Retail Store Case Study

Shobhit Asati

Problem Statement

Assuming you are a data analyst/scientist at some retail store company, you have been assigned the task of analyzing the given datasets to extract valuable insights and provide actionable recommendations.

Report Structure

1. Data Cleaning.
2. Initial exploration.
3. In-depth exploration.
4. Evolution of E-commerce Orders in the Brazil region.
5. Impact on Economy.
6. Analysis on Sales, Freight & Delivery Time.
7. Analysis based on the payments.
8. Actionable Insights & Recommendations.

1 Data Cleaning

```sql
DROP DOMAIN IF EXISTS UINT CASCADE;
CREATE DOMAIN UINT AS INTEGER CHECK (VALUE >= 0);
```

Listing 1: Create Domain

```sql
DROP TABLE IF EXISTS customer;

CREATE TABLE customer (
  id TEXT,
  unique_id ID TEXT,
  zip_code_prefix CHAR(5),
  city TEXT,
  state TEXT
);

COPY customer
FROM ’path/to/customers.csv’
WITH (FORMAT CSV, HEADER);
```

Listing 2: Setup Customer Table

```sql
DROP TABLE IF EXISTS seller;

CREATE TABLE seller (
id TEXT,
zip_code_prefix CHAR(5),
city TEXT,
state TEXT
);

COPY seller
FROM ’path/to/sellers.csv’
WITH (FORMAT CSV, HEADER);
```

Listing 3: Setup Seller Table

```sql
DROP TABLE IF EXISTS orders;

CREATE TABLE orders (
  id TEXT,
  customer_id TEXT,
  status TEXT,
  purchase_timestamp TIMESTAMP WITHOUT TIME ZONE,
  approved_at TIMESTAMP WITHOUT TIME ZONE,
  delivered_carrier_date TIMESTAMP WITHOUT TIME ZONE,
  delivered_customer_date TIMESTAMP WITHOUT TIME ZONE,
  estimated_delivery_date DATE
);
COPY orders
FROM ’path/to/orders.csv’
WITH (FORMAT CSV, HEADER);
```

Listing 4: Setup Orders Table

```sql
DROP TABLE IF EXISTS order_item;

CREATE TABLE order_item (
id TEXT,
item_id ID UINT,
product_id TEXT,
seller_id TEXT,
shipping_limit_date TIMESTAMP,
price REAL,
freight_value REAL
);

COPY order_item
FROM 'path/to/order_items.csv'
WITH (FORMAT CSV, HEADER);
```

Listing 5: Setup Order Item Table

```sql
DROP TABLE IF EXISTS order_review;

CREATE TABLE order_review (
id TEXT,
order_id TEXT,
score UINT,
comment_title TEXT,
creation_date TEXT,
answer_timestamp TEXT
);

COPY order_review
FROM 'path/to/order_reviews.csv'
WITH (FORMAT CSV, HEADER);

ALTER TABLE order_review
ALTER COLUMN answer_timestamp TYPE TIMESTAMP WITHOUT TIME ZONE USING
CASE
WHEN ANSWER_TIMESTAMP ~ '^[0-9]{2}/[0-9]{2}/[0-9]{2} [0-9]{1,2}:[0-9]{2}$' THEN TO_TIMESTAMP(ANSWER_TIMESTAMP, 'DD/MM/YY HH24:MI')
ELSE NULL
END;

ALTER TABLE ORDER_REVIEW
ALTER COLUMN creation_date TYPE DATE USING
CASE
WHEN CREATION_DATE ~ '^[0-9]{2}/[0-9]{2}/[0-9]{2} [0-9]{1,2}:[0-9]{2}$' THEN TO_DATE(CREATION_DATE, 'DD/MM/YY')
ELSE NULL -- Set to NULL if the format is invalid
END;
```

Listing 6: Setup Order Review Table

```sql
DROP TABLE IF EXISTS payment;

CREATE TABLE payment (
  order_id TEXT,
  sequential UINT,
  type TEXT,
  installments UINT,
  values REAL
);

COPY payment
FROM 'path/to/payments.csv'
WITH (FORMAT CSV, HEADER);
```

