Targeting Success: A Business Case Analysis of 100k Orders at Target in Brazil by Emma Luk

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

1. Understanding the data

Based on the given datasets, it can identify the following relationships between them:

1. **Customers and Orders:**
   - The "**customers.csv**" file contains information about customers, including their unique IDs, zip codes, cities, and states.
   - The "**orders.csv**" file contains information about orders, including the order IDs, customer IDs, order status, purchase timestamps, delivery dates, and estimated delivery dates.
   - The relationship between these two datasets is established through the "**customer_id**" column in the "**orders.csv**" file, which references the unique IDs of customers in the "**customers.csv**" file.
2. **Sellers and Orders:**
   - The "**sellers.csv**" file contains information about sellers, including their unique IDs, zip codes, cities, and states.
   - The "**order_items.csv**" file contains information about order items, including the order IDs, order item IDs, product IDs, seller IDs, shipping limit dates, prices, and freight values.
   - The relationship between these two datasets is established through the "**seller_id**" column in the "**order_items.csv**" file, which references the unique IDs of sellers in the "sellers.csv" file.
3. **Payments and Orders:**
   - The "**payments.csv**" file contains information about payments, including the order IDs, payment sequential numbers, payment types, payment instalments, and payment values.
   - The "**orders.csv**" file also contains information about orders, including the order IDs, customer IDs, order status, purchase timestamps, delivery dates, and estimated delivery dates.
   - The relationship between these two datasets is established through the "**order_id**" column, which is common in both the "**payments.csv**" and "**orders.csv**" files.
4. **Reviews and Orders:**
   - The "**order_reviews.csv**" file contains information about reviews, including the review IDs, order IDs, review scores, review comment titles, review creation timestamps, and review answer timestamps.

- The "**orders.csv**" file also contains information about orders, including the order IDs, customer IDs, order status, purchase timestamps, delivery dates, and estimated delivery dates.
- The relationship between these two datasets is established through the "**order_id**" column, which is common in both the "**order_reviews.csv**" and "**orders.csv**" files.

5. **Products and Order Items:**
    - The "**products.csv**" file contains information about products, including the product IDs, product category names, product name lengths, product description lengths, product photos quantities, product weight in grams, product length in centimetres, product height in centimetres, and product width in centimetres.
    - The "order_items.csv" file contains information about order items, including the order IDs, order item IDs, product IDs, seller IDs, shipping limit dates, prices, and freight values.
    - The relationship between these two datasets is established through the "**product_id**" column in the "**order_items.csv**" file, which references the unique IDs of products in the "**products.csv**" file.
6. **Geolocation and Customers/Sellers:**
    - The "**geolocation.csv**" file contains information about geolocations, including the zip code prefixes, latitudes, longitudes, cities, and states.
    - The "**customers.csv**" file contains information about customers, including their unique IDs, zip codes, cities, and states.
    - The "**sellers.csv**" file contains information about sellers, including their unique IDs, zip codes, cities, and states.
    - The relationship between the "**geolocation.csv**" file and the "**customers.csv**" and "**sellers.csv**" files is established through the zip code prefixes, which are common in all three files and can be used to join or merge the datasets based on the location information.

Based on these relationships, data analysts at Target could perform various analyses, such as customer segmentation **(Section 7.1 Analysing Customer Sentiment with Natural language)** based on customer reviews and review scores.

1.1Data type of columns in tables

(Section 7.1 Analysing Customer Sentiment with Natural language), very often it is easier to perform analysis using SQL or BigQuery on data we have right in the tables and then move forward to ML/AI/Data science and engineering in Python.

Python codes:

(Figure 1 Python codes) The “**df.shape**” function returns the shape of the “**dataframe**”, “**df.dtypes**” function returns the data types of each column in the dataframe, and “**df.describe()**” function returns the descriptive statistics of the numerical columns in the dataframe.

```
import pandas as pd
import numpy as np
import matplotlib.pylab as plt
import seaborn as sns
plt.style.use('ggplot')
# Display the shape of the dataframe
df.shape

# Display the data types of each column in the dataframe
df.dtypes

# Display descriptive statistics of the numerical columns in the dataframe
df.describe()
```

Figure 1 Python codes

(Figure 1.1 Analyse Data Types of Columns for different tables with Common Table Expression (CTE)), data_type: This is likely meant to display the data types of the columns in the table, which would give information about the type of data stored in each column (e.g., integer, float, string).

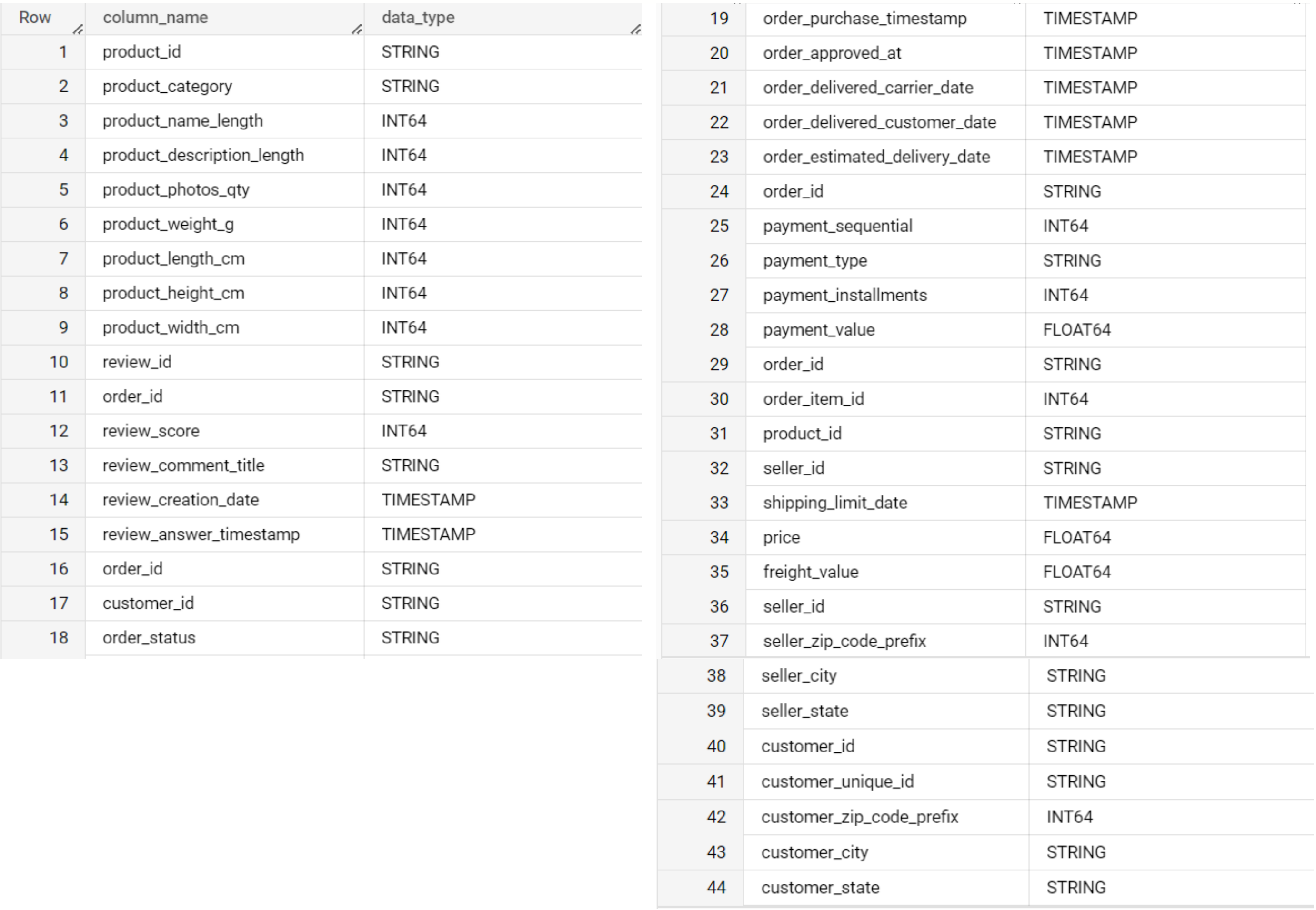
Analyse Data Types of Columns

To analyse the data types of columns in a table, use the following query in BigQuery:

```
--------------------------------------------------------------------------------------
-- Data type of columns in tables
-- Analyse Data Types of Columns for different tables
-- with Common Table Expression (CTE)
--------------------------------------------------------------------------------------
WITH customer_columns AS (
SELECT column_name, data_type
FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'customers'
),
seller_columns AS (
SELECT column_name, data_type
FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'sellers'
),
order_items_columns AS (
SELECT column_name, data_type
FROM`target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'order_items'
),
geolocations_columns AS (
SELECT column_name, data_type
FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'geolocations'
),
payments_columns AS (
SELECT column_name, data_type
FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'payments'
),
orders_columns AS (
SELECT column_name, data_type
FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'orders'
),
reviews_columns AS (
SELECT column_name, data_type
FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'order_reviews'
),
products_columns AS (
SELECT column_name, data_type
FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'products'
)
-- Analyse Data Types of Columns for different tables with Common Table Expression (CTE)
SELECT column_name, data_type FROM customer_columns
UNION ALL
SELECT column_name, data_type FROM seller_columns
UNION ALL
SELECT column_name, data_type FROM order_items_columns
UNION ALL
SELECT column_name, data_type FROM geolocations_columns
UNION ALL
SELECT column_name, data_type FROM payments_columns
UNION ALL
SELECT column_name, data_type FROM orders_columns
UNION ALL
SELECT column_name, data_type FROM reviews_columns
UNION ALL
SELECT column_name, data_type FROM products_columns;
```

Figure 1.1 Analyse Data Types of Columns for different tables with Common Table Expression (CTE)

Query results in the following:



Row	column_name	data_type
1	product_id	STRING
2	product_category	STRING
3	product_name_length	INT64
4	product_description_length	INT64
5	product_photos_qty	INT64
6	product_weight_g	INT64
7	product_length_cm	INT64
8	product_height_cm	INT64
9	product_width_cm	INT64
10	review_id	STRING
11	order_id	STRING
12	review_score	INT64
13	review_comment_title	STRING
14	review_creation_date	TIMESTAMP
15	review_answer_timestamp	TIMESTAMP
16	order_id	STRING
17	customer_id	STRING
18	order_status	STRING
19	order_purchase_timestamp	TIMESTAMP
20	order_approved_at	TIMESTAMP
21	order_delivered_carrier_date	TIMESTAMP
22	order_delivered_customer_date	TIMESTAMP
23	order_estimated_delivery_date	TIMESTAMP
24	order_id	STRING
25	payment_sequential	INT64
26	payment_type	STRING
27	payment_installments	INT64
28	payment_value	FLOAT64
29	order_id	STRING
30	order_item_id	INT64
31	product_id	STRING
32	seller_id	STRING
33	shipping_limit_date	TIMESTAMP
34	price	FLOAT64
35	freight_value	FLOAT64
36	seller_id	STRING
37	seller_zip_code_prefix	INT64
38	seller_city	STRING
39	seller_state	STRING
40	customer_id	STRING
41	customer_unique_id	STRING
42	customer_zip_code_prefix	INT64
43	customer_city	STRING
44	customer_state	STRING

Figure 1.2 Query results

1.2 Data shape of rows and columns in tables

shape: This is likely meant to display the shape of a table, which would be the number of rows and columns in the table.

(Figure 1.3 BigQuery: shape) The code provided is a series of SQL queries written in BigQuery (a cloud-based data warehousing and analytics platform by Google) that are used to determine the shape (i.e., number of columns and rows) of various tables in a dataset named target_business in a BigQuery project named target-business-case-382621.

The tables being queried are:

- **customers**: Provides information about customers.
- **sellers**: Provides information about sellers.
- **order_items**: Provides information about order items.
- **geolocation**: Provides information about geolocations.
- **payments**: Provides information about payments.
- **orders**: Provides information about orders.
- **order_reviews**: Provides information about order reviews.
- **products**: Provides information about products.

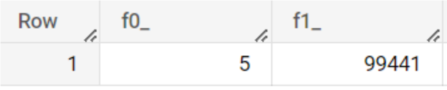
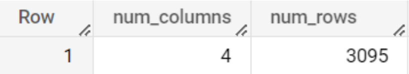
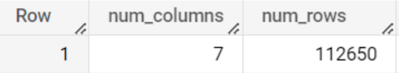
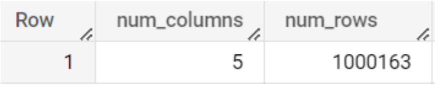
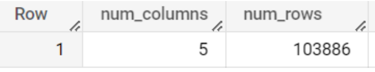
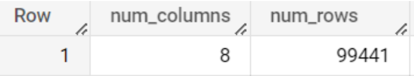
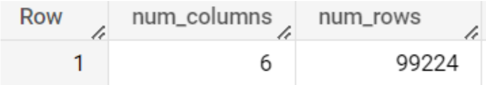
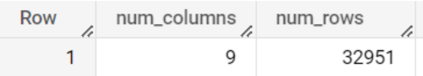
(**Figure 1.3 BigQuery: shape**) Each query uses a Common Table Expression (CTE) to calculate the number of distinct columns (**num_columns)** in each table using the COUNT(DISTINCT **column_name**) function, and the total number of rows **(num_rows**) in each table using a subquery with COUNT(*). The final output of each query is a result set with two columns: **num_columns** and **num_rows**, which represent the shape of the respective table in terms of columns and rows.

```
-- -- Data shape tables in tables
-- Analyse shape tables for different tables
-- with Common Table Expression (CTE)

-- BigQuery shape table for customers table
WITH customer_shape AS (
  SELECT COUNT(DISTINCT column_name) AS num_columns,
         (SELECT COUNT(*) FROM `target-business-case-
382621.target_business.customers`) AS num_rows
  FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
  WHERE table_name = 'customers'
)
SELECT num_columns, num_rows
FROM customer_shape;

-- BigQuery shape table for sellers table
WITH seller_shape AS (
  SELECT COUNT(DISTINCT column_name) AS num_columns,
         (SELECT COUNT(*) FROM `target-business-case-
382621.target_business.sellers`) AS num_rows
  FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
  WHERE table_name = 'sellers'
)
```

```
SELECT num_columns, num_rows
FROM seller_shape;

-- BigQuery shape table for order_items table
WITH order_items_shape AS (
  SELECT COUNT(DISTINCT column_name) AS num_columns,
         (SELECT COUNT(*) FROM `target-business-case-
382621.target_business.order_items`) AS num_rows
  FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
  WHERE table_name = 'order_items'
)
SELECT num_columns, num_rows
FROM order_items_shape;

-- BigQuery shape table for geolocations table
WITH geolocations_shape AS (
  SELECT COUNT(DISTINCT column_name) AS num_columns,
         (SELECT COUNT(*) FROM `target-business-case-
382621.target_business.geolocation`) AS num_rows
  FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
  WHERE table_name = 'geolocation'
)
SELECT num_columns, num_rows
FROM geolocations_shape;

-- BigQuery shape table for payments table
WITH payments_shape AS (
  SELECT COUNT(DISTINCT column_name) AS num_columns,
         (SELECT COUNT(*) FROM `target-business-case-
382621.target_business.payments`) AS num_rows
  FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
  WHERE table_name = 'payments'
)
SELECT num_columns, num_rows
FROM payments_shape;

-- BigQuery shape table for orders table
WITH orders_shape AS (
  SELECT COUNT(DISTINCT column_name) AS num_columns,
         (SELECT COUNT(*) FROM `target-business-case-382621.target_business.orders`) AS num_rows
  FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
  WHERE table_name = 'orders'
)
SELECT num_columns, num_rows
FROM orders_shape;

-- BigQuery shape table for reviews table
WITH reviews_shape AS (
  SELECT COUNT(DISTINCT column_name) AS num_columns,
         (SELECT COUNT(*) FROM `target-business-case-
382621.target_business.order_reviews`) AS num_rows
  FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
  WHERE table_name = 'order_reviews'
)
SELECT num_columns, num_rows
FROM reviews_shape;

-- BigQuery shape table for products table
WITH products_shape AS (
  SELECT COUNT(DISTINCT column_name) AS num_columns,
         (SELECT COUNT(*) FROM `target-business-case-
382621.target_business.products`) AS num_rows
  FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
  WHERE table_name = 'products'
)
SELECT num_columns, num_rows
FROM products_shape;
```

Figure 1.3 BigQuery: shape

Query Results:

the shape (i.e., number of columns and rows) for customers table:  Row | f0_ | f1_ 1 | 5 | 99441	**the shape (i.e., number of columns and rows) for sellers table:**  Row | num_columns | num_rows 1 | 4 | 3095	**the shape (i.e., number of columns and rows) for order_items:**  Row | num_columns | num_rows 1 | 7 | 112650
the shape (i.e., number of columns and rows) for geolocations table:  Row | num_columns | num_rows 1 | 5 | 1000163	**the shape (i.e., number of columns and rows) for payments table:**  Row | num_columns | num_rows 1 | 5 | 103886	**the shape (i.e., number of columns and rows) for orders table:**  Row | num_columns | num_rows 1 | 8 | 99441
the shape (i.e., number of columns and rows) for reviews table:  Row | num_columns | num_rows 1 | 6 | 99224	**the shape (i.e., number of columns and rows) for products table:**  Row | num_columns | num_rows 1 | 9 | 32951	

Here is the interpretation of the output:

- **customers.csv**: The table has 5 columns and 99441 rows.
- **sellers.csv**: The table has 4 columns and 3095 rows.
- **order_items.csv**: The table has 7 columns and 112650 rows.
- **geolocations.csv**: The table has 5 columns and 1000163 rows.
- **payments.csv**: The table has 5 columns and 103886 rows.
- **orders.csv**: The table has 8 columns and 99441 rows.
- **reviews.csv**: The table has 6 columns and 99224 rows.
- **products.csv**: The table has 9 columns and 32951 rows.

1.2 Time period for which the data is given

(Figure 1.4 BigQuery: Time period for which the data is given), here's the BigQuery that uses Common Table Expressions (CTEs) to find out the time period for which the data is given in the Target dataset:

```
-- 1.2. Time period for which the data is given

WITH min_max_dates AS (
  SELECT
```

```
    MIN(order_purchase_timestamp) AS min_date,
    MAX(order_purchase_timestamp) AS max_date
  FROM
    target_business.orders
)
SELECT
  FORMAT_TIMESTAMP('%Y-%m-%d', min_date) AS min_purchase_date,
  FORMAT_TIMESTAMP('%Y-%m-%d', max_date) AS max_purchase_date
FROM
  min_max_dates;
```

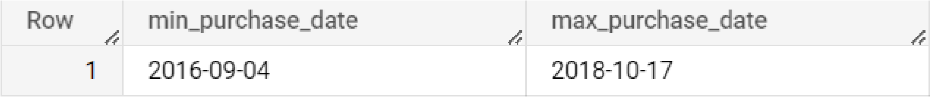
Figure 1.4 BigQuery: Time period for which the data is given

(Figure 1.4 BigQuery: Time period for which the data is given), in this query, first define a Common Table Expression (CTE) called **min_max_dates** which calculates the minimum and maximum purchase timestamps from the orders table using the **MIN**() and **MAX**() functions. Then, in the main query, use the **FORMAT_TIMESTAMP**() function to format the minimum and maximum purchase timestamps as dates in the 'YYYY-MM-DD' format, and alias them as **min_purchase_date** and **max_purchase_date**, respectively.

This query will return the minimum and maximum purchase dates from the orders table, which represent the time period for which the data is given in the Target dataset.

Query Results:

```
-- min_purchase_date: 2016-09-04
-- max_purchase_date: 2018-10-17
```



Row	min_purchase_date	max_purchase_date
1	2016-09-04	2018-10-17

1.3 Cities and States of customers ordered during the given period

(Figure 1.5 BigQuery: Cities and States of customers ordered during the given period using Common Table Expression (CTE)), in this query, a Common Table Expression (CTE) named **orders_cte** is used to retrieve the distinct **customer_city** and **customer_state** from the orders table, customers table, and geolocation table. The ON clause specifies the join conditions between the tables. The WHERE clause filters the orders based on the given period using the **order_purchase_timestamp** column.

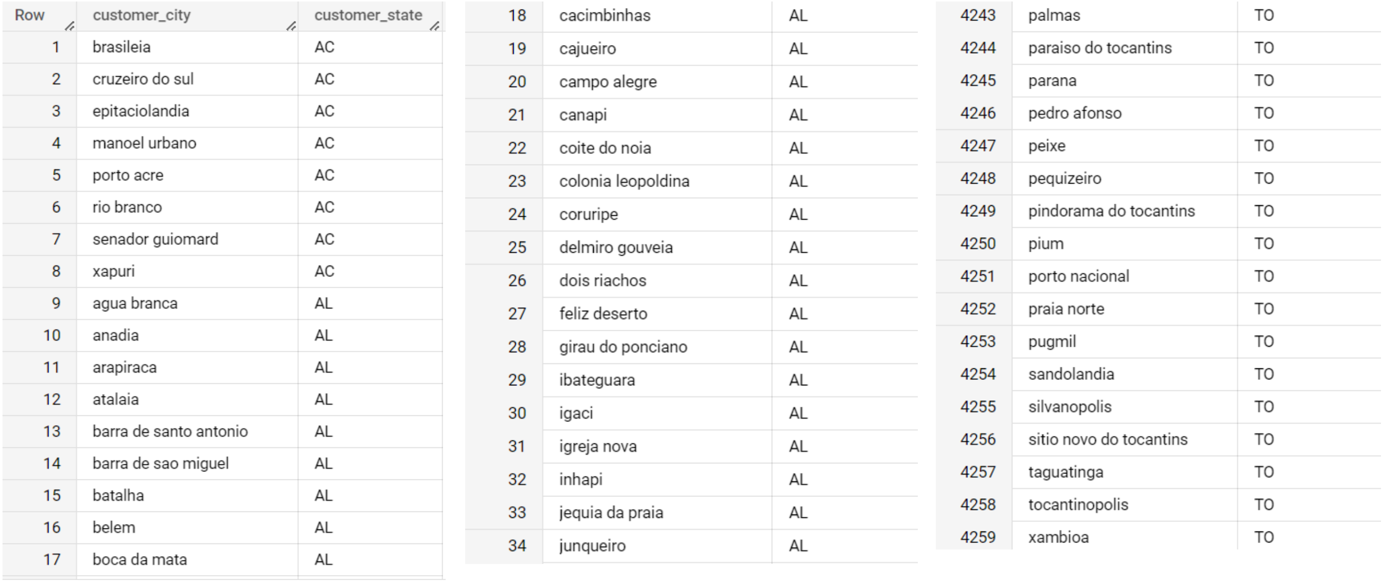
Finally, the main query selects the **customer_city** and **customer_state** columns from the CTE and orders the results by **customer_state** and **customer_city**.

```
-
- Cities and States of customers ordered during the given period using Common Table Expression (
CTE)

WITH orders_cte AS (
  SELECT DISTINCT customer_city, customer_state
  FROM target_business.orders o
  JOIN target_business.customers c ON o.customer_id = c.customer_id
  JOIN target_business.geolocation g ON c.customer_zip_code_prefix = g.geolocation_zip_code_pref
ix
  WHERE o.order_purchase_timestamp BETWEEN '2016-09-04 21:15:19 UTC' AND '2018-10-
17 17:30:18 UTC'
)
SELECT customer_city, customer_state
FROM orders_cte
ORDER BY customer_state, customer_city;
```

Figure 1.5 BigQuery: Cities and States of customers ordered during the given period using Common Table Expression (CTE)

Query Results:

DISTINCT Cities and States of customers ordered during the given period: 4259



Row	customer_city	customer_state
1	brasileia	AC
2	cruzeiro do sul	AC
3	epitaciolandia	AC
4	manoel urbano	AC
5	porto acre	AC
6	rio branco	AC
7	senador guiomard	AC
8	xapuri	AC
9	agua branca	AL
10	anadia	AL
11	arapiraca	AL
12	atalaia	AL
13	barra de santo antonio	AL
14	barra de sao miguel	AL
15	batalha	AL
16	belem	AL
17	boca da mata	AL
18	cacimbinhas	AL
19	cajueiro	AL
20	campo alegre	AL
21	canapi	AL
22	coite do noia	AL
23	colonia leopoldina	AL
24	coruripe	AL
25	delmiro gouveia	AL
26	dois riachos	AL
27	feliz deserto	AL
28	girau do ponciano	AL
29	ibateguara	AL
30	igaci	AL
31	igreja nova	AL
32	inhapi	AL
33	jequia da praia	AL
34	junqueiro	AL
4243	palmas	TO
4244	paraiso do tocantins	TO
4245	parana	TO
4246	pedro afonso	TO
4247	peixe	TO
4248	pequizeiro	TO
4249	pindorama do tocantins	TO
4250	pium	TO
4251	porto nacional	TO
4252	praia norte	TO
4253	pugmil	TO
4254	sandolandia	TO
4255	silvanopolis	TO
4256	sitio novo do tocantins	TO
4257	taguatinga	TO
4258	tocantinopolis	TO
4259	xambioa	TO

Figure 1.6 DISTINCT Cities and States of customers ordered during the given period: 4259

2. In-depth Exploration:

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

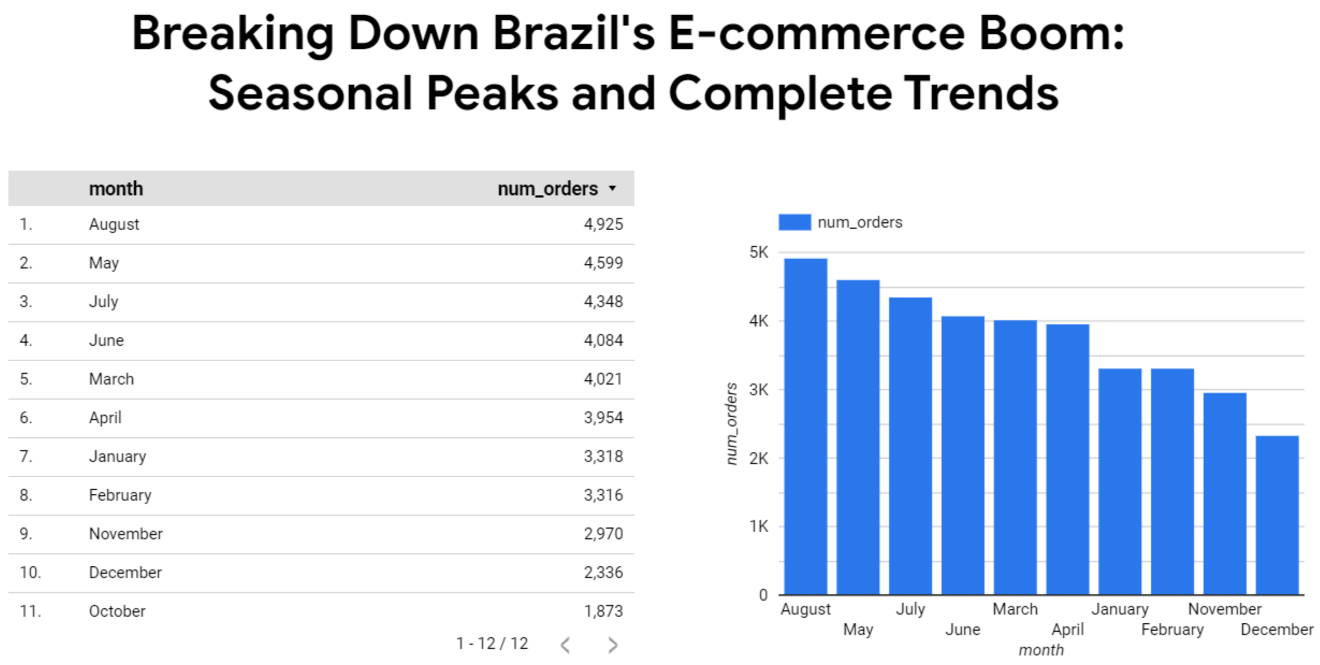
(Figure 2.1 Breaking Down Brazil's E-commerce Boom: Seasonal Peaks and Complete Trends), based on the data, it can see a clear growing trend in e-commerce in Brazil. The number of orders and revenue have steadily increased throughout the year, with a notable increase in the number of orders from May to August, and then a slight decrease in September to December.

The data shows that there were 3,318 orders and 65,731,702.59 BRL revenue in January, while in December, there were 2,336 orders and 45,203,634.93 BRL revenue.

Overall, the data suggests that e-commerce in Brazil is on the rise, and that there are specific months where it can see a peak in orders and revenue. However, to fully describe a complete scenario, it would need to analyse more data, such as the types of products being sold, the demographics of the buyers, and any external factors that may be contributing to the growth in e-commerce.

```
-- Breaking Down Brazil's E-commerce Boom:  Seasonal Peaks and Complete Trends
SELECT
 EXTRACT(MONTH FROM order_purchase_timestamp) AS month,
 COUNT(DISTINCT o.order_id) AS num_orders,
 SUM(oi.price + oi.freight_value) AS revenue
FROM
 `target-business-case-382621.target_business.orders` o
 JOIN `target-business-case-382621.target_business.order_items` oi ON o.order_id = oi.order_id
 JOIN `target-business-case-382621.target_business.customers` c ON o.customer_id = c.customer_id
 JOIN `target-business-case-382621.target_business.geolocation` g ON c.customer_zip_code_prefix = g.geolocation_zip_code_prefix
WHERE
 g.geolocation_state = 'SP'
GROUP BY
 month
ORDER BY
 month ASC;
```

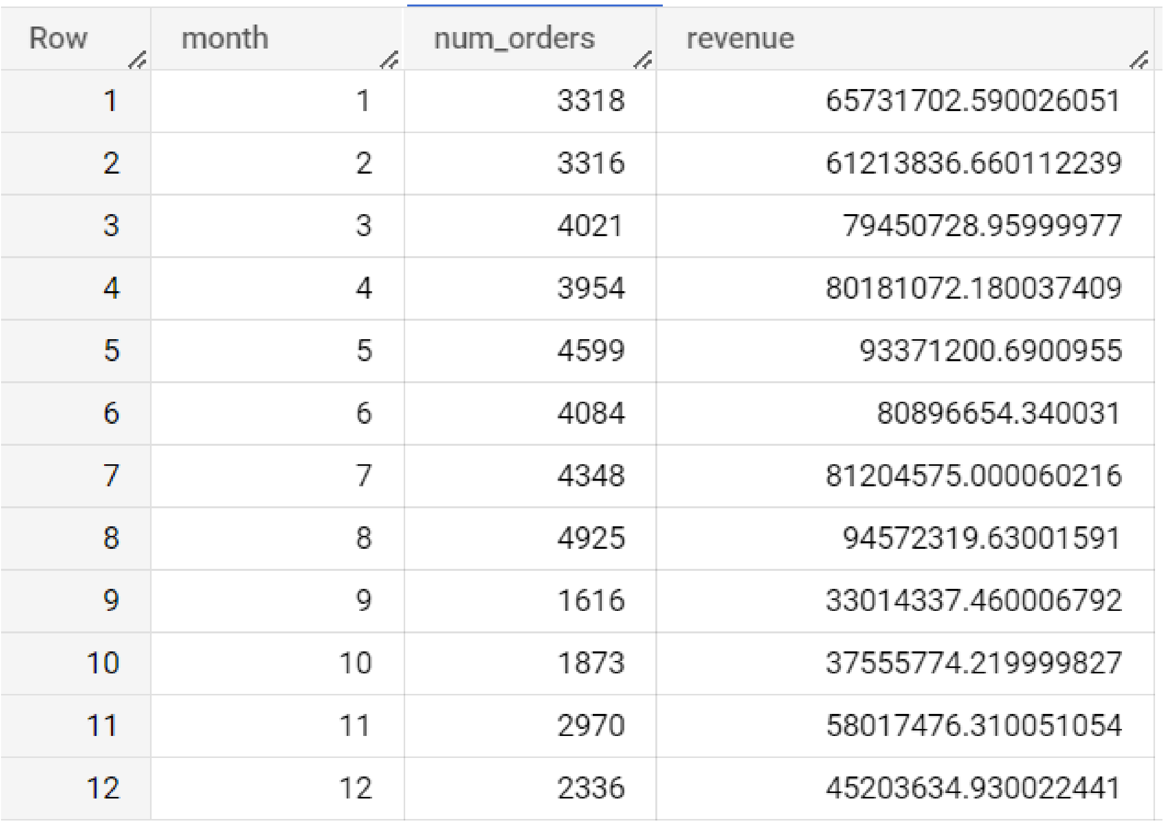
Figure 2 BigQuery: Breaking Down Brazil's E-commerce Boom: Seasonal Peaks and Complete Trends



Breaking Down Brazil's E-commerce Boom: Seasonal Peaks and Complete Trends

	month	num_orders ▾
1.	August	4,925
2.	May	4,599
3.	July	4,348
4.	June	4,084
5.	March	4,021
6.	April	3,954
7.	January	3,318
8.	February	3,316
9.	November	2,970
10.	December	2,336
11.	October	1,873

1 - 12 / 12 < >



Row	month	num_orders	revenue
1	1	3318	65731702.590026051
2	2	3316	61213836.660112239
3	3	4021	79450728.95999977
4	4	3954	80181072.180037409
5	5	4599	93371200.6900955
6	6	4084	80896654.340031
7	7	4348	81204575.000060216
8	8	4925	94572319.63001591
9	9	1616	33014337.460006792
10	10	1873	37555774.219999827
11	11	2970	58017476.310051054
12	12	2336	45203634.930022441

Figure 2.1 Breaking Down Brazil's E-commerce Boom: Seasonal Peaks and Complete Trends

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

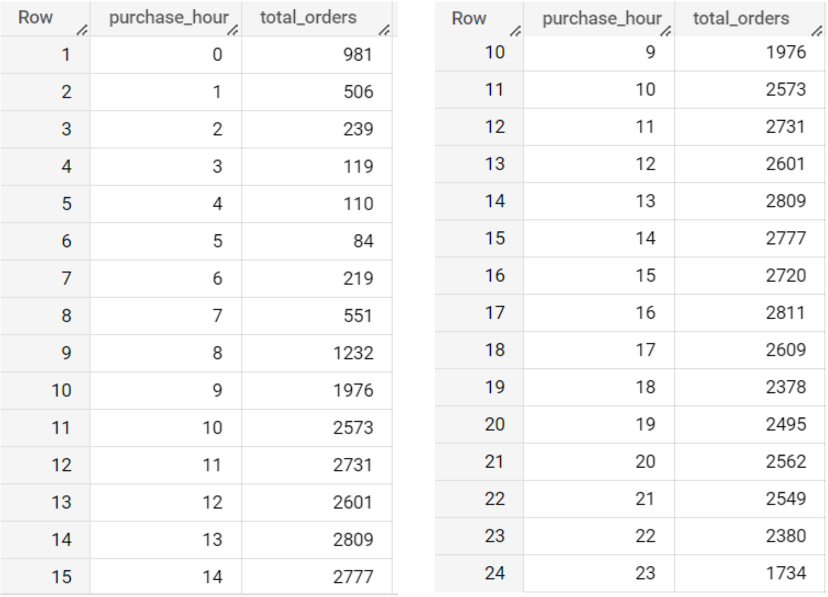
- **(Figure 2.2 BigQuery: Brazilian customers tend to buy (Dawn, Morning, Afternoon or Night),** in this query, it join the **orders** and **customers** tables on the **customer_id** column to get the **order_purchase_timestamp column** and the **customer_state** column in the same result set. It filter the results to only include orders from the Sao Paulo state, which is where Target operates in Brazil.
- It then extract the hour of the day from the **order_purchase_timestamp** column using the EXTRACT function, and group the results by the purchase hour. Finally, it count the number of orders in each hour and sort the results by the purchase hour in ascending order.
- This query will return a table with two columns: **purchase_hour** and **total_orders**. The **purchase_hour** column will contain the hour of the day (in 24-hour format) when the orders were made, and the **total_orders** column will contain the number of orders made in that hour. **(Figure 2.3 Brazilian customers tend to buy (Dawn, Morning, Afternoon or Night))**, it can interpret the results to find out what time Brazilian customers tend to buy more: early morning, morning, afternoon or night.

```
SELECT
  EXTRACT(HOUR FROM order_purchase_timestamp) AS purchase_hour,
  COUNT(*) AS total_orders
FROM
  `target-business-case-382621.target_business.orders` AS o
  JOIN `target-business-case-382621.target_business.customers` AS c ON o.customer_id = c.customer_id
WHERE
```

```
 c.customer_state = 'SP'  -- Select only orders from Sao Paulo state
GROUP BY
 purchase_hour
ORDER BY
 purchase_hour
```

Figure 2.2 BigQuery: Brazilian customers tend to buy (Dawn, Morning, Afternoon or Night)

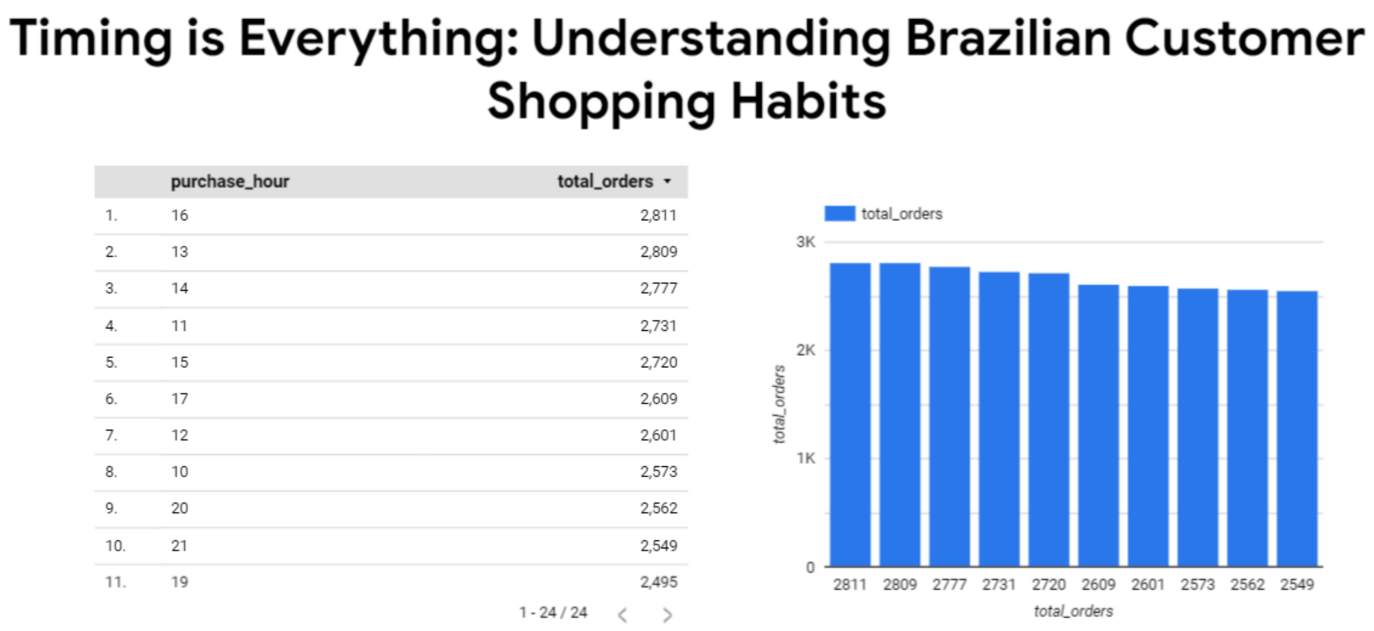
Query results:



