Context:

Target is a globally renowned brand and a prominent retailer in the United States. Target makes itself a preferred shopping destination by offering outstanding value, inspiration, innovation and an exceptional guest experience that no other retailer can deliver.

This particular business case focuses on the operations of Target in Brazil and provides insightful information about 100,000 orders placed between 2016 and 2018. The dataset offers a comprehensive view of various dimensions including the order status, price, payment and freight performance, customer location, product attributes, and customer reviews.

By analyzing this extensive dataset, it becomes possible to gain valuable insights into Target's operations in Brazil. The information can shed light on various aspects of the business, such as order processing, pricing strategies, payment and shipping efficiency, customer demographics, product characteristics, and customer satisfaction levels.

What does 'good' look like?

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

Importing the Dataset to **BigQuery** :

The dataset was imported via BigQuery on Google Cloud.

Here I read on how is BigQuery better than MySQL: https://lxadm.com/bigquery-vs-mysql/

Following are the steps to import data to BigQuery :

a. Add project
b. Create dataset - 3 dots on the project->create a dataset
c. Create table – 3 dots on the dataset->create table
d. Create a table from - click upload
e. File type - csv
f. Browse and upload and name it as it is
g. Toggle ON edit as text
h. Click on Create Table

1b. Time period for which the data is given:

Approach: The dataset contains an "orders" table where the we have purchase date of orders with their respective order Ids. Here, we need to find out the max and min of order purchase date to find out starting from what date till what date were the orders placed – used **min() and max() functions**.

Query:

```
SELECT min(order_purchase_timestamp) as Start_orders,

max(order_purchase_timestamp) as END_orders

FROM `naman-jain-target.TARGET.orders`;
```

Results:

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

Inference & Recommendation:

The data is ranging **from 4/09/2016** (dd/mm/yyyy) **to 17/10/2018** (dd/mm/yyyy).

1c. Cities and States of customers ordered during the given period:

Approach: Here, we need to find out all **distinct** cities and states of Brazil from where the customers have placed orders. To achieve that, the table orders can be grouped by cities and states followed by adding a count-of-customers column for each group of city and state.

Query:

```
--Cities and States of customers ordered during the given period
```

```
select customer_city, customer_state, count (*) as number_of_customers
from `naman-jain-target.TARGET.customers`
where customer_id in (select customer_id from `naman-jain-target.TARGET.orders`)
group by customer_city, customer_state
order by number_of_customers desc
```

Results:

Row	customer_city	customer_state	number_of_cust
1	sao paulo	SP	15540
2	rio de janeiro	RJ	6882
3	belo horizonte	MG	2773
4	brasilia	DF	2131
5	curitiba	PR	1521
6	campinas	SP	1444
7	porto alegre	RS	1379
8	salvador	BA	1245
9	guarulhos	SP	1189
10	sao bernardo do campo	SP	938

Inference & Recommendation: Here, we can see that Sao Paulo city from State SP had the highest orders amount the other city and state groups, followed by Rio de Janeiro.

2. In-depth Exploration:

2a. Is there a growing trend in e-commerce in Brazil? How can we describe a complete scenario? Can we see some seasonality with peaks at specific months?

Approach: So, here as we want to know the seasonality peaks we consider what was the revenue generated per month of each year and compare with each other. For ex. We compare revenue of Jan 2017 to revenue of Jan 2018. By seeing the 25 rows results we can easily understand that the season is highest in

Query:

```
SELECT Sum(p.payment_value) as rev_per_month, Extract(
month FROM o.order_purchase_timestamp) as month,
Extract(year FROM o.order_purchase_timestamp) as year
FROM `naman-jain-target.TARGET.orders` AS o
JOIN `naman-jain-target.TARGET.payments` AS p
ON o.order_id = p.order_id
GROUP BY month,year
order by month,year;
```

Results:

	rev_per_month	month	year
1	138488.0	1	2017
2	1115004.0	1	2018
3	291908.0	2	2017
4	992463.0	2	2018
5	449864.0	3	2017
6	1159652.0	3	2018
7	417788.0	4	2017
8	1160785.0	4	2018
9	592919.0	5	2017
10	1153982.0	5	2018

Inference & Recommendation: After looking at the numbers here, we can see that revenue for months in 2018 is more than revenue for months in 2017, except for some months after August.