Listing 7: Setup Payment Table

```sql
DROP TABLE IF EXISTS geo_location;

CREATE TABLE geo_location (
  zip_code_prefix CHAR(5),
  latitude DOUBLE PRECISION,
  longitude DOUBLE PRECISION,
  city TEXT,
  state TEXT
);

COPY geo_location
FROM 'path/to/geolocation.csv'
WITH (FORMAT CSV, HEADER);
```

Listing 8: Setup Geo-Location Table

```sql
DROP TABLE IF EXISTS product;

CREATE TABLE product (
  id TEXT,
  category TEXT,
  name_length UINT,
  description_length UINT,
  photos_qty UINT,
  weight_g UINT,
  length_cm UINT,
  height_cm UINT,
  width_cm UINT
);

COPY product
FROM 'path/to/products.csv'
WITH (FORMAT CSV, HEADER);
```

Listing 9: Setup Product Table

You can download dataset from here.

2 Initial exploration

Ques 2.1 – Data type of all the columns in *customers* table.

```
    SELECT
      table_name, column_name, data_type
    FROM
      information_schema.columns
    WHERE
      table_name = 'customer';
```

Listing 10: Schema Information

	table_name	column_name	data_type
1	customer	zip_code_prefix	integer
2	customer	id	text
3	customer	unique_id	text
4	customer	city	text
5	customer	state	text

Table 10: Schema Information

Query in `Listing 10` demonstrate how to retrieve information about a schema columns and its data type. We can also list tables from the database by replacing `information_schema.columns` with `information_schema.tables` and specifying the schema in `WHERE` clause.

Ques 2.2 – Get the date range between which the orders were placed.

```
    SELECT
      'first order' AS order_type,
      DATE(MIN(purchase_timestamp)) AS order_date,
      CAST(MIN(purchase_timestamp) AS TIME) AS order_time
    FROM orders
    UNION
    SELECT
      'last order' AS order_type,
      DATE(MAX(purchase_timestamp)) AS order_date,
      CAST(MAX(purchase_timestamp) AS TIME) AS order_time
    FROM orders;
```

Listing 11: Date Range

	order_type	order_date	order_time
1	first order	2016-09-04	21:15:19
2	last order	2018-10-17	17:30:18

Table 11: Date Range

Ques 2.3 – Count the Cities & States of customers who ordered during the given period.

```sql
SELECT
  COUNT(DISTINCT city) AS city_count,
  COUNT(DISTINCT state) AS state_count
FROM customer cst
  JOIN orders ord ON ord.customer_id = cst.id;
```

Listing 12: Count of Cities & States

	city_count	state_count
1	4119	27

Table 12: Count of Cities & States

Ques 2.4 – Count the number of distinct Sellers, Products & Customers

```sql
SELECT
  ’sellers’ AS "# of distinct",
  COUNT(id) AS counts
FROM seller
UNION
SELECT
  ’products’ AS "# of distinct",
  COUNT(id) AS counts
FROM product
UNION
SELECT
  ’customers’ AS "# of distinct",
  COUNT(DISTINCT unique_id) AS counts
FROM customer
ORDER BY counts;
```

Listing 13: Count of distinct Sellers, Products & Customers

	# of distinct	counts
1	sellers	3095
2	products	32951
3	customers	96096

Table 13: Count of distinct Sellers, Products & Customers

Ques 2.5 – Percentage share of review scores.

```
SELECT
  CONCAT(score, ’ stars’) AS review_score,
  ROUND(100.0 * COUNT(score) / (SELECT COUNT(*) FROM order_review), 2)
      || ’%’ AS percentage_share
FROM order_review
GROUP BY score
ORDER BY score;
```

Listing 14: Review Score Percentage Share

review_score	percentage_share
1 stars	11.51%
2 stars	3.18%
3 stars	8.24%
4 stars	19.29%
5 stars	57.78%

Table 14: Review Score Percentage Share

Ques 2.6 – Percentage share of status of orders.

```
SELECT
  INITCAP(status) AS order_status,
  ROUND(100.0 * COUNT(status) / (SELECT COUNT(*) ), 2) || ’%’ AS
     percentage_share
FROM orders
GROUP BY status
ORDER BY percentage_share;
```

Listing 15: Order Status Percentage Share

	order_status	percentage_share
1	Approved	0.00%
2	Created	0.01%
3	Processing	0.30%
4	Invoiced	0.32%
5	Unavailable	0.61%
6	Canceled	0.63%
7	Shipped	1.11%
8	Delivered	97.02%

