Target Business Case Study

🛒 Target Brazil E-Commerce Case Study

📌 Project Overview

Analyzed 100,000+ e-commerce orders from Target Brazil (2016–2018) to uncover operational insights across order processing, pricing, payments, freight, customer behavior, and delivery performance.

🎯 Objectivess

As a data analyst at Target, the goal was to extract actionable insights from a multi-table dataset and recommend improvements in logistics, pricing, and customer experience.

🗂 Dataset Summary

- Source: Google Drive Folder
- Format: 8 CSV files
- Tables: orders, order_items, customers, sellers, payments, reviews, products, geolocation

🔍 Key Explorations & Insights

▯ Exploratory Analysis

- Identified data types and structure across all tables
- Mapped customer distribution by city and state
- Extracted order time range and seasonal patterns

📈 Order Trends

- Analyzed yearly and monthly growth in order volume
- Segmented order times into Dawn, Morning, Afternoon, Night
- Tracked monthly order counts per state

💰 Economic Impact

- Calculated % increase in payment value from Jan–Aug (2017 vs 2018)
- Aggregated total & average order price and freight by state

🚚 Delivery Performance

- Computed delivery time and deviation from estimated delivery
- Ranked states by:
- Highest/lowest average freight cost
- Fastest/slowest delivery time
- Most efficient delivery vs estimated date

💳 Payment Analysis

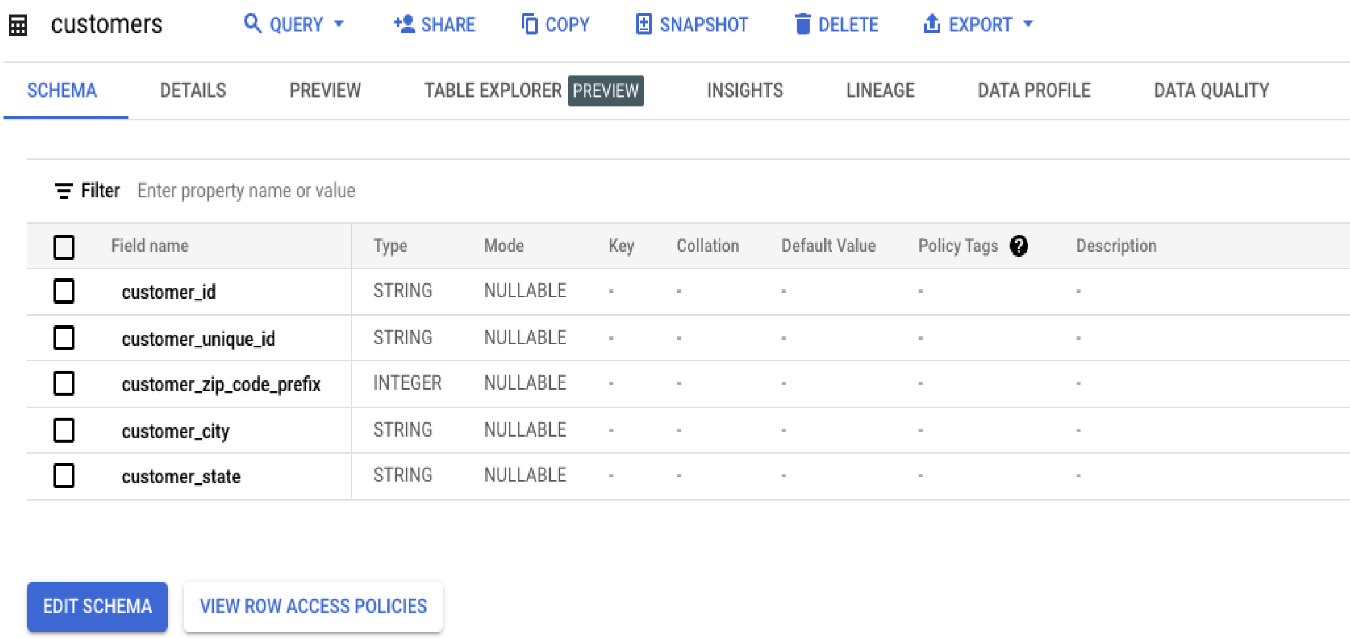
- Monthly breakdown of payment types used
- Distribution of orders by installment count

