Target case study

Data Science and Machine Learning (Scaler Neovarsity)

Business Case 1: Target SQL

Nithisha

```
--I.Import the dataset and do usual exploratory analysis steps like
checking the structure & characteristics of the dataset:
--1. Data type of all columns in the "customers" table.
```

Query:

```
select column_name, data_type from
`Target_Analysis.INFORMATION_SCHEMA.COLUMNS`
Where
table_name = 'customers';
```

Result:

JOB INFORMATION | RESULTS | JSON | EXECUTION

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

Inference:

Data type of columns, with column names can be obtained by using Information schema.
This helps in understanding the structure of the table and the kind of attributes the table stores.

```
--2. Get the time range between which the orders were placed.
```

Query:

```
SELECT
min(order_purchase_timestamp) as minvalue,
max(order_purchase_timestamp) as maxvalue

from `scaler-dsml-sql-393406.Target_Analysis.orders`
```

Result:

JOB INFORMATION RESULTS JSON EXECUTION D

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

Inference:

Time range can be obtained from purchase time details of the orders. Hence the min() and max() functions can give the duration of the orders placed.

--3.Count the Cities & States of customers who ordered during the given period.

Query:

```
select count(distinct c.customer_state),
count(distinct customer_city)
from
 `scaler-dsml-sql-393406.Target_Analysis.customers` c
inner join
`scaler-dsml-sql-393406.Target_Analysis.orders` o
on c.customer_id = o.customer_id
```

Result:

Row	f0_	f1_
1	27	4119

Inference:

Fetching the customers only orders table would give the details of customers who placed orders during the given period.

--II. In-depth Exploration:
--1. 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,
extract(month from order_purchase_timestamp) as month,
```

```sql
count(order_id) as no_of_orders
FROM `scaler-dsml-sql-393406.Target_Analysis.orders`
group by 1,2
order by 1,2
```

Result:

Row	year	month	no_of_orders
1	2016	9	4
2	2016	10	324
3	2016	12	1
4	2017	1	800
5	2017	2	1780
6	2017	3	2682
7	2017	4	2404
8	2017	5	3700
9	2017	6	3245
10	2017	7	4026
11	2017	8	4331
12	2017	9	4285

Inference:

No: of orders can be calculated by count() function grouping by year and month.
As we see the no: of orders do not follow any growing trend in the initial months, it reaches peak during nov 2017 and then lowers during dec because of holiday, then follows a constant pattern till Aug 2018 and falls again.
In all, there is no particular pattern that the no. of orders placed over the years.

--2. Can we see some kind of monthly seasonality in terms of the no. of orders being placed?

Query:

```sql
SELECT
extract(month from order_purchase_timestamp) as year,

count(distinct order_id) as no_of_orders FROM `scaler-dsml-sql-393406.Target_Analysis.orders`
group by 1
order by 1
```

Result:

Row	year	no_of_orders
1	1	8069
2	2	8508
3	3	9893
4	4	9343
5	5	10573
6	6	9412
7	7	10318
8	8	10843
9	9	4305
10	10	4959
11	11	7544
12	12	5674

Inference:

By grouping on month, we can count the number of orders using count() aggregate function.
When we sort them in an order, a comparison analysis can be done.
We notice the presence of seasonality in only specific months where there is no holiday.

--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

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_of_the_day,
count(order_id) as no_of_orders

 FROM `scaler-dsml-sql-393406.Target_Analysis.orders`
```

```
group by time_of_the_day
order by no_of_orders
```

Result:

Query results SAVE RESU

JOB INFORMATION | RESULTS | JSON | EXECUTI

Row	time_of_the_day	no_of_orders
1	Dawn	5242
2	Mornings	27733
3	Night	28331
4	Afternoon	38135

Inference:
From the timestamp of orders, we can get the time of placing an order.
Using case when expression, we can check the condition if the time or order is during dawn or mornings or night or afternoon.
No: of orders can be counted using count() aggregate function by grouping according to the time lap.
Further strategical analysis can be done and company can pull extra add-ons during peak hours to multiply the sales.