Table 15: Order Status Percentage Share

3 In-depth exploration

Ques 3.1 – Is there a growing trend in the no. of orders placed over the past years?

```
WITH total_orders AS (
  SELECT COUNT(*) AS total_order_count
  FROM orders
)
SELECT
  EXTRACT(YEAR FROM purchase_timestamp) AS purchase_year,
  COUNT(*) AS total_orders,
  ROUND(100.0 * COUNT(*) / total_order_count::NUMERIC, 2) || ’%’ AS
      percentage_share
FROM orders, total_orders
GROUP BY purchase_year, total_order_count
ORDER BY purchase_year;
```

Listing 16: Number of Orders Per Year

	purchase_year	total_orders	percentage_share
1	2016	329	0.33%
2	2017	45101	45.35%
3	2018	54011	54.31%

Table 16: Number of Orders Per Year

Ques 3.2 – What is the percentage of orders without reviews and the average time taken to provide a review for each score category?

```
SELECT
score || ’ stars’ AS review_score,
ROUND(100.0 - 100.0 * COUNT(comment_title) / COUNT(1), 2) || ’%’ AS
    non_reviewed_orders,
ROUND(AVG(
EXTRACT(EPOCH FROM answer_timestamp - creation_date) / (24*3600)
), 2) || ’ days’ AS avg_time_to_give_review
FROM order_review
GROUP BY score;
```

Listing 17: Non-reviewed % & Average Review Time

review_score	non_reviewed_orders	avg_time_to_give_review
1 stars	83.62%	3.05 days
2 stars	84.86%	3.00 days
3 stars	89.94%	2.98 days
4 stars	90.95%	3.12 days
5 stars	88.41%	3.21 days

Table 17: Non-reviewed % & Average Review Time

Ques 3.3 – During what time of the day, do the customers mostly place their orders? (Dawn – 0 to 6 , Morning – 7 to 12, Afternoon – 13 to 18 or Night – 19 to 23)

```sql
WITH time_of_day AS (
  SELECT
    CASE
      WHEN EXTRACT(HOUR FROM purchase_timestamp) BETWEEN 0 AND 6 THEN 'Dawn'
      WHEN EXTRACT(HOUR FROM purchase_timestamp) BETWEEN 7 AND 12 THEN 'Morning'
      WHEN EXTRACT(HOUR FROM purchase_timestamp) BETWEEN 13 AND 18 THEN 'Afternoon'
      WHEN EXTRACT(HOUR FROM purchase_timestamp) BETWEEN 19 AND 23 THEN 'Night'
    END AS phase_of_day,
    COUNT(*) AS phase_order_count
  FROM orders
  GROUP BY phase_of_day
), overall_orders AS (
  SELECT COUNT(*) AS total_orders
  FROM orders
)
SELECT
  phase_of_day,
  phase_order_count,
  ROUND(100.0 * phase_order_count / (SELECT total_orders FROM overall_orders), 2) || '%' AS percentage_share
FROM time_of_day
ORDER BY phase_order_count DESC;
```

Listing 18: # Orders by Phase of the Day

	phase_of_day	phase_order_count	percentage_share
1	Afternoon	38135	38.35%
2	Night	28331	28.49%
3	Morning	27733	27.89%
4	Dawn	5242	5.27%

Table 18: # Orders by Phase of the Day

Ques 3.4 – Get the number of orders placed on hourly basis.

```sql
WITH hourly_purchase AS (
  SELECT COUNT(id) AS hourly_count,
    EXTRACT(HOUR FROM purchase_timestamp) AS hour_of_day
  FROM orders
  GROUP BY EXTRACT(HOUR FROM purchase_timestamp)
), overall_purchase AS (
  SELECT COUNT(id) AS total_purchase
  FROM orders
)
SELECT
  hour_of_day, hourly_count,
  ROUND(100.0 * hourly_count / total_purchase::NUMERIC, 2) || '%' AS
     percentage_share
FROM hourly_purchase, overall_purchase
ORDER BY hour_of_day;
```

