

# Case Study: E-commerce Data Analysis for Target

By Shubham V Gupta

19-01-2024

## INTRODUCTION

This case study involves analyzing an e-commerce dataset to extract insights and provide actionable recommendations to Target. The dataset contains details on customer demographics, order history, and payment information. The aim of this analysis is to assess trends in order volumes, customer behavior, payment methods, and delivery performance, with a focus on the Brazilian region.

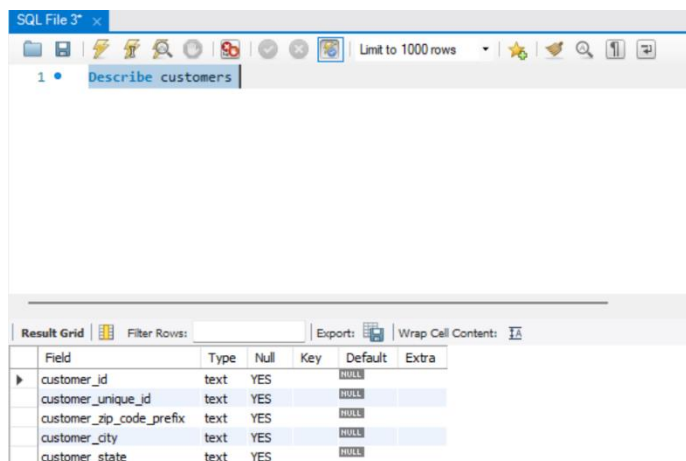
### Problem Statement:

Assuming you are a data analyst/ scientist at Target, you have been assigned the task of analyzing the given dataset to extract valuable insights and provide actionable recommendations.

### What does 'good' look like?

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

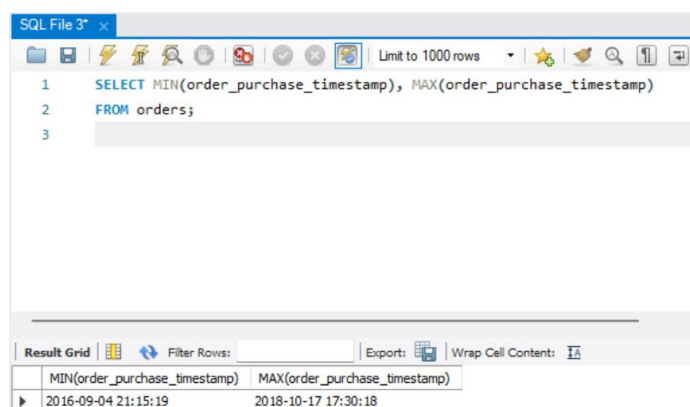
1. Data type of all columns in the "customers" table.



The screenshot shows a SQL interface with a toolbar at the top. Below the toolbar, there is a tab labeled "Describe customers". The main area displays a table structure for the 'customers' table. The table has six columns: customer\_id, customer\_unique\_id, customer\_zip\_code\_prefix, customer\_city, and customer\_state. Each column is of type 'text' and is nullable (YES). The 'customer\_id' column is the primary key (Key). The 'Default' column for all fields is 'NULL'.

Field	Type	Null	Key	Default	Extra
customer_id	text	YES	YES	NULL	
customer_unique_id	text	YES		NULL	
customer_zip_code_prefix	text	YES		NULL	
customer_city	text	YES		NULL	
customer_state	text	YES		NULL	

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



The screenshot shows a SQL interface with a toolbar at the top. Below the toolbar, there is a query editor with the following SQL code:

```
1 SELECT MIN(order_purchase_timestamp), MAX(order_purchase_timestamp)
2 FROM orders;
3
```

The main area displays the result of the query. The result is a single row with two columns: MIN(order\_purchase\_timestamp) and MAX(order\_purchase\_timestamp). The values are 2016-09-04 21:15:19 and 2018-10-17 17:30:18 respectively.

MIN(order_purchase_timestamp)	MAX(order_purchase_timestamp)
2016-09-04 21:15:19	2018-10-17 17:30:18

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

```
SQL File 3* x
1 • SELECT COUNT(DISTINCT customer_city) AS city_count,
2       COUNT(DISTINCT customer_state) AS state_count
3 FROM customers
4 JOIN orders ON customers.customer_id = orders.customer_id;
5
```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
city_count	state_count			
4119	27			

## 2. In-depth Exploration:

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

```
SQL File 3* x
1 • SELECT YEAR(order_purchase_timestamp) AS year, COUNT(*) AS total_orders FROM orders GROUP BY year;
2
```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
year	total_orders			
2017	45101			
2018	54011			
2016	329			