2b. What time do Brazilian customers tend to buy (Dawn, Morning, Afternoon or Night)?

Approach: Here, we need to group the data of customers, by making groups based on time of the day as dawn(12 am to 7am), morning(7am to 12pm), afternoon(12pm to 8pm) and night(8pm to 12am). We basically just have created bin within the time of the day and dropped the respective order in the respective bins, followed by adding a count-of-customers column to compare within the bins.

Query:

```
SELECT
CASE
        WHEN h >= 0 AND h <= 7 THEN 'dawn' -- including 0
        WHEN h >7 AND h <= 12 THEN ‘morning’
        WHEN h > 12 AND h <= 20 THEN 'afternoon'
        WHEN h > 20 THEN 'night'
END AS Time_of_the_day ,count(*) AS num_of_cust
FROM (
         SELECT extract(hour from order_purchase_timestamp) AS h
         FROM `naman-jain-target.TARGET.orders`
         ) x
GROUP BY Time_of_the_day
ORDER BY count(*) DESC;
```

Results:

Row	Time_of_the_day	num_of_cust
1	afternoon	50310
2	morning	26502
3	night	16156
4	dawn	6473

Inference & Recommendation: Customers ordered the **most during the afternoon**, more marketing channels like emails and push notification can be enabled in the same time range while some other criteria has to be figured to raise the order flow in other time ranges during the day.

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

3a. Get month-on-month orders by states

Approach: We need number or orders place in a month of a year of a state. Here we have **grouped by year, month and state** and then counted number of orders for each row of this. We have used the **extract function** here as we do not have **MONTH()** and **YEAR()** functions in BIGQUERY. Hence, here we get the number of order for each state, for each month of each year.

Query:

```
SELECT extract(Year from order_purchase_timestamp) AS Year ,
extract(month from order_purchase_timestamp) AS month, COUNT(distinct order_id) AS Count_Of_Ord ers,
y.customer_state as state
FROM `naman-jain-target.TARGET.orders` x join `naman-jain
target.TARGET.customers` y on x.customer_id = y.customer_id
GROUP BY month, state
ORDER BY month ASC, state;
```

Results:

Row	Year	month	Count_Of_Order:	state
1	2016	9	1	RR
2	2016	9	1	RS
3	2016	9	2	SP
4	2016	10	2	AL
5	2016	10	4	BA
6	2016	10	8	CE
7	2016	10	6	DF
8	2016	10	4	ES
9	2016	10	9	GO
10	2016	10	4	MA
11	2016	10	40	MG
12	2016	10	3	MT
13	2016	10	4	PA

Inference & Recommendation: As we scroll down to the end of this results table, we see that the number of **orders have increased drastically over the months** and years, **except in the months from Sept 2016 to Oct 2018.**

3b. Distribution of customers across the states in Brazil

Approach: Distribution of customer across the states, could be broken down to number of customer in each state. Then percentage of customers in each state is calculated by using group by w.r.t states and total unique customers.

Query:

```
with t as -- gives out total uniques customers(used CTE)
(select count(customer_unique_id) x from `naman-jain-target.TARGET.customers`) --
generates percentage distribution of all uniques customers
select customer_state, round(((count(customer_unique_id)/t.x)*100),2) as unique_customers from
`naman-jain-target.TARGET.customers`, t
group by customer_state, t.x
order by unique_customers DESC;
```

Results:

	customer_state	unique_custome
1	SP	41.98
2	RJ	12.92
3	MG	11.7
4	RS	5.5
5	PR	5.07
6	SC	3.66
7	BA	3.4
8	DF	2.15
9	ES	2.04
10	GO	2.03

Inference & Recommendation: Approximately **42% of total customers** are based out of **state SP in Brazil**. The recommendation would be enhance the business via marketing and branding in the other potential states, while catering the major customer base in SP.