🛠 Tools & Skills Used

- Python (pandas, NumPy, Seaborn)
- Data cleaning & transformation
- Time-series analysis
- Geo-mapping & state-level aggregation
- Business storytelling with data

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

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


customers QUERY SHARE COPY SNAPSHOT DELETE EXPORT

SCHEMA DETAILS PREVIEW TABLE EXPLORER PREVIEW INSIGHTS LINEAGE DATA PROFILE DATA QUALITY

Filter Enter property name or value

Field name	Type	Mode	Key	Collation	Default Value	Policy Tags	Description
customer_id	STRING	NULLABLE	-	-	-	-	-
customer_unique_id	STRING	NULLABLE	-	-	-	-	-
customer_zip_code_prefix	INTEGER	NULLABLE	-	-	-	-	-
customer_city	STRING	NULLABLE	-	-	-	-	-
customer_state	STRING	NULLABLE	-	-	-	-	-

EDIT SCHEMA VIEW ROW ACCESS POLICIES

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

```
SELECT MIN(order_purchase_timestamp) AS order_start_date,
     MAX(order_purchase_timestamp) AS order_end_date,
```

```sql
FROM `TARGET_SQL.orders`
```

Query results

JOB INFORMATION | RESULTS | CHART | JSON | EXECUTION DETAILS | EXECUTION GRAPH

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

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

```sql
WITH min_max_dates AS (
   SELECT
       MIN(order_purchase_timestamp) AS min_date,
       MAX(order_purchase_timestamp) AS max_date
   FROM `TARGET_SQL.orders`
)
SELECT COUNT(DISTINCT cust.customer_id) AS cities_state_count
FROM `TARGET_SQL.customers` AS cust
JOIN `TARGET_SQL.orders` AS ord
ON cust.customer_id = ord.customer_id
JOIN min_max_dates AS dates
ON ord.order_purchase_timestamp BETWEEN dates.min_date AND dates.max_date;
```

Query results

JOB INFORMATION | RESULTS | CHART | JSON | EXECUTION DETAILS | EXECUTION GRAPH

Row	cities_state_count
1	99441

In-depth Exploration:

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

```sql
WITH order_data AS (
   SELECT
       order_id,
       order_purchase_timestamp,
       FORMAT_DATETIME('%Y-%m', order_purchase_timestamp) AS past_years
   FROM `TARGET_SQL.orders`
)
SELECT
   past_years,
   COUNT(*) AS order_month_count,
   COUNT(*)-LAG(COUNT(*)) OVER(ORDER BY past_years) AS growth_trend
```

```sql
FROM order_data
GROUP BY past_years
ORDER BY past_years
LIMIT 10
```

Query results

JOB INFORMATION | RESULTS | CHART | JSON | EXECUTION DETAILS | EXECUTION GRAPH

Row	past_years	order_month_count	growth_trend
1	2016-09	4	null
2	2016-10	324	320
3	2016-12	1	-323
4	2017-01	800	799
5	2017-02	1780	980
6	2017-03	2682	902
7	2017-04	2404	-278
8	2017-05	3700	1296
9	2017-06	3245	-455
10	2017-07	4026	781

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

```sql
WITH order_data AS (
   SELECT
       FORMAT_DATETIME('%m', order_purchase_timestamp) AS past_month,
       COUNT(order_id) AS order_month_count
   FROM `TARGET_SQL.orders`
   GROUP BY past_month
)
SELECT
   past_month,
   AVG(order_month_count) OVER() AS avg_month_count,
   order_month_count - ROUND(AVG(order_month_count) OVER(), 2) AS
monthly_seasonal
FROM order_data
ORDER BY past_month
```

