Business Case: Target SQL

Scaler DS ML

Shayantan Dey

30th July

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.

Dataset:

The data is available in 8 csv files at Google Drive

1. customers.csv
2. sellers.csv
3. order_items.csv
4. geolocation.csv
5. payments.csv
6. reviews.csv
7. orders.csv
8. products.csv

The column description for these csv files is given below.

The **customers.csv** contain following features:

Features	Description
customer_id	ID of the consumer who made the purchase
customer_unique_id	Unique ID of the consumer
customer_zip_code_prefix	Zip Code of consumer's location
customer_city	Name of the City from where order is made
customer_state	State Code from where order is made (Eg. são paulo - SP)

The **sellers.csv** contains following features:

Features	Description
seller_id	Unique ID of the seller registered
seller_zip_code_prefix	Zip Code of the seller's location
seller_city	Name of the City of the seller
seller_state	State Code (Eg. são paulo - SP)

The **order_items.csv** contain following features:

Features	Description
order_id	A Unique ID of order made by the consumers
order_item_id	A Unique ID given to each item ordered in the order
product_id	A Unique ID given to each product available on the site
seller_id	Unique ID of the seller registered in Target
shipping_limit_date	The date before which the ordered product must be shipped
price	Actual price of the products ordered
freight_value	Price rate at which a product is delivered from one point to another

The **geolocations.csv** contain following features:

Features	Description
geolocation_zip_code_prefix	First 5 digits of Zip Code
geolocation_lat	Latitude
geolocation_lng	Longitude
geolocation_city	City
geolocation_state	State

The **payments.csv** contain following features:

Features	Description
order_id	A Unique ID of order made by the consumers
payment_sequential	Sequences of the payments made in case of EMI
payment_type	Mode of payment used (Eg. Credit Card)
payment_installments	Number of installments in case of EMI purchase
payment_value	Total amount paid for the purchase order

The **orders.csv** contain following features:

Features	Description
order_id	A Unique ID of order made by the consumers
customer_id	ID of the consumer who made the purchase
order_status	Status of the order made i.e. delivered, shipped, etc.
order_purchase_timestamp	Timestamp of the purchase
order_delivered_carrier_date	Delivery date at which carrier made the delivery
order_delivered_customer_date	Date at which customer got the product
order_estimated_delivery_date	Estimated delivery date of the products

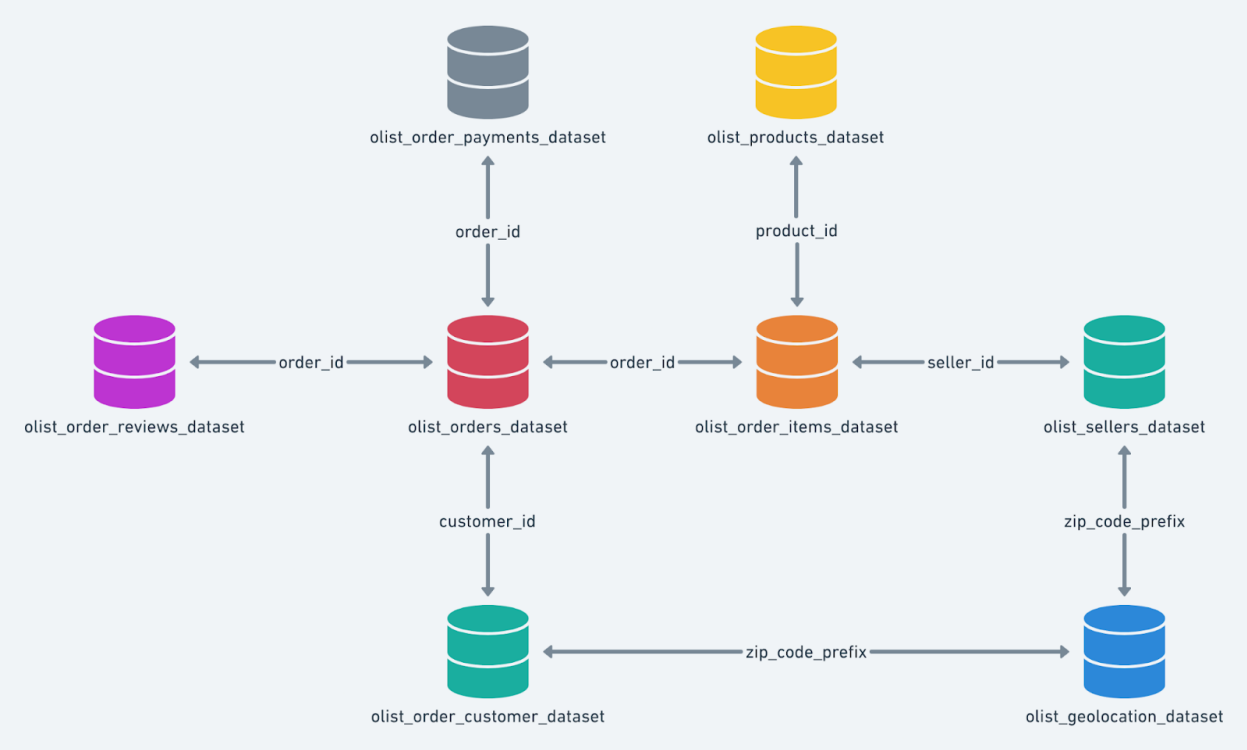
The **reviews.csv** contain following features:

Features	Description
review_id	ID of the review given on the product ordered by the order id
order_id	A Unique ID of order made by the consumers
review_score	Review score given by the customer for each order on a scale of 1-5
review_comment_title	Title of the review
review_comment_message	Review comments posted by the consumer for each order
review_creation_date	Timestamp of the review when it is created
review_answer_timestamp	Timestamp of the review answered

The **products.csv** contain following features:

Features	Description
product_id	A Unique identifier for the proposed project
product_category_name	Name of the product category
product_name_lenght	Length of the string which specifies the name given to the products ordered
product_description_lenght	Length of the description written for each product ordered on the site
product_photos_qty	Number of photos of each product ordered available on the shopping portal
product_weight_g	Weight of the products ordered in grams
product_length_cm	Length of the products ordered in centimeters
product_height_cm	Height of the products ordered in centimeters
product_width_cm	Width of the product ordered in centimeters

Dataset schema:



Observations in the dataset

Two files, order_reviews.csv and geolocation.csv had unclean data.

Issues Identified in the order_reviews.csv file:

Encoding Issue: The file had to be read with ISO-8859-1 encoding instead of UTF-8.

Null Values: The review_comment_title column has many null values.

Date and Time Formatting: The review_creation_date and review_answer_timestamp columns are in string format and not properly parsed as datetime objects.

Steps to Correct Issues:

1. Ensure consistent encoding.
2. Handle null values in review_comment_title.
3. Convert date and time columns to proper datetime format.

Cleaning Data:

1. Strip leading/trailing spaces in text fields.
2. Replace any special characters or non-UTF-8 characters in text fields.
3. Check for null or empty values and handle them appropriately.

4. Convert date and time columns to datetime format.

Issues Identified in the geolocation.csv file:

Encoding Issue: The file had to be read with ISO-8859-1 encoding instead of UTF-8.

Null Values: The review_comment_title column has many null values.

Date and Time Formatting: The review_creation_date and review_answer_timestamp columns are in string format and not properly parsed as datetime objects.

Steps to Correct Issues:

1. Special characters in text fields.
2. Trailing or leading spaces.
3. Null or empty values.
4. Ensure that the file does not have any rows that might cause issues.

Cleaning Data:

1. Strip leading/trailing spaces in text fields.
2. Replace any special characters or non-UTF-8 characters in text fields.
3. Check for null or empty values and handle them appropriately.

All the 27 *geolocation_state* listed in the *geolocations.csv* file and *customer_state* in *customers.csv* are 26 states and 1 federal territory of Brazil. Hence, the data is specific to Brazil customers.