Row	purchase_hour	total_orders
1	0	981
2	1	506
3	2	239
4	3	119
5	4	110
6	5	84
7	6	219
8	7	551
9	8	1232
10	9	1976
11	10	2573
12	11	2731
13	12	2601
14	13	2809
15	14	2777

Row	purchase_hour	total_orders
10	9	1976
11	10	2573
12	11	2731
13	12	2601
14	13	2809
15	14	2777
16	15	2720
17	16	2811
18	17	2609
19	18	2378
20	19	2495
21	20	2562
22	21	2549
23	22	2380
24	23	1734

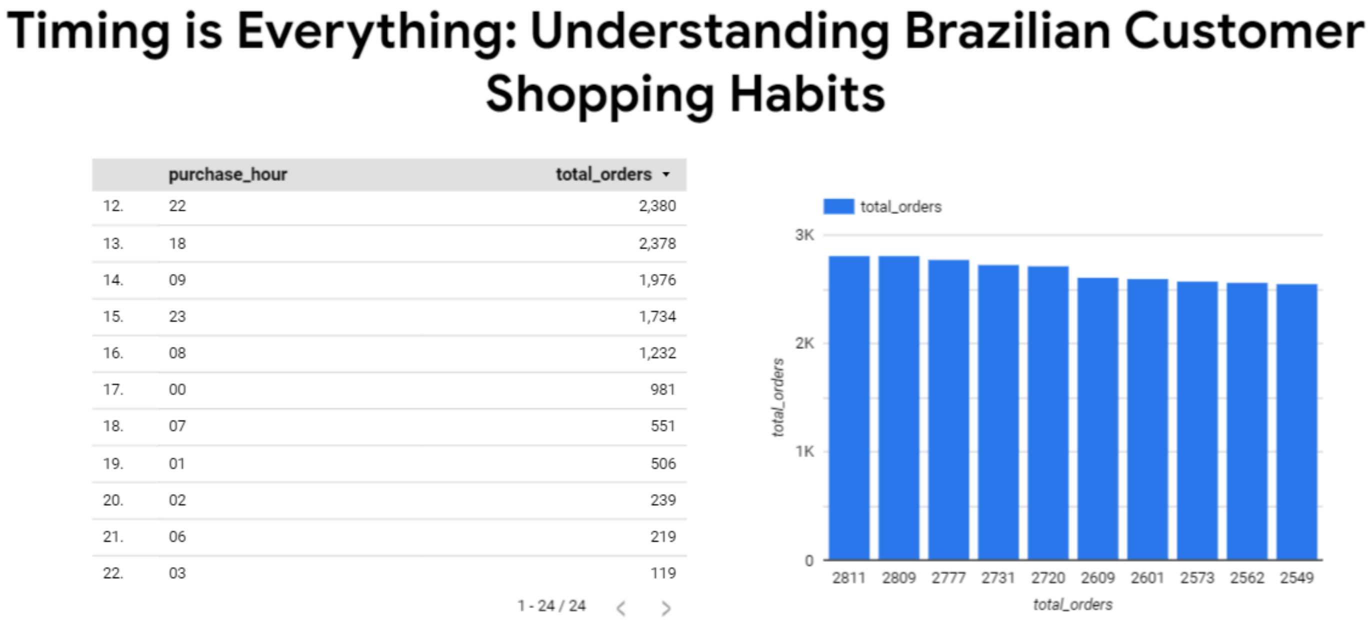
Figure 2.3 Brazilian customers tend to buy (Dawn, Morning, Afternoon or Night)



Timing is Everything: Understanding Brazilian Customer Shopping Habits

	purchase_hour	total_orders ▾
1.	16	2,811
2.	13	2,809
3.	14	2,777
4.	11	2,731
5.	15	2,720
6.	17	2,609
7.	12	2,601
8.	10	2,573
9.	20	2,562
10.	21	2,549
11.	19	2,495

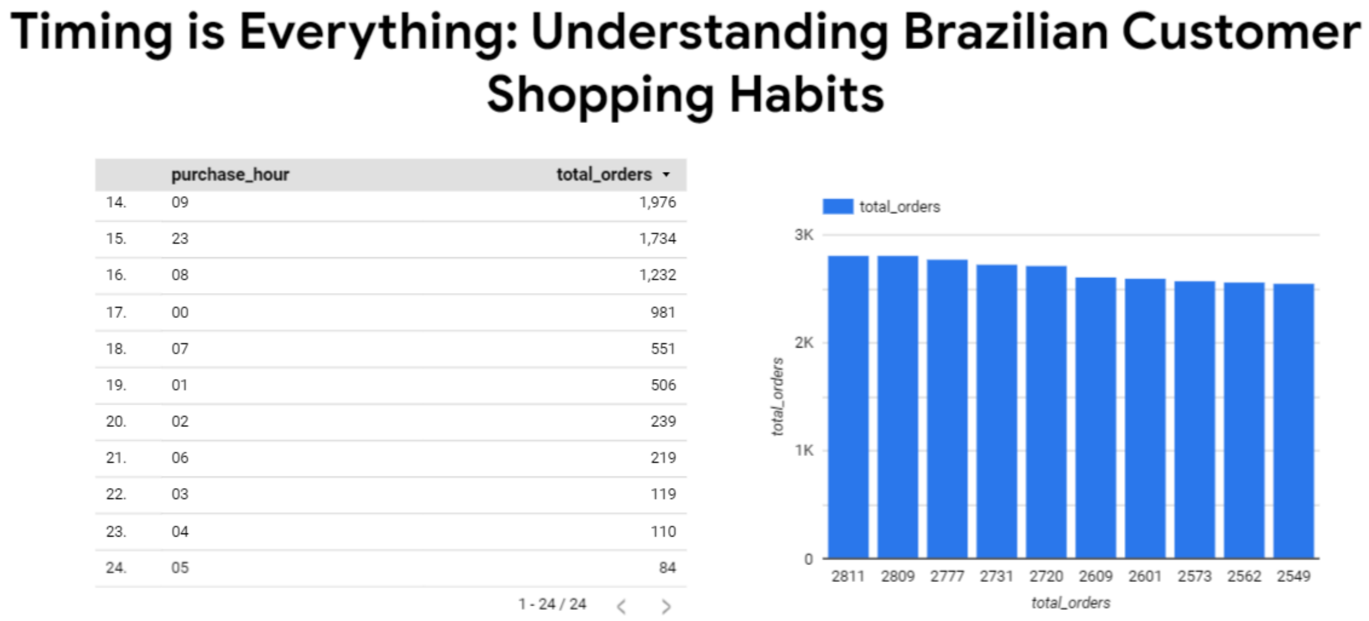
1 - 24 / 24 < >



Timing is Everything: Understanding Brazilian Customer Shopping Habits

	purchase_hour	total_orders ▾
12.	22	2,380
13.	18	2,378
14.	09	1,976
15.	23	1,734
16.	08	1,232
17.	00	981
18.	07	551
19.	01	506
20.	02	239
21.	06	219
22.	03	119

1 - 24 / 24 < >



Timing is Everything: Understanding Brazilian Customer Shopping Habits

	purchase_hour	total_orders ▾
14.	09	1,976
15.	23	1,734
16.	08	1,232
17.	00	981
18.	07	551
19.	01	506
20.	02	239
21.	06	219
22.	03	119
23.	04	110
24.	05	84

1 - 24 / 24 < >

Figure 2.4 Brazilian customers tend to buy (Dawn, Morning, Afternoon or Night)

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

3.1 Get month on month orders by states

(Figure 3 BigQuery: Get month on month orders by states), this query begins with a SELECT statement that specifies three columns to be retrieved: "EXTRACT(MONTH FROM **order_purchase_timestamp**)" aliased as "**order_month**", "**c.customer_state**", and "COUNT(DISTINCT **o.order_id**)" aliased as "**order_count**." The "**order_purchase_timestamp**" column is likely a timestamp column that represents the date and time when the order was made, and the "**o.order_id**" column is likely a unique identifier for each order. The EXTRACT() function is used to extract the month component from the "**order_purchase_timestamp**" column, which will be used to group the results by month. The "**c.customer_state**" column likely represents the state where the customer who made the order is located. The COUNT(DISTINCT) function is used to count the number of distinct order IDs, which represents the number of orders made in each month for each state.

The query then uses a JOIN clause to combine the "**orders**" and "**customers**" tables based on the condition "**o.customer_id = c.customer_id**". This indicates that the "**customer_id**" column in the "orders" table is being matched with the "customer_id" column in the "**customers**" table, presumably to link the order data with the corresponding customer data.

Next, the query uses a WHERE clause to filter the results based on the "**order_purchase_timestamp**" column, specifying a date range between '2016-09-04 21:15:19 UTC' and '2018-10-17 17:30:18 UTC'. This restricts the analysis to orders made within this time frame.

The query then uses a GROUP BY clause to group the results by "**order_month**" and "**c.customer_state**," which represents the month and state of the orders, respectively. This allows for calculating the order counts for each month and state separately.

Finally, the query uses an ORDER BY clause to sort the results by "**order_month**" and "**c.customer_state**," which represents the chronological order of the months and the alphabetical order of the states, respectively.

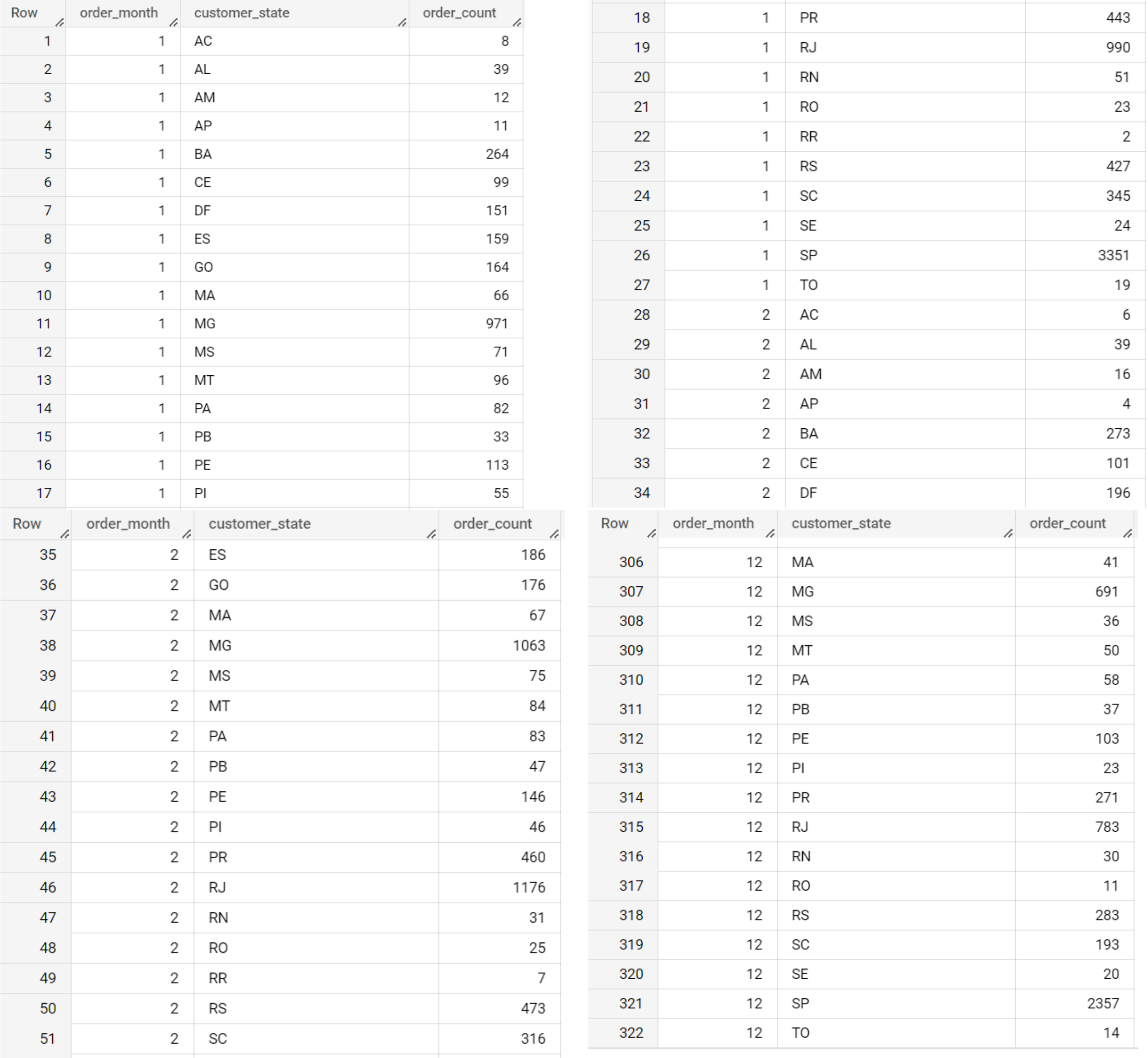
In summary, this SQL query retrieves order data from a database, joins it with customer data, filters the results by a specific date range, groups the results by month and state, and orders them chronologically by month and alphabetically by state to analyse the evolution of e-commerce orders in the Brazil region over time.

```
-- Evolution of E-commerce orders in the Brazil region:

-- Get month on month orders by states
SELECT
  EXTRACT(MONTH FROM order_purchase_timestamp) AS order_month,
  -- DATE_TRUNC('month', o.order_purchase_timestamp) AS order_month,
  c.customer_state,
  COUNT(DISTINCT o.order_id) AS order_count
```

```
FROM
  target_business.orders o
  JOIN target_business.customers c ON o.customer_id = c.customer_id
WHERE
  o.order_purchase_timestamp >= '2016-09-
04 21:15:19 UTC' AND o.order_purchase_timestamp < '2018-10-17 17:30:18 UTC'
GROUP BY
  order_month,
  c.customer_state
ORDER BY
  order_month,
  c.customer_state;
```

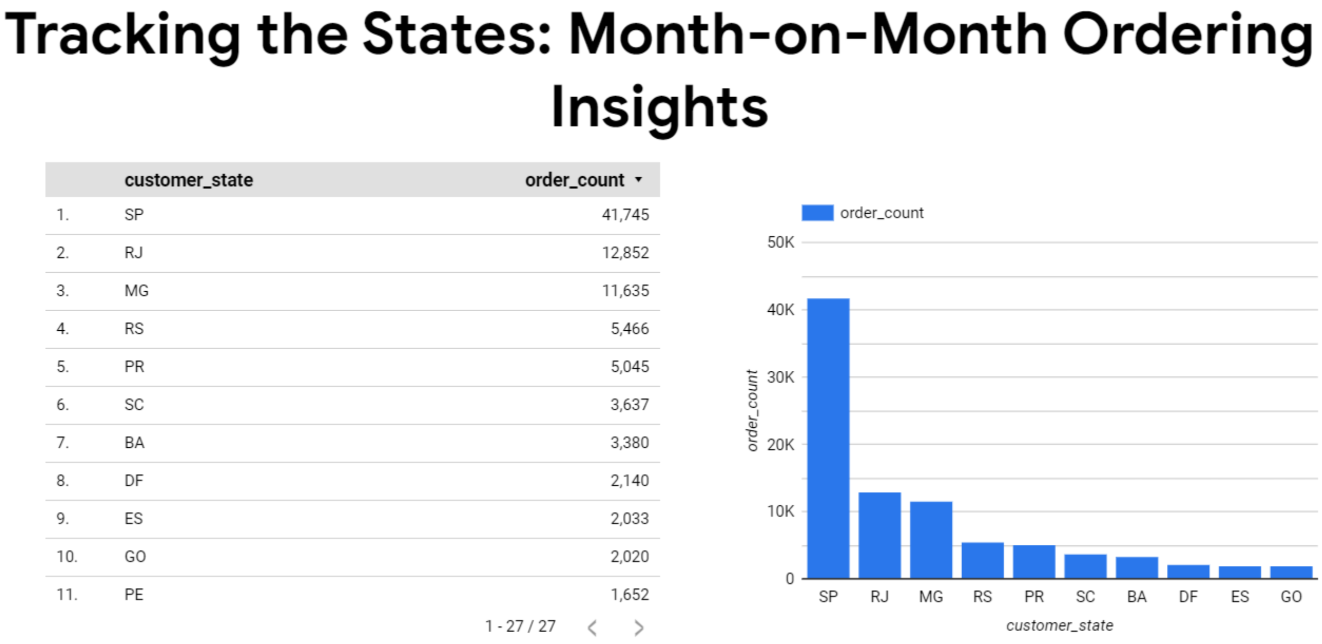
Figure 3 BigQuery: Get month on month orders by states

Query results:



Row	order_month	customer_state	order_count
1	1	AC	8
2	1	AL	39
3	1	AM	12
4	1	AP	11
5	1	BA	264
6	1	CE	99
7	1	DF	151
8	1	ES	159
9	1	GO	164
10	1	MA	66
11	1	MG	971
12	1	MS	71
13	1	MT	96
14	1	PA	82
15	1	PB	33
16	1	PE	113
17	1	PI	55
18	1	PR	443
19	1	RJ	990
20	1	RN	51
21	1	RO	23
22	1	RR	2
23	1	RS	427
24	1	SC	345
25	1	SE	24
26	1	SP	3351
27	1	TO	19
28	2	AC	6
29	2	AL	39
30	2	AM	16
31	2	AP	4
32	2	BA	273
33	2	CE	101
34	2	DF	196
35	2	ES	186
36	2	GO	176
37	2	MA	67
38	2	MG	1063
39	2	MS	75
40	2	MT	84
41	2	PA	83
42	2	PB	47
43	2	PE	146
44	2	PI	46
45	2	PR	460
46	2	RJ	1176
47	2	RN	31
48	2	RO	25
49	2	RR	7
50	2	RS	473
51	2	SC	316
306	12	MA	41
307	12	MG	691
308	12	MS	36
309	12	MT	50
310	12	PA	58
311	12	PB	37
312	12	PE	103
313	12	PI	23
314	12	PR	271
315	12	RJ	783
316	12	RN	30
317	12	RO	11
318	12	RS	283
319	12	SC	193
320	12	SE	20
321	12	SP	2357
322	12	TO	14

Figure 3.1 Get month on month orders by states

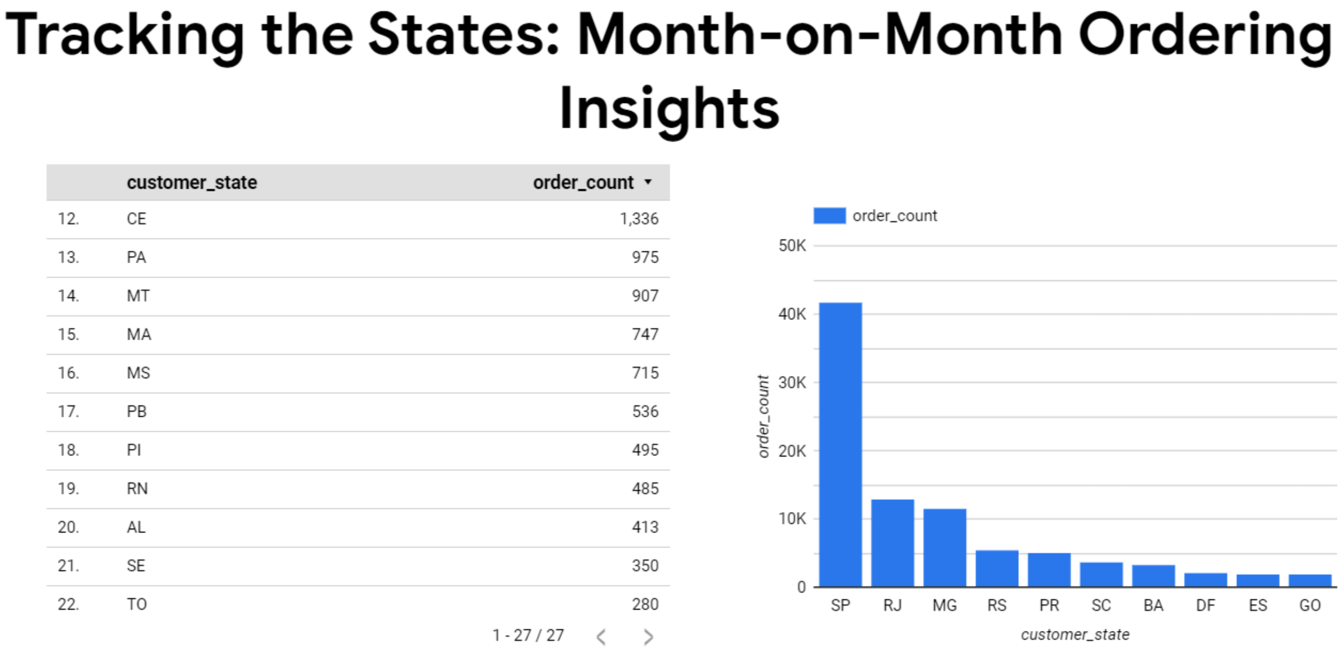


Tracking the States: Month-on-Month Ordering Insights

	customer_state	order_count ▾
1.	SP	41,745
2.	RJ	12,852
3.	MG	11,635
4.	RS	5,466
5.	PR	5,045
6.	SC	3,637
7.	BA	3,380
8.	DF	2,140
9.	ES	2,033
10.	GO	2,020
11.	PE	1,652

1 - 27 / 27 ‹ ›

Figure 3.2 Get month on month orders by states



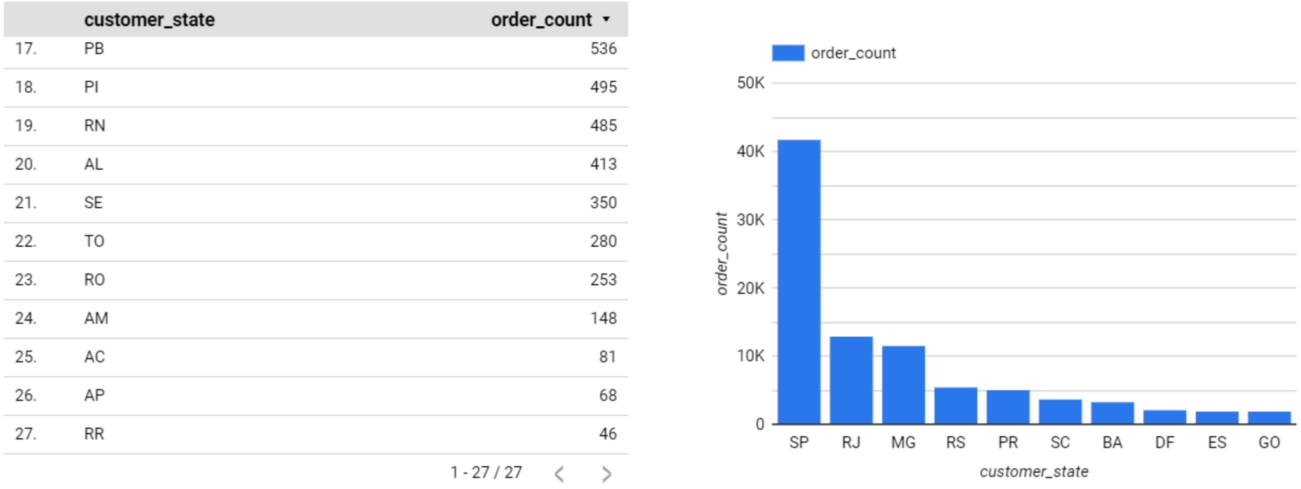
Tracking the States: Month-on-Month Ordering Insights

	customer_state	order_count ▾
12.	CE	1,336
13.	PA	975
14.	MT	907
15.	MA	747
16.	MS	715
17.	PB	536
18.	PI	495
19.	RN	485
20.	AL	413
21.	SE	350
22.	TO	280

1 - 27 / 27 ‹ ›

Figure 3.2 Get month on month orders by states

Tracking the States: Month-on-Month Ordering Insights



	customer_state	order_count ▾
17.	PB	536
18.	PI	495
19.	RN	485
20.	AL	413
21.	SE	350
22.	TO	280
23.	RO	253
24.	AM	148
25.	AC	81
26.	AP	68
27.	RR	46

1 - 27 / 27

Figure 3.2 Get month on month orders by states

3.2 Distribution of customers across the states in Brazil

(**Figure 3.3 BigQuery: From North to South: Exploring Customer Distribution in Brazil**) This query begins with a SELECT statement that specifies two columns to be retrieved: "**c.customer_state**" and "COUNT(**c.customer_id**)" aliased as "**customer_count**." The "**c.customer_state"** column likely represents the state where the customers are located, and the "**c.customer_id**" column is likely a unique identifier for each customer. The COUNT() function is used to count the number of customers in each state.

The query then uses a JOIN clause to combine the "**customers**" and "**geolocation**" tables based on the condition "**c.customer_zip_code_prefix = g.geolocation_zip_code_prefix**". This indicates that the "**customer_zip_code_prefix**" column in the "**customers**" table is being matched with the "**geolocation_zip_code_prefix**" column in the "**geolocation**" table, presumably to link the customer data with their corresponding geographical location.

Next, the query uses a GROUP BY clause to group the results by the "**customer_state**" column, which represents the state where the customers are located. This is followed by an ORDER BY clause that sorts the results in descending order based on the "**customer_count**" column, which represents the count of customers in each state.

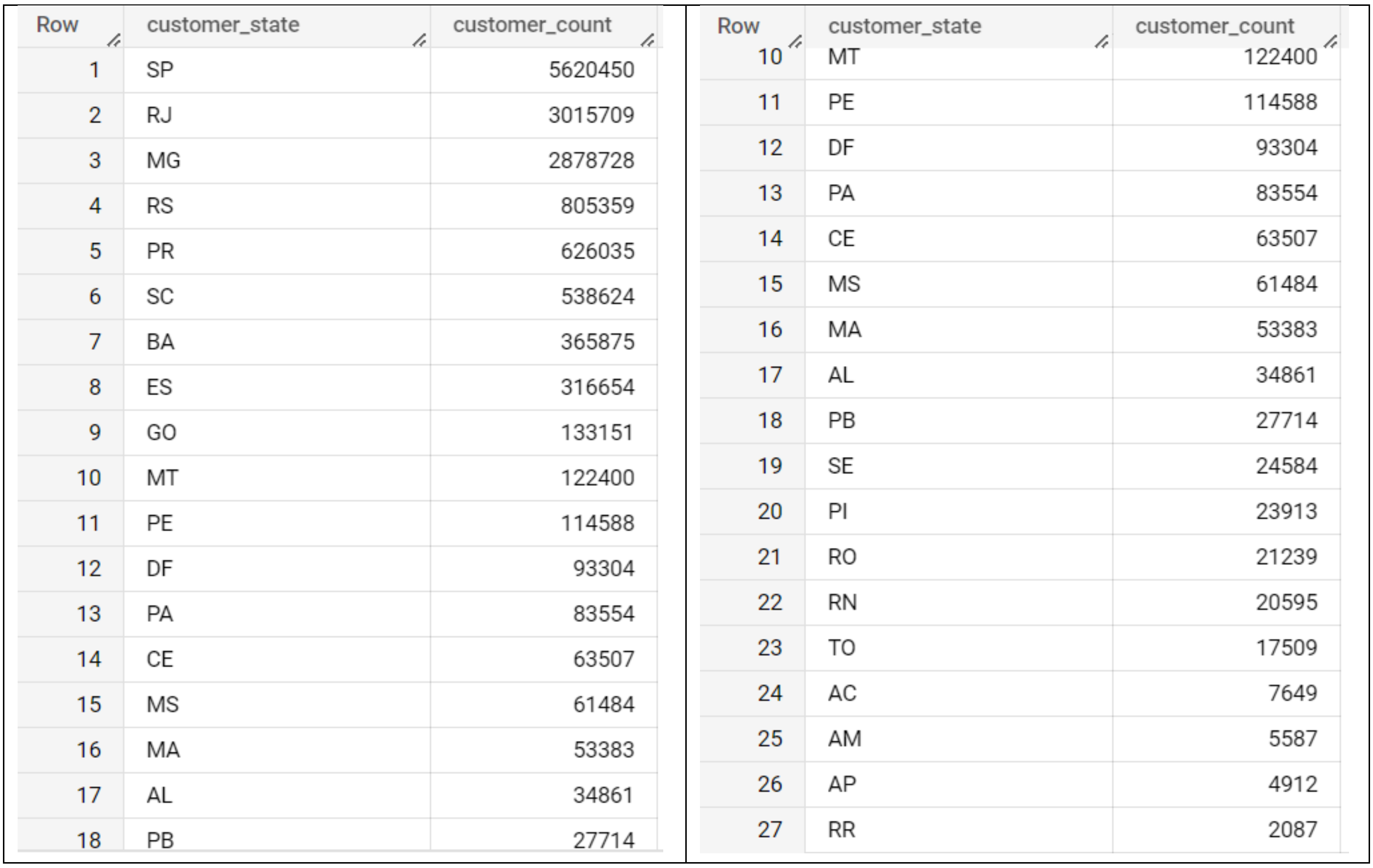
In summary, this SQL query retrieves customer data from a database, joins it with geolocation data, groups the results by state, and orders them by the number of customers in each state, providing insights into customer distribution in Brazil from north to south.

```sql
--From North to South: Exploring Customer Distribution in Brazil

SELECT c.customer_state, COUNT(c.customer_id) as customer_count
FROM target_business.customers c
JOIN target_business.geolocation g ON c.customer_zip_code_prefix = g.geolocation_zip_code_prefix
GROUP BY c.customer_state
ORDER BY customer_count DESC;
```

Figure 3.3 BigQuery: From North to South: Exploring Customer Distribution in Brazil

The distribution of customers across the states in Brazil is as follows:

Query results:



Row	customer_state	customer_count
1	SP	5620450
2	RJ	3015709
3	MG	2878728
4	RS	805359
5	PR	626035
6	SC	538624
7	BA	365875
8	ES	316654
9	GO	133151
10	MT	122400
11	PE	114588
12	DF	93304
13	PA	83554
14	CE	63507
15	MS	61484
16	MA	53383
17	AL	34861
18	PB	27714
19	SE	24584
20	PI	23913
21	RO	21239
22	RN	20595
23	TO	17509
24	AC	7649
25	AM	5587
26	AP	4912
27	RR	2087

Figure 3.4 Query results: From North to South: Exploring Customer Distribution in Brazil

(Figure 3.4 Query results: From North to South: Exploring Customer Distribution in Brazil), the query results provided show data for customer counts in different states of Brazil. Here is a breakdown of the information:

1. SP: São Paulo - 5,620,450 customers
2. RJ: Rio de Janeiro - 3,015,709 customers
3. MG: Minas Gerais - 2,878,728 customers
4. RS: Rio Grande do Sul - 805,359 customers
5. PR: Paraná - 626,035 customers
6. SC: Santa Catarina - 538,624 customers

7. BA: Bahia - 365,875 customers
8. ES: Espírito Santo - 316,654 customers
9. GO: Goiás - 133,151 customers
10. MT: Mato Grosso - 122,400 customers
11. PE: Pernambuco - 114,588 customers
12. DF: Distrito Federal - 93,304 customers
13. PA: Pará - 83,554 customers
14. CE: Ceará - 63,507 customers
15. MS: Mato Grosso do Sul - 61,484 customers
16. MA: Maranhão - 53,383 customers
17. AL: Alagoas - 34,861 customers
18. PB: Paraíba - 27,714 customers
19. SE: Sergipe - 24,584 customers
20. PI: Piauí - 23,913 customers
21. RO: Rondônia - 21,239 customers
22. RN: Rio Grande do Norte - 20,595 customers
23. TO: Tocantins - 17,509 customers
24. AC: Acre - 7,649 customers
25. AM: Amazonas - 5,587 customers
26. AP: Amapá - 4,912 customers
27. RR: Roraima - 2,087 customers

(**Figure 3.4 Query results: From North to South: Exploring Customer Distribution in Brazil**), these results provide the customer count for each state in Brazil, arranged in descending order from the highest count in São Paulo (SP) to the lowest count in Roraima (RR).

(**Figure 3.4 Query results: From North to South: Exploring Customer Distribution in Brazil**), this distribution shows the number of customers in each state of Brazil based on the dataset provided, with the highest number of customers in São Paulo (SP) and the lowest number of customers in Roraima (RR).

Here is more information about São Paulo (SP) and Roraima (RR):

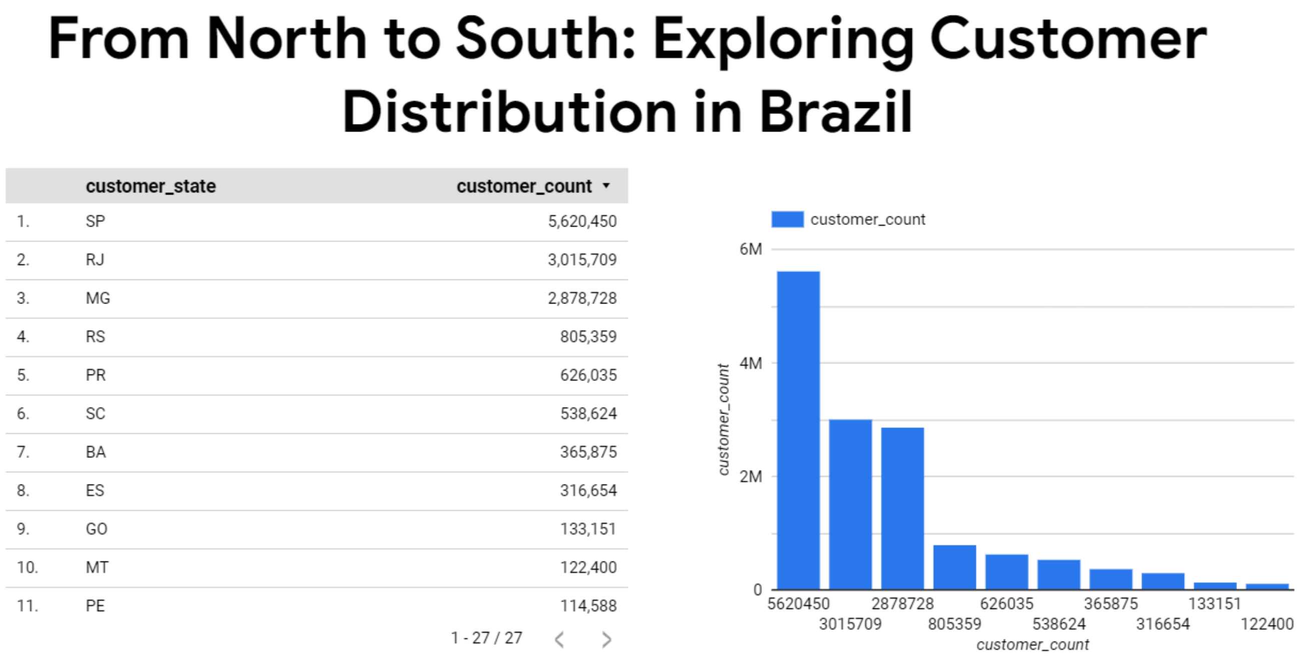
São Paulo (SP):

- São Paulo is a state located in the south-eastern region of Brazil and is the most populous state in the country.
- It has the highest number of customers among all the states listed in the query, with a customer count of 5,620,450.
- São Paulo is known for its diverse economy, with a strong focus on industries such as finance, services, manufacturing, and agriculture.
- The capital of São Paulo state is São Paulo City, which is also the largest city in Brazil and one of the largest cities in the world in terms of population and economic activity.

- São Paulo is known for its cultural richness, with a vibrant arts scene, diverse cuisine, and numerous cultural events and festivals.

Roraima (RR):

- Roraima is a state located in the northern region of Brazil, and it has the lowest number of customers among all the states listed in the query, with a customer count of 2,087.
- Roraima is the least populous state in Brazil and is known for its unique geographical feature, Mount Roraima, which is a tabletop mountain and a popular tourist destination.
- The capital of Roraima state is Boa Vista, which is the only capital city in Brazil located entirely north of the equator.
- Roraima is characterised by its rich indigenous culture, with a significant population of indigenous peoples, and has a unique cultural heritage.
- Roraima has a relatively small economy with a focus on agriculture, mining, and renewable energy resources. It is also known for its natural beauty and ecotourism opportunities, with several protected areas and national parks within its borders.