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

--1. 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 monthly_orders,
format_datetime('%b', O.order_purchase_timestamp) as month,
count(O.order_id) as no_of_orders
FROM `scaler-dsml-sql-393406.Target_Analysis.orders` O
inner join
`scaler-dsml-sql-393406.Target_Analysis.customers` C
on C.customer_id = O.customer_id
group by 1,2,3
order by 1,2
```

Result:

JOB INFORMATION	RESULTS	JSON	EXECUTION DETAILS	CHART

Row	customer_state	monthly_orders	month	no_of_orders
1	AC	1	Jan	8
2	AC	2	Feb	6
3	AC	3	Mar	4
4	AC	4	Apr	9
5	AC	5	May	10
6	AC	6	Jun	7
7	AC	7	Jul	9
8	AC	8	Aug	7
9	AC	9	Sep	5
10	AC	10	Oct	6
11	AC	11	Nov	5
12	AC	12	Dec	5
13	AL	1	Jan	39
14	AL	2	Feb	39
15	AL	3	Mar	40
16	AL	4	Apr	51

Inference:
This analysis helps to get insights into customer purchase trends on state by state basis.
The state that has highest no. of orders in an given months or least no. of orders can be found out.

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

Query:

```
SELECT customer_state,
count(customer_unique_id) as no_of_customers
FROM `scaler-dsml-sql-393406.Target_Analysis.customers`
group by 1
order by 1
```

Result:

JOB INFORMATION RESULTS JSON

Row	customer_state	no_of_customers
1	AC	81
2	AL	413
3	AM	148
4	AP	68
5	BA	3380
6	CE	1336
7	DF	2140
8	ES	2033
9	GO	2020
10	MA	747
11	MG	11635

Inference:

Grouping the states and counting the no: of customers would help us know from which state the orders are being placed more.
Further orders placed can be calculated to improve the product patterns accordingly.

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

--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 cte1 as(
select
sum(p.payment_value) as orderval2017,
extract(year from o.order_purchase_timestamp) as year,

format_datetime('%b', o.order_purchase_timestamp) as month,
from `scaler-dsml-sql-393406.Target_Analysis.orders` o
join
`scaler-dsml-sql-393406.Target_Analysis.payments` p
on o.order_id = p.order_id
where extract(year from o.order_purchase_timestamp) = 2017
and extract(month from order_purchase_timestamp) between 1 and 8

group by 2,3
order by 2,3),
```

```
cte2 as (select
sum(p.payment_value) as orderval2018,
extract(year from o.order_purchase_timestamp) as year,

format_datetime('%b', o.order_purchase_timestamp) as month,
extract(month from o.order_purchase_timestamp) as monthnum
from `scaler-dsml-sql-393406.Target_Analysis.orders` o
join
`scaler-dsml-sql-393406.Target_Analysis.payments` p
on o.order_id = p.order_id
where extract(year from o.order_purchase_timestamp) = 2018
and extract(month from order_purchase_timestamp) between 1 and 8
group by 2,3,4
order by 2,3)

select
a.month,
((b.orderval2018-a.orderval2017)/a.orderval2017)*100 as
percent_change
from cte1 a inner join cte2 b
on a.month = b.month
order by b.monthnum
```

Result:

Row	month	percent_change
1	Jan	705.1266954171...
2	Feb	239.9918145445...
3	Mar	157.7786066709...
4	Apr	177.8407701149...
5	May	94.62734375677...
6	Jun	100.2596912456...
7	Jul	80.04245463390...
8	Aug	51.60600520477...

Inference:

Cte, common table expression is a temporary relational table which can be used later in a SQL statement.

The table is called temp because it exits only during the scope of the sql statement written after the CTE.

As attributes of 2 years are to be observed, 2 cte tables are used to filter the data.

Further % increase formula is applied to observe the change in the cost of orders.

January shows highest percentage change followed by Feb and April.

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

Query:

```
SELECT
c.customer_state,
round(sum(oi.price)) as sum_price,
round(avg(oi.price)) as avg_price,
FROM `scaler-dsml-sql-393406.Target_Analysis.order_items` oi
inner join
`scaler-dsml-sql-393406.Target_Analysis.orders` o
on o.order_id = oi.order_id
inner join
`scaler-dsml-sql-393406.Target_Analysis.customers` c
on o.customer_id = c.customer_id

group by c.customer_state
order by 1
```

Result:

JOB INFORMATION RESULTS JSON EXECUTION DETAIL

Row	customer_state	sum_price	avg_price
1	AC	15983.0	174.0
2	AL	80315.0	181.0
3	AM	22357.0	135.0
4	AP	13474.0	164.0
5	BA	511350.0	135.0
6	CE	227255.0	154.0
7	DF	302604.0	126.0
8	ES	275037.0	122.0
9	GO	294592.0	126.0
10	MA	119648.0	145.0
11	MG	1585308.0	121.0

Inference:
The avg and sum of order price can be calculated by joining the orders and customers table