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:

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

```
DESCRIBE customers;
```

Table 9: 5 records

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

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

```sql
SELECT
    MIN(order_purchase_timestamp) AS order_start_date,
    MAX(order_purchase_timestamp) AS order_end_date,
    DATEDIFF(MAX(order_purchase_timestamp), MIN(order_purchase_timestamp))
      AS order_time_range_days
FROM
    orders;
```

Table 10: 1 records

order_start_date	order_end_date	order_time_range_days
2016-09-04 21:15:19	2018-10-17 17:30:18	773

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

```sql
SELECT DISTINCT c.customer_city, c.customer_state, COUNT(*) AS customer_count
FROM orders AS o
JOIN customers AS c
ON o.customer_id = c.customer_id
GROUP BY c.customer_city, c.customer_state
ORDER BY customer_count DESC
```

Table 11: Displaying records 1 - 10

customer_city	customer_state	customer_count
sao paulo	SP	15540
rio de janeiro	RJ	6882
belo horizonte	MG	2773
brasilia	DF	2131
curitiba	PR	1521
campinas	SP	1444
porto alegre	RS	1379
salvador	BA	1245
guarulhos	SP	1189
sao bernardo do campo	SP	938

2. In-depth Exploration:

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

The purchases were made in the year 2016, 2017 and 2018.

```sql
SELECT DISTINCT YEAR(order_purchase_timestamp) AS year_of_orders
FROM orders
ORDER BY year_of_orders;
```

Table 12: 3 records

year_of_orders
2016
2017
2018

Trend for 2016 does not show conclusive evidence of a growing trend.

```
SELECT DISTINCT CONCAT(MONTHNAME(order_purchase_timestamp), " ", "2016") AS month,
       COUNT(order_id) OVER (PARTITION BY MONTH(order_purchase_timestamp))
          AS order_count,
       MONTH(order_purchase_timestamp) AS month_number
FROM orders
WHERE YEAR(order_purchase_timestamp) = 2016
ORDER BY MONTH(order_purchase_timestamp);
```

Table 13: 3 records

month	order_count	month_number
September 2016	4	9
October 2016	324	10
December 2016	1	12

Trend for 2017 shows growth in month-on-month sale throughout the year.

```
SELECT DISTINCT CONCAT(MONTHNAME(order_purchase_timestamp), " ", "2017") AS month,
       COUNT(order_id) OVER (PARTITION BY MONTH(order_purchase_timestamp))
        AS order_count,
       MONTH(order_purchase_timestamp) AS month_number
FROM orders
WHERE YEAR(order_purchase_timestamp) = 2017
ORDER BY MONTH(order_purchase_timestamp);
```

Table 14: Displaying records 1 - 10

month	order_count	month_number
January 2017	800	1
February 2017	1780	2
March 2017	2682	3
April 2017	2404	4
May 2017	3700	5
June 2017	3245	6
July 2017	4026	7
August 2017	4331	8
September 2017	4285	9
October 2017	4631	10

Trend for 2018 shows growth in month-on-month sale throughout the year.

```sql
SELECT DISTINCT CONCAT(MONTHNAME(order_purchase_timestamp), " ", "2018") AS month,
       COUNT(order_id) OVER (PARTITION BY MONTH(order_purchase_timestamp))
        AS order_count,
       MONTH(order_purchase_timestamp) AS month_number
FROM orders
WHERE YEAR(order_purchase_timestamp) = 2018
ORDER BY MONTH(order_purchase_timestamp);
```

Table 15: Displaying records 1 - 10

month	order_count	month_number
January 2018	7269	1
February 2018	6728	2
March 2018	7211	3
April 2018	6939	4
May 2018	6873	5
June 2018	6167	6
July 2018	6292	7
August 2018	6512	8
September 2018	16	9
October 2018	4	10

Finding the sales per year shows a year-on-year growing trend.

```sql
SELECT DISTINCT YEAR(order_purchase_timestamp) AS year,
       COUNT(order_id) OVER(PARTITION BY YEAR(order_purchase_timestamp))
          AS count_of_orders
FROM orders;
```

Table 16: 3 records

year	count_of_orders
2016	329
2017	45101
2018	54011

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

Highest monthly sales in the given data is as follows, but it fails to show any seasonal trend:

```sql
SELECT YEAR(order_purchase_timestamp) AS year,
       MONTHNAME(order_purchase_timestamp) AS month,
       COUNT(*) AS order_count
FROM orders
GROUP BY year, month
ORDER BY order_count DESC;
```

Table 17: Displaying records 1 - 10

year	month	order_count
2017	November	7544
2018	January	7269
2018	March	7211
2018	April	6939
2018	May	6873
2018	February	6728
2018	August	6512
2018	July	6292
2018	June	6167
2017	December	5673

While checking the year-wise monthly sales data, we do not see any monthly seasonality:

```
SELECT DISTINCT CONCAT(MONTHNAME(order_purchase_timestamp), " ", "2016") AS month,
       MONTH(order_purchase_timestamp) AS month_number,
       COUNT(order_id) OVER (PARTITION BY MONTH(order_purchase_timestamp))
        AS order_count
FROM orders
WHERE YEAR(order_purchase_timestamp) = 2016
ORDER BY order_count DESC;
```

Table 18: 3 records

month	month_number	order_count
October 2016	10	324
September 2016	9	4
December 2016	12	1

```
SELECT DISTINCT CONCAT(MONTHNAME(order_purchase_timestamp), " ", "2017") AS month,
       MONTH(order_purchase_timestamp) AS month_number,
       COUNT(order_id) OVER (PARTITION BY MONTH(order_purchase_timestamp))
        AS order_count
FROM orders
WHERE YEAR(order_purchase_timestamp) = 2017
ORDER BY order_count DESC;
```

Table 19: Displaying records 1 - 10

month	month_number	order_count
November 2017	11	7544
December 2017	12	5673
October 2017	10	4631
August 2017	8	4331
September 2017	9	4285
July 2017	7	4026
May 2017	5	3700

month	month_number	order_count
June 2017	6	3245
March 2017	3	2682
April 2017	4	2404

```
SELECT DISTINCT CONCAT(MONTHNAME(order_purchase_timestamp), " ", "2018") AS month,
       MONTH(order_purchase_timestamp) AS month_number,
       COUNT(order_id) OVER (PARTITION BY MONTH(order_purchase_timestamp))
        AS order_count
FROM orders
WHERE YEAR(order_purchase_timestamp) = 2018
ORDER BY order_count DESC;
```

Table 20: Displaying records 1 - 10

month	month_number	order_count
January 2018	1	7269
March 2018	3	7211
April 2018	4	6939
May 2018	5	6873
February 2018	2	6728
August 2018	8	6512
July 2018	7	6292
June 2018	6	6167
September 2018	9	16
October 2018	10	4

2.3. 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

As per the data, Brazilian customers prefer placing their orders during afternoon.

```
WITH d AS (SELECT customer_id, order_purchase_timestamp,
       CASE
          WHEN FLOOR(EXTRACT(HOUR FROM order_purchase_timestamp)) BETWEEN 0 AND 6 THEN
            "Dawn"
          WHEN FLOOR(EXTRACT(HOUR FROM order_purchase_timestamp)) BETWEEN 7 AND 12 THEN
            "Mornings"
          WHEN FLOOR(EXTRACT(HOUR FROM order_purchase_timestamp)) BETWEEN 13 AND 18 THEN
            "Afternoon"
          WHEN FLOOR(EXTRACT(HOUR FROM order_purchase_timestamp)) BETWEEN 19 AND 23 THEN
            "Night"
       END AS time_of_day
FROM orders)
SELECT DISTINCT d.time_of_day, COUNT(d.time_of_day) OVER(PARTITION BY d.time_of_day)
        AS count_of_orders
```

```
FROM d
ORDER BY count_of_orders DESC;
```

Table 21: 4 records

time_of_day	count_of_orders
Afternoon	38135
Night	28331
Mornings	27733
Dawn	5242

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

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

```
SELECT c.customer_state,
    EXTRACT(YEAR FROM o.order_purchase_timestamp) AS year,
    MONTHNAME(o.order_purchase_timestamp) AS month_name,
    EXTRACT(MONTH FROM o.order_purchase_timestamp) AS month_number,
    COUNT(o.order_id) AS order_count
FROM customers AS c
JOIN orders AS o
ON c.customer_id = o.customer_id
GROUP By c.customer_state,
    EXTRACT(YEAR FROM o.order_purchase_timestamp),
    MONTHNAME(o.order_purchase_timestamp),
    EXTRACT(MONTH FROM o.order_purchase_timestamp)
ORDER BY c.customer_state, year, month_number;
```

Table 22: Displaying records 1 - 10

customer_state	year	month_name	month_number	order_count
AC	2017	January	1	2
AC	2017	February	2	3
AC	2017	March	3	2
AC	2017	April	4	5
AC	2017	May	5	8
AC	2017	June	6	4
AC	2017	July	7	5
AC	2017	August	8	4
AC	2017	September	9	5
AC	2017	October	10	6