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

```
SQL File 3* x
1 • SELECT YEAR(order_purchase_timestamp) AS year, MONTH(order_purchase_timestamp) AS month, COUNT(*) AS total_orders
2 FROM orders
3 GROUP BY year, month;
```

Result Grid			Filter Rows:	Export:	Wrap Cell Content:
year	month	total_orders			
2017	10	4631			
2018	7	6292			
2018	8	6512			
2017	11	7544			
2018	2	6728			
2017	7	4026			
2017	4	2404			
2017	5	3700			
2017	1	800			
2018	6	6167			
2018	3	7211			
2018	1	7269			
2017	12	5673			
2017	9	4285			
2018	5	6873			
2017	8	4331			
2018	4	6939			
2017	3	2682			
2017	6	3245			
2017	2	1780			
2016	10	324			
2016	9	16			

3. During what time of the day, do the Brazilian customers mostly place their orders? (Dawn, Morning, Afternoon or Night)
  1. 0-6 hrs : Dawn
  2. 7-12 hrs : Mornings
  3. 13-18 hrs : Afternoon
  4. 19-23 hrs : Night

SQL File 3\*

```

1 SELECT CASE WHEN HOUR(order_purchase_timestamp) BETWEEN 0 AND 6 THEN 'Dawn'
2             WHEN HOUR(order_purchase_timestamp) BETWEEN 7 AND 12 THEN 'Morning'
3             WHEN HOUR(order_purchase_timestamp) BETWEEN 13 AND 18 THEN 'Afternoon'
4             ELSE 'Night' END AS time_of_day, COUNT(*) AS total_orders
5 FROM orders
6 JOIN customers ON orders.customer_id = customers.customer_id
7 WHERE customer_state = 'BA'
8 GROUP BY time_of_day;
9

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

time_of_day	total_orders
Afternoon	1272
Night	1006
Morning	895
Dawn	207

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

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

SQL File 3\*

```

1 SELECT customer_state, YEAR(order_purchase_timestamp) AS year, MONTH(order_purchase_timestamp) AS month, COUNT(*) AS total_orders
2 FROM orders
3 JOIN customers ON orders.customer_id = customers.customer_id
4 WHERE customer_state = 'BA'
5 GROUP BY customer_state, year, month;

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

customer_state	year	month	total_orders
BA	2018	7	250
BA	2017	10	166
BA	2018	6	201
BA	2018	8	165
BA	2017	5	127
BA	2018	1	239
BA	2017	1	25
BA	2017	11	250
BA	2017	8	158
BA	2018	3	249
BA	2018	4	225
BA	2017	12	192
BA	2017	2	59
BA	2017	9	170
BA	2017	3	91
BA	2018	5	241
BA	2017	6	106
BA	2017	7	155
BA	2018	2	214

- How are the customers distributed across all the states?

SQL File 3\*

```

1 • SELECT customer_state, COUNT(*) AS total_customers
2 FROM customers
3 GROUP BY customer_state
4 ORDER BY total_customers DESC;

```

Result Grid | Filter Rows: | Export: | Wrap Cell C

	customer_state	total_customers
▶	SP	41746
	RJ	12852
	MG	11635
	RS	5466
	PR	5045
	SC	3637
	BA	3380
	DF	2140
	ES	2033
	GO	2020
	PE	1652
	CE	1336

#### 4.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 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.

SQL File 3\*

```

1 • SELECT 100.0 * (SUM(CASE WHEN YEAR(order_purchase_timestamp) = 2018 THEN payment_value ELSE 0 END) -
2 SUM(CASE WHEN YEAR(order_purchase_timestamp) = 2017 THEN payment_value ELSE 0 END)) /
3 SUM(CASE WHEN YEAR(order_purchase_timestamp) = 2017 THEN payment_value ELSE 0 END) AS percent_increase
4 FROM payments
5 JOIN orders ON payments.order_id = orders.order_id
6 WHERE (YEAR(order_purchase_timestamp) = 2017 OR YEAR(order_purchase_timestamp) = 2018)
7 AND MONTH(order_purchase_timestamp) BETWEEN 1 AND 8;

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

	percent_increase
▶	136.97687164666303

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

SQL File 3\*

```

1 • SELECT customer_state, SUM(payment_value) AS total_order_value, AVG(payment_value) AS avg_order_value
2 FROM payments
3 JOIN orders ON payments.order_id = orders.order_id
4 JOIN customers ON orders.customer_id = customers.customer_id
5 GROUP BY customer_state;