Listing 19: Hourly Purchase

hour_of_day	hourly_count	percentage_share
0	2394	2.41%
1	1170	1.18%
2	510	0.51%
3	272	0.27%
4	206	0.21%
5	188	0.19%
6	502	0.50%
7	1231	1.24%
8	2967	2.98%
9	4785	4.81%
10	6177	6.21%
11	6578	6.61%
12	5995	6.03%
13	6518	6.55%
14	6569	6.61%
15	6454	6.49%
16	6675	6.71%
17	6150	6.18%
18	5769	5.80%
19	5982	6.02%
20	6193	6.23%
21	6217	6.25%
22	5816	5.85%
23	4123	4.15%

Table 19: Hourly Purchase

Ques 3.5 – Can we see some kind of monthly seasonality in terms of the no. of orders being placed?

```
WITH monthly_orders AS (
  SELECT
    EXTRACT(MONTH FROM delivered_customer_date) AS month_number,
    SUM(CASE WHEN EXTRACT(YEAR FROM delivered_customer_date) = 2016
       THEN 1 ELSE 0 END) AS orders_2016,
    SUM(CASE WHEN EXTRACT(YEAR FROM delivered_customer_date) = 2017
       THEN 1 ELSE 0 END) AS orders_2017,
    SUM(CASE WHEN EXTRACT(YEAR FROM delivered_customer_date) = 2018
       THEN 1 ELSE 0 END) AS orders_2018,
    ROW_NUMBER() OVER (ORDER BY EXTRACT(MONTH FROM
       delivered_customer_date)) AS row_num
  FROM order_item oi
    JOIN orders o ON oi.order_id = o.id
  GROUP BY EXTRACT(MONTH FROM delivered_customer_date)
  ORDER BY EXTRACT(MONTH FROM delivered_customer_date)
)
SELECT month_number,
  orders_2016,
  CASE
    WHEN orders_2016 - LAG(orders_2016) OVER (ORDER BY row_num) > 0
       THEN ’up-trend’
    WHEN ABS(100.0 * (orders_2016 - LAG(orders_2016) OVER (ORDER BY
       row_num)) /
    (SELECT COUNT(*) AS total_orders_2016  WHERE EXTRACT(YEAR FROM
       delivered_customer_date) = 2016)) > 2 THEN ’down-trend’
    ELSE ’break-even’
  END AS trend_2016,
  orders_2017,
  CASE
    WHEN orders_2017 - LAG(orders_2017) OVER (ORDER BY row_num) > 0
       THEN ’up-trend’
    WHEN ABS(100.0 * (orders_2017 - LAG(orders_2017) OVER (ORDER BY
       row_num)) /
    (SELECT COUNT(*) AS total_orders_2017  WHERE EXTRACT(YEAR FROM
       delivered_customer_date) = 2017)) > 2 THEN ’down-trend’
    ELSE ’break-even’
  END AS trend_2017,
  orders_2018,
  CASE
    WHEN orders_2018 - LAG(orders_2018) OVER (ORDER BY row_num) > 0
       THEN ’up-trend’
    WHEN ABS(100.0 * (orders_2018 - LAG(orders_2018) OVER (ORDER BY
       row_num)) /
    (SELECT COUNT(*) AS total_orders_2016  WHERE EXTRACT(YEAR FROM
       delivered_customer_date) = 2018)) > 2 THEN ’down-trend’
    ELSE ’break-even’
  END AS trend_2018
FROM monthly_orders
WHERE month_number IS NOT NULL;
```

Listing 20: Monthly Seasonality Over The Years

month_number	orders_2016	trend_2016	orders_2017	trend_2017	orders_2018	trend_2018
1	0	break-even	326	break-even	7419	break-even
2	0	break-even	1565	up-trend	6623	break-even
3	0	break-even	2724	up-trend	7948	up-trend
4	0	break-even	2072	break-even	8999	up-trend
5	0	break-even	4201	up-trend	8166	break-even
6	0	break-even	3640	break-even	7887	break-even
7	0	break-even	3933	up-trend	6581	down-trend
8	0	break-even	4900	up-trend	9385	up-trend
9	0	break-even	4496	break-even	59	down-trend
10	245	up-trend	5215	up-trend	3	break-even
11	74	down-trend	5412	up-trend	0	break-even
12	4	down-trend	8319	up-trend	0	break-even

Table 20: Monthly Seasonality Over The Years

1. Next month's order count will be classified as a *down-trend* if it decreases by more than 2% compared to the previous month within the current year.
   Otherwise, it will be classified as *break-even*.