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

Distribution of customers across states is as follows:

```
SELECT customer_state,
       COUNT(*) AS count_of_customers
FROM customers
GROUP BY customer_state
ORDER BY customer_state;
```

Table 23: Displaying records 1 - 10

customer_state	count_of_customers
AC	81
AL	413
AM	148
AP	68
BA	3380
CE	1336
DF	2140
ES	2033
GO	2020
MA	747

Distribution of customers across cities in those states is as follows:

```
SELECT customer_state, customer_city,
       COUNT(*) AS count_of_customers
FROM customers
GROUP BY customer_state, customer_city
ORDER BY customer_state, customer_city;
```

Table 24: Displaying records 1 - 10

customer_state	customer_city	count_of_customers
AC	brasileia	1
AC	cruzeiro do sul	3
AC	epitaciolandia	1
AC	manoel urbano	1
AC	porto acre	1
AC	rio branco	70
AC	senador guiomard	2
AC	xapuri	2
AL	agua branca	1
AL	anadia	2

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

4.1. 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.

```
WITH d AS (SELECT EXTRACT(YEAR FROM o.order_purchase_timestamp) AS year,
       p.payment_value
       FROM orders AS o
       JOIN payments AS p
       ON o.order_id = p.order_id
       WHERE (o.order_purchase_timestamp BETWEEN '2017-01-01' AND '2017-08-31')
         OR (o.order_purchase_timestamp BETWEEN '2018-01-01' AND '2018-08-31')),

    d2 AS (SELECT DISTINCT d.year,
```

```
       SUM(d.payment_value) OVER (PARTITION BY d.year) AS yearly_payment_value
       FROM d),

    d3 AS (SELECT d2.year, d2.yearly_payment_value,
           LEAD(d2.yearly_payment_value) OVER(ORDER BY d2.yearly_payment_value)
              AS lead_,
           ((LEAD(d2.yearly_payment_value) OVER(ORDER BY d2.yearly_payment_value)
             - d2.yearly_payment_value)) AS diff
           FROM d2)

SELECT d3.yearly_payment_value AS 2017_payment_value,
       d3.lead_ AS 2018_payment_value,
       ROUND((d3.diff/d3.yearly_payment_value) * 100, 2)
          AS 2017_to_2018_percentage_increase
FROM d3
WHERE d3.lead_ IS NOT NULL;
```

Table 25: 1 records

2017_payment_value	2018_payment_value	2017_to_2018_percentage_increase
3645107	8694670	138.53

Analyzing the *order_status* we find the following status. Including or excluding orders with certain status will change the percentage increase.

```
SELECT DISTINCT order_status FROM orders;
```

Table 26: 8 records

order_status
delivered
invoiced
shipped
processing
unavailable
canceled
created
approved

Foe example, excluding the orders with *canceled* status changes the results as follows:

```
WITH d AS (SELECT EXTRACT(YEAR FROM o.order_purchase_timestamp) AS year,
       p.payment_value
       FROM orders AS o
       JOIN payments AS p
       ON o.order_id = p.order_id
       WHERE ((o.order_purchase_timestamp BETWEEN '2017-01-01' AND '2017-08-31')
         OR (o.order_purchase_timestamp BETWEEN '2018-01-01' AND '2018-08-31'))
         AND (o.order_status <> "canceled")),
```

```
    d2 AS (SELECT DISTINCT d.year,
       SUM(d.payment_value) OVER (PARTITION BY d.year) AS yearly_payment_value
       FROM d),

    d3 AS (SELECT d2.year, d2.yearly_payment_value,
            LEAD(d2.yearly_payment_value) OVER(ORDER BY d2.yearly_payment_value)
               AS lead_,
            ((LEAD(d2.yearly_payment_value) OVER(ORDER BY d2.yearly_payment_value)
              - d2.yearly_payment_value)) AS diff
            FROM d2)

SELECT d3.yearly_payment_value AS 2017_payment_value,
       d3.lead_ AS 2018_payment_value,
       ROUND((d3.diff/d3.yearly_payment_value) * 100, 2)
          AS 2017_to_2018_percentage_increase
FROM d3
WHERE d3.lead_ IS NOT NULL;
```

Table 27: 1 records

2017_payment_value	2018_payment_value	2017_to_2018_percentage_increase
3604147	8624465	139.29

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

```
SELECT DISTINCT c.customer_state,
       ROUND(SUM(p.payment_value) OVER(PARTITION BY c.customer_state), 2)
          AS total_order_price,
       ROUND(AVG(p.payment_value) OVER(PARTITION BY c.customer_state), 2)
          AS average_order_price
FROM customers AS c
JOIN orders AS o
ON c.customer_id = o.customer_id
JOIN payments AS p
ON o.order_id = p.order_id
ORDER BY c.customer_state;
```

Table 28: Displaying records 1 - 10

customer_state	total_order_price	average_order_price
AC	19680.62	234.29
AL	96962.06	227.08
AM	27966.93	181.60
AP	16262.80	232.33
BA	616645.82	170.82
CE	279464.03	199.90
DF	355141.08	161.13
ES	325967.55	154.71
GO	350092.31	165.76
MA	152523.02	198.86

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

```
SELECT DISTINCT c.customer_state,
       ROUND(SUM(oi.freight_value) OVER(PARTITION BY c.customer_state), 2)
          AS total_freight_price,
       ROUND(AVG(oi.freight_value) OVER(PARTITION BY c.customer_state), 2)
          AS average_freight_price
FROM customers AS c
JOIN orders AS o
ON c.customer_id = o.customer_id
JOIN order_items AS oi
ON o.order_id = oi.order_id
ORDER BY c.customer_state;
```

Table 29: Displaying records 1 - 10

customer_state	total_freight_price	average_freight_price
AC	3686.75	40.07
AL	15914.59	35.84
AM	5478.89	33.21
AP	2788.50	34.01
BA	100156.68	26.36
CE	48351.59	32.71
DF	50625.50	21.04
ES	49764.60	22.06
GO	53114.98	22.77
MA	31523.77	38.26

5. Analysis based on sales, freight and delivery time.

5.1. 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.

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

Checking for the missing values, we find that *order_delivered_customer_date* is is missing in the dataset for many rows across all *order_status*. This should be considered as unclean data and it will adversely affect the analysis.

```
SELECT DISTINCT order_status,
       COUNT(CASE WHEN order_delivered_customer_date = "" THEN 1 END)
          OVER (PARTITION BY order_status ORDER BY order_delivered_customer_date)
              AS null_delivered_date,
       COUNT(CASE WHEN order_purchase_timestamp = "" THEN 1 END)
          OVER (PARTITION BY order_status ORDER BY order_purchase_timestamp)
              AS null_purchase_timestamp,
       COUNT(CASE WHEN order_estimated_delivery_date = "" THEN 1 END)
          OVER (PARTITION BY order_status ORDER BY order_estimated_delivery_date)
              AS null_estimated_delivery
FROM orders;
```

Table 30: 8 records

order_status	null_delivered_date	null_purchase_timestamp	null_estimated_delivery
approved	2	0	0
canceled	619	0	0
created	5	0	0
delivered	8	0	0
invoiced	314	0	0
processing	301	0	0
shipped	1107	0	0
unavailable	609	0	0

Calculating *time_to_deliver* and *diff_estimated_delivery*, we observe that
time to deliver and estimated delivery are a major concern that requires improvement.

```
SELECT order_id, order_status,
       DATEDIFF(order_delivered_customer_date, order_purchase_timestamp)
          AS time_to_deliver,
       DATEDIFF(order_delivered_customer_date, order_estimated_delivery_date)
          AS diff_estimated_delivery
FROM orders
ORDER BY diff_estimated_delivery DESC;
```

Table 31: Displaying records 1 - 10

order_id	order_status	time_to_deliver	diff_estimated_delivery
1b3190b2dfa9d789e1f14c05b647a14a	delivered	208	188
ca07593549f1816d26a572e06dc1eab6	delivered	210	181
47b40429ed8cce3aee9199792275433f	delivered	191	175
2fe324febf907e3ea3f2aa9650869fa5	delivered	190	167
285ab9426d6982034523a855f55a885e	delivered	195	166
440d0d17af552815d15a9e41abe49359	delivered	196	165
c27815f7e3dd0b926b58552628481575	delivered	188	162
d24e8541128cea179a11a65176e0a96f	delivered	175	161
0f4519c5f1c541ddec9f21b3bddd533a	delivered	194	161
2d7561026d542c8dbd8f0daeadf67a43	delivered	188	159

