that a maximum number of orders are placed in the afternoon time.
2. The least orders are placed during dawn.
3. A similar kind of sales is seen in the mornings and afternoons based on the given data

RECOMMENDATIONS:

1. Sales Offers: Early bird offers during the early hours, and midnight sales offers could boost the sales at dawn.
2. Discount on Fresh Produce during early hours could increase the sales in the 0-6 hrs.

# ****Evolution of E-commerce orders in the Brazil region****

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

Query:

--Getting the orders placed by each city and state

SELECT

       c.customer\_state, t.years, t.months, count(\*) as orders\_per\_state

FROM

       `targetsql.customers` c

JOIN (SELECT EXTRACT(YEAR FROM order\_purchase\_timestamp) as years,

              EXTRACT(MONTH FROM order\_purchase\_timestamp) as months,

              customer\_id

       FROM `targetsql.orders`) t

ON

       c.customer\_id = t.customer\_id

GROUP BY

       c.customer\_state, t.years, t.months

order by

       t.years, t.months;

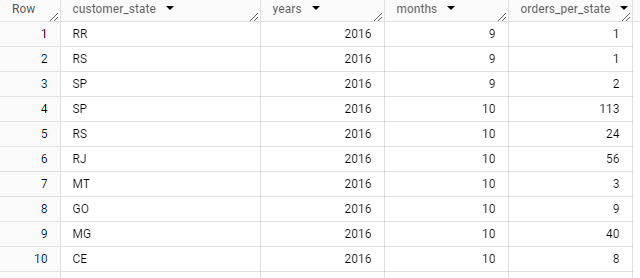


Figure 4—1: Number of orders of each state per month

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

## Get the % increase in the cost of orders from the 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.

### Getting the orders from 2017 to 2018 from months January to August only

Query:

SELECT \*

FROM (SELECT order\_id,

       EXTRACT(YEAR FROM order\_purchase\_timestamp) as years,

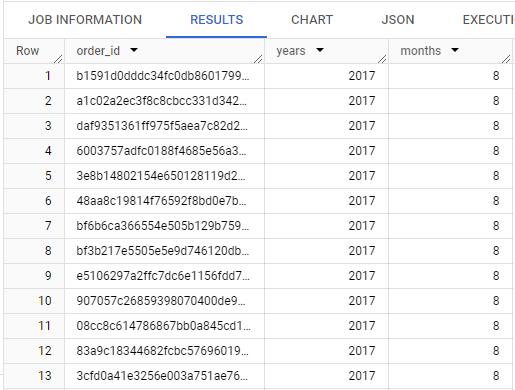
       EXTRACT(MONTH FROM order\_purchase\_timestamp) as months

       FROM `targetsql.orders`) tbl

WHERE (tbl.years = 2017 or tbl.years = 2018)

      AND tbl.months between 01 and 08

ORDER BY tbl.years, tbl.months DESC;



### Getting the cost of orders wrt year-month

Query:

--Getting the cost of orders wrt year, month.

WITH orders\_YM as (SELECT order\_id,

       EXTRACT(YEAR FROM order\_purchase\_timestamp) as years,

       EXTRACT(MONTH FROM order\_purchase\_timestamp) as months,

       order\_status

FROM `targetsql.orders`)

SELECT o.years,o.months,

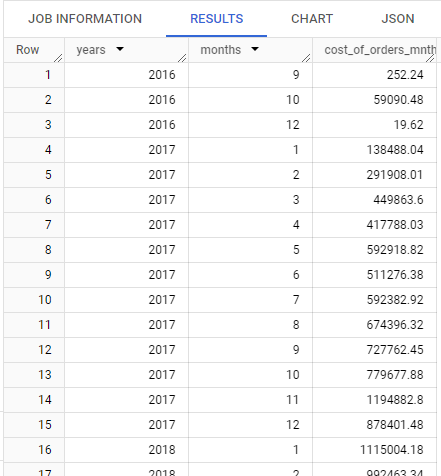
       ROUND(SUM(payment\_value),3) as cost\_of\_orders\_mnthwise