Query results

JOB INFORMATION | RESULTS | CHART | JSON | EXECUTION DETAILS | EXECUTION GRAPH

Row	past_month	avg_month_count	monthly_seasonal
1	01	8286.749999999...	-217.75
2	02	8286.749999999...	221.25
3	03	8286.749999999...	1606.25
4	04	8286.749999999...	1056.25
5	05	8286.749999999...	2286.25
6	06	8286.749999999...	1125.25
7	07	8286.749999999...	2031.25
8	08	8286.749999999...	2556.25
9	09	8286.749999999...	-3981.75
10	10	8286.749999999...	-3327.75
11	11	8286.749999999...	-742.75
12	12	8286.749999999...	-2612.75

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

```
SELECT order_purchase_timezone, count(order_purchase_timezone) FROM (
SELECT order_purchase_timestamp,
CASE
 WHEN extract(hour FROM order_purchase_timestamp) between 0 and 6 THEN 'Dawan'
 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'
 ELSE 'Night'
END AS order_purchase_timezone
FROM `TARGET_SQL.orders`)
GROUP BY order_purchase_timezone
```

Query results

JOB INFORMATION | RESULTS | CHART | JSON | EXECUTION DETAILS | EXECUTION GRAPH

Row	order_purchase_timezone	f0_
1	Mornings	27733
2	Dawan	5242
3	Afternoon	38135
4	Night	28331

Evolution of E-commerce orders in the Brazil region:

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

```
SELECT geo.geolocation_state,
   extract(month FROM  ord.order_purchase_timestamp) AS order_month,
   count(ord.order_id) AS month_on_month
FROM `TARGET_SQL.geolocation` AS geo
JOIN `TARGET_SQL.customers` AS cust
 ON geo.geolocation_zip_code_prefix = cust.customer_zip_code_prefix
JOIN `TARGET_SQL.orders` AS ord
 ON  cust.customer_id = ord.customer_id
GROUP BY geo.geolocation_state, order_month
order by geo.geolocation_state, order_month
limit 15
```

Query results

JOB INFORMATION | RESULTS | CHART | JSON | EXECUTION DETAILS | EXECUTION GRAPH

Row	geolocation_state	order_month	month_on_month
1	AC	1	694
2	AC	2	515
3	AC	3	516
4	AC	4	789
5	AC	5	1161
6	AC	6	563
7	AC	7	937
8	AC	8	1060
9	AC	9	161
10	AC	10	535
11	AC	11	368
12	AC	12	389
13	AL	1	3645
14	AL	2	2902
15	AL	3	5279

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

```
SELECT geo.geolocation_state,count(distinct cust.customer_id) AS no_customers
FROM `TARGET_SQL.geolocation` AS geo
JOIN `TARGET_SQL.customers` AS cust
 ON geo.geolocation_zip_code_prefix = cust.customer_zip_code_prefix
GROUP BY geo.geolocation_state
limit 10
```

Query results

JOB INFORMATION | RESULTS | CHART | JSON | EXECUTION DETAILS | EXECUTION GRAPH

Row	geolocation_state	no_customers
1	SE	349
2	AL	412
3	PI	492
4	AP	68
5	AM	148
6	RR	46
7	AC	120
8	RO	256
9	TO	279
10	BA	3371

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).

```
WITH monthly_avg_payment AS (
   SELECT
       EXTRACT(YEAR FROM ord.order_purchase_timestamp) AS year_x,
       EXTRACT(MONTH FROM ord.order_purchase_timestamp) AS month_x,
       AVG(pmt.payment_value) AS avg_payment
   FROM `TARGET_SQL.orders` AS ord
   JOIN `TARGET_SQL.payments` AS pmt
       ON ord.order_id = pmt.order_id
   WHERE EXTRACT(YEAR FROM ord.order_purchase_timestamp) IN (2017, 2018)
       AND EXTRACT(MONTH FROM ord.order_purchase_timestamp) BETWEEN 1 AND 8
   GROUP BY year_x, month_x
)
SELECT
   year_x,
   month_x,
   avg_payment,
   ROUND(
       (avg_payment - LAG(avg_payment) OVER (ORDER BY year_x, month_x)) * 100
       / LAG(avg_payment) OVER (ORDER BY year_x, month_x), 2) AS percentage_increase
FROM monthly_avg_payment
ORDER BY year_x, month_x;
```