6535 orders had actual delivery later than the estimated delivery.
Estimation of delivery date should be revised.

```
SELECT COUNT(*) AS delivery_estimate_miss
FROM orders
WHERE DATEDIFF(order_delivered_customer_date, order_estimated_delivery_date) >= 1;
```

Table 32: 1 records

delivery_estimate_miss
6535

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

```sql
SELECT DISTINCT c.customer_state,
       AVG(oi.freight_value) OVER (PARTITION BY c.customer_state) AS highest_5_avg
FROM order_items AS oi
JOIN orders AS o
ON oi.order_id = o.order_id
JOIN customers AS c
ON o.customer_id = c.customer_id
ORDER BY highest_5_avg DESC
LIMIT 5;
```

Table 33: 5 records

customer_state	highest_5_avg
RR	42.98442
PB	42.72380
RO	41.06971
AC	40.07337
PI	39.14797

```sql
SELECT DISTINCT c.customer_state,
       AVG(oi.freight_value) OVER (PARTITION BY c.customer_state) AS lowest_5_avg
FROM order_items AS oi
JOIN orders AS o
ON oi.order_id = o.order_id
JOIN customers AS c
ON o.customer_id = c.customer_id
ORDER BY lowest_5_avg
LIMIT 5;
```

Table 34: 5 records

customer_state	lowest_5_avg
SP	15.14728
PR	20.53165
MG	20.63017
RJ	20.96092
DF	21.04135

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

```sql
SELECT DISTINCT c.customer_state,
       AVG(DATEDIFF(order_delivered_customer_date, order_purchase_timestamp))
          OVER (PARTITION BY c.customer_state) AS highest_5_avg
FROM orders AS o
JOIN customers AS c
ON o.customer_id = c.customer_id
ORDER BY highest_5_avg DESC
LIMIT 5;
```

Table 35: 5 records

customer_state	highest_5_avg
RR	29.3415
AP	27.1791
AM	26.3586
AL	24.5013
PA	23.7252

```sql
SELECT DISTINCT c.customer_state,
       AVG(DATEDIFF(order_delivered_customer_date, order_purchase_timestamp))
          OVER (PARTITION BY c.customer_state) AS lowest_5_avg
FROM orders AS o
JOIN customers AS c
ON o.customer_id = c.customer_id
ORDER BY lowest_5_avg
LIMIT 5;
```

Table 36: 5 records

customer_state	lowest_5_avg
SP	8.7005
PR	11.9380
MG	11.9465
DF	12.8990
SC	14.9075

5.4. 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.

```sql
SELECT DISTINCT c.customer_state,
       AVG(DATEDIFF(order_estimated_delivery_date, order_delivered_customer_date))
          OVER (PARTITION BY c.customer_state) AS top_5_fastest_delivery
FROM orders AS o
JOIN customers AS c
ON o.customer_id = c.customer_id
ORDER BY top_5_fastest_delivery
LIMIT 5;
```

Table 37: 5 records

customer_state	top_5_fastest_delivery
AL	8.7078
MA	9.5718
SE	10.0209
ES	10.4962
BA	10.7945

6. Analysis based on the payments:

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

The various payment types in the dataset is as follows:

```
SELECT DISTINCT payment_type, COUNT(payment_type) AS count_payment_type
FROM payments
GROUP BY payment_type
HAVING payment_type <> "not_defined"
```

Table 38: 4 records

payment_type	count_payment_type
credit_card	76795
UPI	19784
voucher	5775
debit_card	1529

Payment type is not defined for 3 entries in the dataset.

```
SELECT payment_type, COUNT(payment_type) AS count_payment_type
FROM payments
GROUP BY payment_type
HAVING payment_type = "not_defined"
```

Table 39: 1 records

payment_type	count_payment_type
not_defined	3

Month on month no. of orders placed using different payment types is as follows

```
WITH d AS (
  SELECT p.payment_type, o.order_purchase_timestamp, p.order_id,
         EXTRACT(YEAR FROM o.order_purchase_timestamp) AS year,
         MONTHNAME(o.order_purchase_timestamp) AS month,
         EXTRACT(MONTH FROM o.order_purchase_timestamp) AS month_number
  FROM payments AS p
  JOIN orders AS o
  ON p.order_id = o.order_id
)
SELECT
  d.year,
  d.month,
  COUNT(CASE WHEN d.payment_type = 'credit_card' THEN d.order_id ELSE NULL END)
        AS credit_card,
  COUNT(CASE WHEN d.payment_type = 'UPI' THEN d.order_id ELSE NULL END)
        AS UPI,
  COUNT(CASE WHEN d.payment_type = 'voucher' THEN d.order_id ELSE NULL END)
        AS voucher,
```

```
  COUNT(CASE WHEN d.payment_type = 'debit_card' THEN d.order_id ELSE NULL END)
        AS debit_card,
  d.month_number
FROM d
GROUP BY d.year, d.month, d.month_number
ORDER BY d.year, d.month_number;
```

Table 40: Displaying records 1 - 10

year	month	credit_card	UPI	voucher	debit_card	month_number
2016	September	3	0	0	0	9
2016	October	254	63	23	2	10
2016	December	1	0	0	0	12
2017	January	583	197	61	9	1
2017	February	1356	398	119	13	2
2017	March	2016	590	200	31	3
2017	April	1846	496	202	27	4
2017	May	2853	772	289	30	5
2017	June	2463	707	239	27	6
2017	July	3086	845	364	22	7

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

```
SELECT DISTINCT payment_installments,
       COUNT(order_id) OVER (PARTITION BY payment_installments) AS count_orders
FROM payments
```