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

4a. Get % increase in cost of orders from 2017 to 2018 (include months between Jan to Aug only) - You can use "payment_value" column in payments table

Approach: First filtered all required data for year 2017 and 2018 and month from Jan to Aug. Calculated number of orders, total cost per month for each of those via group by month, year. Lastly, calculated the percentage increase in comparing every month total cost per month in 2017 and 2018.

Query:

```
select *,
round((((x18.avg_order_cost - x17.avg_order_cost)/(x17.avg_order_cost))*100)) as percent_inc_cost from
(select
Extract(year FROM o.order_purchase_timestamp) as year, -- year - 2017
Extract(month from o.order_purchase_timestamp) as month, -- month 1 to 8
count(distinct p.order_id) as num_of_orders, -- count of orders
round(avg(p.payment_value)) as avg_order_cost
FROM `naman-jain-target.TARGET.orders` AS o join `naman-jain-target.TARGET.payments` as p on
o.order_id = p.order_id
where Extract(year FROM o.order_purchase_timestamp) = 2017 -- filter
and Extract(month from o.order_purchase_timestamp) between 1 and 8 -- filter
group by year, month
order by year, month ASC) x17
join
(select
```

```
Extract(year FROM o.order_purchase_timestamp) as year, -- year - 2017 and 2018
Extract(month from o.order_purchase_timestamp) as month, -- month 1 to 8
count(distinct p.order_id) as num_of_orders, -- count of orders
round(avg(p.payment_value)) as avg_order_cost
FROM `naman-jain-target.TARGET.orders` AS o join `naman-jain-target.TARGET.payments` as p on
o.order_id = p.order_id
where Extract(year FROM o.order_purchase_timestamp) = 2018
and Extract(month from o.order_purchase_timestamp) between 1 and 8
group by year, month
order by year, month ASC) x18 on x17.month = x18.month order by x17.month
```

Results:

Row	year	month	num_of_orders	total_cost_per_n	year_1	month_1	num_of_orders_	total_cost_per_n	percent_inc_cos
1	2017	1	800	138488.0	2018	1	7269	1115004.0	705.0
2	2017	2	1780	291908.0	2018	2	6728	992463.0	240.0
3	2017	3	2682	449864.0	2018	3	7211	1159652.0	158.0
4	2017	4	2404	417788.0	2018	4	6939	1160785.0	178.0
5	2017	5	3700	592919.0	2018	5	6873	1153982.0	95.0
6	2017	6	3245	511276.0	2018	6	6167	1023880.0	100.0
7	2017	7	4026	592383.0	2018	7	6292	1066541.0	80.0
8	2017	8	4331	674396.0	2018	8	6512	1022425.0	52.0

Inference:

All 8 months from jan to aug of years 2017 and 2018 compared here on the basis of parameters:

Number of order - **num_of_orders - both years and all months**

Total cost per month **- total_cost_per_month - both years and all months**

Percentage increase in cost **- Percent_inc_cost –**

Compares cost of Every month of 2017 vs every month of 2018

Here, we can also infer that the **average percentage increased in cost on month on month** for the years 2017 and 2018 correspondingly is **200%**.

4b. Mean & Sum of price and freight value by customer state

Approach: Here, we first took the order_id from the order_items(which has price and freight value) table and found the corresponding customer_id for that order via orders table and finally joined these two with a another table which gave customer_state of the customer of order_id in order_items from customers table.

So, now after finally finding and joining table we took out customer state, price and freight value as the query output. We then grouped by customer state as we need avg of price, avg of freight value, sum of price and sum of freight value for each state.