FROM orders\_YM o join `targetsql.payments` p

ON o.order\_id = p.order\_id

GROUP BY o.years,o.months

ORDER BY o.years,o.months;



### Displaying the cost of orders from 2017 to 2018(include months between JAN and AUG only)

Query:

-- The cost of orders from year 2017 to 2018 (include months between Jan to Aug only)

WITH orders\_YM as (SELECT order\_id,

                          EXTRACT(YEAR FROM order\_purchase\_timestamp) as years,

                          EXTRACT(MONTH FROM order\_purchase\_timestamp) as months,

                          FORMAT\_DATETIME("%Y - %m", order\_purchase\_timestamp) as year\_month,

                          order\_purchase\_timestamp

                   FROM `targetsql.orders`)

SELECT o.year\_month,

       ROUND(SUM(payment\_value),3) as cost\_of\_orders\_mnthwise

FROM orders\_YM o JOIN `targetsql.payments` p

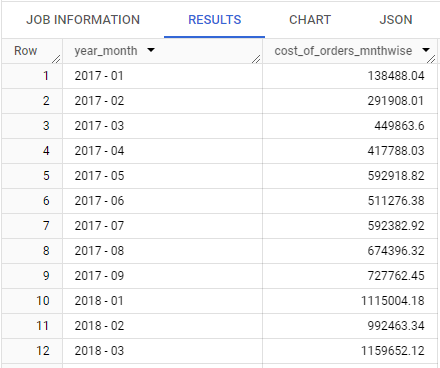
ON o.order\_id = p.order\_id

WHERE (o.years = 2017 or o.years = 2018)

      AND o.months between 01 and 09

GROUP BY o.year\_month

ORDER BY o.year\_month;



### Calculating %increase in the cost of orders **2017 month on 2018 month**

Query:

--% of increase in cost of orders month-wise from 2017 to 2018

WITH orders\_YM as (SELECT

                          order\_id,

                          EXTRACT(YEAR FROM order\_purchase\_timestamp) as years,

                          EXTRACT(MONTH FROM order\_purchase\_timestamp) as months,

                          FORMAT\_DATETIME("%Y - %m", order\_purchase\_timestamp) as year\_month,

                          order\_purchase\_timestamp

                   FROM

                     `targetsql.orders`),

cost\_of\_order\_1718 as (SELECT

                            o.years as YEAR, o.months as MONTH,

                            ROUND(SUM(payment\_value),3) as cost\_of\_orders\_mnthwise

                       FROM

                            orders\_YM o

                       JOIN

                            `targetsql.payments` p  ON o.order\_id = p.order\_id

                       WHERE

                            (o.years = 2017 or o.years = 2018)

                            AND o.months between 01 and 08

                       GROUP BY

                            o.years, o.months

                       ORDER BY

                            o.years, o.months )

SELECT

       p.YEAR, p.MONTH, p.cost\_of\_orders\_mnthwise, n.YEAR as NXT\_YR, n.MONTH as NXT\_MNTH,

n.cost\_of\_orders\_mnthwise,

       ROUND((n.cost\_of\_orders\_mnthwise - p.cost\_of\_orders\_mnthwise)/p.cost\_of\_orders\_mnthwise \* 100,2) as `%increase\_monthwise`

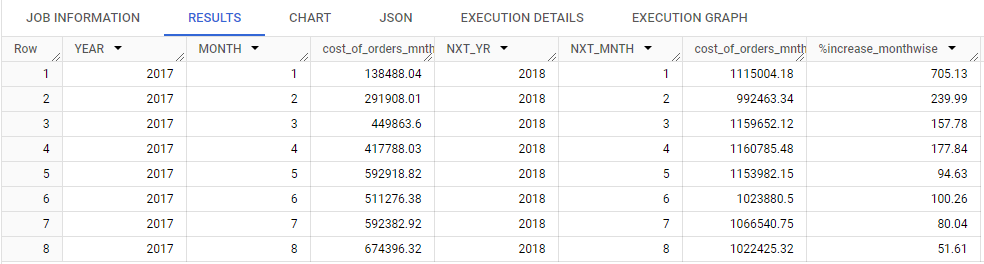
FROM

       cost\_of\_order\_1718 p

JOIN

       cost\_of\_order\_1718 n   ON  p.YEAR = 2017 and n.YEAR = 2018 and n.MONTH = p.MONTH