2. Number of orders are growing year-on-year on monthly basis i.e., from left to right.

4 Evolution of E-Commerce orders in the Brazil region

Ques 4.1 – Get the month on month number of orders placed in each state.

```
    WITH order_counts AS (
      SELECT
        cst.state AS state ,
        EXTRACT(MONTH FROM ord.delivered_customer_date) AS order_month ,
        COUNT(*) AS counts
       FROM orders o
        JOIN customer cst ON cst.id = o.customer_id
      WHERE EXTRACT(MONTH FROM delivered_customer_date) IS NOT NULL
      GROUP BY state , order_month
    )
    SELECT
      state ,
      SUM(counts) FILTER (WHERE order_month = 1) AS jan ,
      SUM(counts) FILTER (WHERE order_month = 2) AS feb ,
      SUM(counts) FILTER (WHERE order_month = 3) AS mar ,
      SUM(counts) FILTER (WHERE order_month = 4) AS apr ,
      SUM(counts) FILTER (WHERE order_month = 5) AS may ,
      SUM(counts) FILTER (WHERE order_month = 6) AS jun ,
      SUM(counts) FILTER (WHERE order_month = 7) AS jul ,
      SUM(counts) FILTER (WHERE order_month = 8) AS aug ,
      SUM(counts) FILTER (WHERE order_month = 9) AS sep ,
      SUM(counts) FILTER (WHERE order_month = 10) AS oct ,
      SUM(counts) FILTER (WHERE order_month = 11) AS nov ,
      SUM(counts) FILTER (WHERE order_month = 12) AS decm
    FROM order_counts
    GROUP BY state
    ORDER BY
      jan DESC , feb DESC , mar DESC , apr DESC ,
      may DESC , jun DESC , jul DESC , aug DESC ,
      sep DESC , oct DESC , nov DESC , decm DESC;
```

Listing 21: Top 5 States

	state	jan	feb	mar	apr	may	jun	jul	aug	sep	oct	nov	decm
1	SP	2731	3046	3924	4004	4458	4346	3967	5607	1520	1819	1997	3076
2	RJ	931	942	1084	1303	1511	1203	1136	1523	543	678	624	875
3	MG	815	897	1093	1210	1172	1191	1069	1327	493	568	517	1003
4	PR	369	396	504	512	524	527	479	653	165	208	228	358
5	RS	352	371	541	603	581	528	520	672	274	288	251	363
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table 21: Top 5 States

Ques 4.2 – This analysis will examine the geographical distribution of customers and sellers across all states. Additionally, we will investigate the correlation between freight costs and total spending.

```sql
WITH customer_counts AS (
  SELECT state, COUNT(unique_id) AS customer_count
  FROM customer
  GROUP BY state
), seller_counts AS (
  SELECT state, COUNT(id) AS seller_count
  FROM seller
  GROUP BY state
), freight_values AS (
  SELECT state, SUM(freight_value) AS total_freight, SUM(price) AS total_price
  FROM order_item oi
    JOIN orders o ON oi.order_id = o.id
    JOIN customer cst ON o.customer_id = cst.id
  GROUP BY state
)
SELECT cc.state, cc.customer_count, sc.seller_count,
       ROUND((100 * fv.total_freight/(fv.total_price + fv.total_freight))::NUMERIC, 2) || '%' AS freight_vs_total_spent
FROM customer_counts cc
  FULL OUTER JOIN seller_counts sc ON cc.state = sc.state
  FULL OUTER JOIN freight_values fv ON cc.state = fv.state
ORDER BY sc.seller_count DESC, cc.customer_count DESC,
         fv.total_freight DESC, state;
```

Listing 22: Customer & Seller count & Freight Percentage

	state	customer_count	seller_count	freight_vs_total_spent
1	AL	413	Null	16.54%
2	TO	280	Null	19.12%
3	AP	68	Null	17.15%
4	RR	46	Null	22.21%
5	SP	41746	1849	12.14%
6	PR	5045	349	14.71%
7	MG	11635	244	14.59%
8	SC	3637	190	14.69%
9	RJ	12852	171	14.35%
10	RS	5466	129	15.30%
11	GO	2020	40	15.28%
12	DF	2140	30	14.33%
13	ES	2033	23	15.32%
	⋮	⋮	⋮	⋮