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

Query:

```
SELECT
c.customer_state,
round(sum(oi.freight_value)) as sum_freight,
round(avg(oi.freight_value)) as avg_freight
FROM `scaler-dsml-sql-393406.Target_Analysis.order_items` oi
inner join
`scaler-dsml-sql-393406.Target_Analysis.orders` o
on o.order_id = oi.order_id
inner join
`scaler-dsml-sql-393406.Target_Analysis.customers` c
on o.customer_id = c.customer_id

group by c.customer_state
order by 1
```

Result:

Row	customer_state	sum_freight	avg_freight
1	AC	3687.0	40.0
2	AL	15915.0	36.0
3	AM	5479.0	33.0
4	AP	2789.0	34.0
5	BA	100157.0	26.0
6	CE	48352.0	33.0
7	DF	50625.0	21.0
8	ES	49765.0	22.0
9	GO	53115.0	23.0
10	MA	31524.0	38.0
11	MG	270853.0	21.0

Inference:
The avg and sum of freight value can be calculated by joining the orders and customers table

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

--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.
```

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 `scaler-dsml-sql-393406.Target_Analysis.orders`
where order_delivered_customer_date is not null
and order_purchase_timestamp is not null
and order_estimated_delivery_date is not null
and order_delivered_customer_date is not null
order by time_to_deliver desc
```

Result:

JOB INFORMATION RESULTS JSON EXECUTION DETAILS

Row	order_id	time_to_deliver	diff_estimated_delive
1	ca07593549f1816d26a572e06...	209	-181
2	1b3190b2dfa9d789e1f14c05b...	208	-188
3	440d0d17af552815d15a9e41a...	195	-165
4	0f4519c5f1c541ddec9f21b3bd...	194	-161
5	285ab9426d6982034523a855f...	194	-166
6	2fb597c2f772eca01b1f5c561b...	194	-155
7	47b40429ed8cce3aee9199792...	191	-175
8	2fe324febf907e3ea3f2aa9650...	189	-167
9	2d7561026d542c8dbd8f0daea...	188	-159
10	437222e3fd1b07396f1d9ba8c...	187	-144
11	c27815f7e3dd0b926b5855262...	187	-162
12	dfe5f68118c2576143240b8d7...	186	-153

Inference:

With the difference in the delivery time and estimated time, potential measures can be taken to enhance the fleet dispatching, route optimization and all activities that reduce chances of delays.

--2. Find out the top 5 states with the highest & lowest average freight value.

Query:

```
with del_time as
(SELECT g.geolocation_state,
avg(date_diff
```

```
(o.order_delivered_customer_date, o.order_purchase_timestamp, day))
as avg_delivery_time,
case when
dense_rank() over(order by
avg(date_diff
(o.order_delivered_customer_date, o.order_purchase_timestamp, day))
desc) <=5 then 'highest_delivery_time'
when
dense_rank() over(order by
avg(date_diff(o.order_delivered_customer_date,
o.order_purchase_timestamp, day)))<=5 then 'lowest_delivery_time'
end as delivery_time_rank

FROM `scaler-dsml-sql-393406.Target_Analysis.orders` o
inner join `scaler-dsml-sql-393406.Target_Analysis.customers` c
on o.customer_id = c.customer_id
inner join `scaler-dsml-sql-393406.Target_Analysis.geolocation` g
on g.geolocation_zip_code_prefix = c.customer_zip_code_prefix
group by 1)

select geolocation_state,
avg_delivery_time,
delivery_time_rank
from del_time
where delivery_time_rank is not null
order by del_time.avg_delivery_time
```

Result:

Row	geolocation_state	avg_delivery_time	delivery_time_rank
1	SP	8.470529714190...	lowest_delivery_time
2	PR	11.03876404770...	lowest_delivery_time
3	MG	11.41862683439...	lowest_delivery_time
4	DF	12.49651789233...	lowest_delivery_time
5	SC	14.49430832817...	lowest_delivery_time
6	PA	22.55023982441...	highest_delivery_time
7	AL	23.14352789271...	highest_delivery_time
8	RR	24.52060133630...	highest_delivery_time
9	AM	24.65119678421...	highest_delivery_time
10	AP	27.99122623772...	highest_delivery_time

Inference:

The result can be used to evaluate asset use, performance, baseline deviations and other focal points.