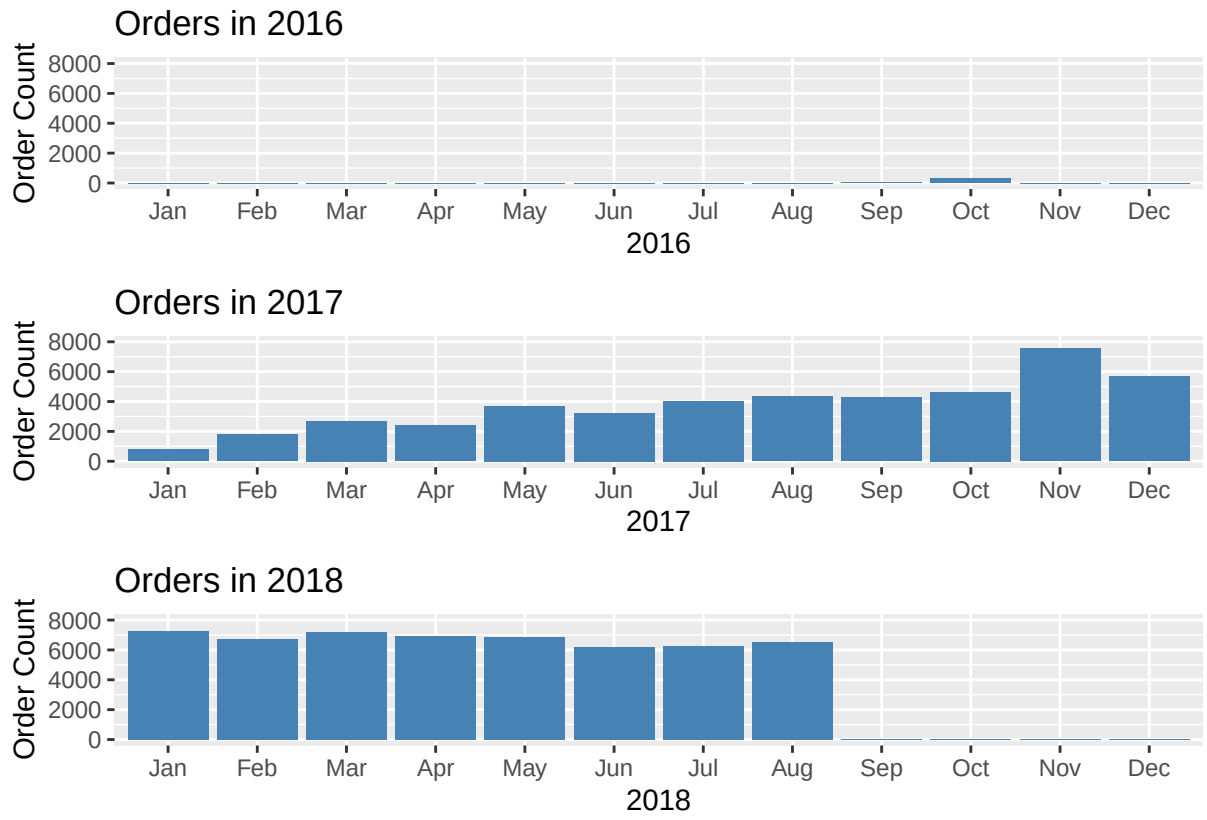
Table 41: Displaying records 1 - 10

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

7. Actionable Insights & Recommendations

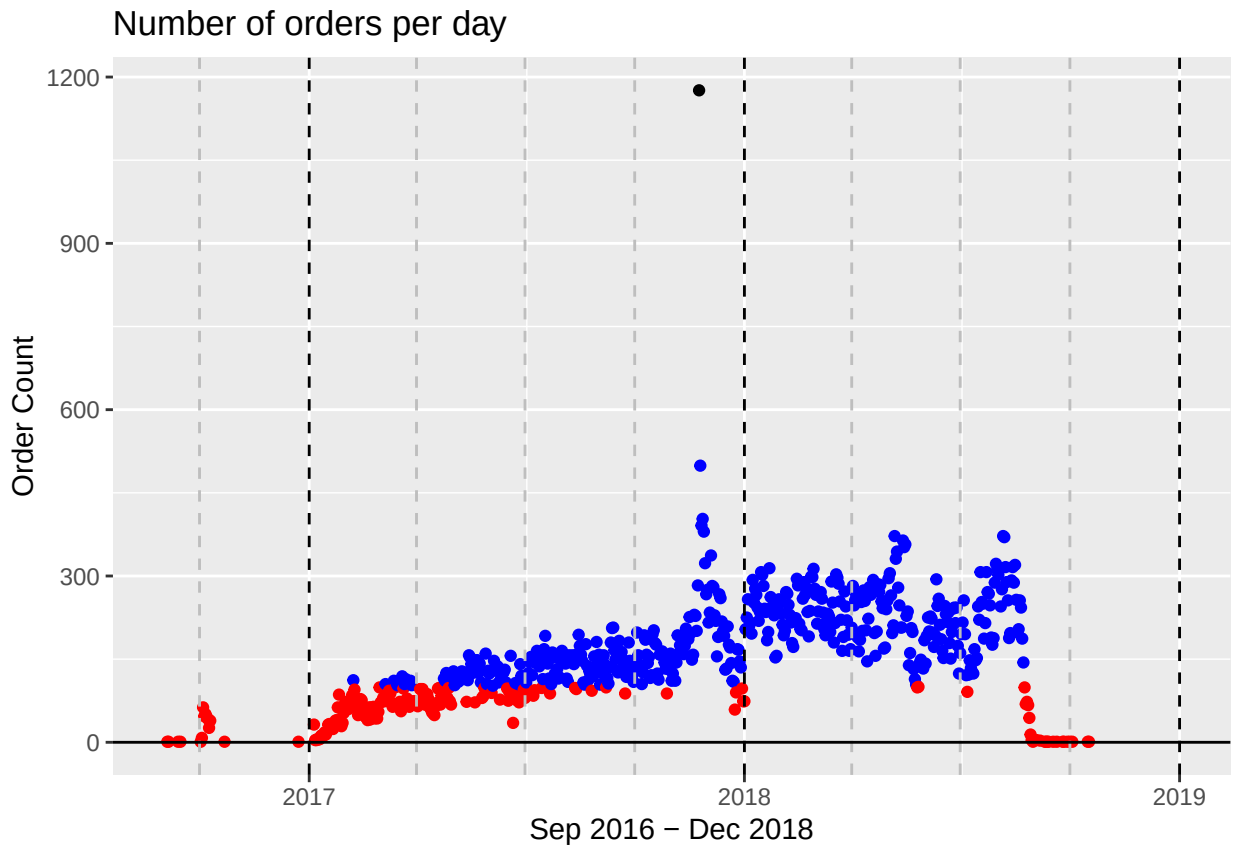
7.1. The data set has data from the year 2016, 2017 and 2018.
7.2. For 2016, only the data for September, October and December is included in the dataset.
7.3. For 2017, January to December data is available.
7.4. For 2018, January to October data is available.
7.5. Two files, order_reviews.csv and geolocation.csv had unclean data.
7.6. The dataset represents the data for 25 states and 1 federal territory of Brazil.

7.7. The sales data does not show any seasonal trends but shows year on year growth.
Plotting the sales data:



7.8. The sales trend plot in the dataset:

```
##        day              order_count
##  Min.   :2016-09-04   Min.   :   1.0
##  1st Qu.:2017-05-28   1st Qu.:  96.0
##  Median :2017-11-02   Median : 148.0
##  Mean   :2017-10-31   Mean   : 156.8
##  3rd Qu.:2018-04-09   3rd Qu.: 215.8
##  Max.   :2018-10-17   Max.   :1176.0
```



The plot reveals the following aspects of the data.
a. The plot shows gaps in between dates when orders were placed.
Further analysis shows the gaps between dates in the dataset as follows:

```
WITH d AS (
    SELECT DATE(order_purchase_timestamp) AS day,
           LAG(DATE(order_purchase_timestamp)) OVER
              (ORDER BY DATE(order_purchase_timestamp)) AS previous_date
    FROM orders
)
SELECT previous_date AS date, day AS next_date,
       datediff(day, previous_date) AS gap_between_dates
FROM d
WHERE DATEDIFF(day, previous_date) > 1
ORDER BY DATEDIFF(day, previous_date) DESC
```

Table 42: Displaying records 1 - 10

date	next_date	gap_between_dates
2016-10-22	2016-12-23	62
2016-09-15	2016-10-02	17
2016-12-23	2017-01-05	13
2018-10-03	2018-10-16	13
2016-10-10	2016-10-22	12
2016-09-05	2016-09-13	8

date	next_date	gap_between_dates
2018-09-20	2018-09-25	5
2018-09-06	2018-09-10	4
2018-09-13	2018-09-17	4
2018-08-31	2018-09-03	3

b. The trend shoots up during the 4th quarter of 2017 and the order placed per day sees a steep rise.

c. One data point shows extreme deviation of 1176 orders on a single day on 24th November 2017. This count deviated greatly from the Median of 148 orders and Mean of 156 orders.

```
WITH d AS (
    SELECT DISTINCT DATE(order_purchase_timestamp) AS date,
           COUNT(order_id) OVER (PARTITION BY DATE(order_purchase_timestamp))
              AS count_orders
    FROM orders
)
SELECT date, count_orders
FROM d
ORDER BY count_orders DESC
```