Table 22: Customer & Seller count & Freight Percentage

5 Impact on Economy

Ques 5.1 – Get the % increase in the cost of orders from year 2017 to 2018 (include months between *January* to *August* only).

```sql
WITH yearly_payment_totals AS (
  SELECT
    EXTRACT(YEAR FROM delivered_customer_date) AS year,
    SUM(value) AS total_value
  FROM payment pmt
    JOIN orders o ON o.id = pmt.order_id
  WHERE EXTRACT(MONTH FROM delivered_customer_date) BETWEEN 1 AND 8
  GROUP BY EXTRACT(YEAR FROM delivered_customer_date)
)
SELECT
  100.0 * ROUND((
        (y2018.total_value - y2017.total_value)
          / (y2018.total_value + y2017.total_value)
      )::NUMERIC, 5
    ) || '%' AS growth_rate
FROM yearly_payment_totals AS y2018
  JOIN yearly_payment_totals AS y2017 ON y2017.year = 2017
    AND y2018.year = 2018;
```

Listing 23: Growth Rate from 2017 to 2018

	growth_rate
1	46.603%

Table 23: Growth Rate from 2017 to 2018

Ques 5.2 – Calculate the Total & Average value of order price for each state.

```
  SELECT
    state,
    ROUND(SUM(price)::NUMERIC, 2) AS total_price,
    ROUND(AVG(price)::NUMERIC, 2) AS average_price
  FROM customer cst
    JOIN orders o ON ord.customer_id = cst.id
    JOIN order_item oi ON oi.order_id = o.id
  GROUP BY state
  ORDER BY state;
```

Listing 24: Total & Average Price per State

	state	total_price	average_price
1	AC	15983.00	173.73
2	AL	80314.80	180.89
3	AM	22356.80	135.50
	⋮	⋮	⋮

Table 24: Total & Average Price per State

Ques 5.3 – Calculate the Total & Average value of order freight for each state.

```
  SELECT
    state,
    ROUND(SUM(freight_value)::NUMERIC, 2) AS total_freight,
    ROUND(AVG(freight_value)::NUMERIC, 2) AS average_freight
  FROM customer cst
    JOIN orders o ON o.customer_id = cst.id
    JOIN order_item oi ON oi.order_id = ord.id
  GROUP BY state
  ORDER BY state;
```

Listing 25: Total & Average Freight per State

	state	total_freight	average_freight
1	AC	3686.75	40.07
2	AL	15914.60	35.84
3	AM	5478.89	33.21
	⋮	⋮	⋮

Table 25: Total & Average Freight per State

6 Analysis on Sales, Freight & Delivery Time

Ques 6.1 – Find the number 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.

```sql
SELECT id,
  delivered_customer_date::DATE - purchase_timestamp::DATE AS
      purchase_to_deliverd,
  delivered_customer_date::DATE - estimated_delivery_date::DATE AS
      estimated_to_deliverd
FROM orders
WHERE delivered_customer_date IS NOT NULL
  AND estimated_delivery_date IS NOT NULL
  AND delivered_customer_date > estimated_delivery_date
ORDER BY
  id, estimated_to_delivered, purchase_to_delivered;
```

Listing 26: Purchase & Estimated vs Actual Delivery Date

	id	purchase_to_delivered	estimated_to_delivered
1	0005a1a1728c9d785b8e2b08b904576c	10	0
2	00063b381e2406b52ad429470734ebd5	11	0
3	000e906b789b55f64edcb1f84030f90d	18	2
4	0017afd5076e074a48f1f1a4c7bac9c5	47	4
5	001c85b5f68d2be0cb0797afc9e8ce9a	28	8
6	001d8f0e34a38c37f7dba2a37d4eba8b	12	2
7	0030d783f979fbc5981e75613b057344	43	22
8	00324b3eda39ba5ecce3945823e3594c	23	3
9	0032d07457ae9c806c79368d7d9ce96b	50	12
10	00335b686d693c7d72deeb12f8e89227	57	32
11	00378c6c981f234634c0b9d6128df6dd	24	0
12	003a94f778ef8cfd50247c8c1b582257	21	9
13	003d804eef0e1b856881cd18e0cc0d4c	54	23
14	003f201cdd39cdd59b6447cff2195456	35	6
15	004f5d8f238e8908e6864b874eda3391	24	2
16	00526a9d4ebde463baee25f386963ddc	9	1
17	005e5166e99d1e4d0c4f808b0540ba94	34	12
18	00685d31ae12e47470ba5c18ba74f22c	38	8
19	0084e195fbd72ae51599af47f04afede	51	24
20	008a1b3db2a8bf63418c2cf7c7f494b1	31	10
	⋮	⋮	⋮