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

	customer_state	total_order_value	avg_order_value
▶	SP	5998226.959999806	137.50462977396282
	MG	1872257.2600000075	154.70643364733164
	RJ	2144379.6899999995	158.52588822355287
	BA	616645.8200000026	170.81601662049934
	PR	811156.3799999995	154.15362599771942
	ES	325967.5500000016	154.7069530137644
	SC	623086.4299999995	165.97933670751183
	GO	350092.3100000004	165.7634043560608
	PA	218295.84999999977	215.92072205736872
	SE	75246.24999999991	208.43836565096927
	CE	279464.0299999997	199.90273962803982
	MA	152523.0200000003	198.85661016949192
	MT	187029.28999999995	195.22890396659702
	PB	141545.71999999994	248.3258245614034
	RS	890898.5399999954	157.18040578687285
	DF	355141.07999999984	161.13479128856616

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

SQL File 3\*

```

1 SELECT
2     customers.customer_state,
3     SUM(order_items.freight_value) AS total_freight_value,
4     AVG(order_items.freight_value) AS avg_freight_value
5 FROM
6     orders
7 JOIN customers ON orders.customer_id = customers.customer_id
8 JOIN order_items ON orders.order_id = order_items.order_id
9 GROUP BY
10    customers.customer_state
11 ORDER BY
12    total_freight_value DESC
13 LIMIT 10;

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

	customer_state	total_freight_value	avg_freight_value
▶	SP	718723.06999999862	15.147275390418896
	RJ	305589.30999999981	20.960923931682427
	MG	270853.460000000305	20.63016680630688
	RS	135522.740000000238	21.735804330393325
	PR	117851.680000000103	20.53165156794443
	BA	100156.679999999903	26.363958936562
	SC	89660.260000000018	21.470368773946404
	PE	59449.659999999991	32.917862679955654
	GO	53114.979999999994	22.766815259322733
	DF	50625.499999999983	21.041354945968344

## 5. 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.

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 File 3\*

```

1 SELECT
2     orders.order_id,
3     DATEDIFF(orders.order_delivered_customer_date, orders.order_purchase_timestamp) AS time_to_deliver,
4     DATEDIFF(orders.order_delivered_customer_date, orders.order_estimated_delivery_date) AS diff_estimated_delivery
5 FROM
6     orders
7 WHERE
8     orders.order_delivered_customer_date IS NOT NULL
9     AND orders.order_estimated_delivery_date IS NOT NULL;
10

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

	order_id	time_to_deliver	diff_estimated_delivery
▶	e481f51cbdc54678b7cc49136f2d6af7	8	-8
	53cdb2fc8bc7dce0b6741e2150273451	14	-6
	47770eb9100c2d0c44946d9cf07ec65d	9	-18
	949d5b44dbf5de918fe9c16f97b45f8a	14	-13
	ad21c59c0840e6cb83a9ceb5573f8159	3	-10
	a4591c265e18cb1dcee52889e2d8acc3	17	-6
	136cce7faa42fdb2cefd53fdc79a6098	NULL	NULL
	6514b8ad8028c9f2cc2374ded245783f	10	-12
	76c6e866289321a7c93b82b54852dc33	10	-32
	e69bfb5eb88e0ed6a785585b27e16dbf	18	-7

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

The screenshot shows a SQL query in a file editor. The query is as follows:

```
1 SELECT
2     customers.customer_state,
3     AVG(DATEDIFF(orders.order_delivered_customer_date, orders.order_estimated_delivery_date)) AS avg_diff_delivery
4 FROM
5     orders
6 JOIN customers ON orders.customer_id = customers.customer_id
7 WHERE
8     orders.order_delivered_customer_date IS NOT NULL
9 AND orders.order_estimated_delivery_date IS NOT NULL
10 GROUP BY
11     customers.customer_state
12 ORDER BY
13     avg_diff_delivery ASC
14 LIMIT 5;
```

Below the query, the result grid is displayed, showing the top 5 states with the lowest average freight value (ordered from lowest to highest):

customer_state	avg_diff_delivery
AC	-20.7250
RO	-20.1029
AP	-19.6866
AM	-19.5655
RR	-17.2927

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

#### HIGHEST AVERAGE DELIVERY TIME

The screenshot shows a SQL query in a file editor. The query is as follows:

```
1 SELECT
2     customers.customer_state,
3     AVG(DATEDIFF(orders.order_delivered_customer_date, orders.order_purchase_timestamp)) AS avg_delivery_time
4 FROM
5     orders
6 JOIN customers ON orders.customer_id = customers.customer_id
7 WHERE
8     orders.order_delivered_customer_date IS NOT NULL
9 GROUP BY
10     customers.customer_state
11 ORDER BY
12     avg_delivery_time DESC
13 LIMIT 5;
```