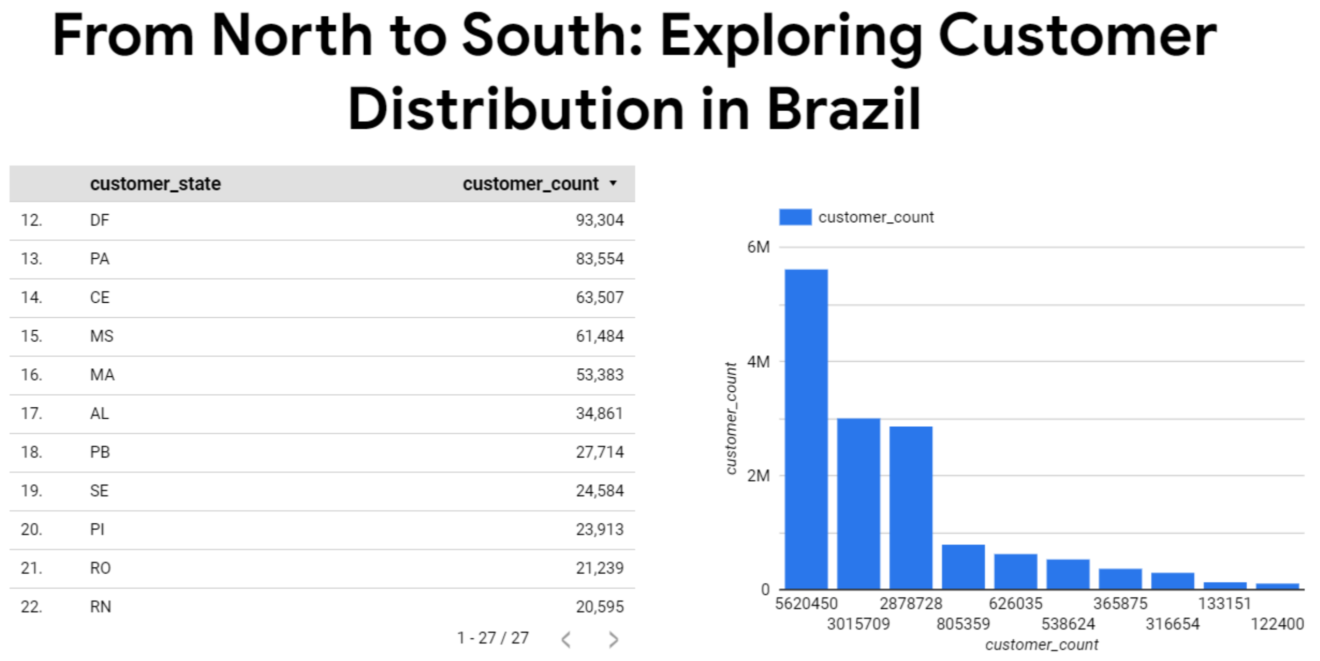


From North to South: Exploring Customer Distribution in Brazil

	customer_state	customer_count ▾
1.	SP	5,620,450
2.	RJ	3,015,709
3.	MG	2,878,728
4.	RS	805,359
5.	PR	626,035
6.	SC	538,624
7.	BA	365,875
8.	ES	316,654
9.	GO	133,151
10.	MT	122,400
11.	PE	114,588

1 - 27 / 27 < >

Figure 3.4 From North to South: Exploring Customer Distribution in Brazil

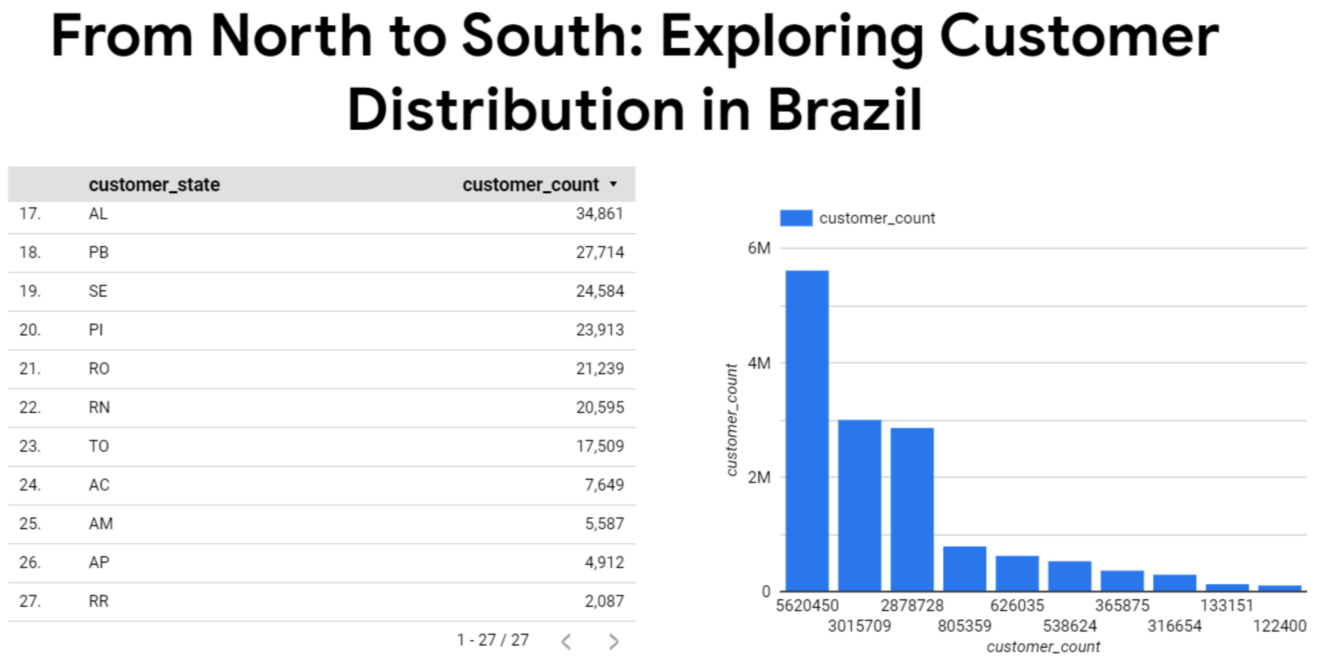


From North to South: Exploring Customer Distribution in Brazil

	customer_state	customer_count ▾
12.	DF	93,304
13.	PA	83,554
14.	CE	63,507
15.	MS	61,484
16.	MA	53,383
17.	AL	34,861
18.	PB	27,714
19.	SE	24,584
20.	PI	23,913
21.	RO	21,239
22.	RN	20,595

1 - 27 / 27 < >

Figure 3.4 From North to South: Exploring Customer Distribution in Brazil



From North to South: Exploring Customer Distribution in Brazil

	customer_state	customer_count ▾
17.	AL	34,861
18.	PB	27,714
19.	SE	24,584
20.	PI	23,913
21.	RO	21,239
22.	RN	20,595
23.	TO	17,509
24.	AC	7,649
25.	AM	5,587
26.	AP	4,912
27.	RR	2,087

1 - 27 / 27 < >

Figure 3.4 From North to South: Exploring Customer Distribution in Brazil

(Figure 3.4 From North to South: Exploring Customer Distribution in Brazil) The table above shows the distribution of customers across states in Brazil, with the **customer_state** column representing the state code and the **customer_count** column representing the total number of customers in each state.

The distribution of customers across states appears to follow a skewed distribution, with a few states having significantly higher customer counts compared to others. The state of São Paulo (SP) has the highest number of customers at 5,620,450, followed by Rio de Janeiro (RJ) with 3,015,709 customers, and Minas Gerais (MG) with 2,878,728 customers. As we move to other states, the number of customers gradually decreases, with states such as AM (Amazonas), AP (Amapá), and RR (Roraima) having relatively lower customer counts.

This skewed distribution is further supported by the fact that the states with higher customer counts are located in the more populous and economically developed regions of Brazil, such as the Southeast and South regions, while states with lower customer counts are located in less populous and economically developed regions, such as the North and Northeast regions.

It's important to note that this distribution is based on the dataset provided and may not necessarily reflect the actual population distribution of customers across states in Brazil. Additionally, other factors such as market size, population density, economic activity, and customer behaviour could also impact the distribution of customers across states.

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

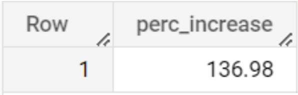
4.1 Get % increase in cost of orders from 2017 to 2018 (include months between Jan to Aug only) - You can use “payment_value” column in payments table

(**Figure 4 BigQuery: From 2017 to 2018: Calculating the Percentage Increase in Order Costs**), this code uses the **EXTRACT** function to extract the year and month from the **order_purchase_timestamp** column in the orders table. It then calculates the total payment value for orders in 2017 and 2018 separately using the **SUM** function with conditional aggregation, and calculates the percentage increase in the total payment value from 2017 to 2018. The result is grouped by year and month, and ordered by year and month for better presentation.

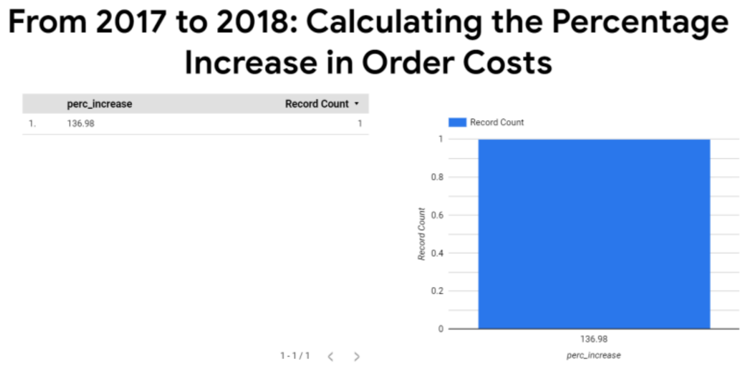
```
--- From 2017 to 2018: Calculating the Percentage Increase in Order Costs
WITH A AS
(
  SELECT
    EXTRACT(YEAR FROM o.order_purchase_timestamp) AS year,
    SUM(p.payment_value) as cost_of_orders
  FROM
    target_business.orders o
  JOIN
    target_business.payments p ON o.order_id = p.order_id
  WHERE
    EXTRACT(month FROM o.order_purchase_timestamp) BETWEEN 1 AND 8
  GROUP BY
    1
)
SELECT
  ROUND(((a2.cost_of_orders / a1.cost_of_orders) - 1) * 100, 2) as perc_increase
FROM
  A as a1, A as a2
WHERE
  a1.year = 2017 AND a2.year = 2018;
```

Figure 4 BigQuery: From 2017 to 2018: Calculating the Percentage Increase in Order Costs

Query results



Row	perc_increase
1	136.98



From 2017 to 2018: Calculating the Percentage Increase in Order Costs

	perc_increase	Record Count ▾
1.	136.98	1

1 - 1 / 1 < >

4.2 Mean & Sum of price and freight value by customer state

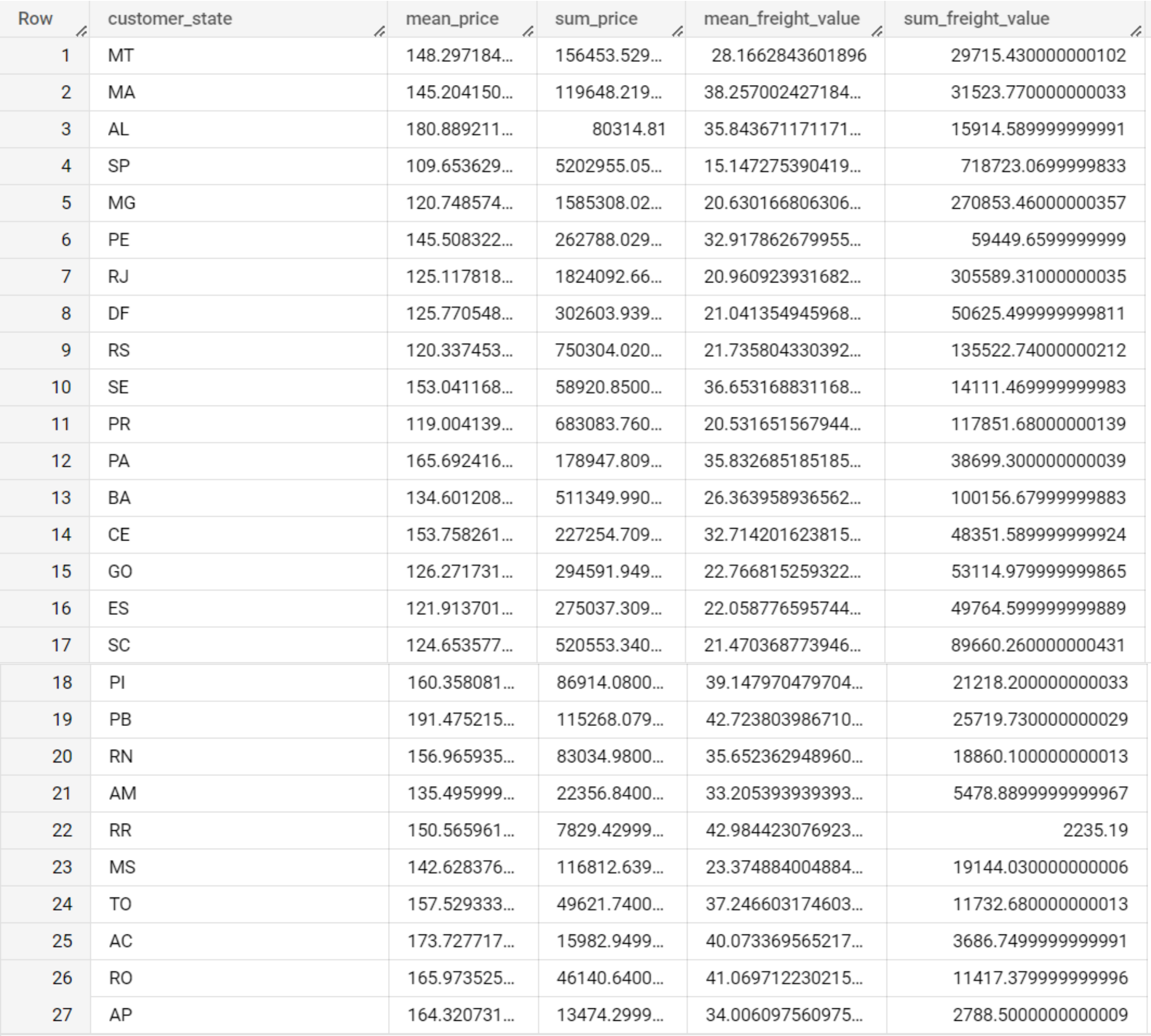
(**Figure 4.2 BigQuery: State-wise E-commerce Insights: Mean and Sum of Price and Freight Values Common Table Expression (CTE) to retrieve order items data**) In this query, it first create two CTEs - **order_items_cte** and **customers_cte** - to extract the relevant columns from the **order_items** and customers CSV files, respectively. Then, it create a combined CTE called **combined_data** by joining the **order_items_cte**, orders, and **customers_cte** tables on their respective keys.

Finally, it use the **combined_data CTE** to perform aggregation using AVG and SUM functions to calculate the mean and sum of price and **freight_value** columns, respectively, grouped by **customer_state**. This will give us the desired result of mean and sum of price and freight value by customer state.

```
-- State-wise E-commerce Insights: Mean and Sum of Price and Freight Values
-- Common Table Expression (CTE) to retrieve order items data
WITH order_items_cte AS (
   SELECT order_id, price, freight_value
   FROM target_business.order_items
),

-- Common Table Expression (CTE) to retrieve customers data
customers_cte AS (
   SELECT customer_id, customer_state
   FROM target_business.customers c
),

-- Common Table Expression (CTE) to combine data from order_items_cte, orders, and customers_cte
combined_data AS (
   SELECT c.customer_state, oi.price, oi.freight_value
   FROM order_items_cte oi
   INNER JOIN target_business.orders o ON oi.order_id = o.order_id
   INNER JOIN customers_cte c ON o.customer_id = c.customer_id
)

-- Main query to calculate mean and sum for each customer state
SELECT customer_state,
     AVG(price) AS mean_price,
     SUM(price) AS sum_price,
     AVG(freight_value) AS mean_freight_value,
    SUM(freight_value) AS sum_freight_value
FROM combined_data
GROUP BY customer_state;
```

Figure 4.2 BigQuery: State-wise E-commerce Insights: Mean and Sum of Price and Freight Values Common Table Expression (CTE) to retrieve order items data

Query results:



Row	customer_state	mean_price	sum_price	mean_freight_value	sum_freight_value
1	MT	148.297184...	156453.529...	28.1662843601896	29715.430000000102
2	MA	145.204150...	119648.219...	38.257002427184...	31523.770000000033
3	AL	180.889211...	80314.81	35.843671171171...	15914.589999999991
4	SP	109.653629...	5202955.05...	15.147275390419...	718723.0699999833
5	MG	120.748574...	1585308.02...	20.630166806306...	270853.46000000357
6	PE	145.508322...	262788.029...	32.917862679955...	59449.6599999999
7	RJ	125.117818...	1824092.66...	20.960923931682...	305589.31000000035
8	DF	125.770548...	302603.939...	21.041354945968...	50625.499999999811
9	RS	120.337453...	750304.020...	21.735804330392...	135522.74000000212
10	SE	153.041168...	58920.8500...	36.653168831168...	14111.469999999983
11	PR	119.004139...	683083.760...	20.531651567944...	117851.68000000139
12	PA	165.692416...	178947.809...	35.832685185185...	38699.300000000039
13	BA	134.601208...	511349.990...	26.363958936562...	100156.67999999883
14	CE	153.758261...	227254.709...	32.714201623815...	48351.589999999924
15	GO	126.271731...	294591.949...	22.766815259322...	53114.979999999865
16	ES	121.913701...	275037.309...	22.058776595744...	49764.599999999889
17	SC	124.653577...	520553.340...	21.470368773946...	89660.260000000431
18	PI	160.358081...	86914.0800...	39.147970479704...	21218.200000000033
19	PB	191.475215...	115268.079...	42.723803986710...	25719.730000000029
20	RN	156.965935...	83034.9800...	35.652362948960...	18860.100000000013
21	AM	135.495999...	22356.8400...	33.205393939393...	5478.8899999999967
22	RR	150.565961...	7829.42999...	42.984423076923...	2235.19
23	MS	142.628376...	116812.639...	23.374884004884...	19144.030000000006
24	TO	157.529333...	49621.7400...	37.246603174603...	11732.680000000013
25	AC	173.727717...	15982.9499...	40.073369565217...	3686.7499999999991
26	RO	165.973525...	46140.6400...	41.069712230215...	11417.379999999996
27	AP	164.320731...	13474.2999...	34.006097560975...	2788.5000000000009

5. Analysis on sales, freight and delivery time

5.1 Calculate days between purchasing, delivering and estimated delivery

(Figure 5 BigQuery: Calculate days between purchasing, delivering and estimated delivery), to calculate the days between purchasing, delivering, and estimated delivery, I use the following formula:

- Days between purchasing and delivering = carrier_delay + customer_delay
- Days between purchasing and estimated delivery = estimated_delivery_delay

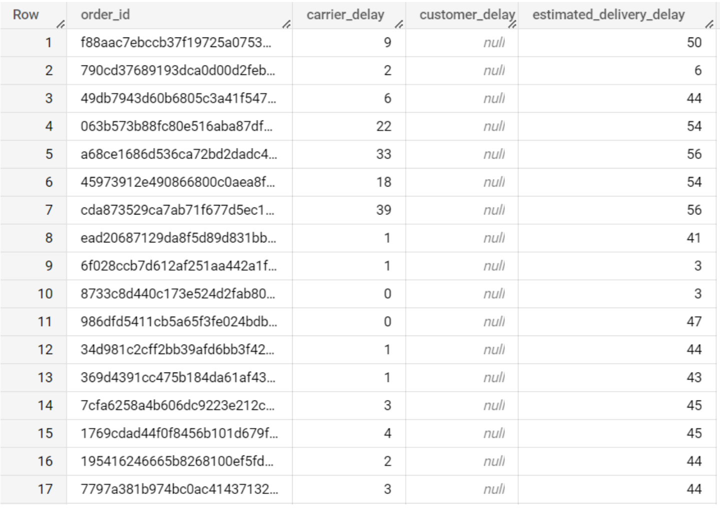
The "**order_id**" is the unique identifier for each order. "**carrier_delay**" represents the number of days of delay caused by the carrier in delivering the order. "**customer_delay**" represents the number of days of delay caused by the customer in receiving the order. "**estimated_delivery_delay**" represents the estimated number of days for delivery as provided by the seller.

The results will provide the time duration in days between the different stages of the order process, including the time it took for the carrier to deliver the order, the estimated delivery time provided by the seller, and any delays caused by the customer. This analysis can help identify patterns or trends in sales, freight, and delivery time, and provide insights for improving order management and customer satisfaction.

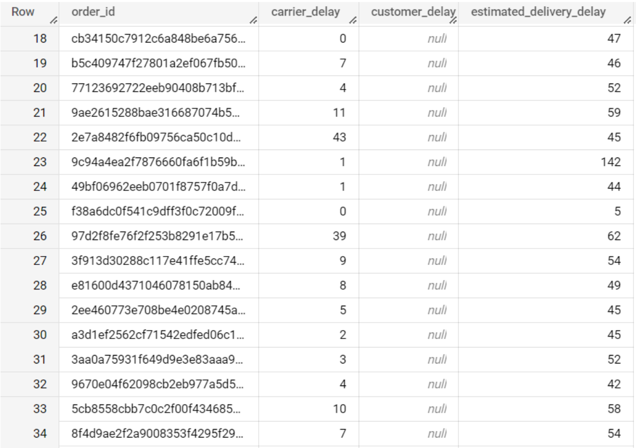
```
--Analysis on sales, freight and delivery time
--Calculate days between purchasing, delivering and estimated delivery
WITH order_info AS (
 SELECT
  o.order_id,
  o.order_purchase_timestamp,
  o.order_delivered_carrier_date,
  o.order_delivered_customer_date,
  o.order_estimated_delivery_date
 FROM
  target_business.orders o
)
, order_delays AS (
 SELECT
  order_id,
  DATE_DIFF(order_delivered_carrier_date, order_purchase_timestamp, DAY) AS carrier_delay,
  DATE_DIFF(order_delivered_customer_date, order_purchase_timestamp, DAY) AS customer_delay,
  DATE_DIFF(order_estimated_delivery_date, order_purchase_timestamp, DAY) AS estimated_delivery_delay
 FROM
  order_info
)
SELECT
 order_id,
 carrier_delay,
 customer_delay,
 estimated_delivery_delay
FROM
 order_delays;
```

Figure 5 BigQuery: Calculate days between purchasing, delivering and estimated delivery

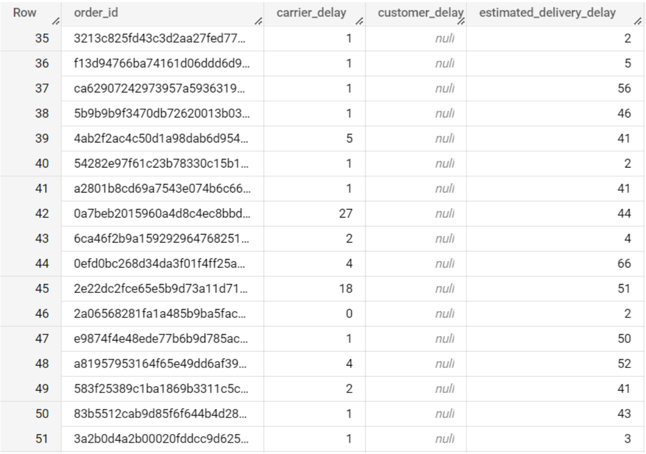
Query results



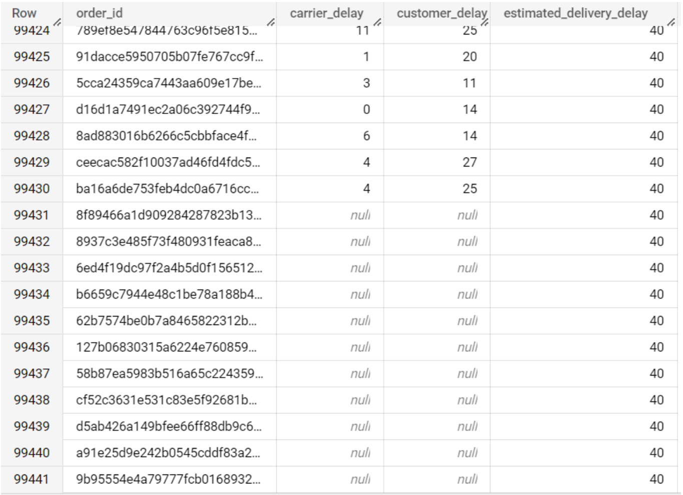
Row	order_id	carrier_delay	customer_delay	estimated_delivery_delay
1	f88aac7ebccb37f19725a0753…	9	null	50
2	790cd37689193dca0d00d2feb…	2	null	6
3	49db7943d60b6805c3a41f547…	6	null	44
4	063b573b88fc80e516aba87df…	22	null	54
5	a68ce1686d536ca72bd2dadc4…	33	null	56
6	45973912e490866800c0aea8f…	18	null	54
7	cda873529ca7ab71f677d5ec1…	39	null	56
8	ead20687129da8f5d89d831bb…	1	null	41
9	6f028ccb7d612af251aa442a1f…	1	null	3
10	8733c8d440c173e524d2fab80…	0	null	3
11	986dfd5411cb5a65f3fe024bdb…	0	null	47
12	34d981c2cff2bb39afd6bb3f42…	1	null	44
13	369d4391cc475b184da61af43…	1	null	43
14	7cfa6258a4b606dc9223e212c…	3	null	45
15	1769cdad44f0f8456b101d679f…	4	null	45
16	195416246665b8268100ef5fd…	2	null	44
17	7797a381b974bc0ac41437132…	3	null	44



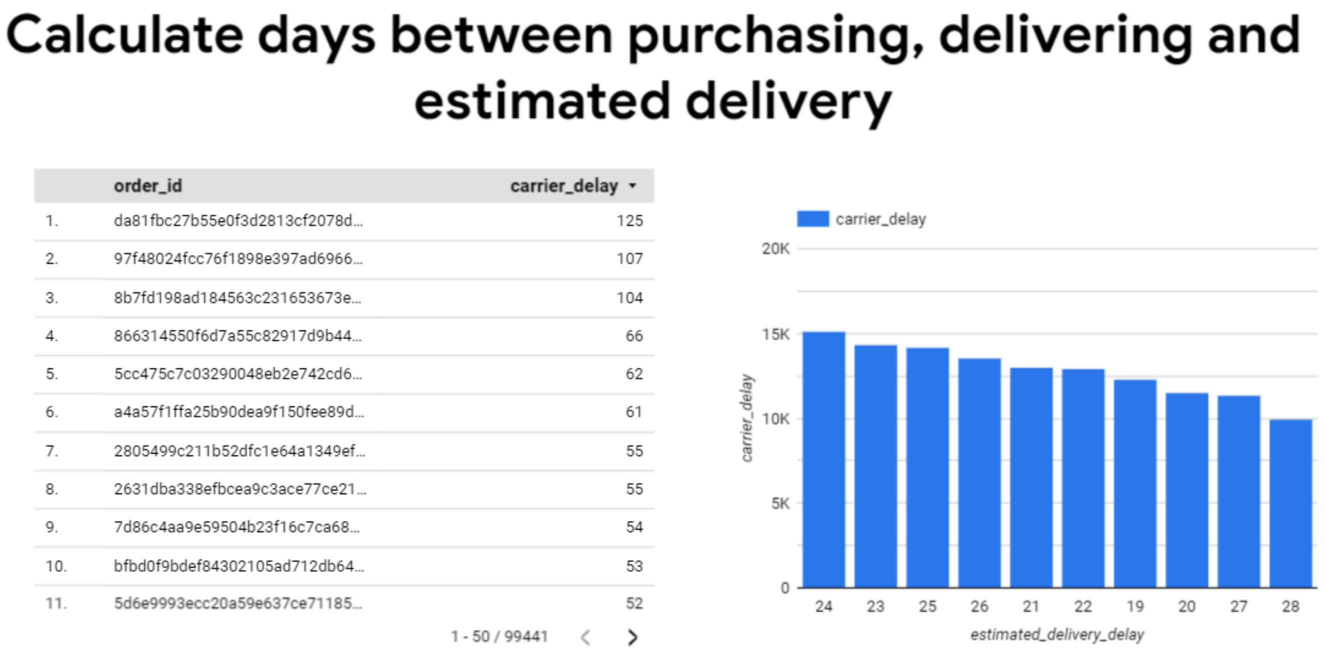
Row	order_id	carrier_delay	customer_delay	estimated_delivery_delay
18	cb34150c7912c6a848be6a756…	0	null	47
19	b5c409747f27801a2ef067fb50…	7	null	46
20	77123692722eeb90408b713bf…	4	null	52
21	9ae2615288bae316687074b5…	11	null	59
22	2e7a8482f6fb09756ca50c10d…	43	null	45
23	9c94a4ea2f7876660fa6f1b59b…	1	null	142
24	49bf06962eeb0701f8757f0a7d…	1	null	44
25	f38a6dc0f541c9dff3f0c72009f…	0	null	5
26	97d2f8fe76f2f253b8291e17b5…	39	null	62
27	3f913d30288c117e41ffe5cc74…	9	null	54
28	e81600d4371046078150ab84…	8	null	49
29	2ee460773e708be4e0208745a…	5	null	45
30	a3d1ef2562cf71542edfed06c1…	2	null	45
31	3aa0a75931f649d9e3e83aaa9…	3	null	52
32	9670e04f62098cb2eb977a5d5…	4	null	42
33	5cb8558cbb7c0c2f00f434685…	10	null	58
34	8f4d9ae2f2a9008353f4295f29…	7	null	54



Row	order_id	carrier_delay	customer_delay	estimated_delivery_delay
35	3213c825fd43c3d2aa27fed77…	1	null	2
36	f13d94766ba74161d06ddd6d9…	1	null	5
37	ca62907242973957a5936319…	1	null	56
38	5b9b9b9f3470db72620013b03…	1	null	46
39	4ab2f2ac4c50d1a98dab6d954…	5	null	41
40	54282e97f61c23b78330c15b1…	1	null	2
41	a2801b8cd69a7543e074b6c66…	1	null	41
42	0a7beb2015960a4d8c4ec8bbd…	27	null	44
43	6ca46f2b9a159292964768251…	2	null	4
44	0efd0bc268d34da3f01f4ff25a…	4	null	66
45	2e22dc2fce65e5b9d73a11d71…	18	null	51
46	2a06568281fa1a485b9ba5fac…	0	null	2
47	e9874f4e48ede77b6b9d785ac…	1	null	50
48	a81957953164f65e49dd6af39…	4	null	52
49	583f25389c1ba1869b3311c5c…	2	null	41
50	83b5512cab9d85f6f644b4d28…	1	null	43
51	3a2b0d4a2b00020fddcc9d625…	1	null	3



Row	order_id	carrier_delay	customer_delay	estimated_delivery_delay
99424	789ef8e547844763c96f5e815…	11	25	40
99425	91dacce5950705b07fe767cc9f…	1	20	40
99426	5cca24359ca7443aa609e17be…	3	11	40
99427	d16d1a7491ec2a06c392744f9…	0	14	40
99428	8ad883016b6266c5cbbface4f…	6	14	40
99429	ceecac582f10037ad46fd4fdc5…	4	27	40
99430	ba16a6de753feb4dc0a6716cc…	4	25	40
99431	8f89466a1d909284287823b13…	null	null	40
99432	8937c3e485f73f480931feaca8…	null	null	40
99433	6ed4f19dc97f2a4b5d0f156512…	null	null	40
99434	b6659c7944e48c1be78a188b4…	null	null	40
99435	62b7574be0b7a8465822312b…	null	null	40
99436	127b06830315a6224e760859…	null	null	40
99437	58b87ea5983b516a65c224359…	null	null	40
99438	cf52c3631e531c83e5f92681b…	null	null	40
99439	d5ab426a149bfee66ff88db9c6…	null	null	40
99440	a91e25d9e242b0545cddf83a2…	null	null	40
99441	9b95554e4a79777fcb0168932…	null	null	40



Calculate days between purchasing, delivering and estimated delivery

	order_id	carrier_delay ▾
1.	da81fbc27b55e0f3d2813cf2078d…	125
2.	97f48024fcc76f1898e397ad6966…	107
3.	8b7fd198ad184563c231653673e…	104
4.	866314550f6d7a55c82917d9b44…	66
5.	5cc475c7c03290048eb2e742cd6…	62
6.	a4a57f1ffa25b90dea9f150fee89d…	61
7.	2805499c211b52dfc1e64a1349ef…	55
8.	2631dba338efbcea9c3ace77ce21…	55
9.	7d86c4aa9e59504b23f16c7ca68…	54
10.	bfbd0f9bdef84302105ad712db64…	53
11.	5d6e9993ecc20a59e637ce71185…	52

Figure 5.1 Calculate days between purchasing, delivering and estimated delivery

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

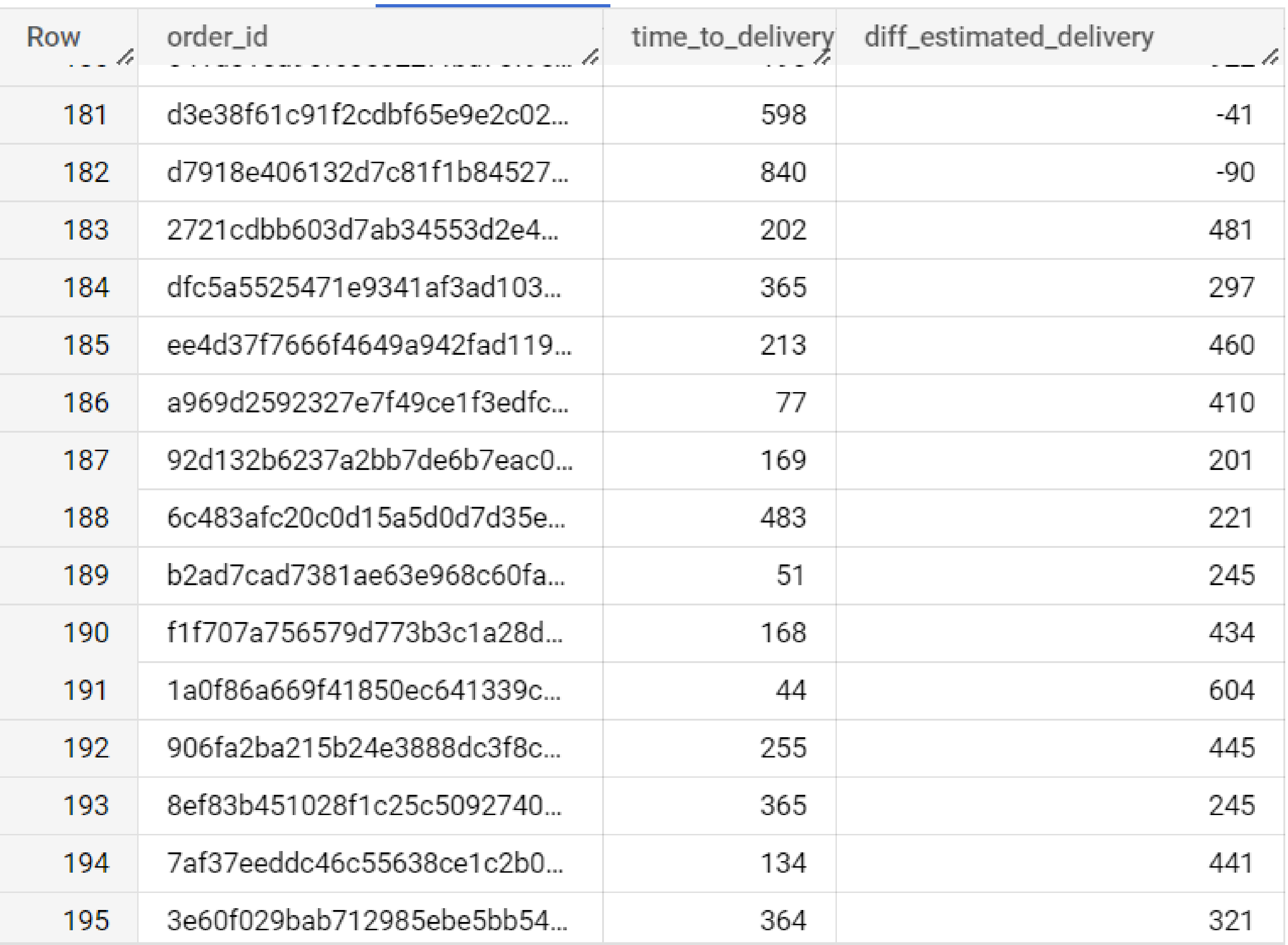
- **time_to_delivery = order_purchase_timestamp-order_delivered_customer_date**
- **diff_estimated_delivery = order_estimated_delivery_date-order_delivered_customer_date**

(Figure 5.2 Find time_to_delivery & diff_estimated_delivery), in this query, I create a CTE called **order_data** that selects the necessary columns from the orders table, including order_id, **order_purchase_timestamp**, **order_delivered_customer_date**, and **order_estimated_delivery_date**. Then, we use the TIMESTAMP_DIFF function to calculate the time difference in hours between **order_purchase_timestamp** and **order_delivered_customer_date** as **time_to_delivery**, and between **order_estimated_delivery_date** and **order_delivered_customer_date** as **diff_estimated_delivery**. Finally, select **order_id**, **time_to_delivery**, and **diff_estimated_delivery** from the **order_data CTE**.

```sql
WITH order_data AS (
 SELECT
  order_id,
  order_purchase_timestamp,
  order_delivered_customer_date,
  order_estimated_delivery_date
 FROM
  `target_business.orders`
)
SELECT
 order_id,
 TIMESTAMP_DIFF(order_delivered_customer_date, order_purchase_timestamp, HOUR) AS time_to_delivery,
 TIMESTAMP_DIFF(order_estimated_delivery_date, order_delivered_customer_date, HOUR) AS diff_estimated_delivery
FROM
 order_data;
```

Figure 5.2 BigQuery: Find time_to_delivery & diff_estimated_delivery

Query Results



Row	order_id	time_to_delivery	diff_estimated_delivery
181	d3e38f61c91f2cdbf65e9e2c02...	598	-41
182	d7918e406132d7c81f1b84527...	840	-90
183	2721cdbb603d7ab34553d2e4...	202	481
184	dfc5a5525471e9341af3ad103...	365	297
185	ee4d37f7666f4649a942fad119...	213	460
186	a969d2592327e7f49ce1f3edfc...	77	410
187	92d132b6237a2bb7de6b7eac0...	169	201
188	6c483afc20c0d15a5d0d7d35e...	483	221
189	b2ad7cad7381ae63e968c60fa...	51	245
190	f1f707a756579d773b3c1a28d...	168	434
191	1a0f86a669f41850ec641339c...	44	604
192	906fa2ba215b24e3888dc3f8c...	255	445
193	8ef83b451028f1c25c5092740...	365	245
194	7af37eeddc46c55638ce1c2b0...	134	441
195	3e60f029bab712985ebe5bb54...	364	321

Figure 5.3 Query results: Find time_to_delivery & diff_estimated_delivery

(Figure 5.3 Query results: Find time_to_delivery & diff_estimated_delivery), the given data appears to be a table with information related to orders, including order IDs, time taken for delivery (in days) denoted as "**time_to_delivery**", and the difference between the estimated delivery date and the actual delivered customer date denoted as "**diff_estimated_delivery**". Let's break down the results:

- Row 182: The order with ID "d7918e406132d7c81f1b845276b03a3b" took 840 days for delivery, and the estimated delivery date was 90 days earlier than the actual delivered customer date.
- Row 183: The order with ID "2721cdbb603d7ab34553d2e44a6f9ae0" took 202 days for delivery, and the estimated delivery date was 481 days after the actual delivered customer date.
- Row 184: The order with ID "dfc5a5525471e9341af3ad103adbef79" took 365 days for delivery, and the estimated delivery date was 297 days after the actual delivered customer date.
- Row 185: The order with ID "ee4d37f7666f4649a942fad1192bb2f4" took 213 days for delivery, and the estimated delivery date was 460 days after the actual delivered customer date.
- Row 186: The order with ID "a969d2592327e7f49ce1f3edfc7658eb" took 77 days for delivery, and the estimated delivery date was 410 days after the actual delivered customer date.

The same pattern continues for the remaining rows, where "**time_to_delivery**" represents the time taken for delivery in days, and "**diff_estimated_delivery**" represents the difference between the estimated delivery date and the actual delivered customer date in days. The specific meaning and implications of these values may depend on the context of the business or system for which this data is relevant.

5.3 Group data by state, take mean of freight_value, time_to_delivery, diff_estimated_delivery

(Figure 5.5 Group data by state, take mean of freight_value, time_to_delivery, diff_estimated_delivery)
avg_freight_value: This column shows the average freight value for orders in each state. Freight value is the cost of shipping for an order, and the average value gives an indication of the average shipping cost incurred by customers in each state.

avg_time_to_delivery: This column shows the average time taken for delivery of orders in each state. Time to delivery is calculated as the difference between the order purchase timestamp and the order delivered customer date, and the average time gives an indication of the average delivery speed in each state.