JOB INFORMATION	RESULTS	CHART	JSON	EXECUTION DETAILS	EXECUTION GRAPH

Row	year_x	month_x	avg_payment	percentage_incre...
1	2017	1	162.9271058823...	null
2	2017	2	154.7762513255...	-5.0
3	2017	3	158.57017976736	2.45
4	2017	4	162.5002061454...	2.48
5	2017	5	150.3343864097...	-7.49
6	2017	6	148.7998777648...	-1.02
7	2017	7	137.2209682649...	-7.78
8	2017	8	148.2189714285...	8.01
9	2018	1	147.4288218960...	-0.53
10	2018	2	142.7593987341...	-3.17
11	2018	3	154.3732854100...	8.14
12	2018	4	161.0189318906...	4.3
13	2018	5	161.7354099509...	0.44
14	2018	6	159.5077893752...	-1.38
15	2018	7	163.9066774243...	2.76
16	2018	8	152.6463601074...	-6.87

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

```
SELECT
   geo.geolocation_state,
   SUM(pmt.payment_value) AS total_price,
   AVG(pmt.payment_value) AS avg_price
FROM `TARGET_SQL.geolocation` AS geo
JOIN `TARGET_SQL.customers` AS cust
   ON geo.geolocation_zip_code_prefix = cust.customer_zip_code_prefix
JOIN `TARGET_SQL.orders` AS ord
   ON ord.customer_id = cust.customer_id
JOIN `TARGET_SQL.payments` AS pmt
   ON ord.order_id = pmt.order_id
GROUP BY geo.geolocation_state
ORDER BY total_price DESC
LIMIT 10;
```

Query results

JOB INFORMATION	RESULTS	CHART	JSON	EXECUTION DETAILS	EXECUTION GRAPH

Row	geolocation_state	total_price	avg_price
1	SP	819324973.1272...	139.1800016902...
2	RJ	516214891.0892...	163.3892353243...
3	MG	468418138.0098...	155.9394130293...
4	RS	131886278.0300...	158.3569008647...
5	PR	101367928.9499...	155.6463115200...
6	SC	96577779.98002...	174.3123905423...
7	BA	74143589.03002...	190.0976312787...
8	ES	51507204.52001...	157.0576229985...
9	MT	27075810.91000...	211.8707522262...
10	GO	24572389.12999...	178.1046716582...

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

```sql
SELECT
     geo.geolocation_state,
     SUM(orditm.freight_value) as total_freight_value,
     AVG(orditm.freight_value) as avg_freight_value
FROM `TARGET_SQL.order_items` AS orditm
JOIN `TARGET_SQL.orders` AS ord
 ON orditm.order_id = ord.order_id
JOIN `TARGET_SQL.customers` AS cust
 ON ord.customer_id = cust.customer_id
JOIN `TARGET_SQL.geolocation` AS geo
 ON cust.customer_zip_code_prefix = geo.geolocation_zip_code_prefix
GROUP BY geo.geolocation_state
LIMIT 10
```

Query results

JOB INFORMATION　RESULTS　CHART　JSON　EXECUTION DETAILS　EXECUTION GRAPH

Row	geolocation_state	total_freight_value	avg_freight_value
1	MT	4177068.029999...	28.72475728422...
2	MA	2275191.8599997	38.07533863275...
3	AL	1237356.220000...	33.83250540015...
4	SP	98574572.42980...	15.40996507007...
5	MG	67058347.09041...	20.45899544954...
6	PE	4195977.719998...	32.86555067321...
7	RJ	71966793.75011...	20.89842360439...
8	DF	2214955.550000...	21.01097098246...
9	RS	19910834.35000...	21.52222484648...
10	SE	943582.8300000...	34.67269897846...