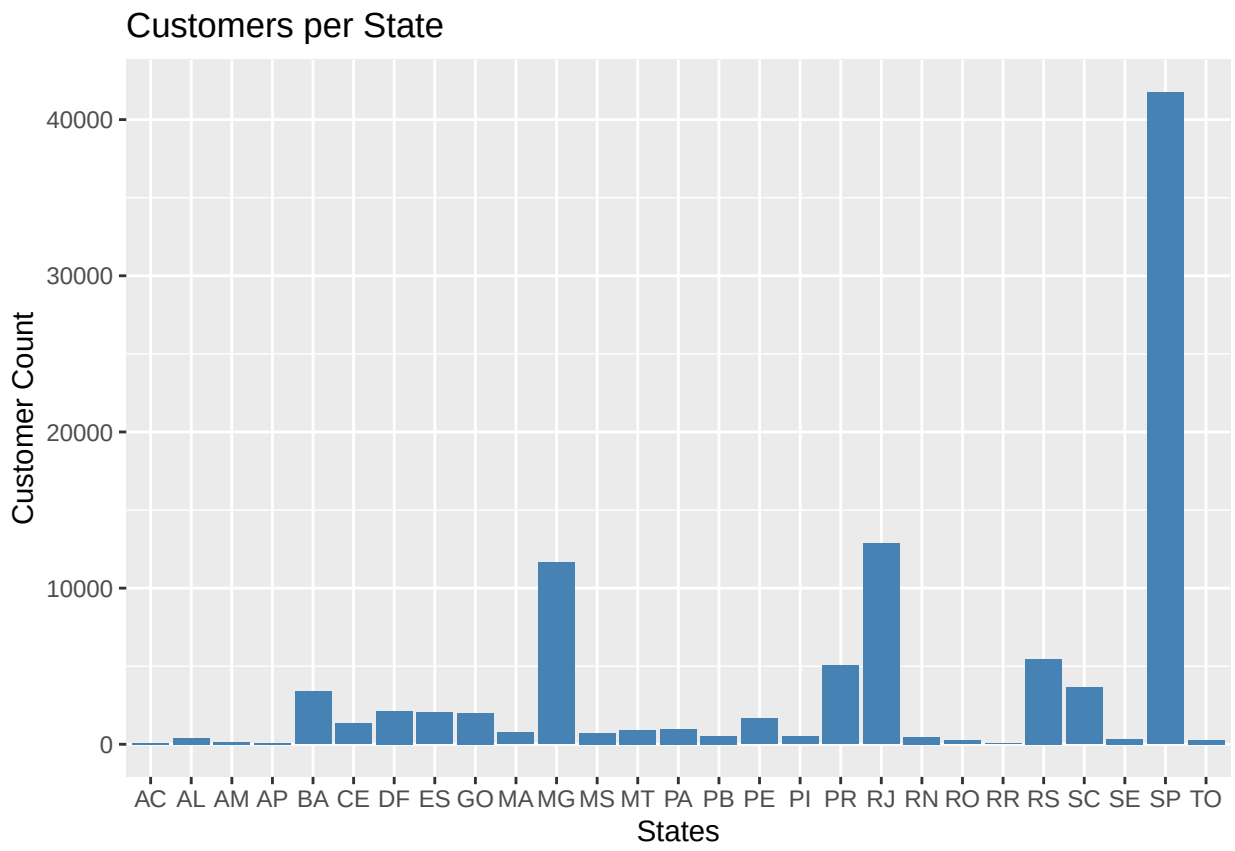
Table 43: Displaying records 1 - 10

date	count_orders
2017-11-24	1176
2017-11-25	499
2017-11-27	403
2017-11-26	391
2017-11-28	380
2018-05-07	372
2018-08-06	372
2018-08-07	370
2018-05-14	364
2018-05-16	357

Hence we can conclude that the data reported through orders.csv is not conducive for creating a predictive model.

7.8. Reviewing the location wise distribution of data:

```
##  customer_state     customer_id_count customer_unique_id_count
##  Length:27          Min.   :   46.0   Min.   :   46.0
##  Class :character   1st Qu.:  381.5   1st Qu.:  381.5
##  Mode  :character   Median :  907.0   Median :  907.0
##                     Mean   : 3683.0   Mean   : 3683.0
##                     3rd Qu.: 2760.0   3rd Qu.: 2760.0
##                     Max.   :41746.0   Max.   :41746.0
```




```sql
WITH d AS (
    SELECT DISTINCT customer_state,
           COUNT(customer_id) OVER (PARTITION BY customer_state)
              AS customer_id_count,
           COUNT(customer_unique_id) OVER (PARTITION BY customer_state)
              AS customer_unique_id_count
    FROM customers
)
SELECT customer_state, customer_id_count, customer_unique_id_count
FROM d
ORDER BY customer_id_count DESC
```

Table 44: Displaying records 1 - 10

customer_state	customer_id_count	customer_unique_id_count
SP	41746	41746
RJ	12852	12852
MG	11635	11635
RS	5466	5466
PR	5045	5045
SC	3637	3637
BA	3380	3380
DF	2140	2140
ES	2033	2033

customer_state	customer_id_count	customer_unique_id_count
GO	2020	2020

```
WITH d AS (
    SELECT DISTINCT c.customer_state,
           COUNT(o.order_id) OVER (PARTITION BY c.customer_state) AS orders_per_state
    FROM customers AS c
    JOIN orders AS o
    ON c.customer_id = o.customer_id
)
SELECT *
FROM d
ORDER BY orders_per_state DESC
```

Table 45: Displaying records 1 - 10

customer_state	orders_per_state
SP	41746
RJ	12852
MG	11635
RS	5466
PR	5045
SC	3637
BA	3380
DF	2140
ES	2033
GO	2020

7.9. Exploratory analysis reveals the following information regarding product deliveries.

a. There is a high deviation in estimated delivery dates and actual delivery to the customer. The estimated delivery dates require further analysis for more realistic estimates.

b. There is a high deviation in delivery as per carrier and actual delivery to the customer. Further investigation is required to drill into the issue.

c. There is a high deviation in delivery as per carrier and estimated delivery to the customer. Further investigation is required to drill into the issue.

```
WITH d AS (
    SELECT c.customer_state,
           COUNT(o.order_id) OVER (PARTITION BY c.customer_state)
              AS total_orders,
           (DATEDIFF(DATE(o.order_estimated_delivery_date),
              DATE(o.order_purchase_timestamp))) AS purchase_estimated_delivery,
           (DATEDIFF(DATE(o.order_delivered_carrier_date),
              DATE(o.order_purchase_timestamp))) AS purchase_delivered_carrier,
           (DATEDIFF(DATE(o.order_delivered_customer_date),
              DATE(o.order_purchase_timestamp))) AS purchase_delivered_customer
```

```
    FROM customers AS c
    JOIN orders AS o
    ON c.customer_id = o.customer_id
)
SELECT DISTINCT d.customer_state, d.total_orders,
       ROUND(AVG(d.purchase_estimated_delivery) OVER (PARTITION BY d.customer_state), 2)
          AS avg_estimated_delivery,
       ROUND(AVG(d.purchase_delivered_carrier) OVER (PARTITION BY d.customer_state), 2)
          AS avg_delivered_carrier,
       ROUND(AVG(d.purchase_delivered_customer) OVER (PARTITION BY d.customer_state), 2)
          AS avg_delivered_customer

FROM d
ORDER BY d.total_orders DESC
```

Table 46: Displaying records 1 - 10

customer_state	total_orders	avg_estimated_delivery	avg_delivered_carrier	avg_delivered_customer
SP	41746	19.81	3.15	8.70
RJ	12852	27.00	3.30	15.24
MG	11635	25.22	3.23	11.95
RS	5466	29.22	3.23	15.25
PR	5045	25.25	3.21	11.94
SC	3637	26.42	3.33	14.91
BA	3380	30.04	3.29	19.28
DF	2140	25.06	3.19	12.90
ES	2033	26.27	3.37	15.72
GO	2020	27.75	3.16	15.54

d. The following states require attention as they deviate from the average delivery time. Further investigation is required to drill into the issue.

```
WITH d AS (
    SELECT c.customer_state,
           COUNT(o.order_id) OVER (PARTITION BY c.customer_state)
              AS total_orders,
           (DATEDIFF(DATE(o.order_estimated_delivery_date),
              DATE(o.order_purchase_timestamp))) AS purchase_estimated_delivery,
           (DATEDIFF(DATE(o.order_delivered_carrier_date),
              DATE(o.order_purchase_timestamp))) AS purchase_delivered_carrier,
           (DATEDIFF(DATE(o.order_delivered_customer_date),
              DATE(o.order_purchase_timestamp))) AS purchase_delivered_customer

    FROM customers AS c
    JOIN orders AS o
    ON c.customer_id = o.customer_id
    ),
  d2 AS (
  SELECT DISTINCT d.customer_state, d.total_orders,
       ROUND(AVG(d.purchase_estimated_delivery) OVER (PARTITION BY d.customer_state), 2)
          AS avg_estimated_delivery,
       ROUND(AVG(d.purchase_delivered_carrier) OVER (PARTITION BY d.customer_state), 2)
```