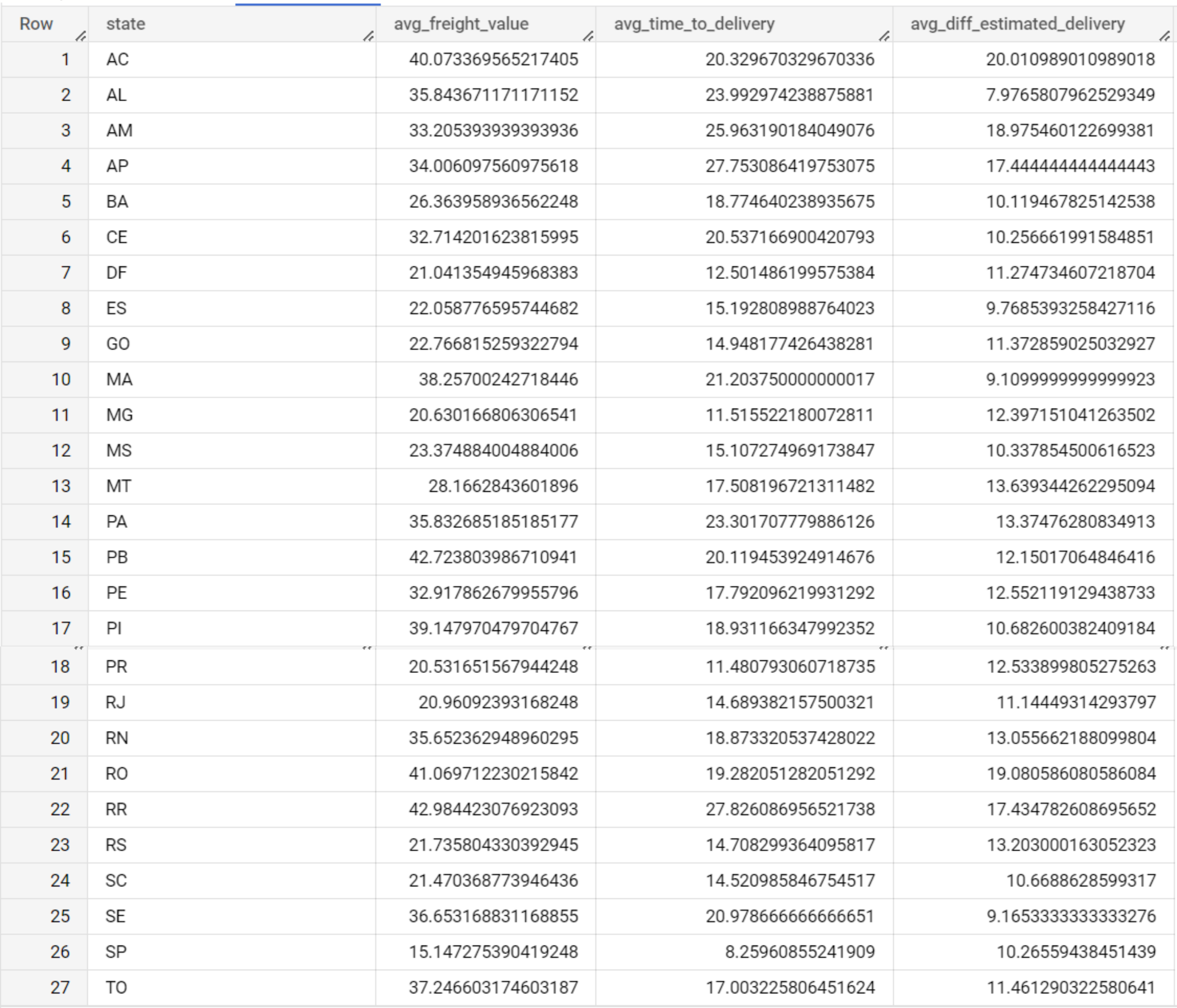
avg_diff_estimated_delivery: This column shows the average difference between the estimated delivery date and the actual delivered customer date for orders in each state. The estimated delivery date is subtracted from the delivered customer date to calculate this difference, and the average value gives an indication of how closely the estimated delivery dates align with the actual delivery dates in each state.

Overall, these results provide insights into the average shipping cost, delivery speed, and accuracy of estimated delivery dates for orders in different states.

```
WITH order_stats AS (
 SELECT
  c.customer_state AS state,
  AVG(oi.freight_value) AS avg_freight_value,
  AVG(date_diff( o.order_delivered_customer_date, o.order_purchase_timestamp, day)) AS avg_time_to_delivery,
  AVG(date_diff(o.order_estimated_delivery_date, o.order_delivered_customer_date, day)) AS avg_diff_estimated_delivery
 FROM
  target_business.customers c
 JOIN
  target_business.orders o ON c.customer_id = o.customer_id
 JOIN
  target_business.order_items oi ON o.order_id = oi.order_id
 GROUP BY
  state
)
SELECT
 state,
 avg_freight_value,
 avg_time_to_delivery,
 avg_diff_estimated_delivery
FROM
 order_stats
ORDER BY
 state;
```

Figure 5.4 BigQuery: Group data by state, take mean of freight_value, time_to_delivery, diff_estimated_delivery

Query Results



Row	state	avg_freight_value	avg_time_to_delivery	avg_diff_estimated_delivery
1	AC	40.073369565217405	20.329670329670336	20.010989010989018
2	AL	35.843671171171152	23.992974238875881	7.9765807962529349
3	AM	33.205393939393936	25.963190184049076	18.97546012269381
4	AP	34.006097560975618	27.753086419753075	17.444444444444443
5	BA	26.363958936562248	18.774640238935675	10.119467825142538
6	CE	32.714201623815995	20.537166900420793	10.256661991584851
7	DF	21.041354945968383	12.501486199575384	11.274734607218704
8	ES	22.058776595744682	15.192808988764023	9.7685393258427116
9	GO	22.766815259322794	14.948177426438281	11.372859025032927
10	MA	38.25700242718446	21.203750000000017	9.1099999999999923
11	MG	20.630166806306541	11.515522180072811	12.397151041263502
12	MS	23.374884004884006	15.107274969173847	10.337854500616523
13	MT	28.1662843601896	17.508196721311482	13.639344262295094
14	PA	35.832685185185177	23.301707779886126	13.37476280834913
15	PB	42.723803986710941	20.119453924914676	12.15017064846416
16	PE	32.917862679955796	17.792096219931292	12.552119129438733
17	PI	39.147970479704767	18.931166347992352	10.682600382409184
18	PR	20.531651567944248	11.480793060718735	12.533899805275263
19	RJ	20.96092393168248	14.689382157500321	11.14449314293797
20	RN	35.652362948960295	18.873320537428022	13.055662188099804
21	RO	41.069712230215842	19.282051282051292	19.080586080586084
22	RR	42.984423076923093	27.826086956521738	17.434782608695652
23	RS	21.735804330392945	14.708299364095817	13.203000163052323
24	SC	21.470368773946436	14.520985846754517	10.6688628599317
25	SE	36.653168831168855	20.978666666666651	9.1653333333333276
26	SP	15.147275390419248	8.25960855241909	10.26559438451439
27	TO	37.246603174603187	17.003225806451624	11.461290322580641

Figure 5.5 Group data by state, take mean of freight_value, time_to_delivery, diff_estimated_delivery

5.4. Sort the data to get the following:

5.5 Top 5 states with highest/lowest average freight value - sort in desc/asc limit 5

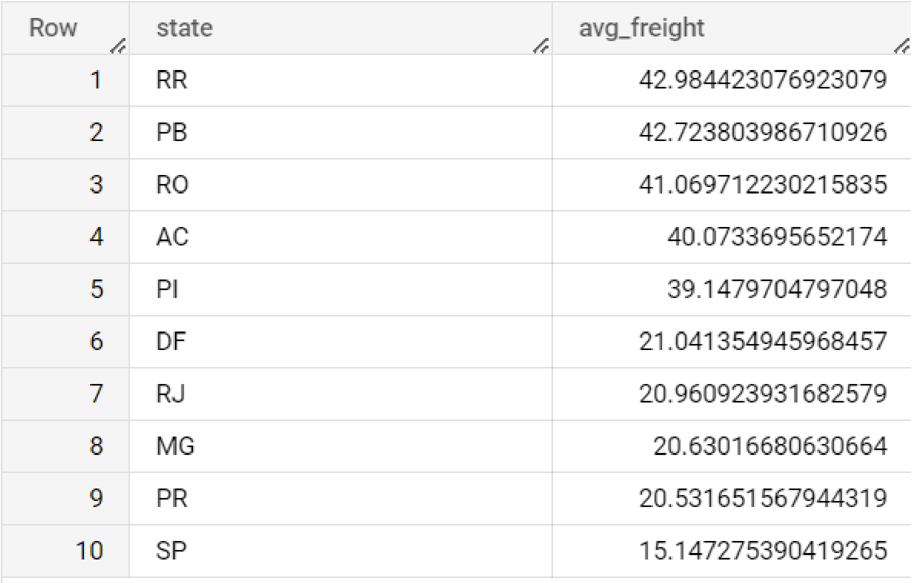
Explanation (Figure 5.6 BigQuery: Top 5 states with highest/lowest average freight value - sort in desc/asc limit 5):

- The **state_freight_avg CTE** calculates the average freight value for each state by joining the **customers**, **orders**, and **order_items** tables, and grouping by the **customer_state** column.
- The outer query selects the state and average freight value from the **state_freight_avg CTE**.
- The ROW_NUMBER() function is used to assign row numbers to the results based on the average freight value, in descending and ascending order separately.

- The results are filtered using the row numbers to limit to the **top 5 states with the highest and lowest average freight value**.
- The final results are sorted by average freight value in descending order and state in ascending order.

```
-- 5. Top 5 states with highest/lowest average freight value - sort in desc/asc limit 5
WITH state_freight_avg AS (
 SELECT
  c.customer_state AS state,
  AVG(oi.freight_value) AS avg_freight
 FROM
  target_business.customers c
 JOIN
  target_business.orders o ON c.customer_id = o.customer_id
 JOIN
  target_business.order_items oi ON o.order_id = oi.order_id
 GROUP BY
  c.customer_state
)
SELECT
 state,
 avg_freight
FROM (
 SELECT
  state,
  avg_freight,
  ROW_NUMBER() OVER (ORDER BY avg_freight DESC) AS rn_desc,
  ROW_NUMBER() OVER (ORDER BY avg_freight ASC) AS rn_asc
 FROM
  state_freight_avg
)
WHERE
 rn_desc <= 5 OR rn_asc <= 5
ORDER BY
 avg_freight DESC, state ASC;
```

Figure 5.6 BigQuery: Top 5 states with highest/lowest average freight value - sort in desc/asc limit 5



Row	state	avg_freight
1	RR	42.984423076923079
2	PB	42.723803986710926
3	RO	41.069712230215835
4	AC	40.0733695652174
5	PI	39.1479704797048
6	DF	21.041354945968457
7	RJ	20.960923931682579
8	MG	20.63016680630664
9	PR	20.531651567944319
10	SP	15.147275390419265

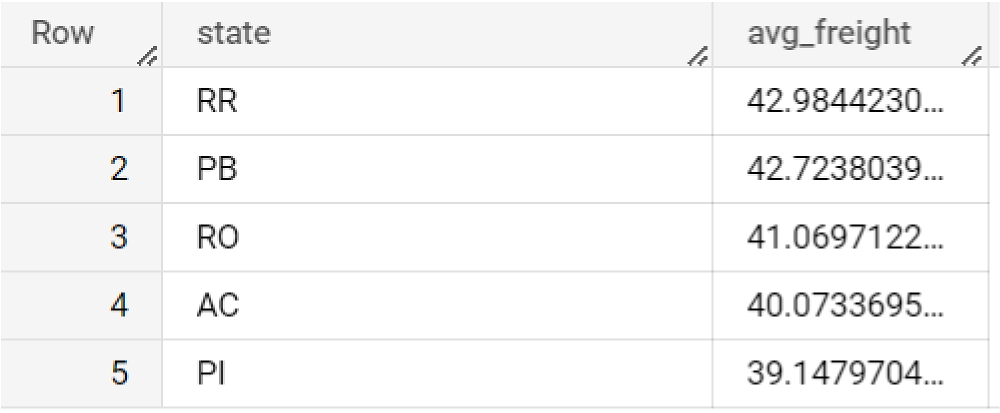
Figure 5.7 Query results: the top 5 states with the highest and lowest average freight value- sort in desc/asc limit 5 - sort in desc/asc limit 5

Here's a query using BigQuery with common table expressions (CTEs) to find the top 5 states with the highest and lowest average freight value, sorted in descending and ascending order, limited to 5 results:

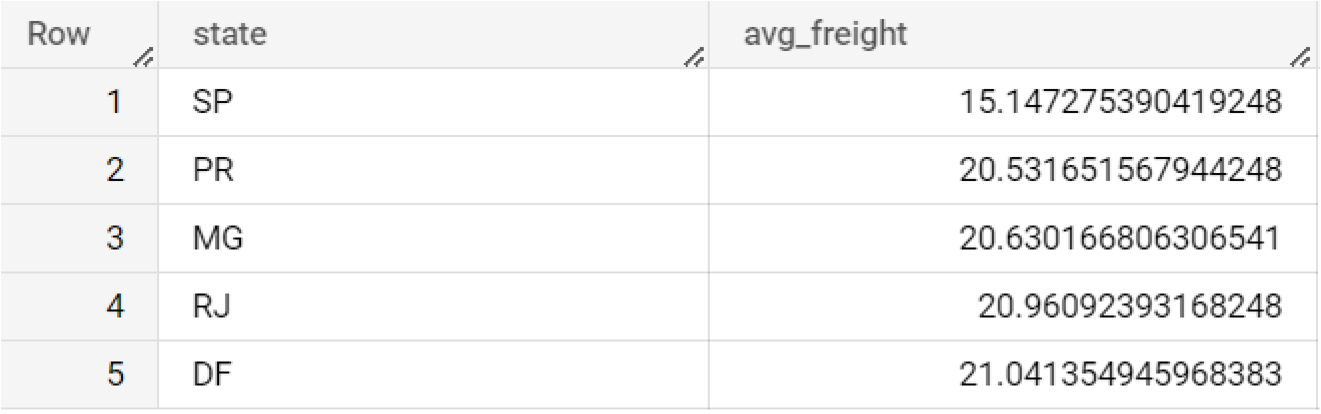
```
WITH freight_avg_by_state AS (
 SELECT
  c.customer_state AS state,
  AVG(oi.freight_value) AS avg_freight
 FROM
  target_business.customers c
 INNER JOIN
  target_business.orders o ON c.customer_id = o.customer_id
 INNER JOIN
  target_business.order_items oi ON o.order_id = oi.order_id
 GROUP BY
  state
)
SELECT
 state,
 avg_freight
FROM
 freight_avg_by_state
ORDER BY
 avg_freight DESC
LIMIT
 5; -- Top 5 states with highest average freight value
```

Figure 5.8 BigQuery: top 5 states with the highest average freight value and the bottom 5 states with the lowest average freight value

In this query, we first create a Common Table Expression (CTE) named avg_freight to calculate the average freight value for each state by joining the customers, orders, and order_items tables based on their respective keys. Then, it use the ORDER BY clause to sort the results in descending order for the top 5 states with the highest average freight value, and in ascending order for the bottom 5 states with the lowest average freight value. Finally, we use the LIMIT clause to limit the results to 5 rows in each case.



Row	state	avg_freight
1	RR	42.9844230...
2	PB	42.7238039...
3	RO	41.0697122...
4	AC	40.0733695...
5	PI	39.1479704...



Row	state	avg_freight
1	SP	15.147275390419248
2	PR	20.531651567944248
3	MG	20.630166806306541
4	RJ	20.96092393168248
5	DF	21.041354945968383

Figure 5.9 Query Results: top 5 states with the highest average freight value and the bottom 5 states with the lowest average freight value

5.6 Top 5 states with highest/lowest average time to delivery

(**Figure 6 BigQuery: top 5 states with the highest average freight value and the bottom 5 states with the lowest average freight value**), this query uses two CTEs (Common Table Expressions). The first CTE **order_delivery_time** calculates the delivery time in days for each order by subtracting the **order_purchase_timestamp** from the **order_delivered_customer_date**. The second CTE **avg_delivery_time** calculates the average delivery time for each state by taking the average of delivery times grouped by state.

Finally, the outer query selects the states with the top 5 highest and lowest average delivery times by using ROW_NUMBER() function to rank the states based on average delivery time in ascending and descending order. The WHERE clause filters the states with ranks less than or equal to 5, and the results are ordered by rank in ascending and descending order separately.

The given data appears to be a table showing the average time to delivery in days for orders placed at Target in various states in Brazil. The results are sorted by the average time to delivery, with the top 5 states with the highest average time to delivery and the bottom 5 states with the lowest average time to delivery. Let's analyse the results:

5.6 a Top 5 states with highest average time to delivery:

- RR (Roraima): The state of Roraima has the highest average time to delivery at Target in Brazil, with an average of 28.98 days.
- AP (Amapá): Amapá is the second state with the highest average time to delivery at Target in Brazil, with an average of 26.73 days.
- AM (Amazonas): Amazonas is the third state with the highest average time to delivery at Target in Brazil, with an average of 25.99 days.
- AL (Alagoas): Alagoas is the fourth state with the highest average time to delivery at Target in Brazil, with an average of 24.04 days.

- PA (Pará): Pará is the fifth state with the highest average time to delivery at Target in Brazil, with an average of 23.32 days.

5.6b Lowest 5 states with lowest average time to delivery:

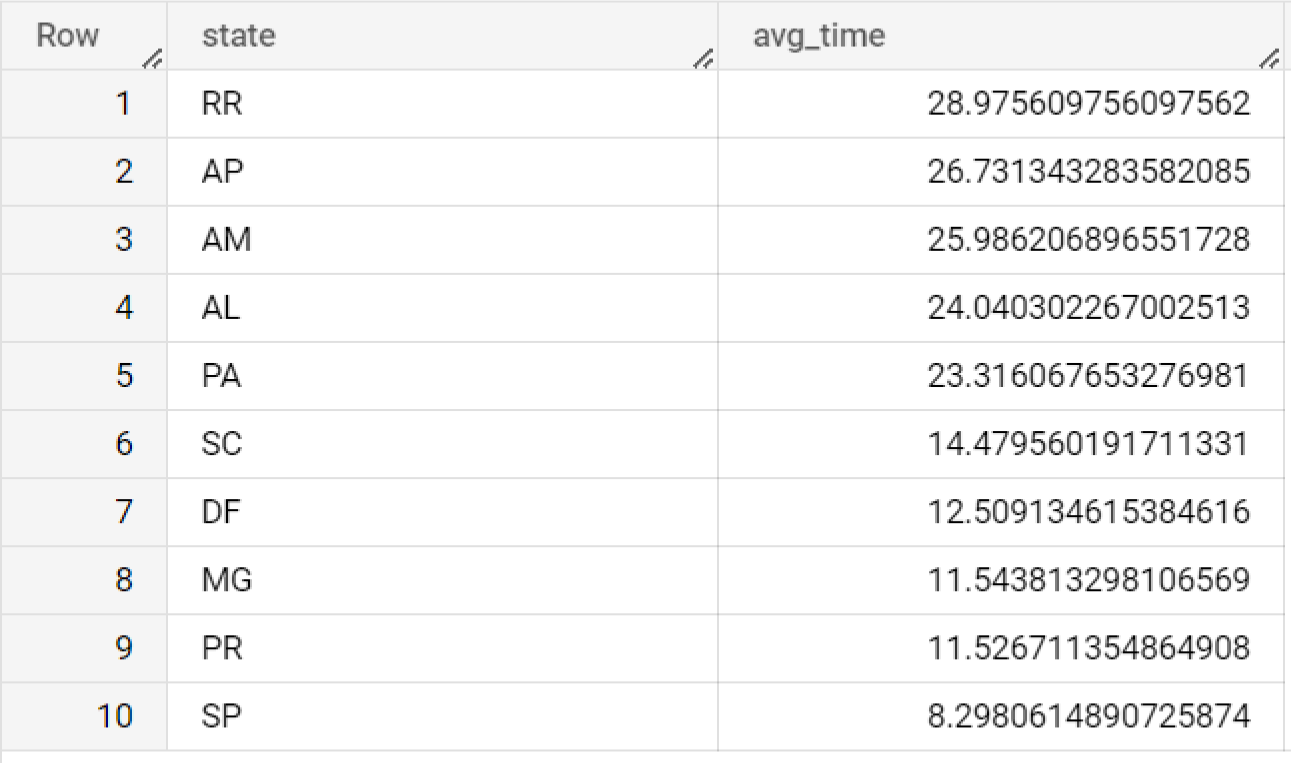
- SP (São Paulo): São Paulo is the state with the lowest average time to delivery at Target in Brazil, with an average of 8.30 days.
- PR (Paraná): Paraná is the second state with the lowest average time to delivery at Target in Brazil, with an average of 11.53 days.
- MG (Minas Gerais): Minas Gerais is the third state with the lowest average time to delivery at Target in Brazil, with an average of 11.54 days.
- DF (Distrito Federal): Distrito Federal is the fourth state with the lowest average time to delivery at Target in Brazil, with an average of 12.51 days.
- SC (Santa Catarina): Santa Catarina is the fifth state with the lowest average time to delivery at Target in Brazil, with an average of 14.48 days.

The results show the states in Brazil where Target has the highest and lowest average time to delivery for orders, which can provide insights into the efficiency of the delivery process in different regions of the country.

```
--
-- 6. Top 5 states with highest/lowest average time to delivery
WITH order_delivery_time AS (
 SELECT
  c.customer_state AS state,
  TIMESTAMP_DIFF(o.order_delivered_customer_date, o.order_purchase_timestamp, DAY) AS delivery_time
 FROM
  target_business.orders o
 JOIN
  target_business.customers c ON o.customer_id = c.customer_id
)
, avg_delivery_time AS (
 SELECT
  state,
  AVG(delivery_time) AS avg_time
 FROM
  order_delivery_time
 GROUP BY
  state
)
SELECT
 state,
 avg_time
FROM (
 SELECT
  state,
  avg_time,
  ROW_NUMBER() OVER (ORDER BY avg_time DESC) AS rank_desc,
  ROW_NUMBER() OVER (ORDER BY avg_time ASC) AS rank_asc
 FROM
  avg_delivery_time
)
```

```
WHERE
  rank_desc <= 5 OR rank_asc <= 5
ORDER BY
  rank_desc ASC,
  rank_asc ASC;
```

Figure 6 BigQuery: top 5 states with the highest average freight value and the bottom 5 states with the lowest average freight value



Row	state	avg_time
1	RR	28.975609756097562
2	AP	26.731343283582085
3	AM	25.986206896551728
4	AL	24.040302267002513
5	PA	23.316067653276981
6	SC	14.479560191711331
7	DF	12.509134615384616
8	MG	11.543813298106569
9	PR	11.526711354864908
10	SP	8.2980614890725874

Figure 6.1 Query results: top 5 states with the highest average freight value and the bottom 5 states with the lowest average freight value

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

Explanation of the query:

- Start by defining a CTE called **order_delivery** which retrieves relevant columns from the orders and customers tables, and joins them on the **customer_id** column.
- Next, in the main query, we select the **customer_state** column from the CTE as state, and use aggregate functions COUNT, SUM, and CASE statements to calculate the total number of orders, fast deliveries (where **order_delivered_customer_date** is less than or equal to **order_estimated_delivery_date**), and delayed deliveries (where **order_delivered_customer_date** is greater than **order_estimated_delivery_date**).
- Then group the results by state and order them by **fast_deliveries** in descending order.
- Finally, limit the results to the top 5 states with the fastest deliveries using the LIMIT clause.

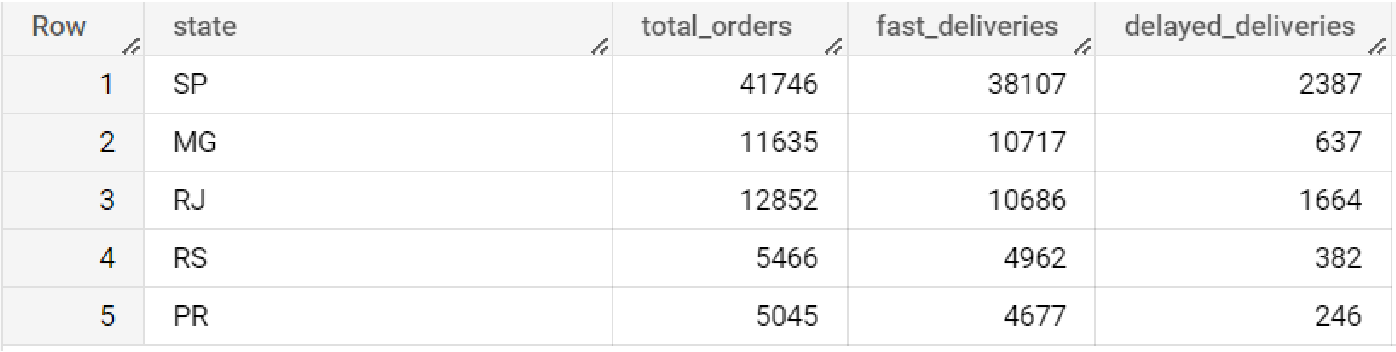
The given data appears to be a table showing the total number of orders, fast deliveries, and delayed deliveries in different states in Brazil. Let's analyse the results:

- SP (São Paulo): São Paulo has a total of 41,746 orders, out of which 38,107 are fast deliveries and 2,387 are delayed deliveries.
- MG (Minas Gerais): Minas Gerais has a total of 11,635 orders, out of which 10,717 are fast deliveries and 637 are delayed deliveries.
- RJ (Rio de Janeiro): Rio de Janeiro has a total of 12,852 orders, out of which 10,686 are fast deliveries and 1,664 are delayed deliveries.
- RS (Rio Grande do Sul): Rio Grande do Sul has a total of 5,466 orders, out of which 4,962 are fast deliveries and 382 are delayed deliveries.
- PR (Paraná): Paraná has a total of 5,045 orders, out of which 4,677 are fast deliveries and 246 are delayed deliveries.

The data provides information about the total number of orders, as well as the number of fast deliveries and delayed deliveries in different states in Brazil. This can be used to assess the performance of delivery services in these states and identify any potential issues or areas for improvement.

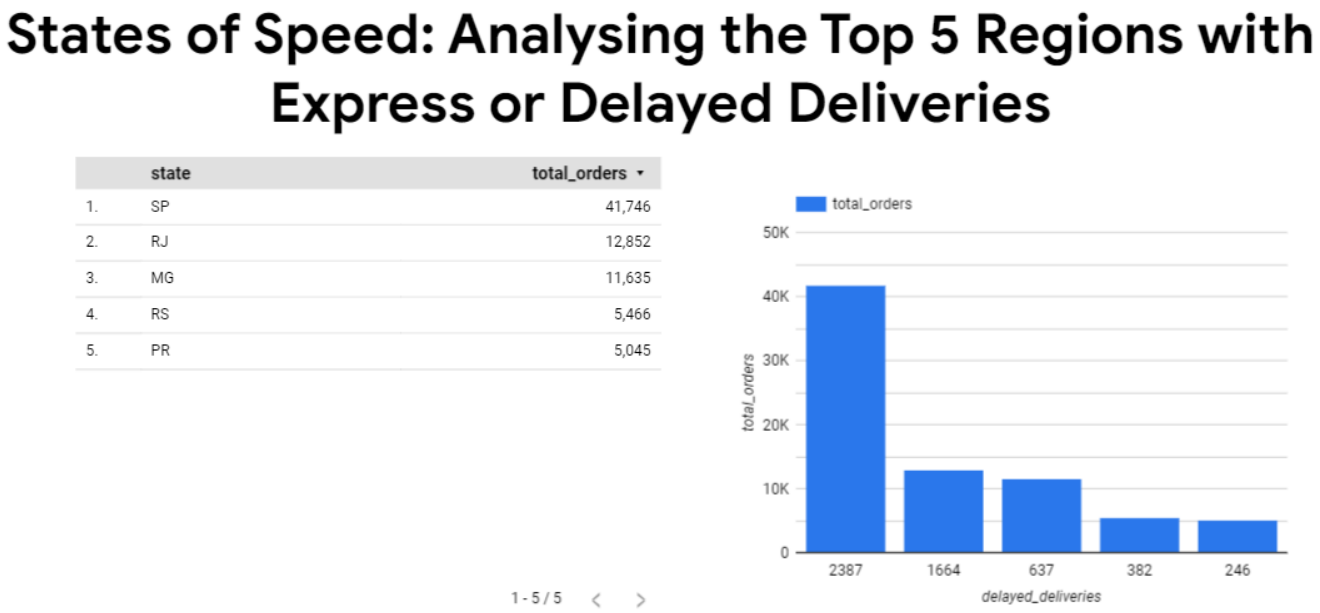
```sql
WITH order_delivery AS (
 SELECT
  o.order_id,
  o.order_status,
  o.order_purchase_timestamp,
  o.order_delivered_carrier_date,
  o.order_delivered_customer_date,
  o.order_estimated_delivery_date,
  c.customer_state AS state
 FROM
  target_business.orders o
 JOIN
  target_business.customers c
 ON
  o.customer_id = c.customer_id
)
SELECT
 state AS state,
 COUNT(*) AS total_orders,
 SUM(CASE
   WHEN order_status = 'delivered' AND order_delivered_customer_date <= order_estimated_delivery_date THEN 1
   ELSE 0
  END) AS fast_deliveries,
 SUM(CASE
   WHEN order_status = 'delivered' AND order_delivered_customer_date > order_estimated_delivery_date THEN 1
   ELSE 0
  END) AS delayed_deliveries
FROM
 order_delivery
GROUP BY
 state
ORDER BY
 fast_deliveries DESC
LIMIT
 5;
```

6.2 BigQuery: Top 5 states where delivery is really fast/ not so fast compared to estimated date



Row	state	total_orders	fast_deliveries	delayed_deliveries
1	SP	41746	38107	2387
2	MG	11635	10717	637
3	RJ	12852	10686	1664
4	RS	5466	4962	382
5	PR	5045	4677	246

6.3 Query results: Top 5 states where delivery is really fast/ not so fast compared to estimated date



States of Speed: Analysing the Top 5 Regions with Express or Delayed Deliveries

	state	total_orders ▾
1.	SP	41,746
2.	RJ	12,852
3.	MG	11,635
4.	RS	5,466
5.	PR	5,045

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

6. Payment type analysis:

6.1 Month over Month count of orders for different payment types

The given data appears to be a table showing the count of orders for different payment types at Target in Brazil on a month-over-month basis. Let's analyse the results:

May 2018: In May 2018, the following counts of orders were made for different payment types:

- UPI (Unified Payments Interface): 1,263 orders
- Credit card: 5,475 orders
- Debit card: 51 orders
- Voucher: 203 orders

June 2018: In June 2018, the following counts of orders were made for different payment types:

- UPI: 1,100 orders
- Credit card: 4,796 orders
- Debit card: 181 orders
- Voucher: 231 orders

July 2018: In July 2018, the following counts of orders were made for different payment types:

- UPI: 1,229 orders
- Credit card: 4,738 orders
- Debit card: 242 orders
- Voucher: 212 orders

August 2018: In August 2018, the following counts of orders were made for different payment types:

- UPI: 1,139 orders
- Credit card: 4,963 orders
- Debit card: 277 orders
- Not defined: 2 orders
- Voucher: 232 orders

September 2018: In September 2018, the following counts of orders were made for different payment types:

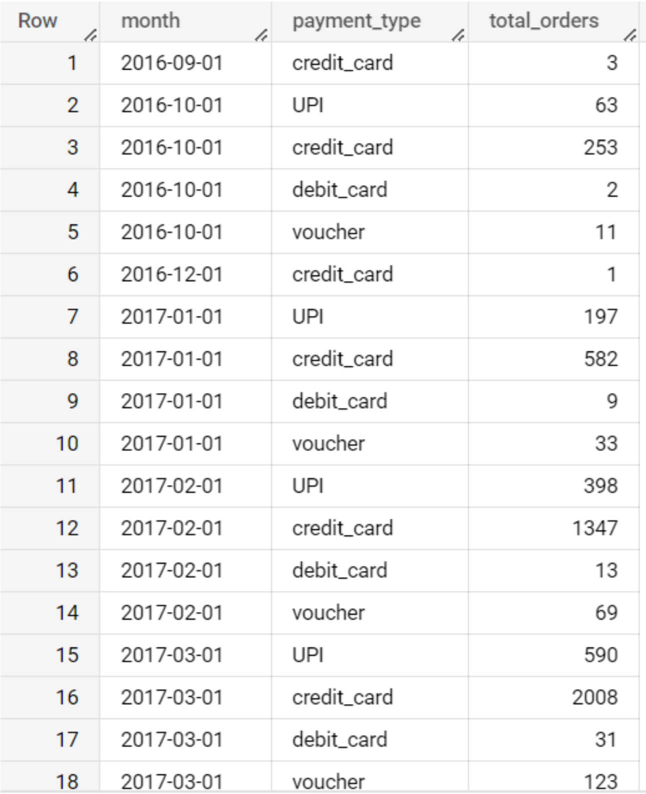
- Not defined: 1 order
- Voucher: 15 orders
- October 2018: In October 2018, the following counts of orders were made for different payment types:

- Voucher: 4 orders

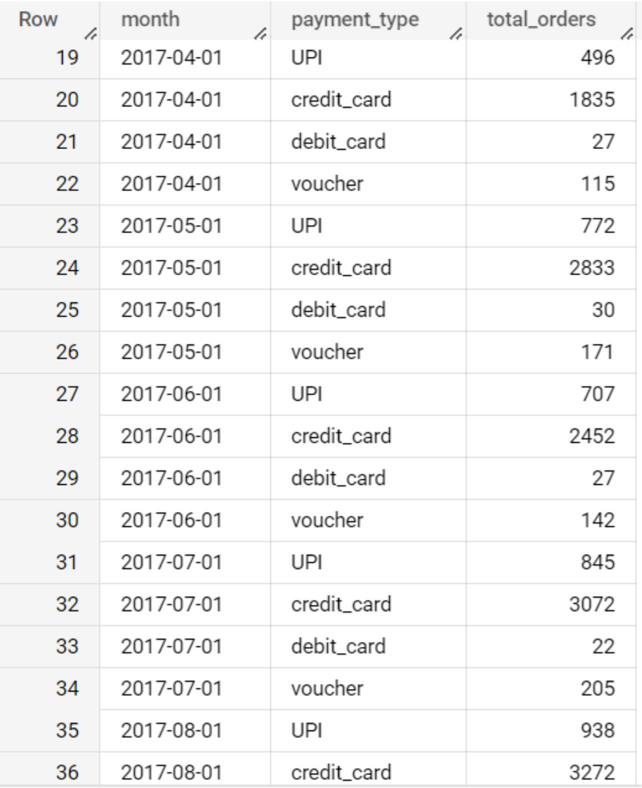
The data provides information about the count of orders for different payment types at Target in Brazil on a month-over-month basis. This can be used to track the trend of payment types used by customers over time and identify any changes or patterns in payment preferences.

```
-- Month over Month count of orders for different payment types
WITH monthly_orders AS (
 SELECT
  DATE_TRUNC(DATE(order_purchase_timestamp), MONTH) AS month,
  p.payment_type,
  COUNT(DISTINCT o.order_id) AS order_count
 FROM
  target_business.orders o
 INNER JOIN
  target_business.payments p ON o.order_id = p.order_id
 GROUP BY
  month,
  payment_type
)
SELECT
 month,
 payment_type,
 SUM(order_count) AS total_orders
FROM
 monthly_orders
GROUP BY
 month,
 payment_type
ORDER BY
 month,
 payment_type;
```

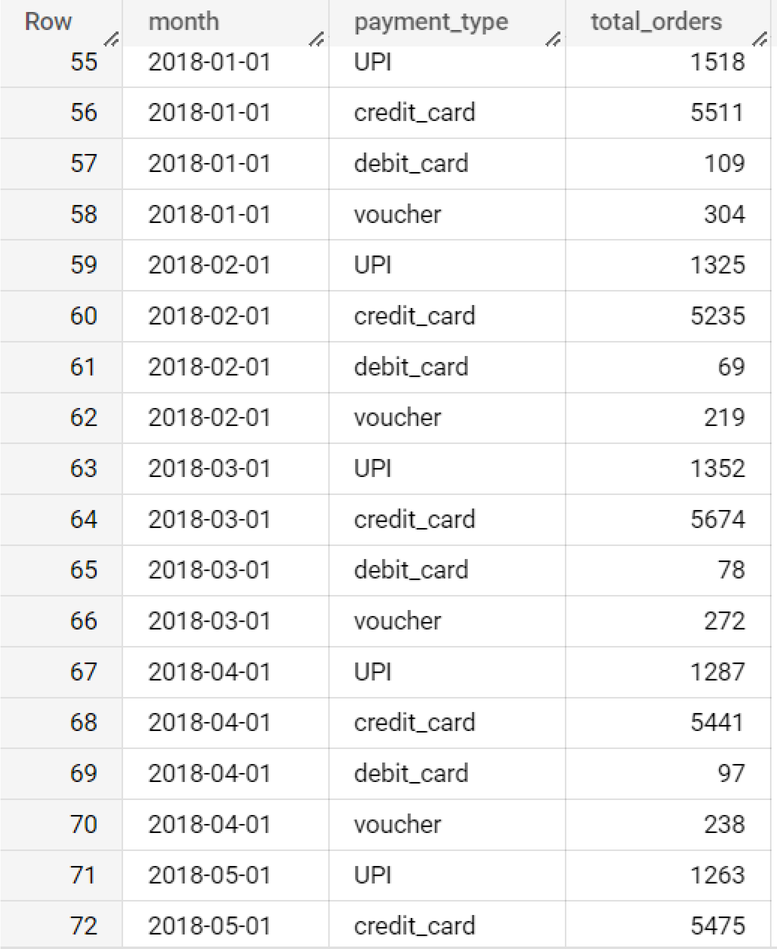
6.5 BigQuery: Month over Month count of orders for different payment types



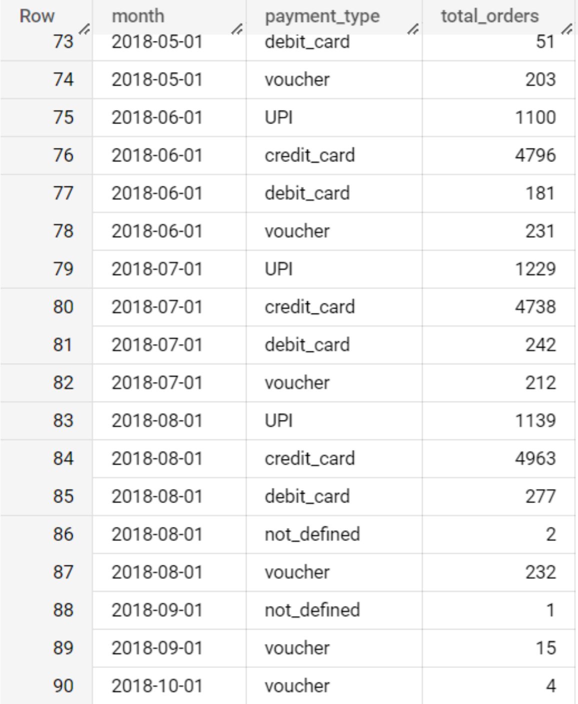
Row	month	payment_type	total_orders
1	2016-09-01	credit_card	3
2	2016-10-01	UPI	63
3	2016-10-01	credit_card	253
4	2016-10-01	debit_card	2
5	2016-10-01	voucher	11
6	2016-12-01	credit_card	1
7	2017-01-01	UPI	197
8	2017-01-01	credit_card	582
9	2017-01-01	debit_card	9
10	2017-01-01	voucher	33
11	2017-02-01	UPI	398
12	2017-02-01	credit_card	1347
13	2017-02-01	debit_card	13
14	2017-02-01	voucher	69
15	2017-03-01	UPI	590
16	2017-03-01	credit_card	2008
17	2017-03-01	debit_card	31
18	2017-03-01	voucher	123



Row	month	payment_type	total_orders
19	2017-04-01	UPI	496
20	2017-04-01	credit_card	1835
21	2017-04-01	debit_card	27
22	2017-04-01	voucher	115
23	2017-05-01	UPI	772
24	2017-05-01	credit_card	2833
25	2017-05-01	debit_card	30
26	2017-05-01	voucher	171
27	2017-06-01	UPI	707
28	2017-06-01	credit_card	2452
29	2017-06-01	debit_card	27
30	2017-06-01	voucher	142
31	2017-07-01	UPI	845
32	2017-07-01	credit_card	3072
33	2017-07-01	debit_card	22
34	2017-07-01	voucher	205
35	2017-08-01	UPI	938
36	2017-08-01	credit_card	3272