Below the query, the result grid is displayed, showing the top 5 states with the highest average delivery time (ordered from highest to lowest):

customer_state	avg_delivery_time
RR	29.3415
AP	27.1791
AM	26.3586
AL	24.5013
PA	23.7252

## LOWEST AVERAGE DELIVERY TIME

```
1 • SELECT
2     customers.customer_state,
3     AVG(DATEDIFF(orders.order_delivered_customer_date, orders.order_purchase_timestamp)) AS avg_delivery_time
4 FROM
5     orders
6 JOIN customers ON orders.customer_id = customers.customer_id
7 WHERE
8     orders.order_delivered_customer_date IS NOT NULL
9 GROUP BY
10    customers.customer_state
11 ORDER BY
12     avg_delivery_time ASC
13 LIMIT 5;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

	customer_state	avg_delivery_time
▶	SP	8.7005
	PR	11.9380
	MG	11.9465
	DF	12.8990
	SC	14.9075

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.

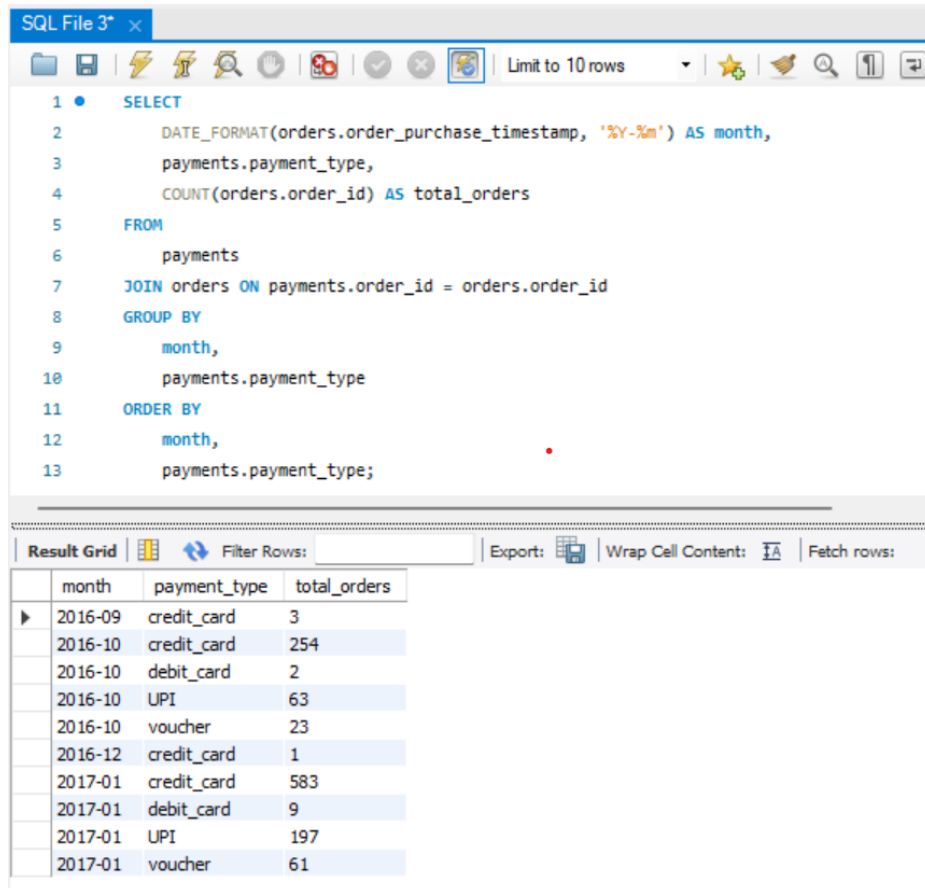
```
1 • SELECT
2     customers.customer_state,
3     AVG(DATEDIFF(orders.order_delivered_customer_date, orders.order_estimated_delivery_date)) AS avg_delivery_difference
4 FROM
5     orders
6 JOIN customers ON orders.customer_id = customers.customer_id
7 WHERE
8     orders.order_delivered_customer_date IS NOT NULL
9     AND orders.order_estimated_delivery_date IS NOT NULL
10 GROUP BY
11    customers.customer_state
12 ORDER BY
13     avg_delivery_difference ASC -- Faster deliveries will have a negative or lower average difference
14 LIMIT 5;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

	customer_state	avg_delivery_difference
▶	AC	-20.7250
	RO	-20.1029
	AP	-19.6866
	AM	-19.5655
	RR	-17.2927