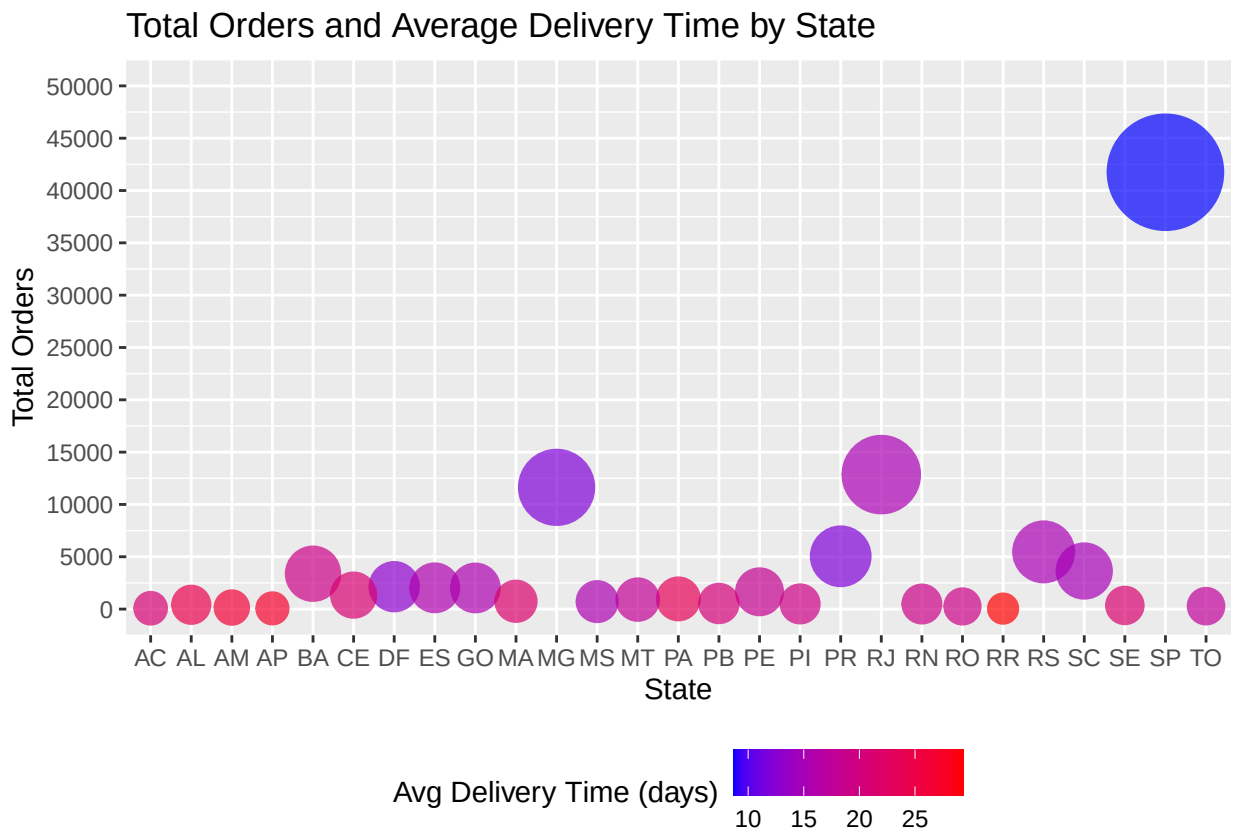
```
          AS avg_delivered_carrier,
       ROUND(AVG(d.purchase_delivered_customer) OVER (PARTITION BY d.customer_state), 2)
          AS avg_delivered_customer
  FROM d
  ),
  d3 AS (
  SELECT customer_state, total_orders, avg_delivered_customer,
       AVG(d2.avg_delivered_customer) OVER () AS avg_delivery_Country
  FROM d2
  )
SELECT customer_state, total_orders, avg_delivered_customer AS avg_delivery_State,
       ROUND(avg_delivery_Country, 2) AS avg_delivery_Country
FROM d3
WHERE ((avg_delivery_Country) - (avg_delivered_customer)) < 0
ORDER BY total_orders DESC
```

Table 47: Displaying records 1 - 10

customer_state	total_orders	avg_delivery_State	avg_delivery_Country
BA	3380	19.28	18.72
CE	1336	21.20	18.72
PA	975	23.73	18.72
MA	747	21.51	18.72
PB	536	20.39	18.72
PI	495	19.40	18.72
RN	485	19.22	18.72
AL	413	24.50	18.72
SE	350	21.46	18.72
RO	253	19.28	18.72

e. The following bubble chart shows total orders vs average delivery time. The size of the circle depicts order count. Colour gradient towards Blue indicates better delivery time. Colour gradient towards Red indicates worse delivery time. This chart will assist in identifying the states that requires attention to improve customer experience by improving delivery time.
For example, Sau Paulo has the largest volume of order and the best average delivery time.

```
##  customer_state      total_orders     avg_estimated_delivery
##  Length:27          Min.   :   46.0   Min.   :19.81
##  Class :character   1st Qu.:  381.5   1st Qu.:26.80
##  Mode  :character   Median :  907.0   Median :31.11
##                     Mean   : 3683.0   Mean   :32.08
##                     3rd Qu.: 2760.0   3rd Qu.:33.41
##                     Max.   :41746.0   Max.   :47.17
##  avg_delivered_carrier avg_delivered_customer
##  Min.   :2.820         Min.   : 8.70
##  1st Qu.:3.185         1st Qu.:15.39
##  Median :3.300         Median :19.22
##  Mean   :3.343         Mean   :18.72
##  3rd Qu.:3.460         3rd Qu.:21.33
##  Max.   :4.530         Max.   :29.34
```



7.10. Analysis of products:

a. Top 2 selling products in each state is given by:

```
WITH d AS (
    SELECT DISTINCT c.customer_state, p.`product category`,
           COUNT(`product category`) OVER (PARTITION BY customer_state, `product category`)
              AS count_product
    FROM customers AS c
    JOIN orders AS o
    ON c.customer_id = o.customer_id
    JOIN order_items AS oi
    ON o.order_id = oi.order_id
    JOIN products AS p
    ON oi.product_id = p.product_id
    ),
  d2 AS (
    SELECT d.customer_state, d.`product category`, d.count_product,
       DENSE_RANK() OVER (PARTITION BY d.customer_state ORDER BY count_product DESC)
          AS product_rank
    FROM d
    )
SELECT d2.customer_state, d2.`product category`, d2.count_product, d2.product_rank
FROM d2
WHERE d2.product_rank <= 2 AND d2.count_product > 500
ORDER BY d2.customer_state, d2.product_rank, d2.count_product DESC
```

Table 48: 9 records

customer_state	product category	count_product	product_rank
MG	bed table bath	1331	1
MG	HEALTH BEAUTY	1086	2
PR	Furniture Decoration	520	1
RJ	bed table bath	1644	1
RJ	Furniture Decoration	1090	2
RS	bed table bath	614	1
RS	Furniture Decoration	561	2
SP	bed table bath	5235	1
SP	HEALTH BEAUTY	4204	2

b. Top selling products in the dataset are as follows:

```
SELECT DISTINCT p.`product category` AS product_category,
       COUNT(oi.order_item_id) OVER (PARTITION BY `product category`)
              AS count_product_category
FROM customers AS c
JOIN orders AS o
    ON c.customer_id = o.customer_id
JOIN order_items AS oi
    ON o.order_id = oi.order_id
JOIN products AS p
    ON oi.product_id = p.product_id
ORDER BY count_product_category DESC
```

Table 49: Displaying records 1 - 10

product_category	count_product_category
bed table bath	11115
HEALTH BEAUTY	9670
sport leisure	8641
Furniture Decoration	8334
computer accessories	7827
housewares	6964
Watches present	5991
telephony	4545
Garden tools	4347
automotive	4235

c. Sales trend in the data set for top selling product category:

