In [1]:
```
import pandas as pd
import pymysql as mysql
from sqlalchemy import create_engine,text
```

In [2]:
```
pd.set_option("display.max_rows",None)
pd.set_option("display.max_columns",None)
```

In [3]:
```
engine=create_engine("mysql+pymysql://root:123456@127.0.0.1:3306/Olist_db",cor
conn=engine.connect()
```

In [6]:
```
query=text(""" SHOW VARIABLES LIKE 'local_infile';""")
pd.read_sql_query(query,conn)
```

Out[6]:

	Variable_name	Value
0	local_infile	OFF

In [8]:
```
query=text("""SET GLOBAL local_infile=1;""")
conn.execute(query)
conn.commit()
```

In [5]:
```
query=text(""" SHOW VARIABLES LIKE 'local_infile';""")
pd.read_sql_query(query,conn)
```

Out[5]:

	Variable_name	Value
0	local_infile	ON

Data Loading

- Bulk data is loaded into staging tables using MySQL LOAD DATA LOCAL INFILE for high-performance ingestion.

customers_data

In [6]:
```
query=text("""SET FOREIGN_KEY_CHECKS = 0;""")
conn.execute(query)
conn.commit()
```

In [7]:
```
check_table_exists_query=text("""
                        SELECT COUNT(*) FROM information_schema.tables
                        WHERE table_schema='olist_db'
                        AND table_name='customers_data';
                      """)

if conn.execute(check_table_exists_query).scalar()==1:
    conn.execute(text("""TRUNCATE TABLE customers_data;"""))
```

```
    conn.commit()
    print("==========All Rows Deleted Successfully✅==========")
else:
    print("==========Table Does Not Exists❌==========")
```

```
==========Table Does Not Exists❌==========
```

In [8]:
```
check_table_exists_query=text("""
                        SELECT COUNT(*) FROM information_schema.tables
                        WHERE table_schema='olist_db'
                        AND table_name='customers_data';
                    """)

if conn.execute(check_table_exists_query).scalar()==0:
    conn.execute(text("""CREATE TABLE customers_data(
                            customer_id VARCHAR(255),
                            customer_unique_id VARCHAR(255),
                            customer_zip_code_prefix INTEGER,
                            customer_city VARCHAR(255),
                            customer_state VARCHAR(255));
                            """))
    conn.commit()
    print("==========Table Created Successfully✅==========")
else:
    print("==========Table Already Exists❌==========")
```

```
==========Table Created Successfully✅==========
```

In [10]:
```
query=text("""SET FOREIGN_KEY_CHECKS = 0;""")
conn.execute(query)
conn.commit()
```

In [11]:
```
query = text("""
LOAD DATA LOCAL INFILE 'D:/Eda project on Olist dataset by Sql/Olist Datasets/
INTO TABLE customers_data
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(customer_id, customer_unique_id, customer_zip_code_prefix, customer_city, cus
""")
conn.execute(query)
conn.commit()
```

In [13]:
```
query=text("""SET FOREIGN_KEY_CHECKS = 1;""")
conn.execute(query)
conn.commit()
```

geolocation_data

In [14]:
```
check_table_exists_query=text("""
                        SELECT COUNT(*) FROM information_schema.tables
```

```
                        WHERE table_schema='olist_db'
                        AND table_name='geolocation_data';
                    """)

if conn.execute(check_table_exists_query).scalar()==1:
    conn.execute(text("""TRUNCATE TABLE geolocation_data;"""))
    conn.commit()
    print("==========All Rows Deleted Successfully✅==========")
else:
    print("==========Table Does Not Exists❌==========")
```

```
==========Table Does Not Exists❌==========
```

In [15]:
```
check_table_exists_query=text("""
                        SELECT COUNT(*) FROM information_schema.tables
                        WHERE table_schema='olist_db'
                        AND table_name='geolocation_data';
                    """)

if conn.execute(check_table_exists_query).scalar()==0:
    conn.execute(text("""CREATE TABLE geolocation_data(
                            geolocation_zip_code_prefix INTEGER,
                            geolocation_lat DOUBLE,
                            geolocation_lng DOUBLE,
                            geolocation_city VARCHAR(255),
                            geolocation_state VARCHAR(255));
                            """))
    conn.commit()
    print("==========Table Created Successfully✅==========")
else:
    print("==========Table Already Exists❌==========")
```

```
==========Table Created Successfully✅==========
```