Table 26: Purchase & Estimated vs Actual Delivery Date

Ques 6.2 Find out the top 5 states with the highest & lowest average freight value.

```sql
SELECT state,
  top_5_avg_freight,
  NULL AS bottom_5_avg_freight
FROM (
  SELECT state,
    ROUND(AVG(freight_value)::NUMERIC, 2) AS top_5_avg_freight,
    RANK() OVER (ORDER BY ROUND(AVG(freight_value)::NUMERIC, 2) DESC)
       AS ranked
  FROM order_item oi
    JOIN seller sell ON oi.seller_id = sell.id
  GROUP BY state
) AS top_5
WHERE ranked <= 5

UNION

SELECT state,
  NULL AS top_5_avg_freight,
  bottom_5_avg_freight
FROM (
    SELECT state,
    ROUND(AVG(freight_value)::NUMERIC, 2) AS bottom_5_avg_freight,
    RANK() OVER (ORDER BY ROUND(AVG(freight_value)::NUMERIC, 2) ASC)
       AS ranked
  FROM order_item oi
    JOIN seller sell ON oi.seller_id = sell.id
  GROUP BY state
) AS bottom_5
WHERE ranked <= 5;
```

Listing 27: Top 5 & Bottom 5 Average Freight

	state	top_5_avg_freight	bottom_5_avg_freight
1	AC	32.84	Null
2	CE	46.38	Null
3	DF	Null	20.57
4	PA	Null	19.39
5	PB	39.19	Null
6	PI	36.94	Null
7	PR	Null	22.72
8	RJ	Null	19.47
9	RO	50.91	Null
10	SP	Null	18.45

Table 27: Top 5 & Bottom 5 Average Freight

Ques 6.3 Find out the top 5 states with the highest & lowest average delivery time.

```
    SELECT state,
      top_5_avg_delivery_time, NULL AS bottom_5_avg_delivery_time
    FROM (
      SELECT state,
        ROUND(AVG(
          delivered_customer_date::DATE - purchase_timestamp::DATE
        ), 2) AS top_5_avg_delivery_time,
        RANK() OVER (ORDER BY ROUND(AVG(
          delivered_customer_date::DATE - purchase_timestamp::DATE
        ), 2) DESC) AS ranked
      FROM orders o
        JOIN customer cst ON cst.id = o.customer_id
      GROUP BY state
    ) AS top_5
    WHERE ranked <= 5
    UNION -- Union
    SELECT state,
      NULL AS top_5_avg_delivery_time, bottom_5_avg_delivery_time
    FROM (
      SELECT state,
        ROUND(AVG(
          delivered_customer_date::DATE - purchase_timestamp::DATE
        ), 2) AS bottom_5_avg_delivery_time,
        RANK() OVER (ORDER BY ROUND(AVG(
          delivered_customer_date::DATE - purchase_timestamp::DATE
        ), 2) ASC) AS ranked
      FROM orders o
        JOIN customer cst ON cst.id = o.customer_id
      GROUP BY state
    ) AS bottom_5
    WHERE ranked <= 5;
```

Listing 28: Top 5 & Bottom 5 Average Delivery Time

	state	top_5_avg_delivery_time	bottom_5_avg_delivery_time
1	AL	24.50	Null
2	AM	26.36	Null
3	AP	27.18	Null
4	DF	Null	12.90
5	MG	Null	11.95
6	PA	23.73	Null
7	PR	Null	11.94
8	RR	29.34	Null
9	SC	Null	14.91
10	SP	Null	8.70