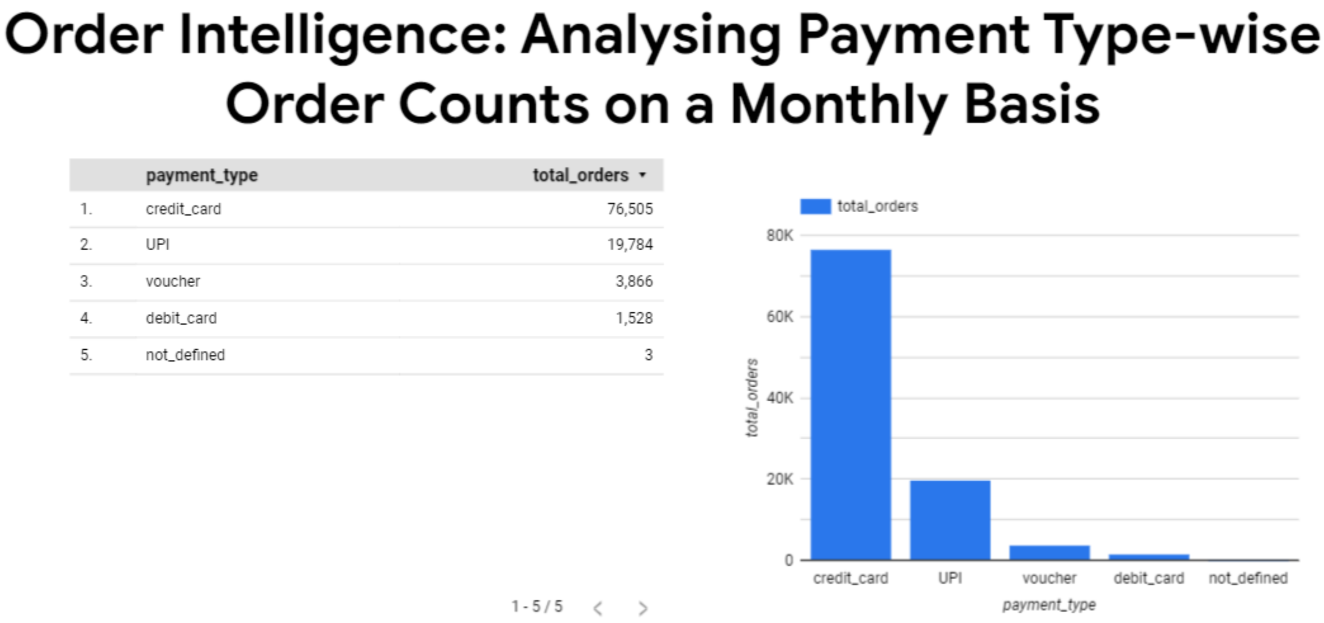


Row	month	payment_type	total_orders
55	2018-01-01	UPI	1518
56	2018-01-01	credit_card	5511
57	2018-01-01	debit_card	109
58	2018-01-01	voucher	304
59	2018-02-01	UPI	1325
60	2018-02-01	credit_card	5235
61	2018-02-01	debit_card	69
62	2018-02-01	voucher	219
63	2018-03-01	UPI	1352
64	2018-03-01	credit_card	5674
65	2018-03-01	debit_card	78
66	2018-03-01	voucher	272
67	2018-04-01	UPI	1287
68	2018-04-01	credit_card	5441
69	2018-04-01	debit_card	97
70	2018-04-01	voucher	238
71	2018-05-01	UPI	1263
72	2018-05-01	credit_card	5475



Row	month	payment_type	total_orders
73	2018-05-01	debit_card	51
74	2018-05-01	voucher	203
75	2018-06-01	UPI	1100
76	2018-06-01	credit_card	4796
77	2018-06-01	debit_card	181
78	2018-06-01	voucher	231
79	2018-07-01	UPI	1229
80	2018-07-01	credit_card	4738
81	2018-07-01	debit_card	242
82	2018-07-01	voucher	212
83	2018-08-01	UPI	1139
84	2018-08-01	credit_card	4963
85	2018-08-01	debit_card	277
86	2018-08-01	not_defined	2
87	2018-08-01	voucher	232
88	2018-09-01	not_defined	1
89	2018-09-01	voucher	15
90	2018-10-01	voucher	4

6.6 Query results: Month over Month count of orders for different payment types



Order Intelligence: Analysing Payment Type-wise Order Counts on a Monthly Basis

	payment_type	total_orders ▾
1.	credit_card	76,505
2.	UPI	19,784
3.	voucher	3,866
4.	debit_card	1,528
5.	not_defined	3

1 - 5 / 5 ‹ ›

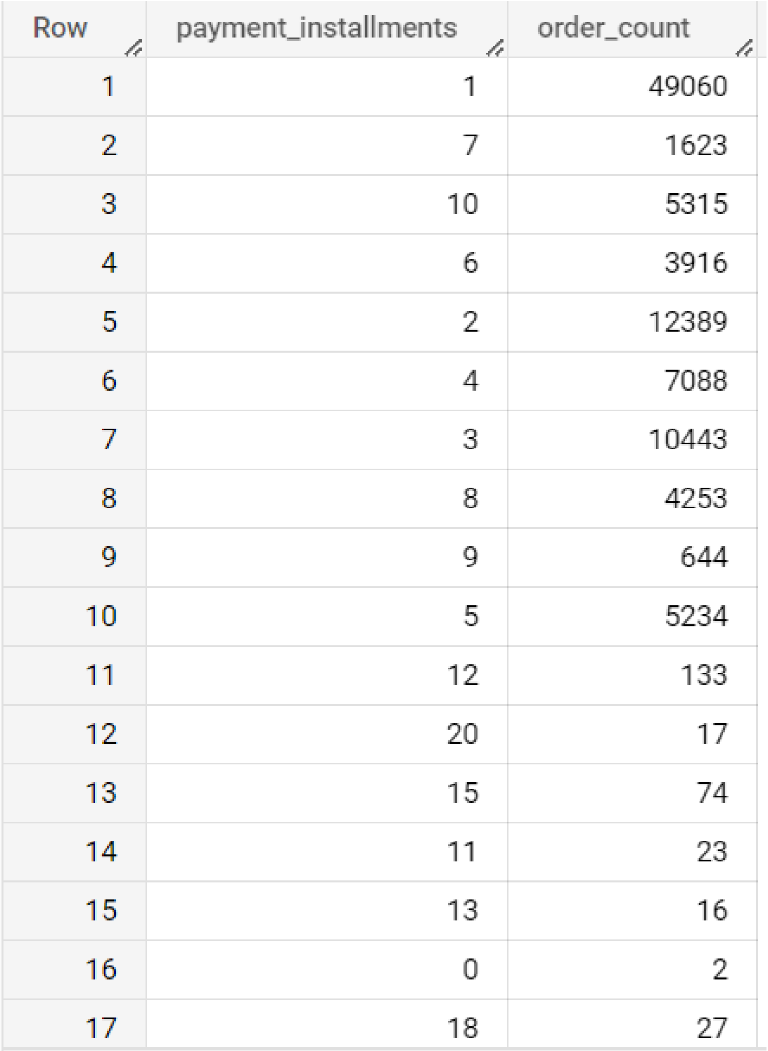
6.7 Month over Month count of orders for different payment types

6.2 Count of orders based on the no. of payment instalments

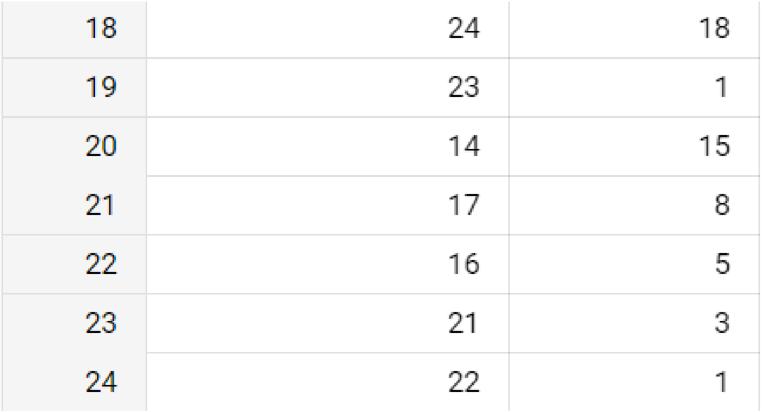
In this SQL, the **order_payments** CTE is created by joining the orders and payments tables and selecting the relevant columns. Then, the main query calculates the count of distinct **order_id** values for each **payment_installments** value from the **order_payments** CTE, using the COUNT function and grouping by **payment_installments**.

```
-- Count of orders based on the no. of payment instalments
WITH order_payments AS (
 SELECT
   o.order_id,
   p.payment_installments
 FROM
   target_business.orders o
 JOIN
   target_business.payments p ON o.order_id = p.order_id
)
SELECT
   payment_installments,
   COUNT(DISTINCT order_id) AS order_count
FROM
   order_payments
GROUP BY
   payment_installments;
```

Figure 6.8 BigQuery: Count of orders based on the no. of payment instalments

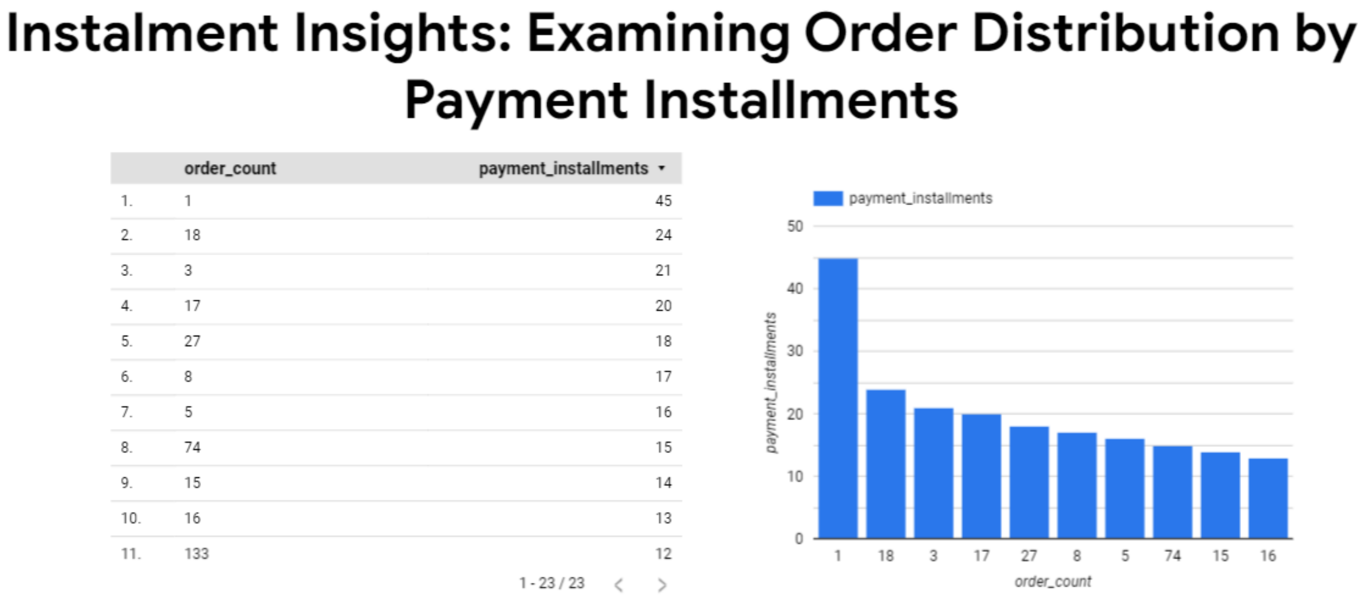


Row	payment_installments	order_count
1	1	49060
2	7	1623
3	10	5315
4	6	3916
5	2	12389
6	4	7088
7	3	10443
8	8	4253
9	9	644
10	5	5234
11	12	133
12	20	17
13	15	74
14	11	23
15	13	16
16	0	2
17	18	27



18	24	18
19	23	1
20	14	15
21	17	8
22	16	5
23	21	3
24	22	1

Figure 6.9 Query results: Count of orders based on the no. of payment instalments



Instalment Insights: Examining Order Distribution by Payment Installments

	order_count	payment_installments ▾
1.	1	45
2.	18	24
3.	3	21
4.	17	20
5.	27	18
6.	8	17
7.	5	16
8.	74	15
9.	15	14
10.	16	13
11.	133	12

1 - 23 / 23 < >

The given data appears to be a table showing the count of orders based on the number of payment installments at Target in Brazil. Let's analyse the results:

1) Payment Installments: 2
   a) Order Count: 12,389
2) Payment Installments: 4
   a) Order Count: 7,088
3) Payment Installments: 3
   a) Order Count: 10,443
4) Payment Installments: 8
   a) Order Count: 4,253
5) Payment Installments: 9
   a) Order Count: 644
6) Payment Installments: 5
   a) Order Count: 5,234
7) Payment Installments: 12

    a) Order Count: 133
8) Payment Installments: 20
    a) Order Count: 17
9) Payment Installments: 15
    a) Order Count: 74
10) Payment Installments: 11
    a) Order Count: 23
11) Payment Installments: 13
    a) Order Count: 16
12) Payment Installments: 0
    a) Order Count: 2
13) Payment Installments: 18
    a) Order Count: 27
14) Payment Installments: 24
    a) Order Count: 18
15) Payment Installments: 23
    a) Order Count: 1
16) Payment Installments: 14
    a) Order Count: 15
17) Payment Installments: 17
    a) Order Count: 8
18) Payment Installments: 16
    a) Order Count: 5
19) Payment Installments: 21
    a) Order Count: 3
20) Payment Installments: 22
    a) Order Count: 1

The data provides information about the count of orders based on the number of payment installments chosen by customers at Target in Brazil. This can be used to analyse the purchasing behaviour and preferences of customers in terms of payment options, and to tailor marketing and sales strategies accordingly.

Actionable Insights:

The given data appears to be a table showing the count of orders based on the number of payment installments at Target in Brazil. Let's analyse the results:

- Payment Installments: 2 - There are 12,389 orders with 2 payment installments.
- Payment Installments: 4 - There are 7,088 orders with 4 payment installments.
- Payment Installments: 3 - There are 10,443 orders with 3 payment installments.
- Payment Installments: 8 - There are 4,253 orders with 8 payment installments.
- Payment Installments: 9 - There are 644 orders with 9 payment installments.

- Payment Installments: 5 - There are 5,234 orders with 5 payment installments.
- Payment Installments: 12 - There are 133 orders with 12 payment installments.
- Payment Installments: 20 - There are 17 orders with 20 payment installments.
- Payment Installments: 15 - There are 74 orders with 15 payment installments.
- Payment Installments: 11 - There are 23 orders with 11 payment installments.
- Payment Installments: 13 - There are 16 orders with 13 payment installments.
- Payment Installments: 0 - There are 2 orders with 0 payment installments.
- Payment Installments: 18 - There are 27 orders with 18 payment installments.
- Payment Installments: 24 - There are 18 orders with 24 payment installments.
- Payment Installments: 23 - There is 1 order with 23 payment installments.
- Payment Installments: 14 - There are 15 orders with 14 payment installments.
- Payment Installments: 17 - There are 8 orders with 17 payment installments.
- Payment Installments: 16 - There are 5 orders with 16 payment installments.
- Payment Installments: 21 - There are 3 orders with 21 payment installments.
- Payment Installments: 22 - There is 1 order with 22 payment installments.

7. Actionable Insights

(In section 6 payment installments), **the given data appears to be a table showing the count of orders based on the number of payment installments at Target in Brazil. The results:**

- Payment Installments: 2 - There are a total of 12,389 orders with 2 payment installments.
- Payment Installments: 4 - There are a total of 7,088 orders with 4 payment installments.
- Payment Installments: 3 - There are a total of 10,443 orders with 3 payment installments.
- Payment Installments: 8 - There are a total of 4,253 orders with 8 payment installments.
- Payment Installments: 9 - There are a total of 644 orders with 9 payment installments.
- Payment Installments: 5 - There are a total of 5,234 orders with 5 payment installments.
- Payment Installments: 12 - There are a total of 133 orders with 12 payment installments.
- Payment Installments: 20 - There are a total of 17 orders with 20 payment installments.
- Payment Installments: 15 - There are a total of 74 orders with 15 payment installments.
- Payment Installments: 11 - There are a total of 23 orders with 11 payment installments.
- Payment Installments: 13 - There are a total of 16 orders with 13 payment installments.
- Payment Installments: 0 - There are a total of 2 orders with 0 payment installments.
- Payment Installments: 18 - There are a total of 27 orders with 18 payment installments.
- Payment Installments: 24 - There are a total of 18 orders with 24 payment installments.
- Payment Installments: 23 - There is only 1 order with 23 payment installments.
- Payment Installments: 14 - There are a total of 15 orders with 14 payment installments.
- Payment Installments: 17 - There are a total of 8 orders with 17 payment installments.
- Payment Installments: 16 - There are a total of 5 orders with 16 payment installments.
- Payment Installments: 21 - There are a total of 3 orders with 21 payment installments.
- Payment Installments: 22 - There is only 1 order with 22 payment installments.

Actionable Insights for payment:

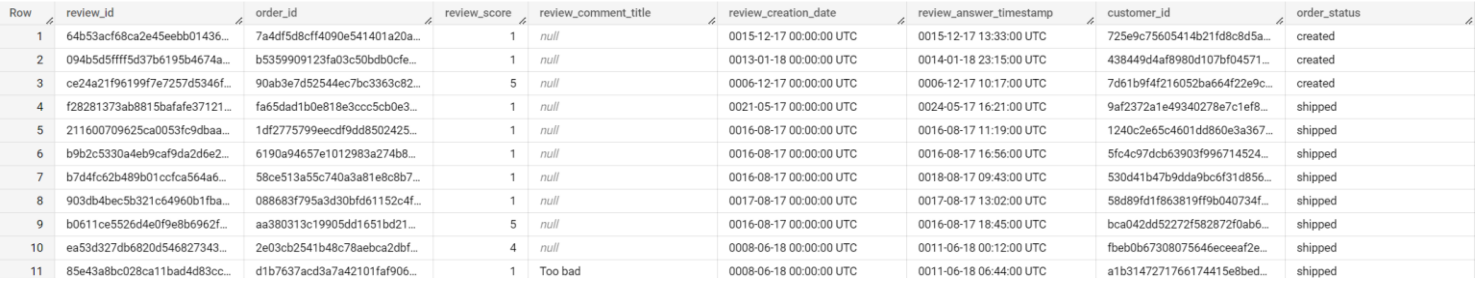
- Most orders have payment installments ranging from 2 to 5, indicating that customers prefer to divide their payments into smaller installments.
- There are a significant number of orders with 0 payment installments, which could indicate that customers are choosing to pay for their orders in full at the time of purchase.
- There are relatively fewer orders with higher payment installments (e.g., 20, 23, 24), indicating that customers may prefer to avoid longer payment plans.
- Target in Brazil may consider analysing the payment installment options and preferences of their customers to optimize their payment offerings and attract more customers.
- Target in Brazil may also consider offering flexible payment plans or promotions to encourage customers to choose higher payment installments and potentially increase order volume.

Overall, the data on payment installments can help Target in Brazil make informed decisions about their payment options, pricing strategies, and marketing efforts

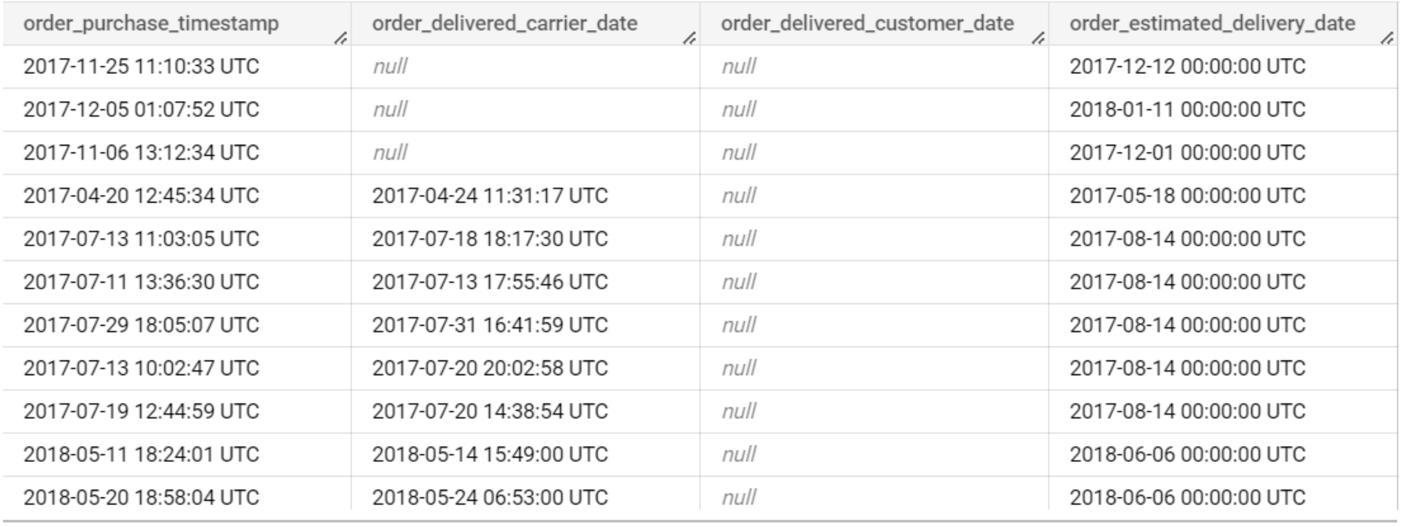
7.1 Analysing Customer Sentiment:

Natural language processing can be used to build predictive models to perform sentiment analysis on social media posts and reviews and predict if customers are happy or not. That way, you can automatically know if your customers are happy or not without manually going through massive number of reviews.

```
---
    WITH review_orders AS (
  SELECT
    r.review_id,
    r.order_id,
    r.review_score,
    r.review_comment_title,
    --r.review_comment_message,
    r.review_creation_date,
    r.review_answer_timestamp,
    o.customer_id,
    o.order_status,
    o.order_purchase_timestamp,
    o.order_delivered_carrier_date,
    o.order_delivered_customer_date,
    o.order_estimated_delivery_date
  FROM
    target_business.order_reviews AS r
  JOIN
    target_business.orders AS o
  ON
    r.order_id = o.order_id
)
SELECT
  ro.review_id,
  ro.order_id,
  ro.review_score,
  ro.review_comment_title,
  --ro.review_comment_message,
  ro.review_creation_date,
  ro.review_answer_timestamp,
  ro.customer_id,
  ro.order_status,
  ro.order_purchase_timestamp,
  ro.order_delivered_carrier_date,
  ro.order_delivered_customer_date,
  ro.order_estimated_delivery_date
FROM
  review_orders AS ro;
```

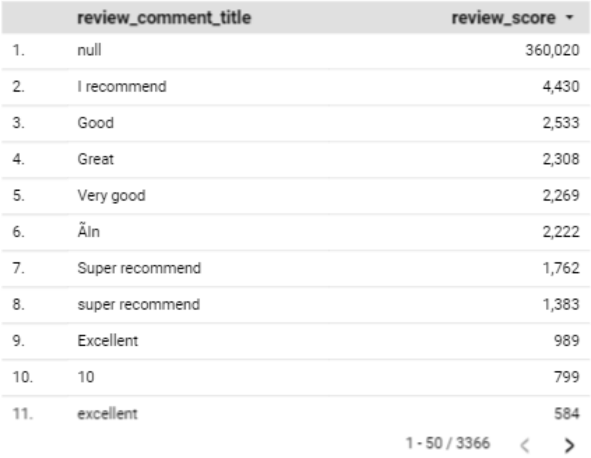


Row	review_id	order_id	review_score	review_comment_title	review_creation_date	review_answer_timestamp	customer_id	order_status
1	64b53acf68ca2e45eebb01436...	7a4df5d8cff4090e541401a20a...	1	null	0015-12-17 00:00:00 UTC	0015-12-17 13:33:00 UTC	725e9c75605414b21fd8c8d5a...	created
2	094b5d5ffff5d37b6195b4674a...	b5359909123fa03c50bdb0cfe...	1	null	0013-01-18 00:00:00 UTC	0014-01-18 23:15:00 UTC	438449d4af8980d107bf04571...	created
3	ce24a21f96199f7e7257d5346f...	90ab3e7d52544ec7bc3363c82...	5	null	0006-12-17 00:00:00 UTC	0006-12-17 10:17:00 UTC	7d61b9f4f216052ba664f22e9c...	created
4	f28281373ab8815bafafe37121...	fa65dad1b0e818e3ccc5cb0e3...	1	null	0021-05-17 00:00:00 UTC	0024-05-17 16:21:00 UTC	9af2372a1e49340278e7c1ef8...	shipped
5	211600709625ca0053fc9dbaa...	1df2775799eecdf9dd8502425...	1	null	0016-08-17 00:00:00 UTC	0016-08-17 11:19:00 UTC	1240c2e65c4601dd860e3a367...	shipped
6	b9b2c5330a4eb9caf9da2d6e2...	6190a94657e1012983a274b8...	1	null	0016-08-17 00:00:00 UTC	0016-08-17 16:56:00 UTC	5fc4c97dcb63903f996714524...	shipped
7	b7d4fc62b489b01ccfca564a6...	58ce513a55c740a3a81e8c8b7...	1	null	0016-08-17 00:00:00 UTC	0018-08-17 09:43:00 UTC	530d41b47b9dda9bc6f31d856...	shipped
8	903db4bec5b321c64960b1fba...	088683f795a3d30bfd61152c4f...	1	null	0017-08-17 00:00:00 UTC	0017-08-17 13:02:00 UTC	58d89fd1f863819ff9b040734f...	shipped
9	b0611ce5526d4e0f9e8b6962f...	aa380313c19905dd1651bd21...	5	null	0016-08-17 00:00:00 UTC	0016-08-17 18:45:00 UTC	bca042dd52272f582872f0ab6...	shipped
10	ea53d327db6820d546827343...	2e03cb2541b48c78aebca2dbf...	4	null	0008-06-18 00:00:00 UTC	0011-06-18 00:12:00 UTC	fbeb0b67308075646eceeaf2e...	shipped
11	85e43a8bc028ca11bad4d83cc...	d1b7637acd3a7a42101faf906...	1	Too bad	0008-06-18 00:00:00 UTC	0011-06-18 06:44:00 UTC	a1b3147271766174415e8bed...	shipped



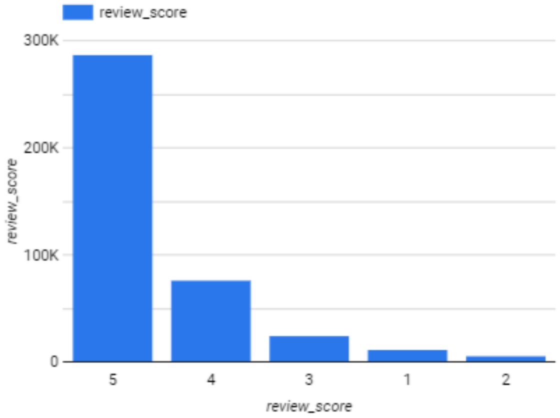
order_purchase_timestamp	order_delivered_carrier_date	order_delivered_customer_date	order_estimated_delivery_date
2017-11-25 11:10:33 UTC	*null*	*null*	2017-12-12 00:00:00 UTC
2017-12-05 01:07:52 UTC	*null*	*null*	2018-01-11 00:00:00 UTC
2017-11-06 13:12:34 UTC	*null*	*null*	2017-12-01 00:00:00 UTC
2017-04-20 12:45:34 UTC	2017-04-24 11:31:17 UTC	*null*	2017-05-18 00:00:00 UTC
2017-07-13 11:03:05 UTC	2017-07-18 18:17:30 UTC	*null*	2017-08-14 00:00:00 UTC
2017-07-11 13:36:30 UTC	2017-07-13 17:55:46 UTC	*null*	2017-08-14 00:00:00 UTC
2017-07-29 18:05:07 UTC	2017-07-31 16:41:59 UTC	*null*	2017-08-14 00:00:00 UTC
2017-07-13 10:02:47 UTC	2017-07-20 20:02:58 UTC	*null*	2017-08-14 00:00:00 UTC
2017-07-19 12:44:59 UTC	2017-07-20 14:38:54 UTC	*null*	2017-08-14 00:00:00 UTC
2018-05-11 18:24:01 UTC	2018-05-14 15:49:00 UTC	*null*	2018-06-06 00:00:00 UTC
2018-05-20 18:58:04 UTC	2018-05-24 06:53:00 UTC	*null*	2018-06-06 00:00:00 UTC

Targeting Excellence: A Comprehensive Analysis of Review Scores and Comments

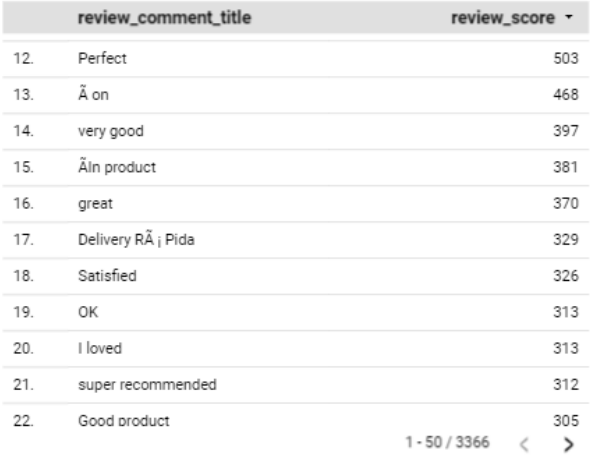


	review_comment_title	review_score ▾
1.	null	360,020
2.	I recommend	4,430
3.	Good	2,533
4.	Great	2,308
5.	Very good	2,269
6.	Ãln	2,222
7.	Super recommend	1,762
8.	super recommend	1,383
9.	Excellent	989
10.	10	799
11.	excellent	584

1 - 50 / 3366 < >

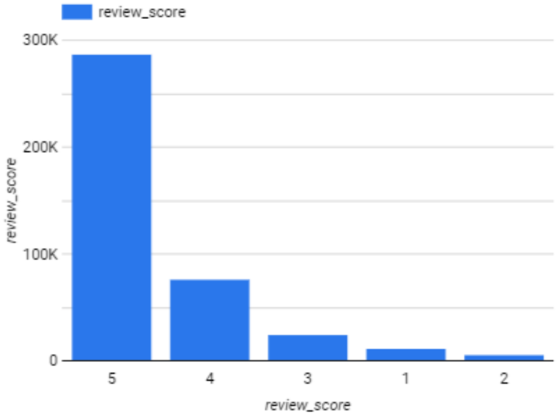


Targeting Excellence: A Comprehensive Analysis of Review Scores and Comments



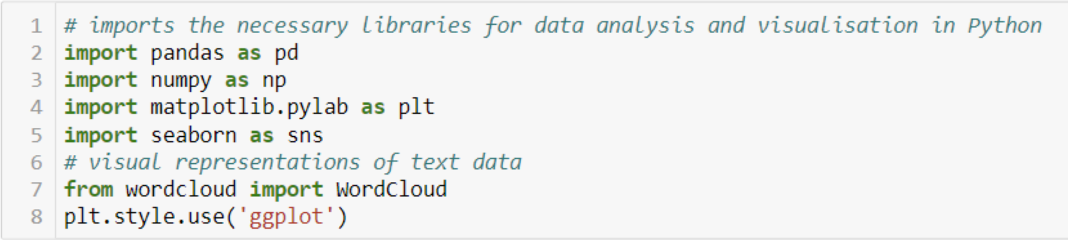
	review_comment_title	review_score ▾
12.	Perfect	503
13.	Ã on	468
14.	very good	397
15.	Ãln product	381
16.	great	370
17.	Delivery RÃ ¡ Pida	329
18.	Satisfied	326
19.	OK	313
20.	I loved	313
21.	super recommended	312
22.	Good product	305

1 - 50 / 3366 < >

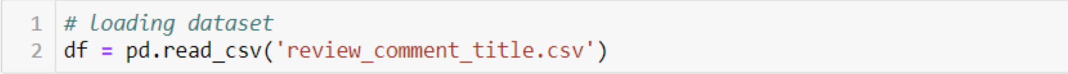


very often it is easier to perform analysis using SQL or BigQuery on data we have right in the tables and then move forward to ML/AI/Data science and engineering in Python.

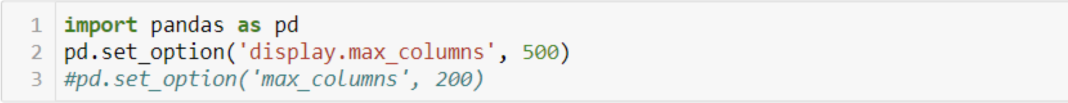
```
- with an additional import statement for WordCloud from the wordcloud library. The WordCloud class is used for generating word clouds, which are visual representations of text data where the size of each word represents its frequency or importance in the text.
  SELECT
    review_comment_title
  FROM
    target_business.order_reviews
  ORDER BY
    review_comment_title DESC;
```



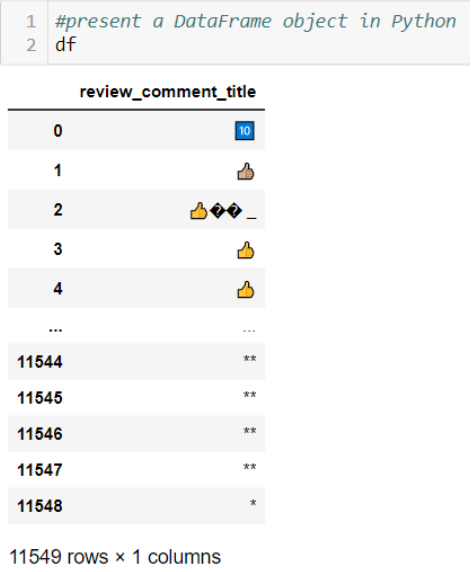
```python
# imports the necessary libraries for data analysis and visualisation in Python
import pandas as pd
import numpy as np
import matplotlib.pylab as plt
import seaborn as sns
# visual representations of text data
from wordcloud import WordCloud
plt.style.use('ggplot')
```



```python
# loading dataset
df = pd.read_csv('review_comment_title.csv')
```



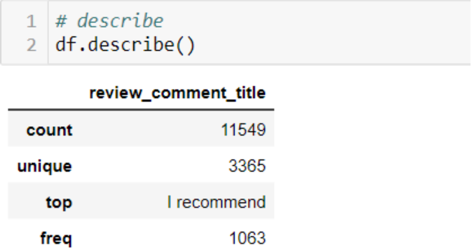
```python
import pandas as pd
pd.set_option('display.max_columns', 500)
#pd.set_option('max_columns', 200)
```



```python
#present a DataFrame object in Python
df
```

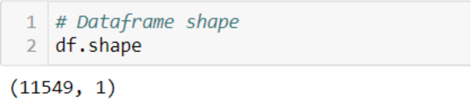
	review_comment_title
0	🔟
1	👍
2	👍��_
3	👍
4	👍
...	...
11544	**
11545	**
11546	**
11547	**
11548	*

11549 rows × 1 columns



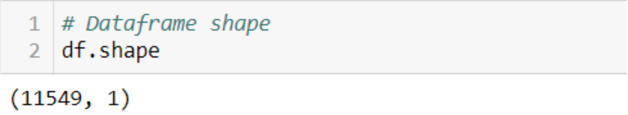
```python
# describe
df.describe()
```

	review_comment_title
count	11549
unique	3365
top	I recommend
freq	1063



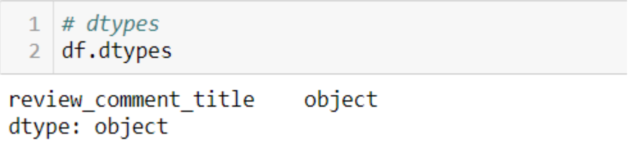
```python
# Dataframe shape
df.shape
```

```
(11549, 1)
```



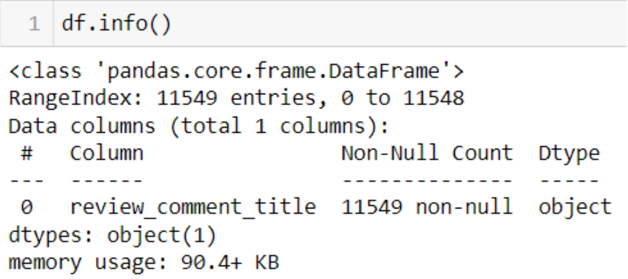
```python
# Dataframe shape
df.shape
```

```
(11549, 1)
```



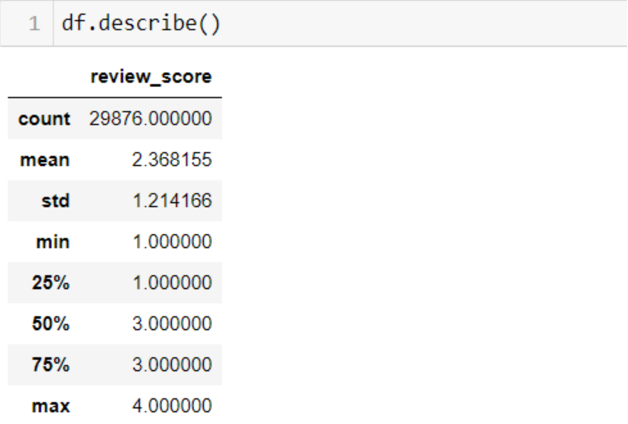
```python
# dtypes
df.dtypes
```

```
review_comment_title    object
dtype: object
```



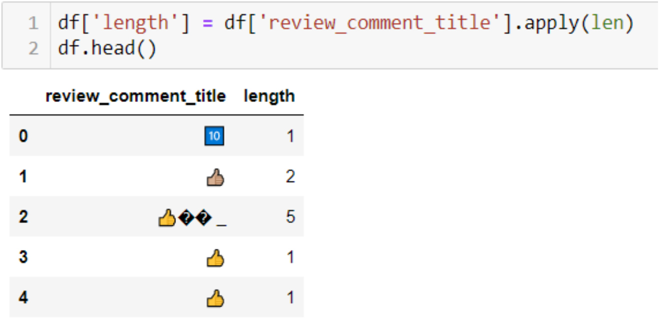
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11549 entries, 0 to 11548
Data columns (total 1 columns):
 #   Column                Non-Null Count  Dtype 
---  ------                --------------  ----- 
 0   review_comment_title  11549 non-null  object
dtypes: object(1)
memory usage: 90.4+ KB
```



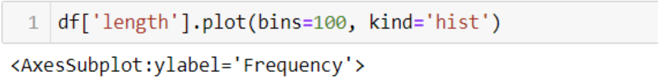
```python
df.describe()
```

	review_score
count	29876.000000
mean	2.368155
std	1.214166
min	1.000000
25%	1.000000
50%	3.000000
75%	3.000000
max	4.000000



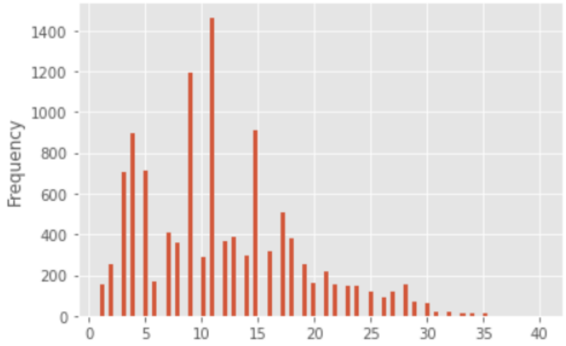
```
df['length'] = df['review_comment_title'].apply(len)
df.head()
```

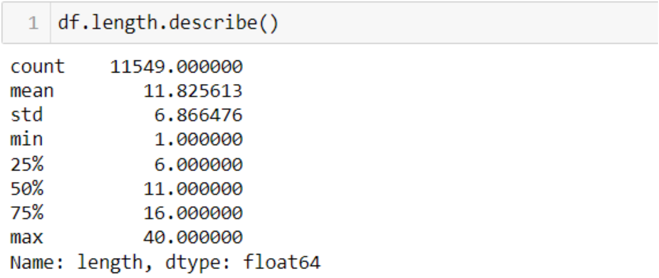
	review_comment_title	length
0	🔟	1
1	👍	2
2	👍��_	5
3	👍	1
4	👍	1