Query:

```
select a.customer_state,
round(sum(c.price)) as sum_price,
```

```
round(sum(c.freight_value)) as sum_freight,
round(avg(c.price)) as avg_price,
round(avg(c.freight_value)) as avg_freight
from `naman-jain-target.TARGET.customers` a
join `naman-jain-target.TARGET.orders` b
        on a.customer_id=b.customer_id
join `naman-jain-target.TARGET.order_items` c
        on c.order_id = b.order_id
group by a.customer_state
order by customer_state;
```

Results:

	customer_state	sum_price	sum_freight	avg_price	avg_freight
1	AC	15983.0	3687.0	174.0	40.0
2	AL	80315.0	15915.0	181.0	36.0
3	AM	22357.0	5479.0	135.0	33.0
4	AP	13474.0	2789.0	164.0	34.0
5	BA	511350.0	100157.0	135.0	26.0
6	CE	227255.0	48352.0	154.0	33.0
7	DF	302604.0	50625.0	126.0	21.0
8	ES	275037.0	49765.0	122.0	22.0
9	GO	294592.0	53115.0	126.0	23.0
10	MA	119648.0	31524.0	145.0	38.0

Inference: We can derive that **average freight value per order** is **30.48** and similarly **average price per order** will be **145.22.** Hence, around **20% of price per order** is being given for the freight.

5.Analysis on sales, freight and delivery time

5a. Calculate days between purchasing, delivering and estimated delivery

Approach: Here, I have calculated the days taken to actually deliver a product for the carrier after the purchase order has been placed, and the number of days estimated for delivery using the order purchase timestamp and the carrier delivery date and the estimated delivery date. Here, I have not used the customer delivery date as the data.

Query:

```
select round(avg(days_for_carrier_delivery)) as avg_carrier_delivery,
round(avg(days_for_estimated_delivery)) as avg_estimated_delivery
from
(SELECT
DATE_DIFF(order_delivered_carrier_date,order_purchase_timestamp, day)
AS days_for_carrier_delivery, -- delivery_carrier – order_purchase
DATE_DIFF(order_estimated_delivery_date,order_purchase_timestamp, day)
AS days_for_estimated_delivery -- estimated_delivery – order_purchase
from `naman-jain-target.TARGET.orders`
```

```
where order_status = "shipped" ) x
```

Results:

Row	avg_actual_deliv	avg_estimated_(
1	3.0	25.0

Inference:

Here, we can easily infer that the **actual average time to deliver an order is ~ 3 days & the estimated delivery time for an order is ~ 25 days.**

5b. Find time_to_delivery & diff_estimated_delivery. Formula for the same given below:

- **time_to_delivery = order_purchase_timestamp order_delivered_customer_date**
- **diff_estimated_delivery = order_estimated_delivery_date order_delivered_customer_date**

Approach: Using the date and time functions like date_diff and converting the same to days, further rounding them up.

Query:

```
select
round((DATE_DIFF(order_delivered_customer_date,order_purchase_timestamp, hour)/24)) AS time
_to_delivery,
round((DATE_DIFF(order_estimated_delivery_date,order_delivered_customer_date, hour)/24))  AS
diff_estimated_delivery
from (select * from `naman-jain-target.TARGET.orders`
where order_delivered_customer_date is not null) x
```

Results:

	time_to_delivery	diff_estimated_(
1	168	1088
2	722	-310
3	743	681
4	181	1065
5	262	989
6	853	397
7	565	228
8	311	-133
9	309	298
10	173	24

5c.Group data by state, take mean of freight_value, time_to_delivery, diff_estimated_delivery

Approach: Here, step 1 is to join the order_items with orders to get the customer_id which will be in-turn used to get the customer_state for the order_id in the order_items table and then we grouped the data by the state where the joining was done based on customer and order ID from orders and customers table. After grouping we added average of freight value column for each state, followed by adding average time_to_delivery and average diff_estimated_delivery columns to each group.