Analysis based on sales, freight and delivery time.

A. Find the no. of days taken to deliver each order from the order's purchase date as delivery time.

Also, calculate the difference (in days) between the estimated & actual delivery date of an order. Do this in a single query.

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

```sql
SELECT
     order_id,
     customer_id,
```

```
    date_diff(order_delivered_customer_date, order_purchase_timestamp, day) as
time_to_deliver,
    date_diff(order_delivered_customer_date, order_estimated_delivery_date, day) as
diff_estimated_delivery
FROM `TARGET_SQL.orders`
WHERE order_delivered_customer_date IS NOT NULL
LIMIT 10;
```

Query results

JOB INFORMATION | RESULTS | CHART | JSON | EXECUTION DETAILS | EXECUTION GRAPH

Row	order_id	customer_id	time_to_deliver	diff_estimated_d...
1	770d331c84e5b214bd9dc70a1...	6c57e6119369185e575b36712...	7	-45
2	1950d777989f6a877539f53795...	1bccb206de9f0f25adc6871a1b...	30	12
3	2c45c33d2f9cb8ff8b1c86cc28...	de4caa97afa80c8eeac2ff4c8da...	30	-28
4	dabf2b0e35b423f94618bf965fc...	5cdec0bb8cbdf53ffc8fdc212cd...	7	-44
5	8beb59392e21af5eb9547ae1a9...	bf609b5741f71697f65ce3852c...	10	-41
6	65d1e226dfaeb8cdc42f665422...	70fc57eeae292675927697fe03...	35	-16
7	c158e9806f85a33877bdfd4f60...	25456ee3b0cf84658015e4668...	23	-9
8	b60b53ad0bb7dacacf2989fe27...	2f9902d85fcd930227f711cf47...	12	5
9	c830f223aae08493ebecb52f29...	af626bcc9c27c08077b02e6d3a...	12	-12
10	a8aa2cd070eeac7e4368cae3d...	2c5519c36277c3f69df911c68c...	7	-1

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

```
WITH freight_value AS(
 SELECT
    geo.geolocation_state as state,
    AVG(orditm.freight_value) as avg_freight_value
FROM `TARGET_SQL.order_items` AS orditm
JOIN `TARGET_SQL.orders` AS ord
ON orditm.order_id = ord.order_id
JOIN `TARGET_SQL.customers` AS cust
ON ord.customer_id = cust.customer_id
JOIN `TARGET_SQL.geolocation` AS geo
ON cust.customer_zip_code_prefix = geo.geolocation_zip_code_prefix
GROUP BY geo.geolocation_state
)
-- HIGHEST
(SELECT *
FROM freight_value
ORDER BY avg_freight_value
LIMIT 5)
UNION ALL
-- Lowest
(SELECT *
```

```
FROM freight_value
ORDER BY avg_freight_value DESC
LIMIT 5)
```

Query results

JOB INFORMATION RESULTS CHART JSON

Row	state	avg_freight_value
1	PB	42.77269312387...
2	RR	42.46960182496...
3	PI	39.47732502831...
4	AC	39.09837253960...
5	MA	38.07533863275...
6	SP	15.40996507007...
7	PR	20.14798071500...
8	MG	20.45899544954...
9	RJ	20.89842360439...
10	DF	21.01097098246...