```
df['length'].plot(bins=100, kind='hist')
```

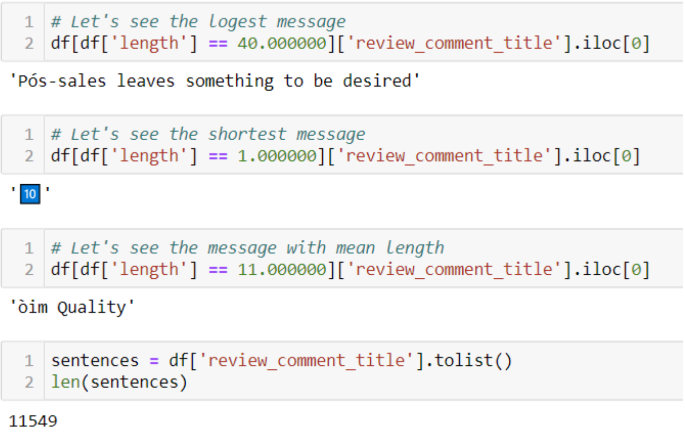
```
<AxesSubplot:ylabel='Frequency'>
```





```
df.length.describe()
```

```
count    11549.000000
mean        11.825613
std          6.866476
min          1.000000
25%          6.000000
50%         11.000000
75%         16.000000
max         40.000000
Name: length, dtype: float64
```



```
# Let's see the logest message
df[df['length'] == 40.000000]['review_comment_title'].iloc[0]
```

```
'Pós-sales leaves something to be desired'
```

```
# Let's see the shortest message
df[df['length'] == 1.000000]['review_comment_title'].iloc[0]
```

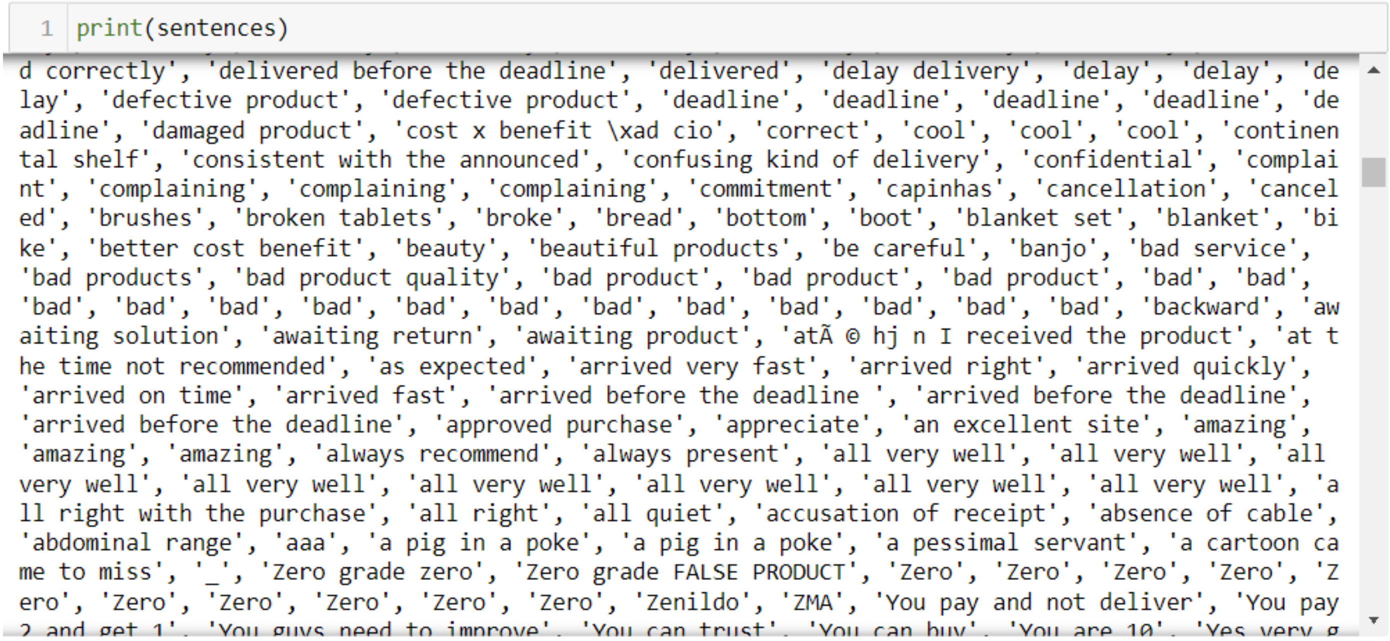
```
'🔟'
```

```
# Let's see the message with mean length
df[df['length'] == 11.000000]['review_comment_title'].iloc[0]
```

```
'òim Quality'
```

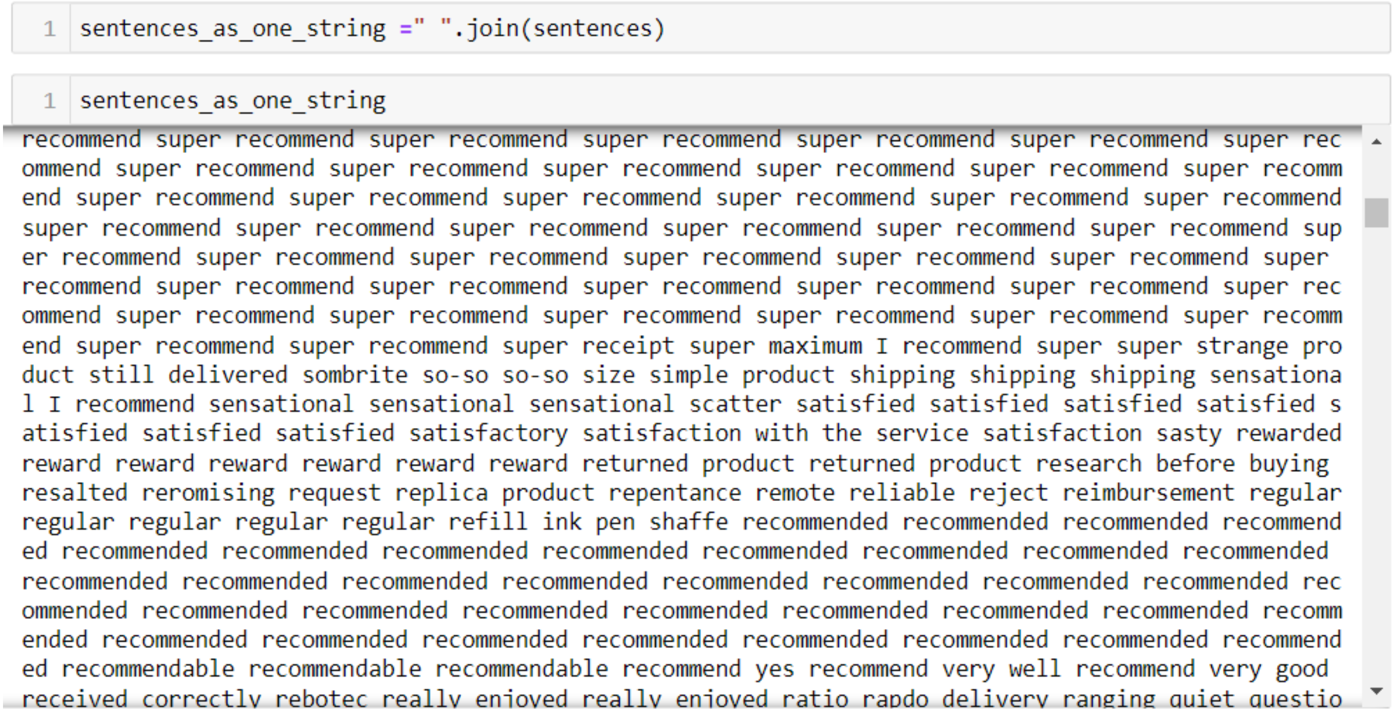
```
sentences = df['review_comment_title'].tolist()
len(sentences)
```

```
11549
```



```
print(sentences)
```

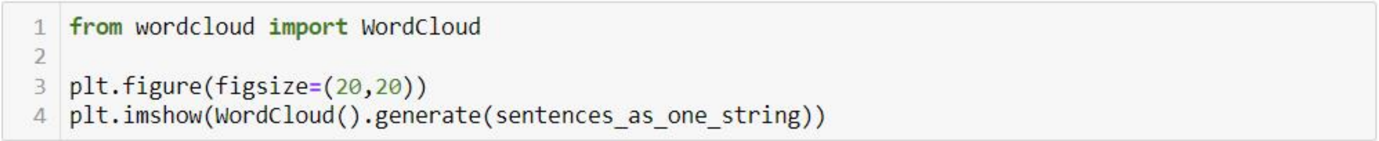
```
d correctly', 'delivered before the deadline', 'delivered', 'delay delivery', 'delay', 'delay', 'de
lay', 'defective product', 'defective product', 'deadline', 'deadline', 'deadline', 'deadline', 'de
adline', 'damaged product', 'cost x benefit \xad cio', 'correct', 'cool', 'cool', 'cool', 'continen
tal shelf', 'consistent with the announced', 'confusing kind of delivery', 'confidential', 'complai
nt', 'complaining', 'complaining', 'complaining', 'commitment', 'capinhas', 'cancellation', 'cancel
ed', 'brushes', 'broken tablets', 'broke', 'bread', 'bottom', 'boot', 'blanket set', 'blanket', 'bi
ke', 'better cost benefit', 'beauty', 'beautiful products', 'be careful', 'banjo', 'bad service',
'bad products', 'bad product quality', 'bad product', 'bad product', 'bad product', 'bad', 'bad',
'bad', 'bad', 'bad', 'bad', 'bad', 'bad', 'bad', 'bad', 'bad', 'bad', 'bad', 'bad', 'backward', 'aw
aiting solution', 'awaiting return', 'awaiting product', 'atÃ © hj n I received the product', 'at t
he time not recommended', 'as expected', 'arrived very fast', 'arrived right', 'arrived quickly',
'arrived on time', 'arrived fast', 'arrived before the deadline ', 'arrived before the deadline',
'arrived before the deadline', 'approved purchase', 'appreciate', 'an excellent site', 'amazing',
'amazing', 'amazing', 'always recommend', 'always present', 'all very well', 'all very well', 'all
very well', 'all very well', 'all very well', 'all very well', 'all very well', 'all very well', 'a
ll right with the purchase', 'all right', 'all quiet', 'accusation of receipt', 'absence of cable',
'abdominal range', 'aaa', 'a pig in a poke', 'a pig in a poke', 'a pessimal servant', 'a cartoon ca
me to miss', '_', 'Zero grade zero', 'Zero grade FALSE PRODUCT', 'Zero', 'Zero', 'Zero', 'Zero', 'Z
ero', 'Zero', 'Zero', 'Zero', 'Zero', 'Zero', 'Zenildo', 'ZMA', 'You pay and not deliver', 'You pay
2 and get 1', 'You guys need to improve', 'You can trust', 'You can buy', 'You are 10', 'Yes very g
```



```python
sentences_as_one_string =" ".join(sentences)
```

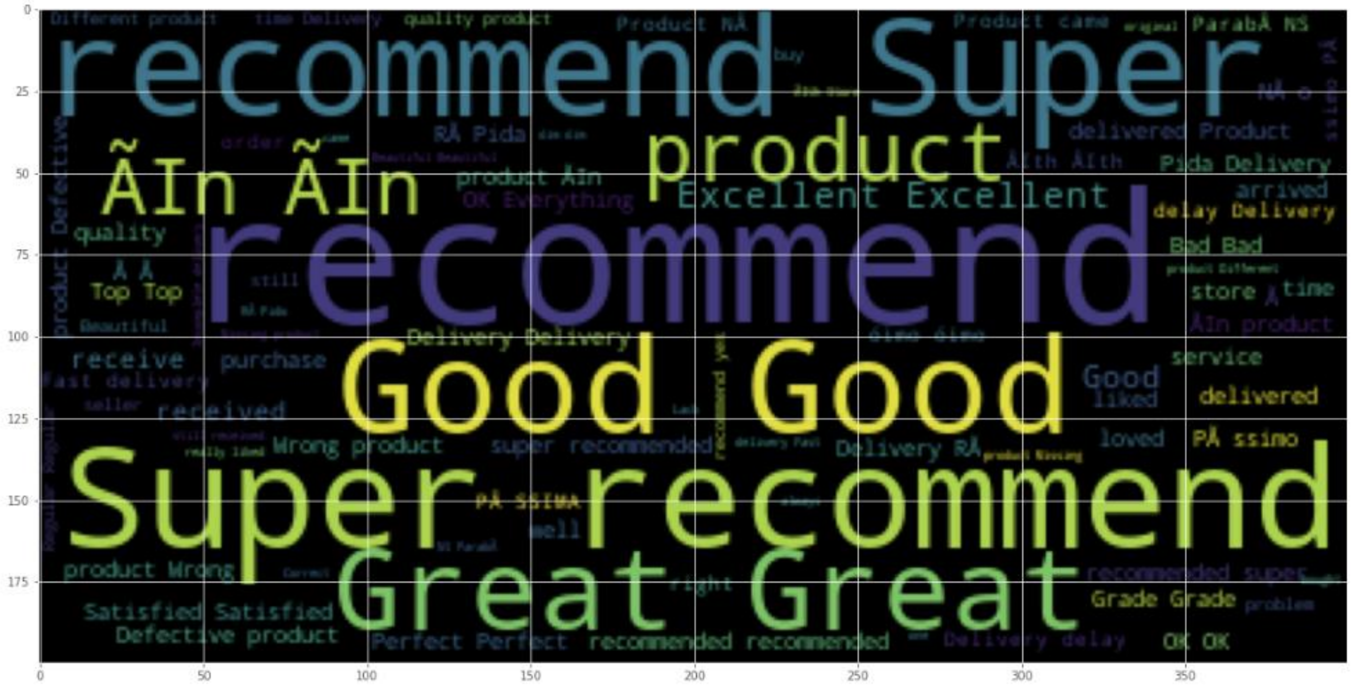
```python
sentences_as_one_string
```

```
recommend super recommend super recommend super recommend super recommend super recommend super rec
ommend super recommend super recommend super recommend super recommend super recommend super recomm
end super recommend super recommend super recommend super recommend super recommend super recommend
super recommend super recommend super recommend super recommend super recommend super recommend sup
er recommend super recommend super recommend super recommend super recommend super recommend super
recommend super recommend super recommend super recommend super recommend super recommend super rec
ommend super recommend super recommend super recommend super recommend super recommend super recomm
end super recommend super recommend super receipt super maximum I recommend super super strange pro
duct still delivered sombrite so-so so-so size simple product shipping shipping shipping sensationa
l I recommend sensational sensational sensational scatter satisfied satisfied satisfied satisfied s
atisfied satisfied satisfied satisfactory satisfaction with the service satisfaction sasty rewarded
reward reward reward reward reward reward returned product returned product research before buying
resalted reromising request replica product repentance remote reliable reject reimbursement regular
regular regular regular regular refill ink pen shaffe recommended recommended recommended recommend
ed recommended recommended recommended recommended recommended recommended recommended recommended
recommended recommended recommended recommended recommended recommended recommended recommended rec
ommended recommended recommended recommended recommended recommended recommended recommended recomm
ended recommended recommended recommended recommended recommended recommended recommended recommend
ed recommendable recommendable recommendable recommend yes recommend very well recommend very good
received correctly rebotec really enjoyed really enjoyed ratio rapdo delivery ranging quiet questio
```

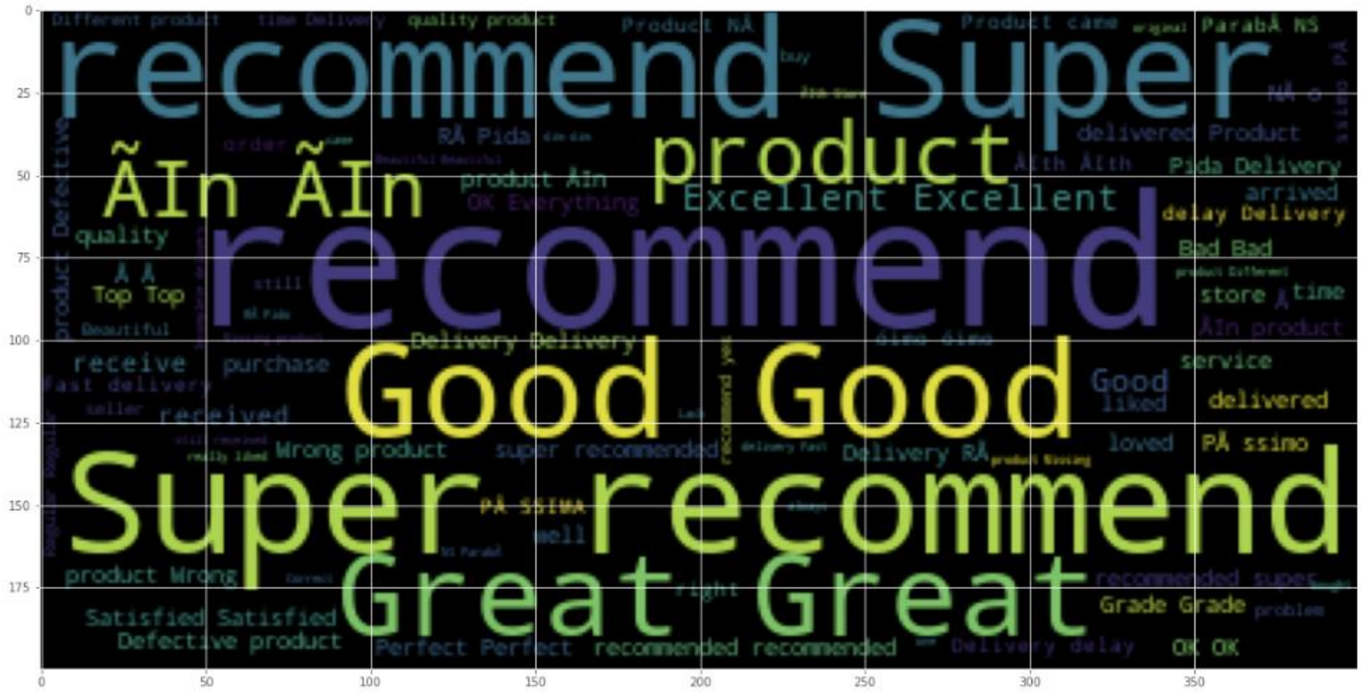


```python
from wordcloud import WordCloud

plt.figure(figsize=(20,20))
plt.imshow(WordCloud().generate(sentences_as_one_string))
```

```
<matplotlib.image.AxesImage at 0x1b3a3d2bb50>
```



```
<matplotlib.image.AxesImage at 0x1b3a3d2bb50>
```



7.2 Actionable Insights

Insights and Recommendations (Figure 1.6 DISTINCT Cities and States of customers ordered during the given period: 4259):

- Geographic Distribution: The customers who made orders during the given period are located in various cities and states in Brazil. Some of the states with a higher number of orders are AP (Amapá), BA (Bahia), and AM (Amazonas). This indicates that Target has a wide reach and customer base across different regions in Brazil.
- Targeting High-Demand States: Based on the data, it appears that Amapá (AP), Bahia (BA), and Amazonas (AM) are the states with higher customer orders. Target can focus on these states to further expand their customer base, increase marketing efforts, and tailor their product offerings to the preferences and demands of customers in these states.
- Understanding Customer Preferences: Analysing the cities and states of customers who made orders can provide insights into their preferences and buying behaviour. Target can use this information to better understand customer needs, preferences, and shopping patterns in different regions. This can help in developing targeted marketing strategies and promotions to cater to the specific preferences of customers in different cities and states.
- Localisation Strategies: Target can leverage the data on cities and states of customers to implement localization strategies. This can include customizing product offerings, pricing, and promotions based on the preferences and demands of customers in different cities and states. For example, offering region-specific products or promotions during local festivals or events can help Target to better connect with customers and increase sales.

Overall, analysing the cities and states of customers who made orders during the given period can provide valuable insights.

8. Recommendations

- Improve Delivery and Shipping Strategies: Analysing the cities and states of customers can also provide insights into the logistics and shipping requirements for different regions. Target can use this information to optimise their delivery and shipping strategies, such as improving delivery times, reducing shipping costs, and enhancing customer experience in different cities and states. This can help in increasing customer satisfaction and loyalty.
- Customer Segmentation: Target can segment customers based on their cities and states to gain a deeper understanding of customer preferences, behaviours, and needs in different regions. This can help in creating targeted marketing campaigns, promotions, and product offerings for different customer segments, leading to increased sales and customer retention.
- Customer Feedback and Reviews: Target can also use customer feedback and reviews from different cities and states to identify any specific pain points, issues, or areas of improvement. Analysing customer comments and reviews can provide valuable insights into customer satisfaction, product quality, and service levels in different regions. Target can use this feedback to address any issues and continuously improve their products and services in different cities and states.
- It may be worth analysing the conversion rates and customer behaviour for different payment installment options to identify any patterns or trends.
- Target in Brazil could consider offering more attractive and flexible payment installment options to cater to customer preferences and drive higher sales.
- Target in Brazil could also consider promoting payment installment options during marketing and promotional campaigns to encourage customers to choose these options at checkout.
- Monitoring customer feedback and conducting surveys to understand customer preferences and satisfaction with payment installment options can provide valuable insights for improving the payment experience at Target in Brazil.

Recommendations for Target to optimise their marketing, localisation, delivery, and customer engagement strategies in different regions of Brazil, leading to increased customer satisfaction, loyalty, and business growth.

References

BigQuery:

```
-- Targeting Success: A Business Case Analysis of 100k Orders
-- at Target in Brazil by Emma Luk

-- BigQuery shape table for customers table
SELECT  count(distinct column_name), (select  count(*) from  `target-business-case-
382621.target_business.customers`)
FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'customers';

-- BigQuery shape table for sellers table
SELECT  count(distinct column_name), (select  count(*) from  `target-business-case-
382621.target_business.sellers`)
FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'sellers';

-- BigQuery shape table for order_items table
SELECT  count(distinct column_name), (select  count(*) from  `target-business-case-
382621.target_business.order_items`)
FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'order_items';

-- BigQuery shape table for geolocations table
SELECT  count(distinct column_name), (select  count(*) from  `target-business-case-
382621.target_business.geolocations`)
FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'geolocations';

-- BigQuery shape table for  payments table
SELECT  count(distinct column_name), (select  count(*) from  `target-business-case-
382621.target_business. payments`)
FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'payments';

-- BigQuery shape table for orders table
SELECT  count(distinct column_name), (select  count(*) from  `target-business-case-
382621.target_business.orders`)
FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'orders';

-- BigQuery shape table for reviews table
SELECT  count(distinct column_name), (select  count(*) from  `target-business-case-
382621.target_business.reviews`)
FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'reviews';

-- BigQuery shape table for products table
SELECT  count(distinct column_name), (select  count(*) from  `target-business-case-
382621.target_business.products`)
FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'products';
--------

-- Analyse Data Types of Columns
SELECT column_name, data_type
FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'customers';
```

```sql
-- Analyse Data Types of Columns
SELECT column_name, data_type
FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'sellers';

-- Analyse Data Types of Columns
SELECT column_name, data_type
FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'order_items';

-- Analyse Data Types of Columns
SELECT column_name, data_type
FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'geolocations';

-- Analyse Data Types of Columns
SELECT column_name, data_type
FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'payments';

-- Analyse Data Types of Columns
SELECT column_name, data_type
FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'orders';

-- Analyse Data Types of Columns
SELECT column_name, data_type
FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'order_reviews';

-- Analyse Data Types of Columns
SELECT column_name, data_type
FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'products';

------------------------------------------------------
-- Data type of columns in a table
-- Analyse Data Types of Columns for different tables
-- with Common Table Expression (CTE)
------------------------------------------------------
WITH customer_columns AS (
SELECT column_name, data_type
FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'customers'
),
seller_columns AS (
SELECT column_name, data_type
FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'sellers'
),
order_items_columns AS (
SELECT column_name, data_type
FROM`target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'order_items'
),
geolocations_columns AS (
SELECT column_name, data_type
FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'geolocations'
),
payments_columns AS (
SELECT column_name, data_type
FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'payments'
),
orders_columns AS (
SELECT column_name, data_type
FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'orders'
),
reviews_columns AS (
SELECT column_name, data_type
FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'order_reviews'
),
products_columns AS (
SELECT column_name, data_type
FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'products'
)
-- Analyse Data Types of Columns for different tables with Common Table Expression (CTE)
SELECT column_name, data_type FROM customer_columns
UNION ALL
SELECT column_name, data_type FROM seller_columns
UNION ALL
SELECT column_name, data_type FROM order_items_columns
UNION ALL
SELECT column_name, data_type FROM geolocations_columns
UNION ALL
SELECT column_name, data_type FROM payments_columns
UNION ALL
SELECT column_name, data_type FROM orders_columns
UNION ALL
SELECT column_name, data_type FROM reviews_columns
UNION ALL
SELECT column_name, data_type FROM products_columns;

----


-- ------------------------------------------------------
-- Data type of columns in a table
-- Analyse Data Types of Columns for different tables
-- with Common Table Expression (CTE)
------------------------------------------------------
--
-- BigQuery shape table for customers table
--
WITH customer_shape AS (
  SELECT COUNT(DISTINCT column_name) AS num_columns,
         (SELECT COUNT(*) FROM `target-business-case-
382621.target_business.customers`) AS num_rows
  FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
  WHERE table_name = 'customers'
)
SELECT num_columns, num_rows
FROM customer_shape;

-- BigQuery shape table for sellers table
WITH seller_shape AS (
  SELECT COUNT(DISTINCT column_name) AS num_columns,
         (SELECT COUNT(*) FROM `target-business-case-
382621.target_business.sellers`) AS num_rows
  FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
  WHERE table_name = 'sellers'
)
SELECT num_columns, num_rows
FROM seller_shape;

-- BigQuery shape table for order_items table
WITH order_items_shape AS (
  SELECT COUNT(DISTINCT column_name) AS num_columns,
         (SELECT COUNT(*) FROM `target-business-case-
382621.target_business.order_items`) AS num_rows
  FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
  WHERE table_name = 'order_items'
)
SELECT num_columns, num_rows
FROM order_items_shape;

-- BigQuery shape table for geolocations table
WITH geolocations_shape AS (
  SELECT COUNT(DISTINCT column_name) AS num_columns,
         (SELECT COUNT(*) FROM `target-business-case-
382621.target_business.geolocation`) AS num_rows
  FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
  WHERE table_name = 'geolocation'
)
SELECT num_columns, num_rows
FROM geolocations_shape;

-- BigQuery shape table for payments table
WITH payments_shape AS (
  SELECT COUNT(DISTINCT column_name) AS num_columns,
         (SELECT COUNT(*) FROM `target-business-case-
382621.target_business.payments`) AS num_rows
  FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
  WHERE table_name = 'payments'
)
SELECT num_columns, num_rows
FROM payments_shape;

-- BigQuery shape table for orders table
WITH orders_shape AS (
  SELECT COUNT(DISTINCT column_name) AS num_columns,
         (SELECT COUNT(*) FROM `target-business-case-382621.target_business.orders`) AS num_rows
  FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
  WHERE table_name = 'orders'
)
SELECT num_columns, num_rows
FROM orders_shape;

-- BigQuery shape table for reviews table
WITH reviews_shape AS (
  SELECT COUNT(DISTINCT column_name) AS num_columns,
         (SELECT COUNT(*) FROM `target-business-case-
382621.target_business.order_reviews`) AS num_rows
  FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
  WHERE table_name = 'order_reviews'
)
SELECT num_columns, num_rows
FROM reviews_shape;
```

```sql
-- BigQuery shape table for products table
WITH products_shape AS (
  SELECT COUNT(DISTINCT column_name) AS num_columns,
         (SELECT COUNT(*) FROM `target-business-case-
382621.target_business.products`) AS num_rows
  FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
  WHERE table_name = 'products'
)
SELECT num_columns, num_rows
FROM products_shape;

---
-- You can now use INFORMATION_SCHEMA - a series of views that provide access to metadata
--- about datasets, tables, and views
SELECT * EXCEPT(is_generated, generation_expression, is_stored, is_updatable)
FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'customers';

SELECT * EXCEPT(is_generated, generation_expression, is_stored, is_updatable)
FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'order_items';

SELECT * EXCEPT(is_generated, generation_expression, is_stored, is_updatable)
FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'order_reviews';

SELECT * EXCEPT(is_generated, generation_expression, is_stored, is_updatable)
FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'orders';

SELECT * EXCEPT(is_generated, generation_expression, is_stored, is_updatable)
FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'payments';

SELECT * EXCEPT(is_generated, generation_expression, is_stored, is_updatable)
FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'products';

SELECT * EXCEPT(is_generated, generation_expression, is_stored, is_updatable)
FROM `target-business-case-382621.target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'sellers';

SELECT * EXCEPT(is_generated, generation_expression, is_stored, is_updatable)
FROM `target_business.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'sellers';

-- 2. Time period for which the data is given

WITH min_max_dates AS (
  SELECT
    MIN(order_purchase_timestamp) AS min_date,
    MAX(order_purchase_timestamp) AS max_date
  FROM
    target_business.orders
)
SELECT
  FORMAT_TIMESTAMP('%Y-%m-%d', min_date) AS min_purchase_date,
  FORMAT_TIMESTAMP('%Y-%m-%d', max_date) AS max_purchase_date
FROM
  min_max_dates;
```

```
-- min_purchase_date: 2016-09-04
-- max_purchase_date: 2018-10-17
--

WITH orders_cte AS (
  SELECT
    customer_id,
    customer_city,
    customer_state
  FROM
    `target_business.customers`
  WHERE
    order_purchase_timestamp BETWEEN TIMESTAMP('2016-09-04') AND TIMESTAMP('2018-10-17')
)
SELECT
  customer_city AS city,
  customer_state AS state
FROM
  orders_cte;


SELECT
  o.order_purchase_timestamp AS order_purchase_timestamp,
  FROM
    target_business.orders o;

------
WITH order_dates AS (
  SELECT
    order_id,
    customer_id,
    customer_city,
    customer_state,
    TIMESTAMP(order_purchase_timestamp) AS purchase_timestamp
  FROM
    target_business.orders o
  WHERE
    TIMESTAMP(order_purchase_timestamp) BETWEEN TIMESTAMP('start_date') AND TIMESTAMP('end_date'
)
),
order_items AS (
  SELECT
    order_id,
    product_id,
    seller_id,
    price,
    freight_value
  FROM
    target_business.order_items oi
)
SELECT
  od.customer_city,
  od.customer_state
FROM
  order_dates od
JOIN
  order_items oi
ON
  od.order_id = oi.order_id
```

```
JOIN
  target_business.customers c
ON
  c.customer_id = od.customer_id
JOIN
  target_business.geolocation g
ON
  c.customer_zip_code_prefix = g.geolocation_zip_code_prefix;

--
WITH orders_cte AS (
  SELECT
    order_id,
    customer_id,
    order_purchase_timestamp
  FROM
    target_business.orders
  WHERE
    order_purchase_timestamp >= TIMESTAMP("2016-09-04 21:15:19 UTC") -
- Replace with the start date and time of the period
    AND order_purchase_timestamp <= TIMESTAMP("2018-10-17 17:30:18 UTC") -
- Replace with the end date and time of the period
),

customers_cte AS (
  SELECT
    customer_id,
    customer_city,
    customer_state
  FROM
    target_business.customers
)

SELECT
  c.customer_city AS city,
  c.customer_state AS state
FROM
  customers_cte c
JOIN
  orders_cte o
ON
  c.customer_id = o.customer_id
ORDER BY
  c.customer_state, c.customer_city;


-- Define the start and end date for the period
DECLARE @start_date DATE;
DECLARE @end_date DATE;
SET @start_date = '2016-09-04';
SET @end_date = '2018-10-17';

-- CTE to get the order IDs and customer IDs for orders placed during the given period
WITH orders_cte AS (
  SELECT order_id, customer_id
  FROM target_business.orders
  WHERE order_purchase_timestamp BETWEEN @start_date AND @end_date
)

-- CTE to get the customer city and state information
, customer_cte AS (
  SELECT customer_id, customer_city, customer_state
  FROM target_business.customers
)

-- CTE to get the geolocation city and state information
, geolocation_cte AS (
  SELECT geolocation_zip_code_prefix, geolocation_city, geolocation_state
  FROM target_business.geolocation
)

-- Join the order, customer, and geolocation CTEs to get the final result
SELECT o.order_id, c.customer_city, c.customer_state, g.geolocation_city, g.geolocation_state
FROM orders_cte o
LEFT JOIN customer_cte c ON o.customer_id = c.customer_id
LEFT JOIN geolocation_cte g ON c.customer_zip_code_prefix = g.geolocation_zip_code_prefix;

--
-
- Cities and States of customers ordered during the given period using Common Table Expression (
CTE)
WITH orders_cte AS (
  SELECT DISTINCT customer_city, customer_state
  FROM target_business.orders o
  JOIN target_business.customers c ON o.customer_id = c.customer_id
  JOIN target_business.geolocation g ON c.customer_zip_code_prefix = g.geolocation_zip_code_pref
ix
  WHERE o.order_purchase_timestamp BETWEEN '2016-09-04 21:15:19 UTC' AND '2018-10-
17 17:30:18 UTC'
)
SELECT customer_city, customer_state
FROM orders_cte
ORDER BY customer_state, customer_city;



------

SELECT city, state
FROM customer_orders
WHERE order_date BETWEEN 'start_date' AND 'end_date'
GROUP BY city, state
ORDER BY state, city;

SELECT DISTINCT customer_city, customer_state
FROM target_business.orders o
JOIN target_business.customers c ON o.customer_id = c.customer_id
-- JOIN geolocations g ON c.customer_zip_code_prefix = g.geolocation_zip_code_prefix
WHERE o.order_purchase_timestamp BETWEEN '2016-09-04 21:15:19 UTC' AND '2018-10-17 17:30:18 UTC'
ORDER BY customer_state, customer_city;

SELECT DISTINCT customer_city, customer_state
FROM target_business.orders o
JOIN target_business.customers c ON o.customer_id = c.customer_id
JOIN target_business.geolocation g ON c.customer_zip_code_prefix = g.geolocation_zip_code_prefix
WHERE o.order_purchase_timestamp BETWEEN '2016-09-04 21:15:19 UTC' AND '2018-10-17 17:30:18 UTC'
ORDER BY customer_state, customer_city;

-- 1. Is there a growing trend on e-
commerce in Brazil? How can we describe a complete scenario? Can we see some seasonality with pe
aks at specific months?
```

```
WITH customer_locations AS (
  SELECT
    c.customer_unique_id,
    c.customer_zip_code_prefix,
    c.customer_city,
    c.customer_state
  FROM `target_business.customers` c
),
seller_locations AS (
  SELECT
    seller_id,
    seller_zip_code_prefix,
    seller_city,
    seller_state
  FROM `target-business-case-382621.target_business.sellers` s
),
order_items_info AS (
  SELECT
    oi.order_id,
    oi.order_item_id,
    oi.product_id,
    oi.seller_id,
    oi.price,
    oi.freight_value
  FROM `target-business-case-382621.target_business.order_items` oi
),
orders_info AS (
  SELECT
    o.order_id,
    --c.customer_unique_id,
    o.customer_id,
    o.order_purchase_timestamp,
    o.order_delivered_customer_date
  FROM `target-business-case-382621.target_business.orders` o
),
payments_info AS (
  SELECT
    pa.order_id,
    pa.payment_type,
    pa.payment_installments,
    pa.payment_value
  FROM `target-business-case-382621.target_business.payments` pa
),
product_info AS (
  SELECT
    p.product_id,
    p.product_category
  FROM `target-business-case-382621.target_business.products` p
),
order_items_with_product_info AS (
  SELECT
    oi.order_id,
    oi.order_item_id,
    oi.product_id,
    oi.seller_id,
    oi.price,
    oi.freight_value,
    pi.product_category
  FROM order_items_info AS oi
  JOIN product_info AS pi
  ON oi.product_id = pi.product_id
```

```sql
),
orders_with_payment_info AS (
  SELECT
    oi.order_id,
    oi.product_category,
    oi.price,
    oi.freight_value,
    pi.payment_type,
    pi.payment_installments,
    pi.payment_value
  FROM order_items_with_product_info AS oi
  JOIN payments_info AS pi
  ON oi.order_id = pi.order_id
),
orders_with_customer_info AS (
  SELECT
    oi.order_id,
    oi.product_category,
    oi.price,
    oi.freight_value,
    pi.payment_type,
    pi.payment_installments,
    pi.payment_value,
    ci.customer_state,
    ci.customer_city
  FROM orders_with_payment_info AS oi
  JOIN customer_locations AS ci
  ON oi.order_id = ci.customer_unique_id
),
orders_with_seller_info AS (
  SELECT
    oi.order_id,
    oi.product_category,
    oi.price,
    oi.freight_value,
    pi.payment_type,
    pi.payment_installments,
    pi.payment_value,
    oi.seller_id,
    si.seller_state,
    si.seller_city
  FROM orders_with_customer_info AS oi
  JOIN seller_locations AS si
  ON oi.seller_id = si.seller_id
),
orders_with_dates AS (
  SELECT
    o.order_id,
    o.product_category,
    o.price,
    o.freight_value,
    o.payment_type,
    o.payment_installments,
    o.payment_value,
    o.customer_state,
    o.customer_city,
    o.seller_state,
    o.seller_city,
    o.order_purchase_timestamp,
    o.order_delivered_customer_date,
    EXTRACT(MONTH FROM o.order_purchase_timestamp) AS order_month
  FROM orders_with_seller_info AS o
),
monthly_orders AS (
  SELECT
    order_month,
    COUNT(DISTINCT order_id) AS num_orders,
    SUM(price) AS total_revenue,
    SUM(freight_value) AS total_freight,
    SUM(payment_value) AS total_payment,
    COUNT(DISTINCT customer_city) AS num_cities,
    COUNT(DISTINCT seller_city) AS num_seller_cities
  FROM orders_with_dates
  GROUP BY order_month
)
SELECT
  order_month,;

----

WITH customer_locations AS (
  SELECT
    customer_unique_id,
    customer_zip_code_prefix,
    customer_city,
    customer_state
  FROM `target_business.customers`
),
seller_locations AS (
  SELECT
    seller_id,
    seller_zip_code_prefix,
    seller_city,
    seller_state
  FROM `target_business.sellers`
),
order_items_info AS (
  SELECT
    order_id,
    order_item_id,
    product_id,
    seller_id,
    price,
    freight_value
  FROM `target_business.order_items`
),
orders_info AS (
  SELECT
    order_id,
    --customer_unique_id,
    customer_id,
    order_purchase_timestamp,
    order_delivered_customer_date
  FROM `target_business.orders`
),
payments_info AS (
  SELECT
    order_id,
    payment_type,
    payment_installments,
    payment_value
  FROM `target_business.payments`
```