Query:

```
select a.customer_state, -- state
round(avg(c.freight_value)) as avg_freight, -- mean of freight value
round(avg(d.time_to_delivery)) as avg_time_to_delivery, -- mean of time to delivery
round(avg(d.diff_estimated_delivery)) as avg_diff_estimated_delivery -- mean diff estimated delivery from
`naman-jain-target.TARGET.customers` a -- for join over customer id
join `naman-jain-target.TARGET.orders` b -- for join over customer id
on a.customer_id=b.customer_id
join `naman-jain-target.TARGET.order_items` c -- join over order_id
on c.order_id = b.order_id
join --over order_id
 (select order_id,
 (DATE_DIFF(order_delivered_customer_date,order_purchase_timestamp, hour)/24)  AS
time_to_delivery, --- will give out time_to_delivery in days
 (DATE_DIFF(order_estimated_delivery_date,order_delivered_customer_date, hour)/24)  AS
diff_estimated_delivery --- will give out diff_estimated_delivery in days
 from (select * from `naman-jain-target.TARGET.orders`
 where order_delivered_customer_date is not null)) d
on c.order_id=d.order_id
group by a.customer_state -- all data will be grouped by states
order by customer_state
```

27 STATES -> 27 ROWS OF DATA

Results: in hours

	customer_state	avg_freight	avg_time_to_del	avg_diff_estimat
1	AC	40.0	497.0	488.0
2	AL	36.0	587.0	193.0
3	AM	33.0	633.0	461.0
4	AP	34.0	676.0	426.0
5	BA	26.0	461.0	247.0
6	CE	33.0	503.0	250.0
7	DF	21.0	311.0	275.0
8	ES	22.0	375.0	238.0
9	GO	23.0	369.0	278.0
10	MA	38.0	519.0	221.0

Sort the data to get the following:
5d. Top 5 states with highest/lowest average freight value - sort in desc/asc limit 5

Query:

```
select a.customer_state, round(avg(c.freight_value)) as avg_freight
from `naman-jain-target.TARGET.customers` a join `naman-jain-target.TARGET.orders` b on
a.customer_id=b.customer_id
join `naman-jain-target.TARGET.order_items` c
on c.order_id = b.order_id
group by a.customer_state
order by avg_freight DESC – descending order
LIMIT 5 – top 5
```

Results: Highest 5 states

	customer_state	avg_freight
1	PB	43.0
2	RR	43.0
3	RO	41.0
4	AC	40.0
5	PI	39.0

customer_state

Inference: These states have the **highest freight value**, hence here we can suggest the company to come up new **supply chain solutions** or **better warehousing** facilities to reduce the same.

5e. Top 5 states with highest/lowest average time to delivery

Query:

```
select a.customer_state, -- state
ROUND(avg(d.time_to_delivery)) as avg_time_to_delivery -- mean of time to delivery from
`naman-jain-target.TARGET.customers` a -- for join over customer id
join `naman-jain-target.TARGET.orders` b -- for join over customer id
on a.customer_id=b.customer_id
join --over order_id
 (select order_id,
 DATE_DIFF(order_delivered_customer_date,order_purchase_timestamp, hour)   AS
time_to_delivery --- will give out time_to_delivery
 from (select * from `naman-jain-target.TARGET.orders`
 where order_delivered_customer_date is not null)) d
on b.order_id=d.order_id
group by a.customer_state -- all data will be grouped by states
order by avg_time_to_delivery DESC
limit 5
```

Results:

	customer_state	avg_time_to_del
1	RR	705.0
2	AP	652.0
3	AM	634.0
4	AL	589.0
5	PA	570.0

Inference: Here, we have the average time to delivery per state **in hours**, and sorted by the same.

5f. Top 5 states where delivery is really fast/ not so fast compared to estimated date

Approach: The approach for the **joining** of tables was shared earlier, but here also need to sort. Now sorting, sort time to delivery in **descending** and estimated time in **ascending – basically highest difference between the two**.