ORDER BY 2;



### Calculating the % increase in the cost of orders from year 2017 to 2018 **YEAR-WISE** (include months between Jan to Aug only)

Query:

--% of increase in cost of orders year-wise from 2017 to 2018

WITH orders\_YM as

       (SELECT

              order\_id,

              EXTRACT(YEAR FROM order\_purchase\_timestamp) as years,

              EXTRACT(MONTH FROM order\_purchase\_timestamp) as months,

              FORMAT\_DATETIME("%Y - %m", order\_purchase\_timestamp) as year\_month,

              order\_purchase\_timestamp

       FROM

              `targetsql.orders`)

SELECT

       o.years as YEAR,

       ROUND(SUM(payment\_value),3) as cost\_of\_orders,

       CASE

              WHEN o.years = 2017 THEN 0

              ELSE ROUND((SUM(CASE WHEN o.years = 2018 THEN p.payment\_value ELSE 0 END)- LAG(ROUND(SUM(payment\_value),3)) over(order by o.years))/ LAG(ROUND(SUM(payment\_value),3)) over(order by o.years) \*100, 2)

       END as nxt\_yr\_cost

FROM

       orders\_YM o

JOIN

       `targetsql.payments` p  ON o.order\_id = p.order\_id

WHERE

       (o.years = 2017 or o.years = 2018)

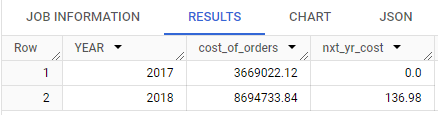
       AND o.months between 01 and 08

GROUP BY

       o.years

ORDER BY

       o.years;



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

Query:

--Displaying the Total & Average value of order price for each state and city

SELECT

       c.customer\_state, c.customer\_city,

       ROUND(SUM(p.payment\_value), 2) as total\_price\_city,

       ROUND(AVG(p.payment\_value),2) as avg\_price\_city

FROM

       `targetsql.customers` c

JOIN

       `targetsql.orders` o ON c.customer\_id = o.customer\_id

JOIN

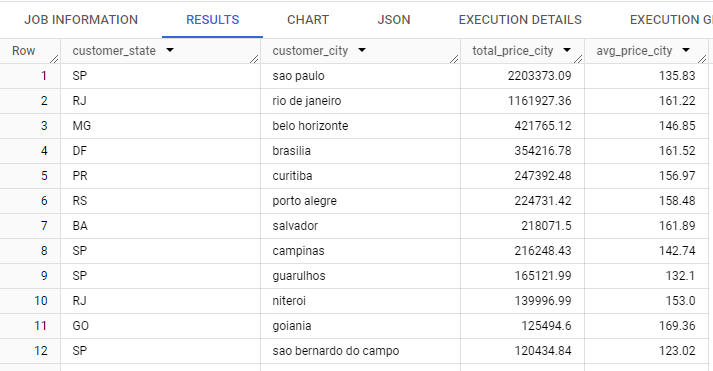
       `targetsql.payments` p ON o.order\_id = p.order\_id

GROUP BY

       c.customer\_state, c.customer\_city

ORDER BY

       total\_price\_city desc, avg\_price\_city;



--Statewise

SELECT

       c.customer\_state,

       ROUND(SUM(p.payment\_value), 2) as total\_price,

       ROUND(AVG(p.payment\_value),2) as avg\_price

FROM

       `targetsql.customers` c

JOIN

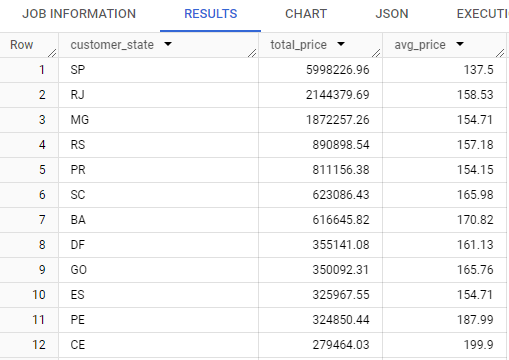
       `targetsql.orders` o ON c.customer\_id = o.customer\_id