```
),
product_info AS (
  SELECT
    product_id,
    product_category
  FROM `target_business.products`
),
order_items_with_product_info AS (
  SELECT
    oi.order_id,
    oi.order_item_id,
    oi.product_id,
    oi.seller_id,
    oi.price,
    oi.freight_value,
    pi.product_category
  FROM order_items_info AS oi
  JOIN product_info AS pi
  ON oi.product_id = pi.product_id
),
orders_with_payment_info AS (
  SELECT
    oi.order_id,
    oi.product_category,
    oi.price,
    oi.freight_value,
    pi.payment_type,
    pi.payment_installments,
    pi.payment_value
  FROM order_items_with_product_info AS oi
  JOIN payments_info AS pi
  ON oi.order_id = pi.order_id
),
orders_with_customer_info AS (
  SELECT
    oi.order_id,
    oi.product_category,
    oi.price,
    oi.freight_value,
    pi.payment_type,
    pi.payment_installments,
    pi.payment_value,
    ci.customer_state,
    ci.customer_city
  FROM orders_with_payment_info AS oi
  JOIN customer_locations AS ci
  ON oi.order_id = ci.customer_unique_id
),
orders_with_seller_info AS (
  SELECT
    oi.order_id,
    oi.product_category,
    oi.price,
    oi.freight_value,
    pi.payment_type,
    pi.payment_installments,
    pi.payment_value,
    oi.seller_id,
    si.seller_state,
    si.seller_city
  FROM orders_with_customer_info AS oi
```

```sql
  JOIN seller_locations AS si
  ON oi.seller_id = si.seller_id
),
orders_with_dates AS (
  SELECT
    o.order_id,
    o.product_category,
    o.price,
    o.freight_value,
    o.payment_type,
    o.payment_installments,
    o.payment_value,
    o.customer_state,
    o.customer_city,
    o.seller_state,
    o.seller_city,
    o.order_purchase_timestamp,
    o.order_delivered_customer_date,
    EXTRACT(MONTH FROM o.order_purchase_timestamp) AS order_month
  FROM orders_with_seller_info AS o
),
monthly_orders AS (
  SELECT
    order_month,
    COUNT(DISTINCT order_id) AS num_orders,
    SUM(price) AS total_revenue,
    SUM(freight_value) AS total_freight,
    SUM(payment_value) AS total_payment,
    COUNT(DISTINCT customer_city) AS num_cities,
    COUNT(DISTINCT seller_city) AS num_seller_cities
  FROM orders_with_dates
  GROUP BY order_month
)
SELECT
  order_month,;

--

--UPDATE `target_business.products`
--SET product_category_name = product_category
--WHERE TRUE;
SELECT
EXTRACT(MONTH FROM order_purchase_timestamp) AS month
FROM
`target-business-case-382621.target_business.orders`;



----
-- Breaking Down Brazil's E-commerce Boom:
-- Seasonal Peaks and Complete Trends
SELECT
  EXTRACT(MONTH FROM order_purchase_timestamp) AS month,
  COUNT(DISTINCT o.order_id) AS num_orders,
  SUM(oi.price + oi.freight_value) AS revenue
FROM
  `target-business-case-382621.target_business.orders` o
  JOIN `target-business-case-382621.target_business.order_items` oi ON o.order_id = oi.order_id
  JOIN `target-business-case-
382621.target_business.customers` c ON o.customer_id = c.customer_id
```

```
  JOIN `target-business-case-382621.target_business.geolocation` g ON c.customer_zip_code_prefix = g.geolocation_zip_code_prefix
WHERE
  g.geolocation_state = 'SP'
GROUP BY
  month
ORDER BY
  month ASC;

--
SELECT
EXTRACT(MONTH FROM order_purchase_timestamp) AS month
FROM
`target-business-case-382621.target_business.orders`;

--

SELECT DATE_TRUNC('week', order_purchase_timestamp) as week, COUNT(*) as num_orders
FROM `target-business-case-382621.target_business.orders`
GROUP BY week;

SELECT
  DATE_TRUNC('month', order_purchase_timestamp) AS month
FROM
  `target-business-case-382621.target_business.orders`;


SELECT
  MONTH(order_purchase_timestamp) AS ProducedMonth
FROM
  `target-business-case-382621.target_business.orders`;

  ---

  WITH orders_and_customers AS (
  SELECT
    o.order_id,
    o.customer_id,
    c.customer_city,
    c.customer_state,
    TIMESTAMP_TRUNC(o.order_purchase_timestamp, HOUR) AS order_hour
  FROM
    `your_project_id.orders` AS o
  JOIN
    `your_project_id.customers` AS c
  ON
    o.customer_id = c.customer_id
), orders_and_customers_and_geolocation AS (
  SELECT
    oac.order_id,
    oac.customer_id,
    oac.customer_city,
    oac.customer_state,
    oac.order_hour,
    g.geolocation_city
  FROM
    orders_and_customers AS oac
  JOIN
    `your_project_id.geolocations` AS g
  ON
```

```
    oac.customer_zip_code_prefix = g.geolocation_zip_code_prefix
), orders_and_customers_and_geolocation_and_payments AS (
  SELECT
    ocg.order_id,
    ocg.customer_id,
    ocg.customer_city,
    ocg.customer_state,
    ocg.order_hour,
    ocg.geolocation_city,
    p.payment_installments,
    TIMESTAMP_TRUNC(ocg.order_hour, DAY) AS order_day
  FROM
    orders_and_customers_and_geolocation AS ocg
  JOIN
    `your_project_id.payments` AS p
  ON
    ocg.order_id = p.order_id
), orders_and_customers_and_geolocation_and_payments_and_order_items AS (
  SELECT
    ocgp.order_id,
    ocgp.customer_id,
    ocgp.customer_city,
    ocgp.customer_state,
    ocgp.order_hour,
    ocgp.geolocation_city,
    ocgp.payment_installments,
    ocgp.order_day,
    oi.price,
    oi.freight_value
  FROM
    orders_and_customers_and_geolocation_and_payments AS ocgp
  JOIN
    `your_project_id.order_items` AS oi
  ON
    ocgp.order_id = oi.order_id
)
SELECT
  CASE
    WHEN order_hour BETWEEN TIMESTAMP('2023-04-07 00:00:00', 'UTC') AND TIMESTAMP('2023-04-
07 06:00:00', 'UTC') THEN 'Dawn'
    WHEN order_hour BETWEEN TIMESTAMP('2023-04-07 06:00:00', 'UTC') AND TIMESTAMP('2023-04-
07 12:00:00', 'UTC') THEN 'Morning'
    WHEN order_hour BETWEEN TIMESTAMP('2023-04-07 12:00:00', 'UTC') AND TIMESTAMP('2023-04-
07 18:00:00', 'UTC') THEN 'Afternoon'
    ELSE 'Night'
  END AS time_of_day,
  COUNT(DISTINCT order_id) AS num_orders,
  AVG(price + freight_value) AS avg_order_amount
FROM
  orders_and_customers_and_geolocation_and_payments_and_order_items
WHERE
  customer_state = 'SP' -- change this to the desired state code
  AND payment_installments = 1 -- only consider non-EMI purchases
GROUP BY
  time_of_day
ORDER BY
  time_of_day;
--

WITH orders_and_customers AS (
  SELECT
```

```
    o.order_id,
    o.customer_id,
    c.customer_city,
    c.customer_state,
    TIMESTAMP_TRUNC(o.order_purchase_timestamp, HOUR) AS order_hour
  FROM
    `your_project_id.orders` AS o
  JOIN
    `your_project_id.customers` AS c
  ON
    o.customer_id = c.customer_id
), orders_and_customers_and_geolocation AS (
  SELECT
    oac.order_id,
    oac.customer_id,
    oac.customer_city,
    oac.customer_state,
    oac.order_hour,
    g.geolocation_city
  FROM
    orders_and_customers AS oac
  JOIN
    `your_project_id.geolocations` AS g
  ON
    oac.customer_zip_code_prefix = g.geolocation_zip_code_prefix
), orders_and_customers_and_geolocation_and_payments AS (
  SELECT
    ocg.order_id,
    ocg.customer_id,
    ocg.customer_city,
    ocg.customer_state,
    ocg.order_hour,
    ocg.geolocation_city,
    p.payment_installments,
    TIMESTAMP_TRUNC(ocg.order_hour, DAY) AS order_day
  FROM
    orders_and_customers_and_geolocation AS ocg
  JOIN
    `your_project_id.payments` AS p
  ON
    ocg.order_id = p.order_id
), orders_and_customers_and_geolocation_and_payments_and_order_items AS (
  SELECT
    ocgp.order_id,
    ocgp.customer_id,
    ocgp.customer_city,
    ocgp.customer_state,
    ocgp.order_hour,
    ocgp.geolocation_city,
    ocgp.payment_installments,
    ocgp.order_day,
    oi.price,
    oi.freight_value
  FROM
    orders_and_customers_and_geolocation_and_payments AS ocgp
  JOIN
    `your_project_id.order_items` AS oi
  ON
    ocgp.order_id = oi.order_id
)
SELECT
```

```
  CASE
    WHEN order_hour BETWEEN TIMESTAMP('2023-04-07 00:00:00', 'UTC') AND TIMESTAMP('2023-04-07 06:00:00', 'UTC') THEN 'Dawn'
    WHEN order_hour BETWEEN TIMESTAMP('2023-04-07 06:00:00', 'UTC') AND TIMESTAMP('2023-04-07 12:00:00', 'UTC') THEN 'Morning'
    WHEN order_hour BETWEEN TIMESTAMP('2023-04-07 12:00:00', 'UTC') AND TIMESTAMP('2023-04-07 18:00:00', 'UTC') THEN 'Afternoon'
    ELSE 'Night'
  END AS time_of_day,
  COUNT(DISTINCT order_id) AS num_orders,
  AVG(price + freight_value) AS avg_order_amount
FROM
  orders_and_customers_and_geolocation_and_payments_and_order_items
WHERE
  customer_state = 'SP' -- change this to the desired state code
  AND payment_installments = 1 -- only consider non-EMI purchases
GROUP BY
  time_of_day
ORDER BY
  time_of_day;

--

-- Breaking Down Brazil's E-commerce Boom:
-- Seasonal Peaks and Complete Trends
SELECT
  EXTRACT(HOUR FROM order_purchase_timestamp) AS purchase_hour,
  COUNT(*) AS total_orders
FROM
  `target-business-case-382621.target_business.orders` AS o
  JOIN `target-business-case-382621.target_business.customers` AS c ON o.customer_id = c.customer_id
WHERE
  c.customer_state = 'SP'  -- Select only orders from Sao Paulo state
GROUP BY
  purchase_hour
ORDER BY
  purchase_hour;

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

-- Get month on month orders by states
SELECT
  EXTRACT(MONTH FROM order_purchase_timestamp) AS order_month,
  -- DATE_TRUNC('month', o.order_purchase_timestamp) AS order_month,
  c.customer_state,
  COUNT(DISTINCT o.order_id) AS order_count
FROM
  target_business.orders o
  JOIN target_business.customers c ON o.customer_id = c.customer_id
WHERE
  o.order_purchase_timestamp >= '2016-09-04 21:15:19 UTC' AND o.order_purchase_timestamp < '2018-10-17 17:30:18 UTC'
GROUP BY
  order_month,
  c.customer_state
ORDER BY
  order_month,
  c.customer_state;
```

```sql
--From North to South: Exploring Customer Distribution in Brazil

SELECT c.customer_state, COUNT(c.customer_id) as customer_count
FROM target_business.customers c
JOIN target_business.geolocation g ON c.customer_zip_code_prefix = g.geolocation_zip_code_prefix
GROUP BY c.customer_state
ORDER BY customer_count DESC;


---

SELECT
    ROUND(((SUM(p2.payment_value) - SUM(p1.payment_value)) / SUM(p1.payment_value)) * 100, 2) AS
 percentage_increase
FROM
    target_business.payments p1
JOIN
    target_business.payments p2 ON p1.order_id = p2.order_id
WHERE
    DATE_TRUNC('month', p1.order_purchase_timestamp) >= '2017-01-01'
    AND DATE_TRUNC('month', p1.order_purchase_timestamp) <= '2017-08-31'
    AND DATE_TRUNC('month', p2.order_purchase_timestamp) >= '2018-01-01'
    AND DATE_TRUNC('month', p2.order_purchase_timestamp) <= '2018-08-31';

-- EXTRACT(MONTH FROM order_purchase_timestamp) AS order_month,
    --DATE_TRUNC('month', p1.order_purchase_timestamp) >= '2017-01-01'
    --AND DATE_TRUNC('month', p1.order_purchase_timestamp) <= '2017-08-31'
    --AND DATE_TRUNC('month', p2.order_purchase_timestamp) >= '2018-01-01'
    --AND DATE_TRUNC('month', p2.order_purchase_timestamp) <= '2018-08-31';
--

with A as
(Select extract(year from o.order_purchase_timestamp) as yr, sum(p.payment_value) as cost_of_ord
ers
from target_business.orders o join target_business.payments p
on o.order_id = p.order_id
where extract(month from o.order_purchase_timestamp) between 1 and 8
group by 1)
Select ((a2.cost_of_orders/a1.cost_of_orders) - 1)*100 as perc_increase
from A as a1, A as a2
where a1.yr = 2017 and a2.yr = 2018;


--- From 2017 to 2018: Calculating the Percentage Increase in Order Costs
WITH A AS
(
    SELECT
        EXTRACT(YEAR FROM o.order_purchase_timestamp) AS year,
        SUM(p.payment_value) as cost_of_orders
    FROM
        target_business.orders o
    JOIN
        target_business.payments p ON o.order_id = p.order_id
    WHERE
        EXTRACT(month FROM o.order_purchase_timestamp) BETWEEN 1 AND 8
    GROUP BY
        1
)
SELECT
    ROUND(((a2.cost_of_orders / a1.cost_of_orders) - 1) * 100, 2) as perc_increase
FROM
```

```sql
    A as a1, A as a2
WHERE
    a1.year = 2017 AND a2.year = 2018;

----
with A AS
(SELECT
    EXTRACT(YEAR FROM o.order_purchase_timestamp) AS year,
    SUM(p.payment_value) as cost_of_orders
FROM
    target_business.orders o
JOIN
    target_business.payments p ON o.order_id = p.order_id
WHERE
    EXTRACT(month FROM o.order_purchase_timestamp) between 1 and 8
GROUP BY
   1)
SELECT
((a2.cost_of_orders/a1.cost_of_orders) - 1)*100 as perc_increase
FROM
  A as a1, A as a2
WHERE a1.year = 2017 and a2.year = 2018;

-- State-wise E-commerce Insights: Mean and Sum of Price and Freight Values
-- Common Table Expression (CTE) to retrieve order items data
WITH order_items_cte AS (
    SELECT order_id, price, freight_value
    FROM target_business.order_items
), customers_cte AS (
    SELECT customer_id, customer_state
    FROM target_business.customers c
), combined_data AS (
    SELECT c.customer_state, oi.price, oi.freight_value
    FROM order_items_cte oi
    INNER JOIN target_business.orders o ON oi.order_id = o.order_id
    INNER JOIN customers_cte c ON o.customer_id = c.customer_id
)
SELECT customer_state, AVG(price) AS mean_price, SUM(price) AS sum_price, AVG(freight_value) AS
mean_freight_value, SUM(freight_value) AS sum_freight_value
FROM combined_data
GROUP BY customer_state;
--SELECT * from order_items_cte

-- State-wise E-commerce Insights: Mean and Sum of Price and Freight Values
-- Common Table Expression (CTE) to retrieve order items data
WITH order_items_cte AS (
    SELECT order_id, price, freight_value
    FROM target_business.order_items
),

-- Common Table Expression (CTE) to retrieve customers data
customers_cte AS (
    SELECT customer_id, customer_state
    FROM target_business.customers c
),

-- Common Table Expression (CTE) to combine data from order_items_cte, orders, and customers_cte
combined_data AS (
    SELECT c.customer_state, oi.price, oi.freight_value
    FROM order_items_cte oi
    INNER JOIN target_business.orders o ON oi.order_id = o.order_id
    INNER JOIN customers_cte c ON o.customer_id = c.customer_id
)

-- Main query to calculate mean and sum for each customer state
SELECT customer_state,
       AVG(price) AS mean_price,
       SUM(price) AS sum_price,
       AVG(freight_value) AS mean_freight_value,
       SUM(freight_value) AS sum_freight_value
FROM combined_data
GROUP BY customer_state;
--
--Analysis on sales, freight and delivery time
--Calculate days between purchasing, delivering and estimated delivery
WITH order_info AS (
  SELECT
    o.order_id,
    o.order_purchase_timestamp,
    o.order_delivered_carrier_date,
    o.order_delivered_customer_date,
    o.order_estimated_delivery_date
  FROM
    target_business.orders o
)
, order_delays AS (
  SELECT
    order_id,
    DATE_DIFF(order_delivered_carrier_date, order_purchase_timestamp, DAY) AS carrier_delay,
    DATE_DIFF(order_delivered_customer_date, order_purchase_timestamp, DAY) AS customer_delay,
    DATE_DIFF(order_estimated_delivery_date, order_purchase_timestamp, DAY) AS estimated_deliver
y_delay
  FROM
    order_info
)
SELECT
  order_id,
  carrier_delay,
  customer_delay,
  estimated_delivery_delay
FROM
  order_delays;


--
--
-- 2. Find time_to_delivery & diff_estimated_delivery. Formula for the same given below:
-- o  time_to_delivery = order_purchase_timestamp-order_delivered_customer_date
-- o  diff_estimated_delivery = order_estimated_delivery_date-order_delivered_customer_date

WITH order_data AS (
  SELECT
    order_id,
    order_purchase_timestamp,
    order_delivered_customer_date,
    order_estimated_delivery_date
  FROM
    `target_business.orders`
)
SELECT
  order_id,
  TIMESTAMP_DIFF(order_delivered_customer_date, order_purchase_timestamp, HOUR) AS time_to_deliv
ery,
  TIMESTAMP_DIFF(order_estimated_delivery_date, order_delivered_customer_date, HOUR) AS diff_est
imated_delivery
FROM
  order_data;


-- Group data by state, take mean of freight_value, time_to_delivery, diff_estimated_delivery
WITH order_stats AS (
  SELECT
    c.customer_state AS state,
    AVG(oi.freight_value) AS avg_freight_value,
    AVG(date_diff( o.order_delivered_customer_date, o.order_purchase_timestamp, day)) AS avg_tim
e_to_delivery,
    AVG(date_diff(o.order_estimated_delivery_date, o.order_delivered_customer_date, day)) AS avg
_diff_estimated_delivery
  FROM
    target_business.customers c
  JOIN
    target_business.orders o ON c.customer_id = o.customer_id
  JOIN
    target_business.order_items oi ON o.order_id = oi.order_id
  GROUP BY
    state
)
SELECT
  state,
  avg_freight_value,
  avg_time_to_delivery,
  avg_diff_estimated_delivery
FROM
  order_stats
ORDER BY
  state;

--

-- 5. Top 5 states with highest/lowest average freight value - sort in desc/asc limit 5
WITH state_freight_avg AS (
  SELECT
    c.customer_state AS state,
    AVG(oi.freight_value) AS avg_freight
  FROM
    target_business.customers c
  JOIN
    target_business.orders o ON c.customer_id = o.customer_id
  JOIN
    target_business.order_items oi ON o.order_id = oi.order_id
  GROUP BY
    c.customer_state
)
SELECT
  state,
  avg_freight
FROM (
  SELECT
    state,
    avg_freight,
    ROW_NUMBER() OVER (ORDER BY avg_freight DESC) AS rn_desc,
    ROW_NUMBER() OVER (ORDER BY avg_freight ASC) AS rn_asc
  FROM
    state_freight_avg
)
WHERE
  rn_desc <= 5 OR rn_asc <= 5
ORDER BY
  avg_freight DESC, state ASC;
-- 5. Top 5 states with highest/lowest average freight value - sort in desc/asc limit 5
WITH freight_avg_by_state AS (
  SELECT
    c.customer_state AS state,
    AVG(oi.freight_value) AS avg_freight
  FROM
    target_business.customers c
  INNER JOIN
    target_business.orders o ON c.customer_id = o.customer_id
  INNER JOIN
    target_business.order_items oi ON o.order_id = oi.order_id
  GROUP BY
    state
)
SELECT
  state,
  avg_freight
FROM
  freight_avg_by_state
ORDER BY
  avg_freight DESC
LIMIT
  5; -- Top 5 states with highest average freight value

WITH freight_avg_by_state AS (
  SELECT
    c.customer_state AS state,
    AVG(oi.freight_value) AS avg_freight
  FROM
    target_business.customers c
  INNER JOIN
    target_business.orders o ON c.customer_id = o.customer_id
  INNER JOIN
    target_business.order_items oi ON o.order_id = oi.order_id
  GROUP BY
    state
)
SELECT
  state,
  avg_freight
FROM
  freight_avg_by_state
ORDER BY
  avg_freight ASC
LIMIT
  5; -- Top 5 states with lowest average freight value

--
WITH avg_freight AS (
  SELECT
    c.customer_state AS state,
    AVG(oi.freight_value) AS avg_freight
  FROM
    `target_business.customers` c
  JOIN
    `target_business.orders` o ON c.customer_id = o.customer_id
  JOIN
    `target_business.order_items` oi ON o.order_id = oi.order_id
  GROUP BY
    state
)
SELECT
  state,
  avg_freight
FROM
  avg_freight
ORDER BY
  avg_freight DESC
LIMIT
  5

-- The above query finds the top 5 states with the highest average freight value

UNION ALL

SELECT
  state,
  avg_freight
FROM (
  SELECT
    state,
    avg_freight
  FROM
    avg_freight
  ORDER BY
    avg_freight ASC
  LIMIT
    5
)
ORDER BY
  avg_freight ASC;

-- The above query finds the bottom 5 states with the lowest average freight value
--
-- 7. Top 5 states with highest/lowest average time to delivery
WITH order_delivery_time AS (
  SELECT
    c.customer_state AS state,
    TIMESTAMP_DIFF(o.order_delivered_customer_date, o.order_purchase_timestamp, DAY) AS delivery
_time
  FROM
    target_business.orders o
  JOIN
    target_business.customers c ON o.customer_id = c.customer_id
)
, avg_delivery_time AS (
  SELECT
    state,
    AVG(delivery_time) AS avg_time
  FROM
    order_delivery_time
  GROUP BY
    state
)
SELECT
```

```
  state,
  avg_time
FROM (
  SELECT
    state,
    avg_time,
    ROW_NUMBER() OVER (ORDER BY avg_time DESC) AS rank_desc,
    ROW_NUMBER() OVER (ORDER BY avg_time ASC) AS rank_asc
  FROM
    avg_delivery_time
)
WHERE
  rank_desc <= 5 OR rank_asc <= 5
ORDER BY
  rank_desc ASC,
  rank_asc ASC;

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

WITH order_delivery AS (
  SELECT
    o.order_id,
    o.order_status,
    o.order_purchase_timestamp,
    o.order_delivered_carrier_date,
    o.order_delivered_customer_date,
    o.order_estimated_delivery_date,
    c.customer_state AS state
  FROM
    target_business.orders o
  JOIN
    target_business.customers c
  ON
    o.customer_id = c.customer_id
)
SELECT
  state AS state,
  COUNT(*) AS total_orders,
  SUM(CASE
      WHEN order_status = 'delivered' AND order_delivered_customer_date <= order_estimated_deliv
ery_date THEN 1
      ELSE 0
    END) AS fast_deliveries,
  SUM(CASE
      WHEN order_status = 'delivered' AND order_delivered_customer_date > order_estimated_delive
ry_date THEN 1
      ELSE 0
    END) AS delayed_deliveries
FROM
  order_delivery
GROUP BY
  state
ORDER BY
  fast_deliveries DESC
LIMIT
  5;

--
-- Month over Month count of orders for different payment types
WITH monthly_orders AS (
  SELECT
```

```
    DATE_TRUNC(DATE(order_purchase_timestamp), MONTH) AS month,
    p.payment_type,
    COUNT(DISTINCT o.order_id) AS order_count
  FROM
    target_business.orders o
  INNER JOIN
    target_business.payments p ON o.order_id = p.order_id
  GROUP BY
    month,
    payment_type
)
SELECT
  month,
  payment_type,
  SUM(order_count) AS total_orders
FROM
  monthly_orders
GROUP BY
  month,
  payment_type
ORDER BY
  month,
  payment_type;

--

-- Count of orders based on the no. of payment instalments
WITH order_payments AS (
  SELECT
    o.order_id,
    p.payment_installments
  FROM
    target_business.orders o
  JOIN
    target_business.payments p ON o.order_id = p.order_id
)
SELECT
    payment_installments,
    COUNT(DISTINCT order_id) AS order_count
FROM
    order_payments
GROUP BY
    payment_installments;

--
WITH order_delivery_time AS (
  SELECT
    c.customer_state AS state,
    o.order_delivered_customer_date AS delivered_date,
    o.order_purchase_timestamp AS purchase_date
  FROM
    target_business.orders o
  JOIN
    target_business.customers c ON o.customer_id = c.customer_id
)
, delivery_time AS (
  SELECT
    state,
    TIMESTAMP_DIFF(delivered_date, purchase_date, DAY) AS delivery_days
  FROM
    order_delivery_time
```

```
)
, avg_delivery_time AS (
  SELECT
    state,
    AVG(delivery_days) AS avg_delivery_time
  FROM
    delivery_time
  GROUP BY
    state
)
, top_states AS (
  SELECT
    state,
    avg_delivery_time,
    RANK() OVER (ORDER BY avg_delivery_time DESC) AS rank_high,
    RANK() OVER (ORDER BY avg_delivery_time ASC) AS rank_low
  FROM
    avg_delivery_time
)
SELECT
  state,
  avg_delivery_time
FROM
  top_states
WHERE
  rank_high <= 5
ORDER BY
  avg_delivery_time DESC;

---
WITH order_delivery AS (
  SELECT
    o.order_id,
    o.customer_state,
    o.order_purchase_timestamp,
    o.order_delivered_customer_date,
    TIMESTAMP_DIFF(o.order_delivered_customer_date, o.order_purchase_timestamp, HOUR) AS delivery_time
  FROM
    `target_business.orders` o
)
SELECT
  customer_state AS state,
  AVG(delivery_time) AS avg_delivery_time
FROM
  order_delivery
GROUP BY
  customer_state
ORDER BY
  avg_delivery_time DESC
LIMIT
  5 -- Top 5 states with highest average time to delivery




--

WITH orders_info AS (
  SELECT
    o.order_id,
```

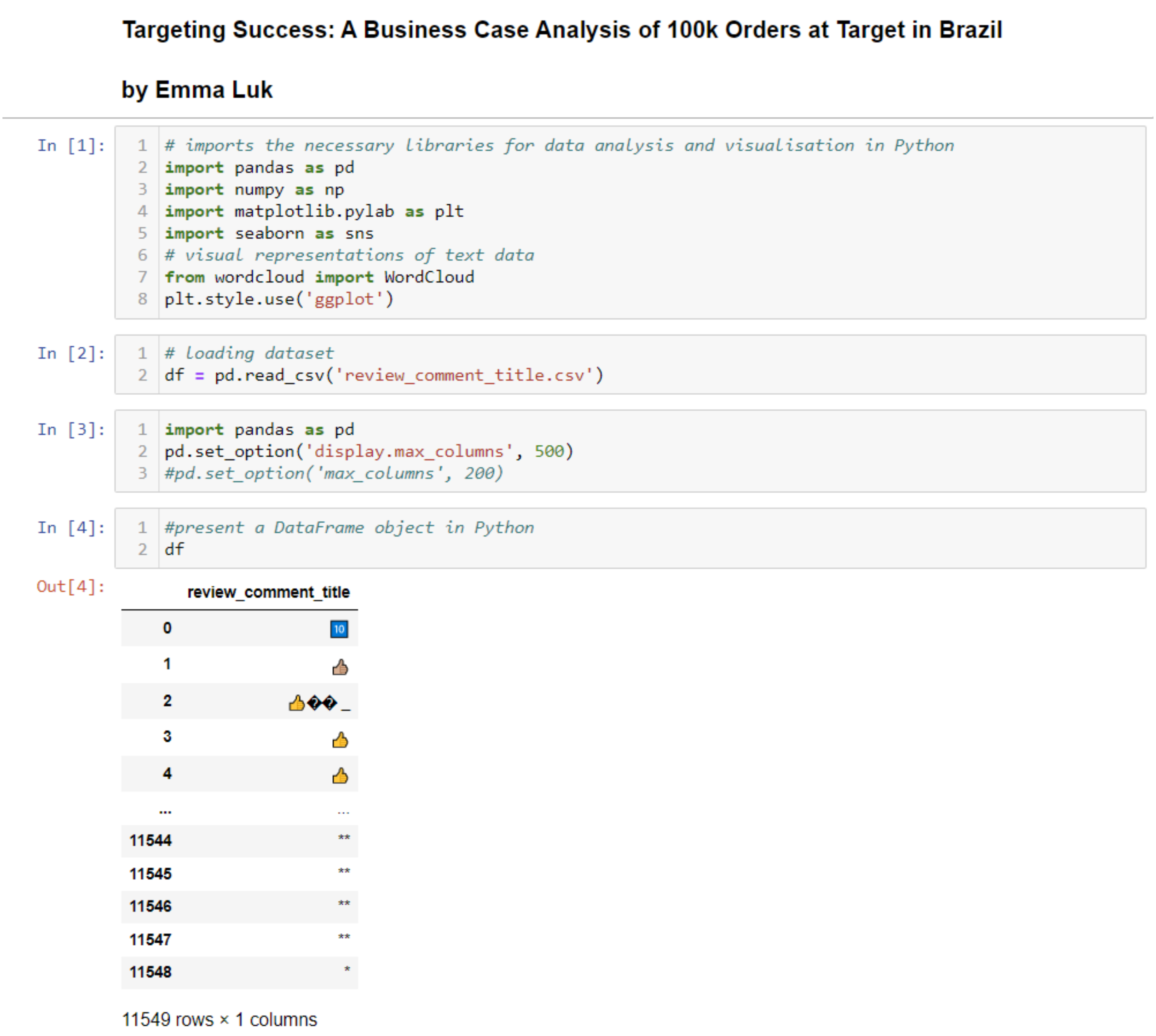
```sql
    o.customer_state,
    o.order_purchase_timestamp,
    o.order_delivered_customer_date,
    TIMESTAMP_DIFF(o.order_delivered_customer_date, o.order_purchase_timestamp, HOUR) AS time_to
_delivery
  FROM
    `your_dataset.orders` o
)
SELECT
  state,
  AVG(time_to_delivery) AS avg_time_to_delivery
FROM
  orders_info
GROUP BY
  state
ORDER BY
  avg_time_to_delivery DESC
LIMIT 5 -- Top 5 states with highest average time to delivery


--
SELECT customer_state, AVG(freight_value) AS avg_freight_value
FROM target_business.orders
GROUP BY customer_state
ORDER BY avg_freight_value DESC
LIMIT 5;

SELECT customer_state, AVG(freight_value) AS avg_freight_value
FROM target_business.orders
GROUP BY customer_state
ORDER BY avg_freight_value ASC
LIMIT 5;

SELECT
    c.customer_state AS state,
  FROM
    target_business.customers c;


--

WITH order_data AS (
  SELECT
    order_id,
    order_purchase_timestamp,
    order_delivered_customer_date,
    order_estimated_delivery_date
  FROM
    `target_business.orders` -- Replace with your actual project and dataset name
)
SELECT
  order_id,
  TIMESTAMP_DIFF(order_delivered_customer_date, order_purchase_timestamp, HOUR) AS time_to_deliv
ery,
  TIMESTAMP_DIFF(order_estimated_delivery_date, order_delivered_customer_date, HOUR) AS diff_est
imated_delivery
FROM
  order_data;

--
```

```
WITH order_data AS (
  SELECT
    o.order_id,
    o.order_purchase_timestamp,
    o.order_delivered_customer_date,
    o.order_estimated_delivery_date,
    o.customer_state,
    oi.freight_value,
    TIMESTAMP_DIFF(o.order_delivered_customer_date, o.order_purchase_timestamp, HOUR) AS time_to_delivery,
    TIMESTAMP_DIFF(o.order_estimated_delivery_date, o.order_delivered_customer_date, HOUR) AS diff_estimated_delivery
  FROM
    `target_business.orders` AS o
  JOIN
    `target_business.order_items` AS oi
  ON
    o.order_id = oi.order_id
)
SELECT
  customer_state,
  AVG(freight_value) AS avg_freight_value,
  AVG(time_to_delivery) AS avg_time_to_delivery,
  AVG(diff_estimated_delivery) AS avg_diff_estimated_delivery
FROM
  order_data
GROUP BY
  customer_state;

--


WITH order_stats AS (
  SELECT
    --o.customer_state AS state,
    AVG(oi.freight_value) AS avg_freight_value,
    AVG(DATEDIFF(o.order_delivered_customer_date, o.order_purchase_timestamp)) AS avg_time_to_delivery,
    AVG(DATEDIFF(o.order_delivered_customer_date, o.order_estimated_delivery_date)) AS avg_diff_estimated_delivery
  FROM
    target_business.orders o
  JOIN
    target_business.order_items oi ON o.order_id = oi.order_id
  GROUP BY
    state
)
SELECT
  state,
  avg_freight_value,
  avg_time_to_delivery,
  avg_diff_estimated_delivery
FROM
  target_business.order_stats
ORDER BY
  state;

--

WITH order_stats AS (
  SELECT
```

```
    c.customer_state AS state,
    AVG(oi.freight_value) AS avg_freight_value,
    AVG(DATEDIFF(o.order_delivered_customer_date, o.order_purchase_timestamp)) AS avg_time_to_de
livery,
    AVG(DATEDIFF(o.order_delivered_customer_date, o.order_estimated_delivery_date)) AS avg_diff_
estimated_delivery
  FROM
    target_business.customers c
  JOIN
    target_business.orders o ON c.customer_id = o.customer_id
  JOIN
    target_business.order_items oi ON o.order_id = oi.order_id
  GROUP BY
    state
)
SELECT
  state,
  avg_freight_value,
  avg_time_to_delivery,
  avg_diff_estimated_delivery
FROM
  order_stats
ORDER BY
  state;
--

WITH order_stats AS (
  SELECT
    c.customer_state AS state,
    AVG(oi.freight_value) AS avg_freight_value,
    AVG(date_diff('day', o.order_purchase_timestamp, o.order_delivered_customer_date)) AS avg_ti
me_to_delivery,
    AVG(date_diff('day', o.order_delivered_customer_date, o.order_estimated_delivery_date)) AS a
vg_diff_estimated_delivery
  FROM
    target_business.customers c
  JOIN
    target_business.orders o ON c.customer_id = o.customer_id
  JOIN
    target_business.order_items oi ON o.order_id = oi.order_id
  GROUP BY
    state
)
SELECT
  state,
  avg_freight_value,
  avg_time_to_delivery,
  avg_diff_estimated_delivery
FROM
  order_stats
ORDER BY
  state;


--

WITH order_delivery AS (
  SELECT
    o.order_id,
    o.order_purchase_timestamp,
    o.order_delivered_carrier_date,
    o.order_delivered_customer_date,
    o.order_estimated_delivery_date
  FROM
    target_business.orders o
)
SELECT
  order_id,
  order_purchase_timestamp,
  order_delivered_carrier_date,
  order_delivered_customer_date,
  order_estimated_delivery_date,
  DATE_PART('day', order_delivered_carrier_date::timestamp - order_purchase_timestamp::timestamp) AS days_between_purchasing_and_delivering,
  DATE_PART('day', order_delivered_customer_date::timestamp - order_purchase_timestamp::timestamp) AS days_between_purchasing_and_delivered,
  DATE_PART('day', order_estimated_delivery_date::timestamp - order_purchase_timestamp::timestamp) AS days_between_purchasing_and_estimated_delivery
FROM
  order_delivery;




--end--
---
WITH order_delivery AS (
  SELECT
    o.order_id,
    o.order_purchase_timestamp,
    o.order_delivered_carrier_date,
    o.order_delivered_customer_date,
    o.order_estimated_delivery_date
  FROM
    target_business.orders o
)
SELECT
  order_id,
  order_purchase_timestamp,
  order_delivered_carrier_date,
  order_delivered_customer_date,
  order_estimated_delivery_date,
  DATE_PART('day', order_delivered_carrier_date::date - order_purchase_timestamp::date) AS days_between_purchasing_and_delivering,
  DATE_PART('day', order_delivered_customer_date::date - order_purchase_timestamp::date) AS days_between_purchasing_and_delivered,
  DATE_PART('day', order_estimated_delivery_date::date - order_purchase_timestamp::date) AS days_between_purchasing_and_estimated_delivery
FROM
  order_delivery;
----
WITH order_delivery AS (
  SELECT
    o.order_id,
    o.order_purchase_timestamp,
    o.order_delivered_carrier_date,
    o.order_delivered_customer_date,
    o.order_estimated_delivery_date
  FROM
    target_business.orders o
```

```sql
)
SELECT
  order_id,
  order_purchase_timestamp,
  order_delivered_carrier_date,
  order_delivered_customer_date,
  order_estimated_delivery_date,
  EXTRACT(EPOCH FROM (order_delivered_carrier_date - order_purchase_timestamp)) / 86400 AS days_between_purchasing_and_delivering,
  EXTRACT(EPOCH FROM (order_delivered_customer_date - order_purchase_timestamp)) / 86400 AS days_between_purchasing_and_delivered,
  EXTRACT(EPOCH FROM (order_estimated_delivery_date - order_purchase_timestamp)) / 86400 AS days_between_purchasing_and_estimated_delivery
FROM
  order_delivery;
--
WITH order_delivery AS (
  SELECT
    o.order_id,
    o.order_purchase_timestamp,
    o.order_delivered_carrier_date,
    o.order_delivered_customer_date,
    o.order_estimated_delivery_date
  FROM
    target_business.orders o
)
SELECT
  order_id,
  order_purchase_timestamp,
  order_delivered_carrier_date,
  order_delivered_customer_date,
  order_estimated_delivery_date,
  DATE_PART('day', order_delivered_carrier_date - order_purchase_timestamp) AS days_between_purchasing_and_delivering,
  DATE_PART('day', order_delivered_customer_date - order_purchase_timestamp) AS days_between_purchasing_and_delivered,
  DATE_PART('day', order_estimated_delivery_date - order_purchase_timestamp) AS days_between_purchasing_and_estimated_delivery
FROM
  order_delivery;
---
WITH order_delivery AS (
  SELECT
    o.order_id,
    o.order_purchase_timestamp,
    o.order_delivered_carrier_date,
    o.order_delivered_customer_date,
    o.order_estimated_delivery_date
  FROM
    target_business.orders o
)
SELECT
  order_id,
  order_purchase_timestamp,
  order_delivered_carrier_date,
  order_delivered_customer_date,
  order_estimated_delivery_date,
  (order_delivered_carrier_date::date - order_purchase_timestamp::date) AS days_between_purchasing_and_delivering,
  (order_delivered_customer_date::date - order_purchase_timestamp::date) AS days_between_purchasing_and_delivered,
  (order_estimated_delivery_date::date - order_purchase_timestamp::date) AS days_between_purchasing_and_estimated_delivery
FROM
  order_delivery;

--

WITH order_delivery AS (
  SELECT
    o.order_id,
    o.order_purchase_timestamp,
    o.order_delivered_carrier_date,
    o.order_delivered_customer_date,
    o.order_estimated_delivery_date
  FROM
    target_business.orders o
)
SELECT
  order_id,
  order_purchase_timestamp,
  order_delivered_carrier_date,
  order_delivered_customer_date,
  order_estimated_delivery_date,
  DATE_PART('day', order_delivered_carrier_date - order_purchase_timestamp) AS days_between_purchasing_and_delivering,
  DATE_PART('day', order_delivered_customer_date - order_purchase_timestamp) AS days_between_purchasing_and_delivered,
  DATE_PART('day', order_estimated_delivery_date - order_purchase_timestamp) AS days_between_purchasing_and_estimated_delivery
FROM
  order_delivery;


-- end
-- not working
SELECT
    EXTRACT(YEAR FROM o.order_purchase_timestamp) AS year,
    EXTRACT(MONTH FROM o.order_purchase_timestamp) AS month,
    SUM(p.payment_value) AS total_payment_value_2017,
    SUM(CASE WHEN EXTRACT(YEAR FROM o.order_purchase_timestamp) = 2018 THEN p.payment_value ELSE 0 END) AS total_payment_value_2018,
    ((SUM(CASE WHEN EXTRACT(YEAR FROM o.order_purchase_timestamp) = 2018 THEN p.payment_value ELSE 0 END) - SUM(p.payment_value)) / SUM(p.payment_value)) * 100 AS percentage_increase
FROM
    target_business.orders o
JOIN
    target_business.payments p ON o.order_id = p.order_id
WHERE
    EXTRACT(YEAR FROM o.order_purchase_timestamp) IN (2017, 2018)
    AND EXTRACT(MONTH FROM o.order_purchase_timestamp) BETWEEN 1 AND 8
GROUP BY
    EXTRACT(YEAR FROM o.order_purchase_timestamp),
    EXTRACT(MONTH FROM o.order_purchase_timestamp)
ORDER BY
    EXTRACT(YEAR FROM o.order_purchase_timestamp),
    EXTRACT(MONTH FROM o.order_purchase_timestamp);


--
SELECT
    EXTRACT(YEAR FROM o.order_purchase_timestamp) AS year_2017,
    EXTRACT(YEAR FROM o.order_purchase_timestamp) AS year_2018,
    ROUND(((SUM(p1.payment_value) - SUM(p1.payment_value)) / SUM(p1.payment_value)) * 100, 2) AS
 percentage_increase
FROM
    -- target_business.payments p1
    target_business.orders o
JOIN
    target_business.payments p1 ON p1.order_id = o1.order_id
WHERE
    EXTRACT(YEAR FROM o.order_purchase_timestamp) = 2017
    AND EXTRACT(YEAR FROM o.order_purchase_timestamp) = 2018
    AND EXTRACT(MONTH FROM p1.order_purchase_timestamp) BETWEEN 1 AND 8
    AND EXTRACT(MONTH FROM p2.order_purchase_timestamp) BETWEEN 1 AND 8
GROUP BY
    EXTRACT(YEAR FROM o1.order_purchase_timestamp),
    EXTRACT(YEAR FROM o2.order_purchase_timestamp);

-
- This query joins the "payments" table with itself based on the "order_id" column, and filters
the results to include only orders made between January to August in both 2017 and 2018. It then
 -
- calculates the percentage increase in the total payment value for these orders, comparing the
sum of "payment_value" for 2018 with that of 2017, and rounds the result to two decimal places.
The final result is grouped by the year of purchase for both 2017 and 2018.