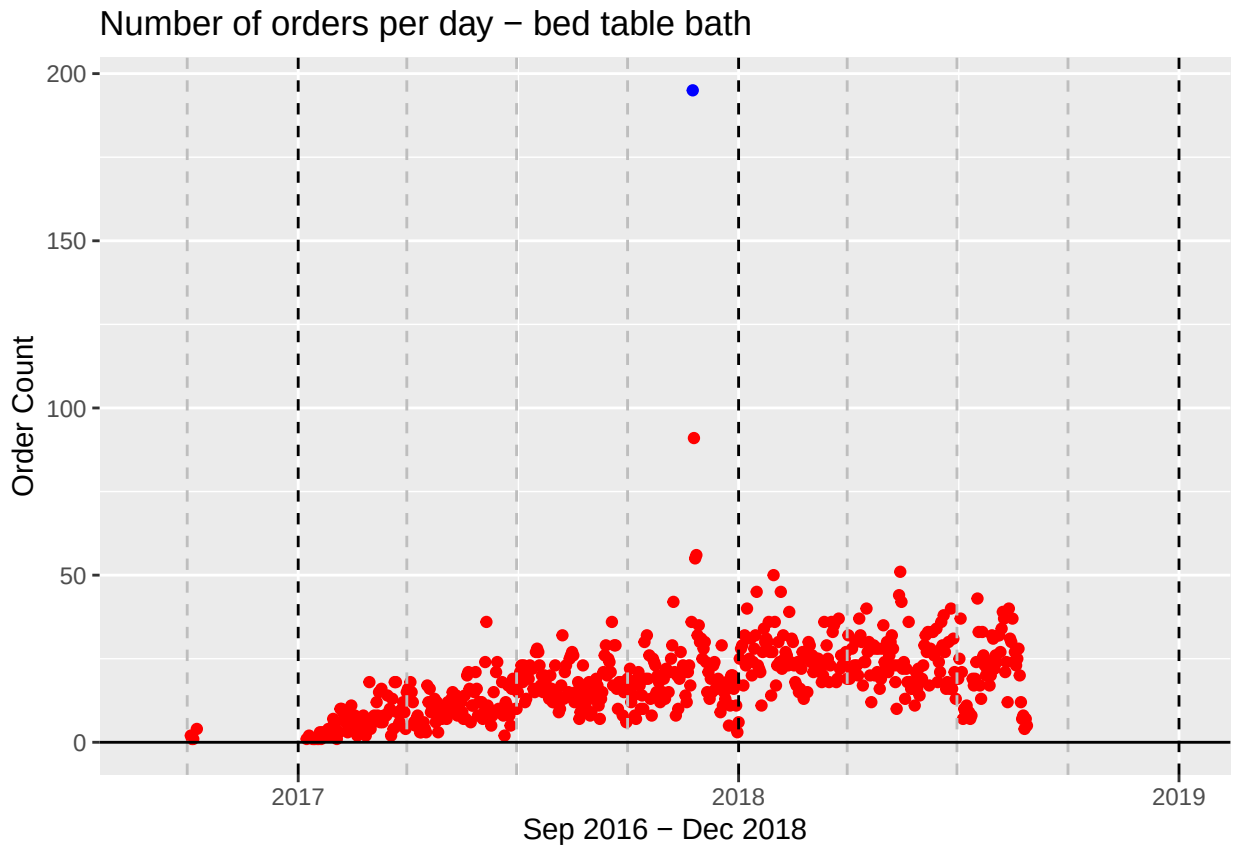
```
WITH d AS (
    SELECT DISTINCT p.`product category` AS product_category,
           COUNT(oi.order_item_id) OVER (PARTITION BY `product category`)
              AS count_product_category
    FROM customers AS c
      JOIN orders AS o
        ON c.customer_id = o.customer_id
    JOIN order_items AS oi
```

```
        ON o.order_id = oi.order_id
    JOIN products AS p
        ON oi.product_id = p.product_id
  ),
  d2 AS (
    SELECT d.product_category, d.count_product_category,
           DENSE_RANK() OVER (ORDER BY d.count_product_category DESC)
                AS rank_category
    FROM d
  ),
  top_prod_cat AS (
    SELECT d2.product_category
    FROM d2
    WHERE d2.rank_category = 1
 )
SELECT DISTINCT DATE(o.order_purchase_timestamp) as day,
       COUNT(o.order_id) OVER (PARTITION BY DATE(o.order_purchase_timestamp))
            AS order_count
FROM customers AS c
  JOIN orders AS o
       ON c.customer_id = o.customer_id
  JOIN order_items AS oi
       ON o.order_id = oi.order_id
  JOIN products AS p
       ON oi.product_id = p.product_id
WHERE p.`product category` IN (SELECT product_category FROM top_prod_cat)
```

Table 50: Displaying records 1 - 10

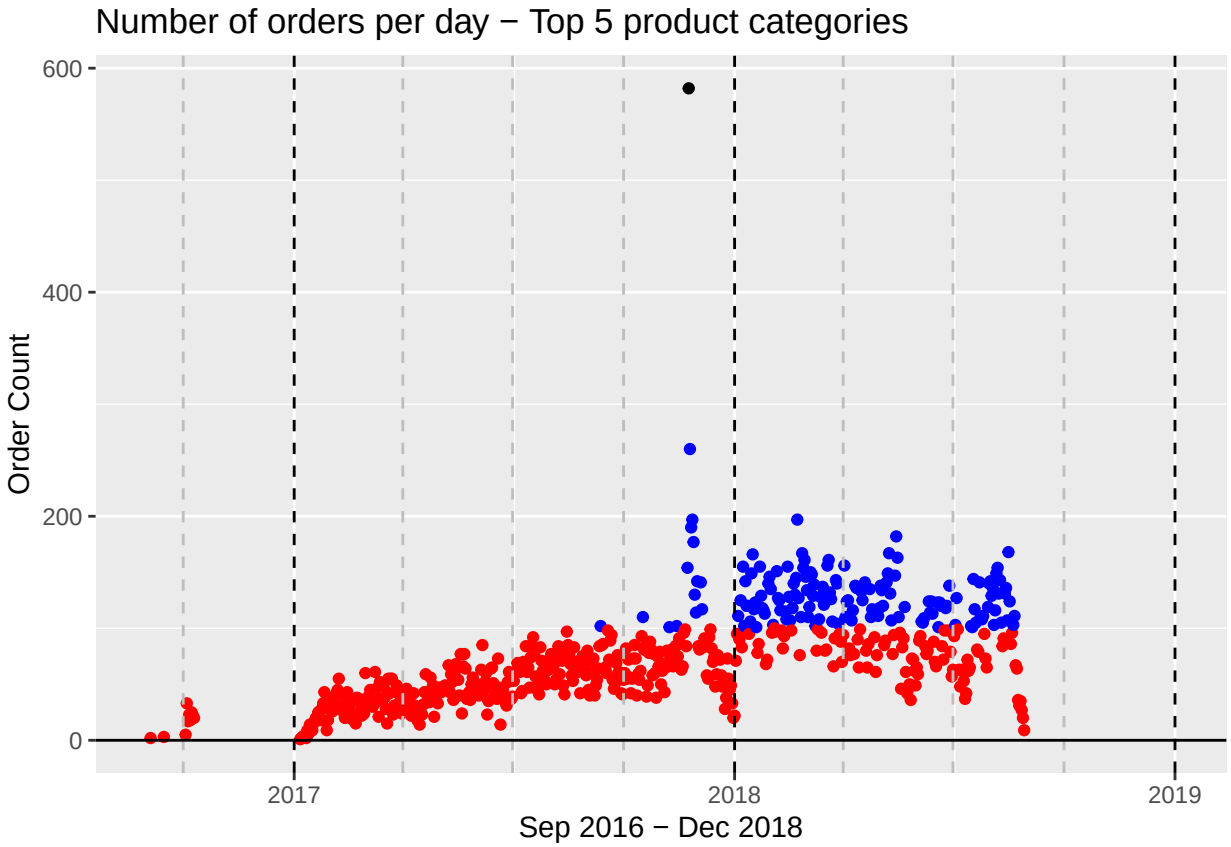
day	order_count
2016-10-04	2
2016-10-05	1
2016-10-06	1
2016-10-09	4
2017-01-08	1
2017-01-10	2
2017-01-13	1
2017-01-14	1
2017-01-15	1
2017-01-17	1

```
##       day              order_count
##  Min.   :2016-10-04   Min.   :  1.00
##  1st Qu.:2017-06-06   1st Qu.: 10.00
##  Median :2017-11-02   Median : 18.00
##  Mean   :2017-11-01   Mean   : 18.59
##  3rd Qu.:2018-03-31   3rd Qu.: 24.00
##  Max.   :2018-08-28   Max.   :195.00
```



d. Sales trend in the data set for top 5 selling product category:

```
##        day              order_count
##  Min.   :2016-09-04   Min.   :  1.00
##  1st Qu.:2017-05-28   1st Qu.: 43.00
##  Median :2017-10-28   Median : 67.00
##  Mean   :2017-10-26   Mean   : 74.61
##  3rd Qu.:2018-03-29   3rd Qu.: 99.50
##  Max.   :2018-08-29   Max.   :582.00
```



Programming environment for analysis of the dataset is as follows

1. RStudio for markdown and plotting.
2. MySQL as database server.
3. Docker for hosting RStudio and MySQL server.
4. Link to the GitHub Repository for the case study.