JOIN

       `targetsql.payments` p ON o.order\_id = p.order\_id

GROUP BY

       c.customer\_state



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

Query:

--Displaying the Total & Average value of order freight for each state and city

SELECT

       c.customer\_state, customer\_city,

       ROUND(SUM(oi.freight\_value),2) as total\_freight\_city,

       ROUND(AVG(oi.freight\_value),2) as avg\_freight\_city

FROM

       `targetsql.customers` c

JOIN

       `targetsql.orders` o ON c.customer\_id = o.customer\_id

JOIN

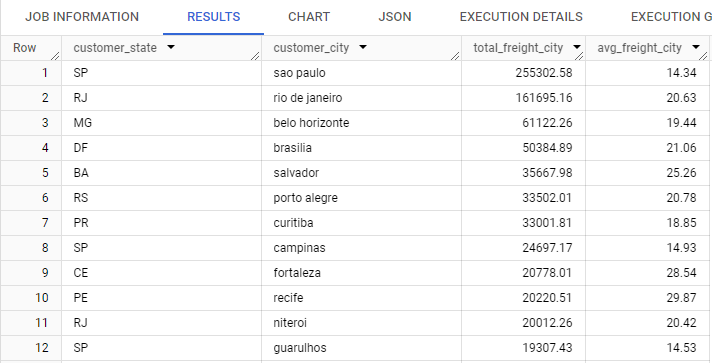
       `targetsql.order\_items` oi ON o.order\_id = oi.order\_id

GROUP BY

       c.customer\_state, customer\_city

ORDER BY

       total\_freight\_city desc, avg\_freight\_city;



--Statewise

SELECT

       c.customer\_state,

       ROUND(SUM(oi.freight\_value),2) as total\_freight,

       ROUND(AVG(oi.freight\_value),2) as avg\_freight

FROM

       `targetsql.customers` c

JOIN

       `targetsql.orders` o ON c.customer\_id = o.customer\_id

JOIN

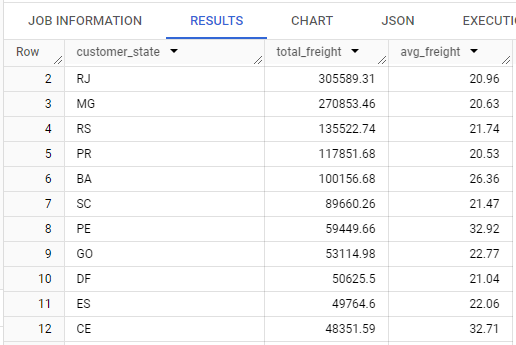
       `targetsql.order\_items` oi ON o.order\_id = oi.order\_id

GROUP BY

       c.customer\_state

ORDER BY

       total\_freight desc, avg\_freight;



# Analysis based on sales, freight and delivery time.

## 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:

#5.11.  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.

-- checking for NUll in order date

SELECT

  \*

FROM

  `targetsql.orders`

WHERE

  order\_purchase\_timestamp is NUll;

--Query

SELECT

  order\_id,

  order\_purchase\_timestamp,

  DATE\_DIFF(DATE(order\_delivered\_customer\_date), DATE(order\_purchase\_timestamp), DAY) as act\_time\_delivery,

  DATE\_DIFF(DATE(order\_estimated\_delivery\_date), DATE(order\_purchase\_timestamp), DAY) as est\_time\_delivery,

  DATE\_DIFF(DATE(order\_delivered\_customer\_date), DATE(order\_estimated\_delivery\_date), DAY) as diff\_delivery

FROM

  `targetsql.orders`

WHERE

  lower(order\_status) = 'delivered';