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

```
WITH high_low as (
SELECT
    geo.geolocation_state as state,
    AVG(DATE_DIFF(ord.order_estimated_delivery_date, ord.order_delivered_customer_date,
day)) as diff_estimated_delivery
FROM `TARGET_SQL.orders` AS ord
JOIN `TARGET_SQL.customers` AS cust
ON ord.customer_id = cust.customer_id
JOIN `TARGET_SQL.geolocation` AS geo
ON cust.customer_zip_code_prefix = geo.geolocation_zip_code_prefix
GROUP BY geo.geolocation_state
)

(SELECT state, diff_estimated_delivery
FROM high_low
ORDER BY diff_estimated_delivery DESC
LIMIT 5)
UNION ALL
(SELECT state, diff_estimated_delivery
FROM high_low
ORDER BY diff_estimated_delivery ASC
LIMIT 5)
```

Query results

JOB INFORMATION	RESULTS	CHART	JSON

Row	state	diff_estimated_d...
1	AL	8.200633858614...
2	SE	8.485723897228...
3	MA	8.842814664110...
4	CE	9.720130879345...
5	ES	9.855011626562...
6	RR	20.42037861915...
7	AM	20.13265119678...
8	RO	18.65209721677...
9	AC	18.46145667198...
10	AP	18.18257781491...

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.

```
with top_five as (
SELECT
    geo.geolocation_state as state,
    AVG(DATE_DIFF(ord.order_estimated_delivery_date, ord.order_purchase_timestamp,
day)) as est_date,
    AVG(DATE_DIFF(ord.order_delivered_customer_date, ord.order_purchase_timestamp,
day)) as delivered_date
FROM `TARGET_SQL.orders` AS ord
JOIN `TARGET_SQL.customers` AS cust
ON ord.customer_id = cust.customer_id
JOIN `TARGET_SQL.geolocation` AS geo
ON cust.customer_zip_code_prefix = geo.geolocation_zip_code_prefix
GROUP BY geo.geolocation_state
)

select top_five.state, top_five.delivered_date
from top_five
where top_five.delivered_date < top_five.est_date
ORDER BY top_five.delivered_date DESC
limit 5
```

Query results

JOB INFORMATION | RESULTS | CHART | JSON | E

Row	state	delivered_date
1	AP	27.99122623772...
2	AM	24.65119678421...
3	RR	24.52060133630...
4	AL	23.14352789271...
5	PA	22.55023982441...

Analysis based on the payments

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

```
SELECT  pmt.payment_type,
EXTRACT(YEAR FROM ord.order_purchase_timestamp) AS year_x,
EXTRACT(MONTH FROM ord.order_purchase_timestamp) AS month_x,
COUNT(pmt.order_id) AS order_count
FROM `TARGET_SQL.payments` AS pmt
JOIN `TARGET_SQL.orders` AS ord
ON pmt.order_id = ord.order_id
GROUP BY pmt.payment_type ,EXTRACT(YEAR FROM ord.order_purchase_timestamp),
EXTRACT(MONTH FROM ord.order_purchase_timestamp)
ORDER BY pmt.payment_type, year_x, month_x
```

Query results

JOB INFORMATION | RESULTS | CHART | JSON | EXECUTION DETAILS | EXECUTION GRAPH

Row	payment_type	year_x	month_x	order_count
1	UPI	2016	10	63
2	UPI	2017	1	197
3	UPI	2017	2	398
4	UPI	2017	3	590
5	UPI	2017	4	496
6	UPI	2017	5	772
7	UPI	2017	6	707
8	UPI	2017	7	845
9	UPI	2017	8	938
10	UPI	2017	9	903
11	UPI	2017	10	993
12	UPI	2017	11	1509
13	UPI	2017	12	1160
14	UPI	2018	1	1518
15	UPI	2018	2	1325
16	UPI	2018	3	1352
17	UPI	2018	4	1287
18	UPI	2018	5	1263
19	UPI	2018	6	1100
20	UPI	2018	7	1229

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

```sql
SELECT payment_type, COUNT(DISTINCT order_id) AS order_count
FROM `TARGET_SQL.payments`
WHERE payment_installments > 0
GROUP BY payment_type
```

Query results

JOB INFORMATION | RESULTS | CHART | JSON

Row	payment_type	order_count
1	voucher	3866
2	not_defined	3
3	credit_card	76503
4	debit_card	1528
5	UPI	19784