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

Query:

```sql
with del_time as
(SELECT g.geolocation_state,
avg(date_diff
(o.order_delivered_customer_date, o.order_purchase_timestamp, day))
as avg_delivery_time,
case when
dense_rank() over(order by
avg(date_diff
(o.order_delivered_customer_date, o.order_purchase_timestamp, day))
desc) <=5 then 'highest_delivery_time'
when
dense_rank() over(order by
avg(date_diff(o.order_delivered_customer_date,
o.order_purchase_timestamp, day)))<=5 then 'lowest_delivery_time'
end as delivery_time_rank

FROM `scaler-dsml-sql-393406.Target_Analysis.orders` o
inner join `scaler-dsml-sql-393406.Target_Analysis.customers` c
on o.customer_id = c.customer_id
inner join `scaler-dsml-sql-393406.Target_Analysis.geolocation` g
on g.geolocation_zip_code_prefix = c.customer_zip_code_prefix
group by 1)

select geolocation_state,
avg_delivery_time,
delivery_time_rank
from del_time
where delivery_time_rank is not null
order by del_time.avg_delivery_time
```

Result

Row	geolocation_state	avg_delivery_time	delivery_time_rank
1	SP	8.470529714190...	lowest_delivery_time
2	PR	11.03876404770...	lowest_delivery_time
3	MG	11.41862683439...	lowest_delivery_time
4	DF	12.49651789233...	lowest_delivery_time
5	SC	14.49430832817...	lowest_delivery_time
6	PA	22.55023982441...	highest_delivery_time
7	AL	23.14352789271...	highest_delivery_time
8	RR	24.52060133630...	highest_delivery_time
9	AM	24.65119678421...	highest_delivery_time
10	AP	27.99122623772...	highest_delivery_time

Inference:

Identifying the state through geolocation_state is preferred as customer_state always might not a valid value.

Cte is used as various tables are joined and subquering would affect the code readability.

--4. Find out the top 5 states where the order delivery is really fast as compared to the estimated date of delivery.

Query:

```
select c.customer_state,
avg(datetime_diff(order_estimated_delivery_date, order_delivered_customer_date, day)) as
fast_deliveries
from `scaler-dsml-sql-393406.Target_Analysis.orders` o
inner join `scaler-dsml-sql-393406.Target_Analysis.customers` c
on o.customer_id = c.customer_id
where order_delivered_customer_date is not null
group by 1
order by 2 desc
limit 5
```

Result:

Row	customer_state	fast_deliveries
1	AC	19.76250000000...
2	RO	19.13168724279...
3	AP	18.73134328358...
4	AM	18.60689655172...
5	RR	16.41463414634...

Inference:

The difference between estimated delivery and actual delivery would help to find the rate at which the orders are delivered.

Result:

Row	customer_state	fast_deliveries
1	AC	19.76250000000...
2	RO	19.13168724279...
3	AP	18.73134328358...
4	AM	18.60689655172...
5	RR	16.41463414634...

Inference:

Avg delivery time and avg estimated time are calculated by filtering according to the state.

--VI. Analysis based on the payments

--1. Find the month on month no. of orders placed using different payment types.

Query:

```
SELECT
p.payment_type,
count(o.order_id) as no_of_orders,
extract(month from o.order_purchase_timestamp) as month,
extract(year from o.order_purchase_timestamp) as year,
format_datetime('%b', o.order_purchase_timestamp) as month_name
FROM `scaler-dsml-sql-393406.Target_Analysis.payments` p
inner join
`scaler-dsml-sql-393406.Target_Analysis.orders` o
on p.order_id = o.order_id
group by 1,3,4,5
order by 1,3,4
```

Result:

Row	payment_type	no_of_orders	month	year	month_name
17	UPI	903	9	2017	Sep
18	UPI	63	10	2016	Oct
19	UPI	993	10	2017	Oct
20	UPI	1509	11	2017	Nov
21	UPI	1160	12	2017	Dec
22	credit_card	583	1	2017	Jan
23	credit_card	5520	1	2018	Jan
24	credit_card	1356	2	2017	Feb
25	credit_card	5253	2	2018	Feb
26	credit_card	2016	3	2017	Mar
27	credit_card	5691	3	2018	Mar

Inference:
To understand the trends in payment types, analysis on month-over-month count of orders for different payment types is done.
Using extract(), month and year are extracted and group statewise.

```
--2. Find the no. of orders placed on the basis of the payment installments that have been paid.
```

Query:

```sql
SELECT
p.payment_installments,
count(o.order_id) as order_count,

FROM `scaler-dsml-sql-393406.Target_Analysis.payments` p
inner join
`scaler-dsml-sql-393406.Target_Analysis.orders` o
on p.order_id = o.order_id
where p.payment_installments!=0
group by 1
order by 1
```

Result:

1	1	52546
2	2	12413
3	3	10461
4	4	7098
5	5	5239
6	6	3920
7	7	1626
8	8	4268
9	9	644
10	10	5328
11	11	23
12	12	133

Inference:

Status of the payments instalments can be found by joining orders and payments table.

But to find the customers who have paid an instalment, filtering and removing the non-paid customers can be obtained by passing condition payment_installments!=0 in where clause.