Query:

```
select a.customer_state, -- state
 round(avg(c.freight_value)) as avg_freight, -- mean of freight value
 round(avg(d.time_to_delivery)) as avg_time_to_delivery, -- mean of time to delivery
round(avg(d.diff_estimated_delivery)) as avg_diff_estimated_delivery -- mean diff estimated delivery from
`naman-jain-target.TARGET.customers` a -- for join over customer id
join `naman-jain-target.TARGET.orders` b -- for join over customer id
on a.customer_id=b.customer_id
join `naman-jain-target.TARGET.order_items` c -- join over order_id
on c.order_id = b.order_id
join --over order_id
 (select order_id,
 (DATE_DIFF(order_delivered_customer_date,order_purchase_timestamp, hour)/24)  AS
time_to_delivery, --- will give out time_to_delivery in days
 (DATE_DIFF(order_estimated_delivery_date,order_delivered_customer_date, hour)/24)  AS
diff_estimated_delivery --- will give out diff_estimated_delivery in days
 from (select * from `naman-jain-target.TARGET.orders`
 where order_delivered_customer_date is not null)) d
on c.order_id=d.order_id
 group by a.customer_state -- all data will be grouped by states
order by avg_time_to_delivery DESC, avg_diff_estimated_delivery
-- high delivery time compared to estimated delivery time
limit 5
```

Results:

	customer_state	avg_freight	avg_time_to_del	avg_diff_estimat
1	AP	34.0	28.0	18.0
2	RR	43.0	28.0	18.0
3	AM	33.0	26.0	19.0
4	AL	36.0	24.0	8.0
5	PA	36.0	24.0	14.0

Inference: We can infer here that the deliveries in these states are slower than the expected ones.

6. Payment type analysis:

6a. Month over Month count of orders for different payment types

Approach: To get the **date of an order(for month) and the payment type** for the same would mean **joining orders and payments** over the order_id and querying the month from the date using the **extract function**. Then **grouping based on the months** and the payment type we add the **count of orders column** in order to get the number of order for a month in a specific payment type.

Query:

```
select Extract(month from o.order_purchase_timestamp) as month, -- month
p.payment_type, -- upi/credit_card/debit_card/voucher/not_defined
count(o.order_id) as count_order – number of orders
from `naman-jain-target.TARGET.orders` o
join `naman-jain-target.TARGET.payments` p
on o.order_id = p.order_id
group by month, payment_type
order by month, count_order desc
```

Inference: We notice from the query results that credit card was most used payment method over the months while voucher, Debit card ranked accordingly.

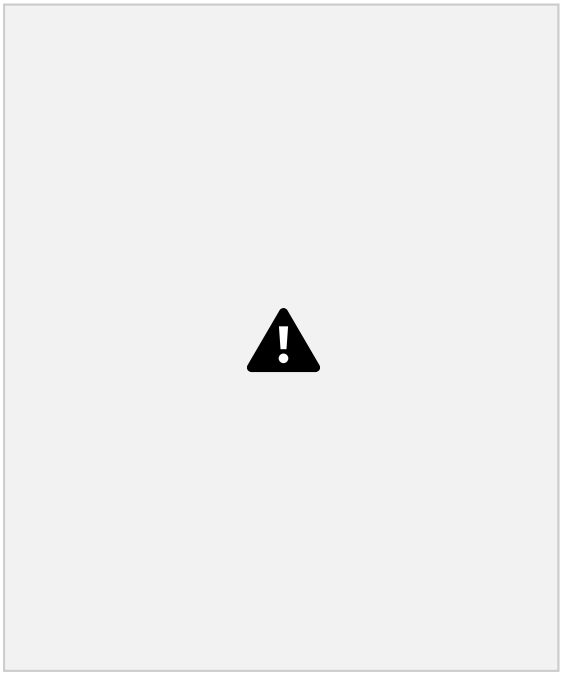
6b. Count of orders based on the no. of payment installments

Approach: Here, we just have to get the distinct **number-of-installments** by which orders are being placed and group by the same alongside creating a new column for counting the number of orders placed via that **number-of-installments.**

Query:

```
select distinct payment_installments,
count(order_id) over(partition by payment_installments order by payment_installments) as num_of_orders -- window
function
from `naman-jain-target.TARGET.payments`
```

Results:



Inference: Here we can infer that the **maximum number of orders where are with 1 installment** i.e, upfront lumpsum payment which is a **good sign** for a brand or company.