Table 28: Top 5 & Bottom 5 Average Delivery Time

Ques 6.4 Find out the top 5 states where the order delivery is really fast as compared to the estimated date of delivery.

```sql
SELECT state,
  ROUND(AVG(
    delivered_customer_date::DATE - estimated_delivery_date::DATE
  ), 2) AS top_5_least_avg_delivery_time
FROM orders o
  JOIN customer cst ON cst.id = o.customer_id
WHERE delivered_customer_date IS NOT NULL
  AND estimated_delivery_date IS NOT NULL
GROUP BY state
ORDER BY AVG(
    delivered_customer_date::DATE - estimated_delivery_date::DATE
  ) ASC
LIMIT 5;
```

Listing 29: Top 5 States with Least Average Delivery Time

	state	top_5_least_avg_delivery_time
1	AC	-20.73
2	RO	-20.10
3	AP	-19.69
4	AM	-19.57
5	RR	-17.29

Table 29: Top 5 States with Least Average Delivery Time

Please note, here *negative sign* indicates that the order is delivered # of days *before* the estimated date.

7 Analysis based on the Payments Type

Ques 7.1 Find the month on month number of orders placed using different payment types.

```sql
WITH order_by_payment_type AS (
  SELECT
    EXTRACT(MONTH FROM purchase_timestamp)::TEXT AS order_month,
    COUNT(type) FILTER (WHERE type = 'credit_card') AS credit_card,
    COUNT(type) FILTER (WHERE type = 'debit_card') AS debit_card,
    COUNT(type) FILTER (WHERE type = 'upi') AS upi,
    COUNT(type) FILTER (WHERE type = 'voucher') AS voucher
  FROM orders o
    JOIN payment pmt ON pmt.order_id = o.id
  GROUP BY order_month
), overall_purchase AS (
  SELECT COUNT(type) AS total_purchase
  FROM orders o
    JOIN payment pmt ON pmt.order_id = o.id
)
SELECT order_month, credit_card, debit_card, upi, voucher
FROM (
  SELECT order_month, credit_card, debit_card, upi, voucher, 1 AS sort_order
  FROM order_by_payment_type

  UNION

  SELECT 'total_orders' AS order_month, SUM(credit_card),
    SUM(debit_card), SUM(upi), SUM(voucher), 2 AS sort_order
  FROM order_by_payment_type

  UNION

  SELECT 'total_orders (in %)' AS order_month,
    ROUND(100.0 * SUM(credit_card) / total_purchase::NUMERIC, 2),
    ROUND(100.0 * SUM(debit_card) / total_purchase::NUMERIC, 2),
    ROUND(100.0 * SUM(upi) / total_purchase::NUMERIC, 2),
    ROUND(100.0 * SUM(voucher) / total_purchase::NUMERIC, 2),
    3 AS sort_order
  FROM order_by_payment_type, overall_purchase
  GROUP BY total_purchase
) AS combined_result
ORDER BY sort_order,
  CASE WHEN order_month ~ '^\d+$' THEN order_month::NUMERIC ELSE NULL END;
```

Listing 30: Number of Orders placed per Month per Payment Type

	order_month	credit_card	debit_card	upi	voucher
1	1	6103	118	1715	477
2	2	6609	82	1723	424
3	3	7707	109	1942	591
4	4	7301	124	1783	572
5	5	8350	81	2035	613
6	6	7276	209	1807	563
7	7	7841	264	2074	645
8	8	8269	311	2077	589
9	9	3286	43	903	302
10	10	3778	54	1056	318
11	11	5897	70	1509	387
12	12	4378	64	1160	294
13	total_orders	76795	1529	19784	5775
14	total_orders (in %)	73.92	1.47	19.04	5.56

Table 30: Number of Orders placed per Month per Payment Type

8 Actionable Insights & Recommendations

1. From `Listing 18` & `Listing 19` we can see maximum purchase happens in Afternoons, to be more precise from 10 O'clock in the morning to 9 O'clock in the evening.

   We can strategize to use this time window to notify user about *limited time deals* or *new product launch*.

2. From `Listing 30` we can see that $\sim$ 74% user make purchase through credit cards.

   We can collaborate/partner with different Credit Card providers to provide some kind of *reward points* system which in-turn they can use to make purchase from our platform.

3. From `Listing 17` we can see that there are more than $\sim$ 83% non-reviewed orders in every review score category.

   We can employ typeform type **MCQ** questions about the product instead of *only* subjective review, that way we can guide user to answer quantified qualities of the products and in-turn makes *us* to quantify the product standards to serve our user with best quality of products.