--
  SELECT
    review_score,
    review_comment_title

  FROM
    target_business.order_reviews
  ORDER BY
    review_score DESC;

--
-
- with an additional import statement for WordCloud from the wordcloud library. The WordCloud cl
ass is used for generating word clouds, which are visual representations of text data where the
size of each word represents its frequency or importance in the text.
  SELECT
    review_comment_title
  FROM
    target_business.order_reviews
  ORDER BY
    review_comment_title DESC;
    ---
---
    WITH review_orders AS (
  SELECT
    r.review_id,
    r.order_id,
    r.review_score,
    r.review_comment_title,
    --r.review_comment_message,
    r.review_creation_date,
    r.review_answer_timestamp,
    o.customer_id,
    o.order_status,
    o.order_purchase_timestamp,
    o.order_delivered_carrier_date,
    o.order_delivered_customer_date,
    o.order_estimated_delivery_date
  FROM
    target_business.order_reviews AS r
  JOIN
    target_business.orders AS o
  ON
    r.order_id = o.order_id
)
SELECT
  ro.review_id,
  ro.order_id,
  ro.review_score,
  ro.review_comment_title,
  --ro.review_comment_message,
  ro.review_creation_date,
  ro.review_answer_timestamp,
  ro.customer_id,
  ro.order_status,
  ro.order_purchase_timestamp,
  ro.order_delivered_carrier_date,
  ro.order_delivered_customer_date,
  ro.order_estimated_delivery_date
FROM
  review_orders AS ro;
```

Notebook:



Targeting Success: A Business Case Analysis of 100k Orders at Target in Brazil

by Emma Luk

In [1]:

```
# imports the necessary libraries for data analysis and visualisation in Python
import pandas as pd
import numpy as np
import matplotlib.pylab as plt
import seaborn as sns
# visual representations of text data
from wordcloud import WordCloud
plt.style.use('ggplot')
```

In [2]:

```
# loading dataset
df = pd.read_csv('review_comment_title.csv')
```

In [3]:

```
import pandas as pd
pd.set_option('display.max_columns', 500)
#pd.set_option('max_columns', 200)
```

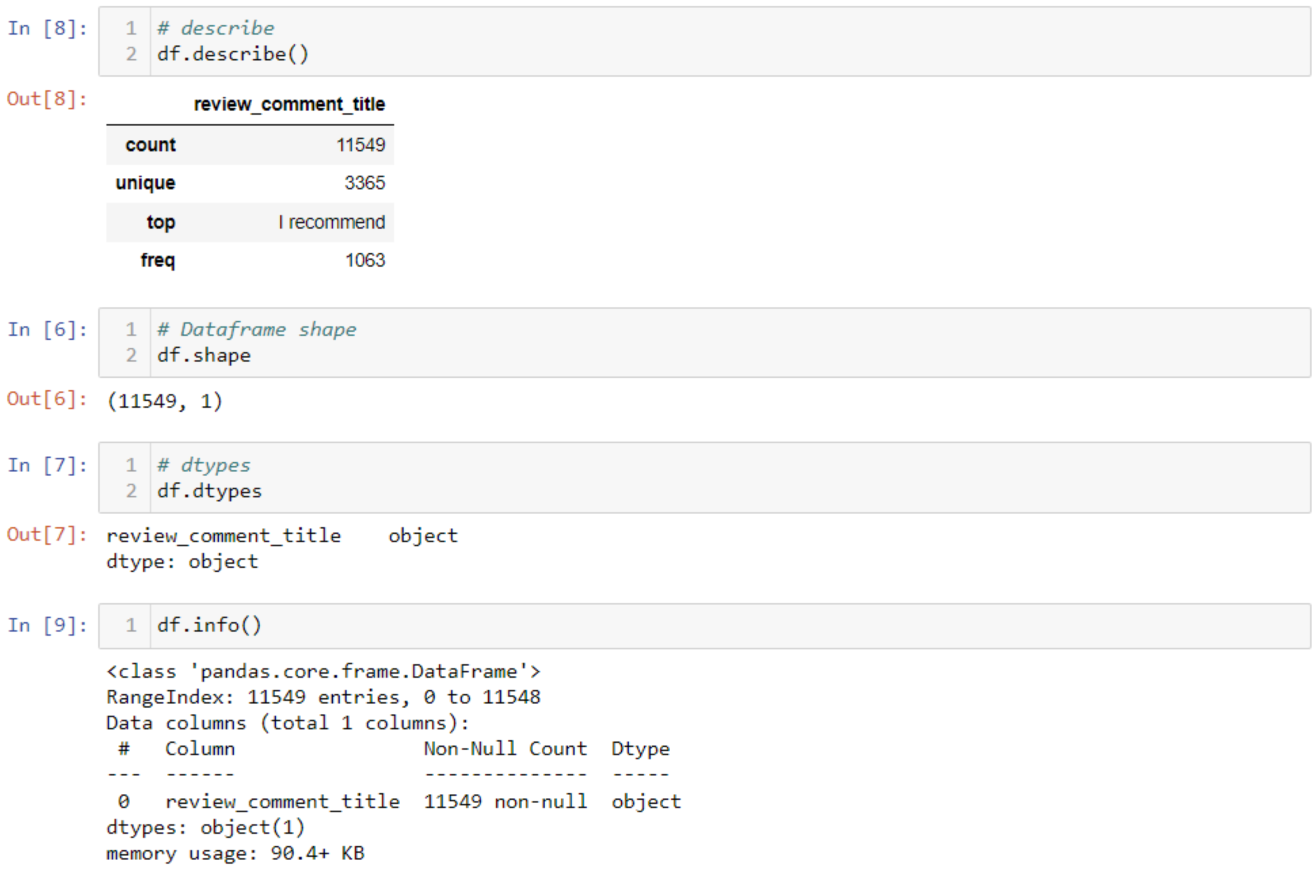
In [4]:

```
#present a DataFrame object in Python
df
```

Out[4]:

	review_comment_title
0	🔟
1	👍
2	👍�� _
3	👍
4	👍
...	...
11544	**
11545	**
11546	**
11547	**
11548	*

11549 rows × 1 columns



In [8]:
```python
# describe
df.describe()
```

Out[8]:

	review_comment_title
count	11549
unique	3365
top	I recommend
freq	1063

In [6]:
```python
# Dataframe shape
df.shape
```

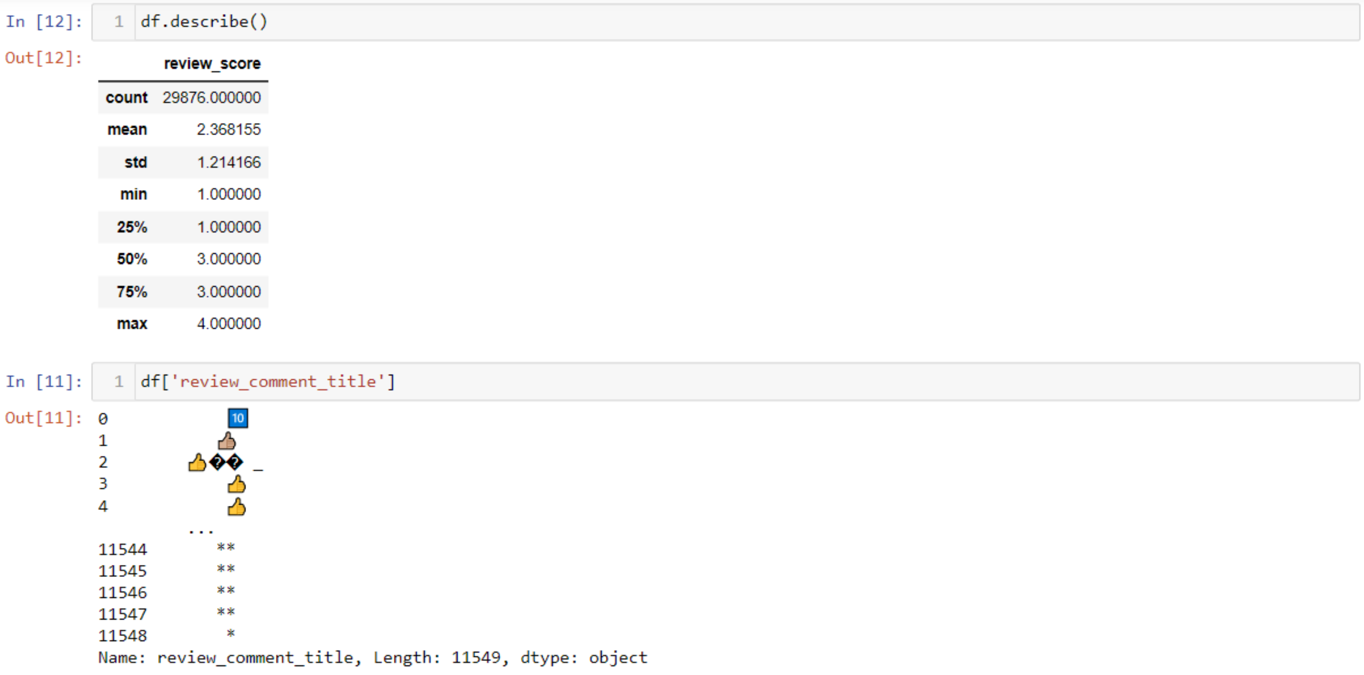
Out[6]:
```
(11549, 1)
```

In [7]:
```python
# dtypes
df.dtypes
```

Out[7]:
```
review_comment_title    object
dtype: object
```

In [9]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11549 entries, 0 to 11548
Data columns (total 1 columns):
 #   Column                Non-Null Count  Dtype 
---  ------                --------------  ----- 
 0   review_comment_title  11549 non-null  object
dtypes: object(1)
memory usage: 90.4+ KB
```



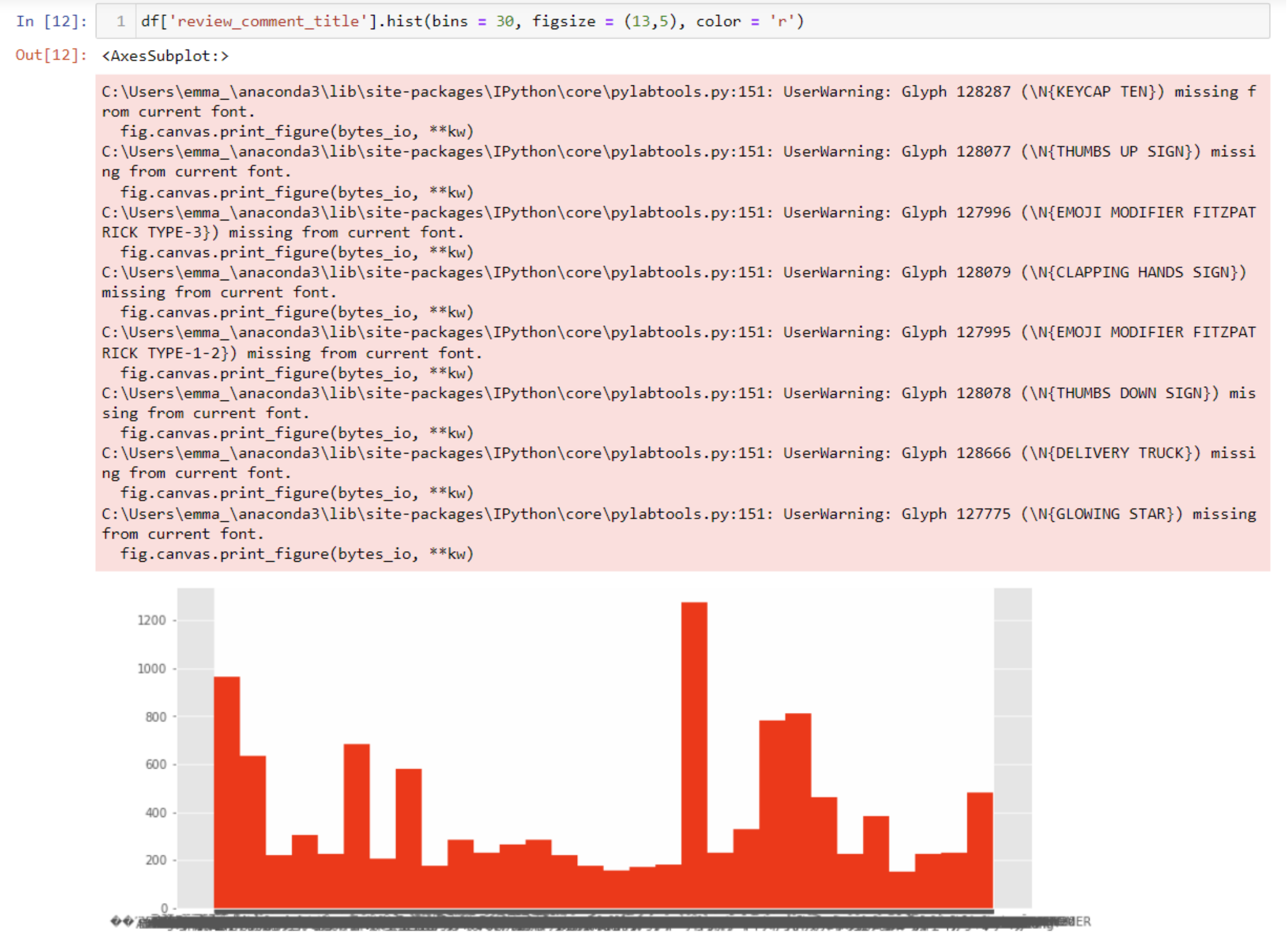
In [12]:
```python
df.describe()
```

Out[12]:

	review_score
count	29876.000000
mean	2.368155
std	1.214166
min	1.000000
25%	1.000000
50%	3.000000
75%	3.000000
max	4.000000

In [11]:
```python
df['review_comment_title']
```

Out[11]:
```
0           🔟
1           👍
2        👍�� _
3           👍
4           👍
          ...
11544       **
11545       **
11546       **
11547       **
11548        *
Name: review_comment_title, Length: 11549, dtype: object
```



```
In [12]: df['review_comment_title'].hist(bins = 30, figsize = (13,5), color = 'r')
```

```
Out[12]: <AxesSubplot:>
```

```
C:\Users\emma_\anaconda3\lib\site-packages\IPython\core\pylabtools.py:151: UserWarning: Glyph 128287 (\N{KEYCAP TEN}) missing f
rom current font.
  fig.canvas.print_figure(bytes_io, **kw)
C:\Users\emma_\anaconda3\lib\site-packages\IPython\core\pylabtools.py:151: UserWarning: Glyph 128077 (\N{THUMBS UP SIGN}) missi
ng from current font.
  fig.canvas.print_figure(bytes_io, **kw)
C:\Users\emma_\anaconda3\lib\site-packages\IPython\core\pylabtools.py:151: UserWarning: Glyph 127996 (\N{EMOJI MODIFIER FITZPAT
RICK TYPE-3}) missing from current font.
  fig.canvas.print_figure(bytes_io, **kw)
C:\Users\emma_\anaconda3\lib\site-packages\IPython\core\pylabtools.py:151: UserWarning: Glyph 128079 (\N{CLAPPING HANDS SIGN})
missing from current font.
  fig.canvas.print_figure(bytes_io, **kw)
C:\Users\emma_\anaconda3\lib\site-packages\IPython\core\pylabtools.py:151: UserWarning: Glyph 127995 (\N{EMOJI MODIFIER FITZPAT
RICK TYPE-1-2}) missing from current font.
  fig.canvas.print_figure(bytes_io, **kw)
C:\Users\emma_\anaconda3\lib\site-packages\IPython\core\pylabtools.py:151: UserWarning: Glyph 128078 (\N{THUMBS DOWN SIGN}) mis
sing from current font.
  fig.canvas.print_figure(bytes_io, **kw)
C:\Users\emma_\anaconda3\lib\site-packages\IPython\core\pylabtools.py:151: UserWarning: Glyph 128666 (\N{DELIVERY TRUCK}) missi
ng from current font.
  fig.canvas.print_figure(bytes_io, **kw)
C:\Users\emma_\anaconda3\lib\site-packages\IPython\core\pylabtools.py:151: UserWarning: Glyph 127775 (\N{GLOWING STAR}) missing
from current font.
  fig.canvas.print_figure(bytes_io, **kw)
```

```
In [13]: df['review_comment_title']
```

```
Out[13]: 0                🔟
         1              👍🏽
         2        👍�� _
         3                👍
         4                👍
                 ...
         11544       **
         11545       **
         11546       **
         11547       **
         11548        *
         Name: review_comment_title, Length: 11549, dtype: object
```

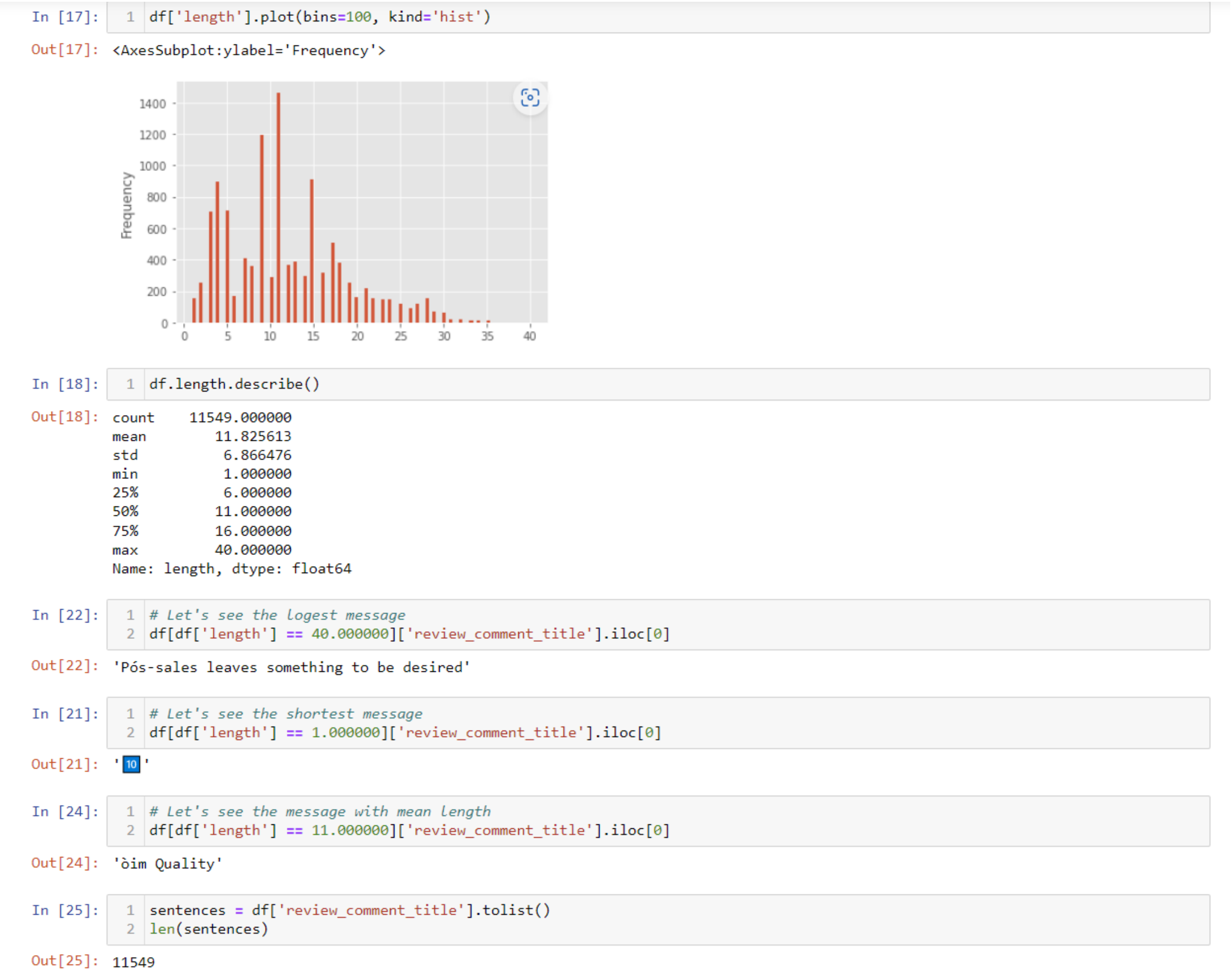
```
In [14]: sns.heatmap(df.isnull(), yticklabels = False, cbar = False, cmap="Blues")
```

```
Out[14]: <AxesSubplot:>
```

```
In [16]: df['length'] = df['review_comment_title'].apply(len)
         df.head()
```

Out[16]:

	review_comment_title	length
0	🔟	1
1	👍🏽	2
2	👍�� _	5
3	👍	1
4	👍	1



In [17]:
```
df['length'].plot(bins=100, kind='hist')
```
Out[17]:
```
<AxesSubplot:ylabel='Frequency'>
```

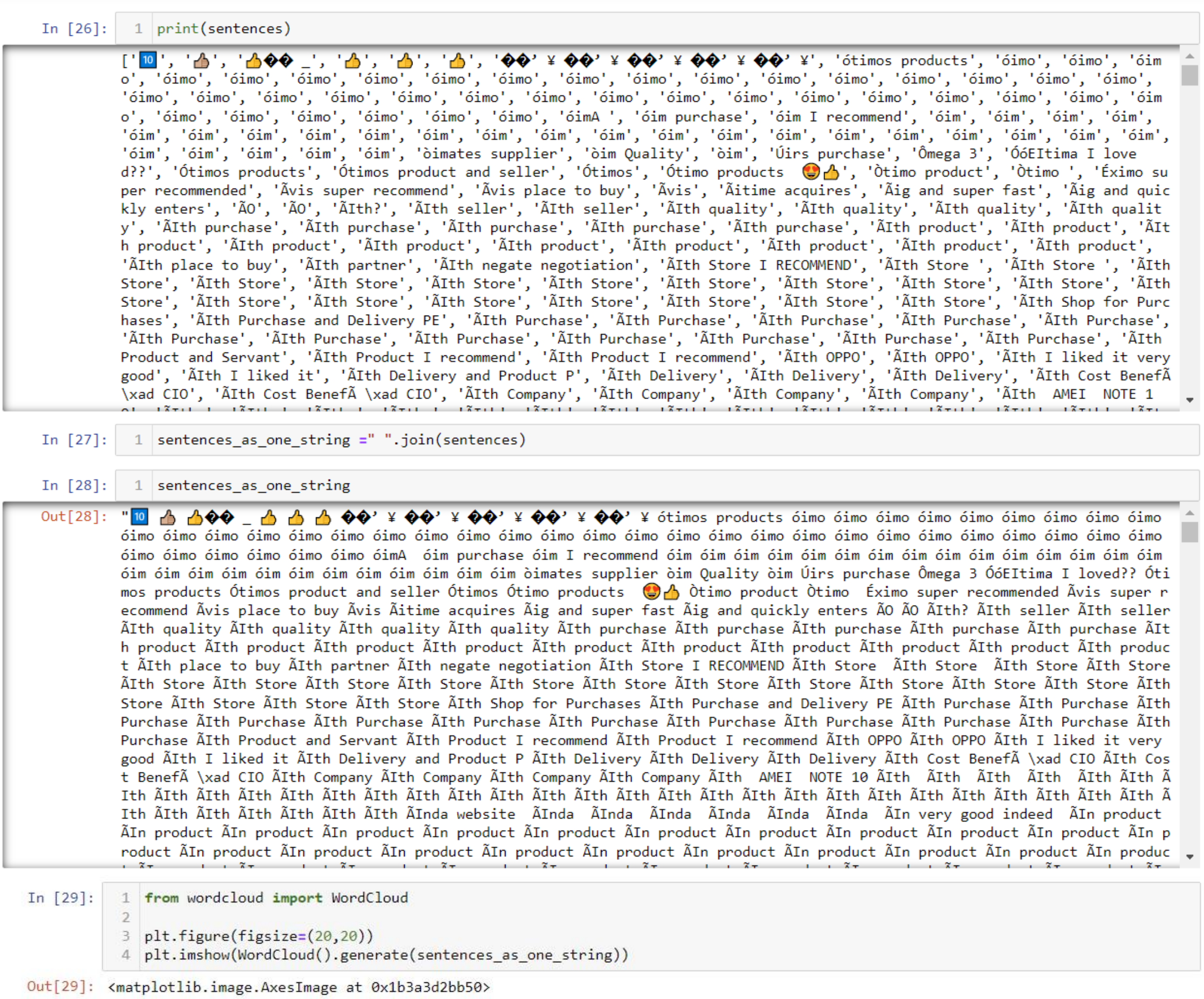
In [18]:
```
df.length.describe()
```
Out[18]:
```
count    11549.000000
mean        11.825613
std          6.866476
min          1.000000
25%          6.000000
50%         11.000000
75%         16.000000
max         40.000000
Name: length, dtype: float64
```

In [22]:
```
# Let's see the logest message
df[df['length'] == 40.000000]['review_comment_title'].iloc[0]
```
Out[22]:
```
'Pós-sales leaves something to be desired'
```

In [21]:
```
# Let's see the shortest message
df[df['length'] == 1.000000]['review_comment_title'].iloc[0]
```
Out[21]:
```
'🔟'
```

In [24]:
```
# Let's see the message with mean length
df[df['length'] == 11.000000]['review_comment_title'].iloc[0]
```
Out[24]:
```
'òim Quality'
```

In [25]:
```
sentences = df['review_comment_title'].tolist()
len(sentences)
```
Out[25]:
```
11549
```



```
In [26]:  print(sentences)
```

```
['🔟', '👍', '👍�� _', '👍', '👍', '👍', '��’ ¥ ��’ ¥ ��’ ¥ ��’ ¥ ��’ ¥', 'ótimos products', 'óimo', 'óimo', 'óim
o', 'óimo', 'óimo', 'óimo', 'óimo', 'óimo', 'óimo', 'óimo', 'óimo', 'óimo', 'óimo', 'óimo', 'óimo', 'óimo', 'óimo', 'óimo',
'óimo', 'óimo', 'óimo', 'óimo', 'óimo', 'óimo', 'óimo', 'óimo', 'óimo', 'óimo', 'óimo', 'óimo', 'óimo', 'óimo', 'óimo', 'óim
o', 'óimo', 'óimo', 'óimo', 'óimo', 'óimo', 'óimo', 'óimA ', 'óim purchase', 'óim I recommend', 'óim', 'óim', 'óim', 'óim',
'óim', 'óim', 'óim', 'óim', 'óim', 'óim', 'óim', 'óim', 'óim', 'óim', 'óim', 'óim', 'óim', 'óim', 'óim', 'óim', 'óim', 'óim',
'óim', 'óim', 'óim', 'óim', 'óim', 'òimates supplier', 'òim Quality', 'òim', 'Úirs purchase', 'Ômega 3', 'ÓóEItima I love
d??', 'Ótimos products', 'Ótimos product and seller', 'Ótimos', 'Ótimo products  😍👍', 'Òtimo product', 'Òtimo ', 'Éximo su
per recommended', 'Ãvis super recommend', 'Ãvis place to buy', 'Ãvis', 'Ãitime acquires', 'Ãig and super fast', 'Ãig and quic
kly enters', 'ÃO', 'ÃO', 'ÃIth?', 'ÃIth seller', 'ÃIth seller', 'ÃIth quality', 'ÃIth quality', 'ÃIth quality', 'ÃIth qualit
y', 'ÃIth purchase', 'ÃIth purchase', 'ÃIth purchase', 'ÃIth purchase', 'ÃIth purchase', 'ÃIth product', 'ÃIth product', 'ÃIt
h product', 'ÃIth product', 'ÃIth product', 'ÃIth product', 'ÃIth product', 'ÃIth product', 'ÃIth product', 'ÃIth product',
'ÃIth place to buy', 'ÃIth partner', 'ÃIth negate negotiation', 'ÃIth Store I RECOMMEND', 'ÃIth Store ', 'ÃIth Store ', 'ÃIth
Store', 'ÃIth Store', 'ÃIth Store', 'ÃIth Store', 'ÃIth Store', 'ÃIth Store', 'ÃIth Store', 'ÃIth Store', 'ÃIth Store', 'ÃIth
Store', 'ÃIth Store', 'ÃIth Store', 'ÃIth Store', 'ÃIth Store', 'ÃIth Store', 'ÃIth Store', 'ÃIth Store', 'ÃIth Shop for Purc
hases', 'ÃIth Purchase and Delivery PE', 'ÃIth Purchase', 'ÃIth Purchase', 'ÃIth Purchase', 'ÃIth Purchase', 'ÃIth Purchase',
'ÃIth Purchase', 'ÃIth Purchase', 'ÃIth Purchase', 'ÃIth Purchase', 'ÃIth Purchase', 'ÃIth Purchase', 'ÃIth Purchase', 'ÃIth
Product and Servant', 'ÃIth Product I recommend', 'ÃIth Product I recommend', 'ÃIth OPPO', 'ÃIth OPPO', 'ÃIth I liked it very
good', 'ÃIth I liked it', 'ÃIth Delivery and Product P', 'ÃIth Delivery', 'ÃIth Delivery', 'ÃIth Delivery', 'ÃIth Cost BenefÃ
\xad CIO', 'ÃIth Cost BenefÃ \xad CIO', 'ÃIth Company', 'ÃIth Company', 'ÃIth Company', 'ÃIth Company', 'ÃIth  AMEI  NOTE 1
```

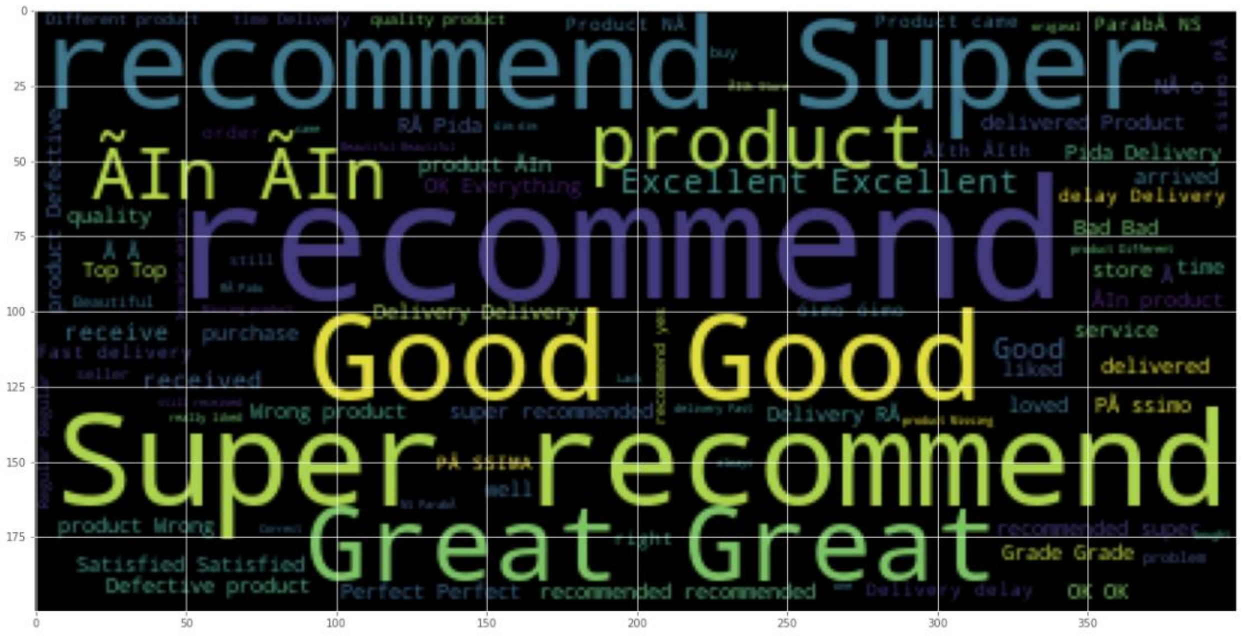
```
In [27]:  sentences_as_one_string =" ".join(sentences)
```

```
In [28]:  sentences_as_one_string
```

```
Out[28]: "🔟 👍 👍�� _ 👍 👍 👍 ��’ ¥ ��’ ¥ ��’ ¥ ��’ ¥ ��’ ¥ ótimos products óimo óimo óimo óimo óimo óimo óimo óimo óimo
óimo óimo óimo óimo óimo óimo óimo óimo óimo óimo óimo óimo óimo óimo óimo óimo óimo óimo óimo óimo óimo óimo óimo óimo óimo óimo
óimo óimo óimo óimo óimo óimo óimA  óim purchase óim I recommend óim óim óim óim óim óim óim óim óim óim óim óim óim óim óim
óim óim óim óim óim óim óim óim óim óim óim óim òimates supplier òim Quality òim Úirs purchase Ômega 3 ÓóEItima I loved?? Óti
mos products Ótimos product and seller Ótimos Ótimo products  😍👍 Òtimo product Òtimo  Éximo super recommended Ãvis super r
ecommend Ãvis place to buy Ãvis Ãitime acquires Ãig and super fast Ãig and quickly enters ÃO ÃO ÃIth? ÃIth seller ÃIth seller
ÃIth quality ÃIth quality ÃIth quality ÃIth quality ÃIth purchase ÃIth purchase ÃIth purchase ÃIth purchase ÃIth purchase ÃIt
h product ÃIth product ÃIth product ÃIth product ÃIth product ÃIth product ÃIth product ÃIth product ÃIth product ÃIth produc
t ÃIth place to buy ÃIth partner ÃIth negate negotiation ÃIth Store I RECOMMEND ÃIth Store  ÃIth Store  ÃIth Store ÃIth Store
ÃIth Store ÃIth Store ÃIth Store ÃIth Store ÃIth Store ÃIth Store ÃIth Store ÃIth Store ÃIth Store ÃIth Store ÃIth Store ÃIth
Store ÃIth Store ÃIth Store ÃIth Store ÃIth Shop for Purchases ÃIth Purchase and Delivery PE ÃIth Purchase ÃIth Purchase ÃIth
Purchase ÃIth Purchase ÃIth Purchase ÃIth Purchase ÃIth Purchase ÃIth Purchase ÃIth Purchase ÃIth Purchase ÃIth Purchase ÃIth
Purchase ÃIth Product and Servant ÃIth Product I recommend ÃIth Product I recommend ÃIth OPPO ÃIth OPPO ÃIth I liked it very
good ÃIth I liked it ÃIth Delivery and Product P ÃIth Delivery ÃIth Delivery ÃIth Delivery ÃIth Cost BenefÃ \xad CIO ÃIth Cos
t BenefÃ \xad CIO ÃIth Company ÃIth Company ÃIth Company ÃIth Company ÃIth  AMEI  NOTE 10 ÃIth  ÃIth  ÃIth  ÃIth  ÃIth ÃIth Ã
Ith ÃIth ÃIth ÃIth ÃIth ÃIth ÃIth ÃIth ÃIth ÃIth ÃIth ÃIth ÃIth ÃIth ÃIth ÃIth ÃIth ÃIth ÃIth ÃIth ÃIth ÃIth ÃIth ÃIth ÃIth Ã
Ith ÃIth ÃIth ÃIth ÃIth ÃIth ÃIth ÃInda website  ÃInda  ÃInda  ÃInda  ÃInda  ÃInda  ÃInda  ÃIn very good indeed  ÃIn product
ÃIn product ÃIn product ÃIn product ÃIn product ÃIn product ÃIn product ÃIn product ÃIn product ÃIn product ÃIn product ÃIn p
roduct ÃIn product ÃIn product ÃIn product ÃIn product ÃIn product ÃIn product ÃIn product ÃIn product ÃIn product ÃIn produc
```

```
In [29]:  from wordcloud import WordCloud

          plt.figure(figsize=(20,20))
          plt.imshow(WordCloud().generate(sentences_as_one_string))
```

```
Out[29]: <matplotlib.image.AxesImage at 0x1b3a3d2bb50>
```


recommend
Super
product
ÃIn ÃIn
Excellent Excellent
recommend
Good Good
Super recommend
Great Great
Different product
time Delivery
quality product
Product NÃ
Product came
ParabÃ NS
delivered Product
Pida Delivery
arrived
delay Delivery
Bad Bad
store
time
ÃIn product
service
delivered
loved
PÃ ssimo
Good
liked
Grade Grade
OK OK
Delivery delay
recommended recommended
Perfect Perfect
Satisfied Satisfied
Defective product
product Wrong
Wrong product
super recommended
Delivery RÃ
receive
purchase
received
Fast delivery
seller
Delivery Delivery
quality
Top Top
Beautiful
still
OK Everything
product Ain
RÃ Pida
order
PÃ SSIMA
well
Defective product