## 6. Analysis based on the payments:

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

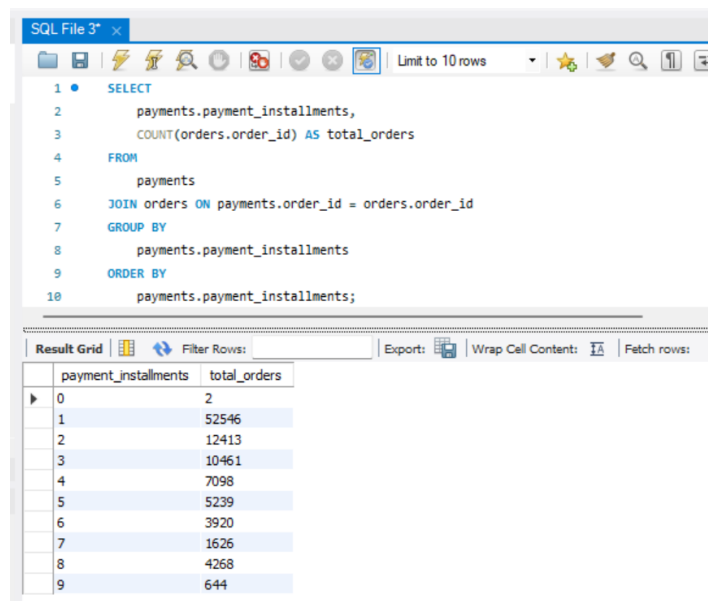


The screenshot shows a SQL query in a file named 'SQL File 3\*'. The query is a SELECT statement that joins the 'orders' and 'payments' tables. It groups the data by month and payment type, and orders the results by month. The result grid shows 12 rows of data.

```
1 SELECT
2     DATE_FORMAT(orders.order_purchase_timestamp, '%Y-%m') AS month,
3     payments.payment_type,
4     COUNT(orders.order_id) AS total_orders
5 FROM
6     payments
7 JOIN orders ON payments.order_id = orders.order_id
8 GROUP BY
9     month,
10    payments.payment_type
11 ORDER BY
12    month,
13    payments.payment_type;
```

month	payment_type	total_orders
2016-09	credit_card	3
2016-10	credit_card	254
2016-10	debit_card	2
2016-10	UPI	63
2016-10	voucher	23
2016-12	credit_card	1
2017-01	credit_card	583
2017-01	debit_card	9
2017-01	UPI	197
2017-01	voucher	61

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



The screenshot shows a SQL query in a file named 'SQL File 3\*'. The query is a SELECT statement that joins the 'orders' and 'payments' tables. It groups the data by payment installment number and orders the results by payment installment number. The result grid shows 10 rows of data.

```
1 SELECT
2     payments.payment_installments,
3     COUNT(orders.order_id) AS total_orders
4 FROM
5     payments
6 JOIN orders ON payments.order_id = orders.order_id
7 GROUP BY
8     payments.payment_installments
9 ORDER BY
10    payments.payment_installments;
```

payment_installments	total_orders
0	2
1	52546
2	12413
3	10461
4	7098
5	5239
6	3920
7	1626
8	4268
9	644



## **Actionable insights & Recommendation**

### **1. Optimize Delivery Operations**

- **Insight:** Certain states exhibit significantly higher average delivery times, indicating potential inefficiencies in the logistics process.
- **Recommendation:** Conduct a thorough analysis of the logistics network in these states to identify bottlenecks. Consider partnering with local courier services or establishing regional distribution centers to expedite delivery and improve overall customer satisfaction.

### **2. Enhance Payment Flexibility**

- **Insight:** A notable percentage of customers are opting for installment payment plans, suggesting a preference for flexibility in payment options.
- **Recommendation:** Introduce promotional financing options for high-ticket items to encourage full upfront payments, while continuing to promote installment plans. This could help balance cash flow while meeting customer preferences.

### **3. Targeted Marketing Based on Payment Preferences**

- **Insight:** Different payment types and installment plans show varying levels of usage across customer demographics.
- **Recommendation:** Develop targeted marketing campaigns that highlight popular payment methods tailored to specific customer segments. Use data analytics to personalize offers based on previous payment behaviors, increasing conversion rates.

### **4. Implement Feedback Mechanisms**

- **Insight:** Customer feedback related to delivery experiences is crucial for continuous improvement.
- **Recommendation:** Set up post-delivery surveys to gather insights on customer satisfaction regarding delivery speed, accuracy, and overall experience. Use this feedback to identify areas for improvement in logistics and customer service.