In [17]:
```
query=text("""SET FOREIGN_KEY_CHECKS = 0;""")
conn.execute(query)
conn.commit()
```

In [18]:
```
query=text("""LOAD DATA LOCAL INFILE 'D:/Eda project on Olist dataset by Sql/C
              INTO TABLE geolocation_data
              FIELDS TERMINATED BY ','
              ENCLOSED BY '"'
              LINES TERMINATED BY '\n'
              IGNORE 1 ROWS
              (geolocation_zip_code_prefix,geolocation_lat,geolocation_lng,geo
conn.execute(query)
conn.commit()
```

In [20]:
```
query=text("""SET FOREIGN_KEY_CHECKS = 1;""")
conn.execute(query)
conn.commit()
```

order_items_data

In [21]:
```
check_table_exists_query=text("""
                        SELECT COUNT(*) FROM information_schema.tables
                        WHERE table_schema='olist_db'
                        AND table_name='order_items_data';
                    """)

if conn.execute(check_table_exists_query).scalar()==1:
    conn.execute(text("""TRUNCATE TABLE order_items_data;"""))
    conn.commit()
    print("==========All Rows Deleted Successfully✅==========")
else:
    print("==========Table Does Not Exists❌==========")
```

```
==========Table Does Not Exists❌==========
```

In [22]:
```
check_table_exists_query=text("""
                        SELECT COUNT(*) FROM information_schema.tables
                        WHERE table_schema='olist_db'
                        AND table_name='order_items_data';
                    """)

if conn.execute(check_table_exists_query).scalar()==0:
    conn.execute(text("""CREATE TABLE order_items_data(
                                            order_id VARCHAR(255),
                                            order_item_id INTEGER,
                                            product_id VARCHAR(255),
                                            seller_id VARCHAR(255),
                                            shipping_limit_date DATETIME,
                                            price DOUBLE,
                                            freight_value DOUBLE)"""))
    conn.commit()
    print("==========Table Created Successfully✅==========")
else:
    print("==========Table Already Exists❌==========")
```

```
==========Table Created Successfully✅==========
```

In [24]:
```
query=text("""SET FOREIGN_KEY_CHECKS = 0;""")
conn.execute(query)
conn.commit()
```

In [25]:
```
query=text("""LOAD DATA LOCAL INFILE 'D:/Eda project on Olist dataset by Sql/C
              INTO TABLE order_items_data
              FIELDS TERMINATED BY ','
              ENCLOSED BY '"'
              LINES TERMINATED BY '\n'
              IGNORE 1 ROWS
              (order_id,order_item_id,product_id,seller_id,shipping_limit_date
conn.execute(query)
conn.commit()
```

In [27]:
```
query=text("""SET FOREIGN_KEY_CHECKS = 1;""")
conn.execute(query)
conn.commit()
```

order_payments_data

In [28]:
```
check_table_exists_query=text("""
                        SELECT COUNT(*) FROM information_schema.tables
                        WHERE table_schema='olist_db'
                        AND table_name='order_payments_data';
                      """)

if conn.execute(check_table_exists_query).scalar()==1:
    conn.execute(text("""TRUNCATE TABLE order_payments_data;"""))
    conn.commit()
    print("==========All Rows Deleted Successfully✅==========")
else:
    print("==========Table Does Not Exists❌==========")
```

```
==========Table Does Not Exists❌==========
```

In [29]:
```
check_table_exists_query=text("""SELECT COUNT(*) FROM information_schema.table
                          WHERE table_schema='olist_db'
                          AND table_name='order_payments_data';""")
if conn.execute(check_table_exists_query).scalar()==0:
    conn.execute(text("""CREATE TABLE order_payments_data(
                           order_id VARCHAR(255),
                           payment_sequential INTEGER,
                           payment_type VARCHAR(255),
                           payment_installments INTEGER,
                           payment_value DOUBLE);"""))
    conn.commit()
    print("==========Table Created Successfully✅==========")
else:
    print("==========Table Already Exists❌==========")
```

```
==========Table Created Successfully✅==========
```

In [31]:
```
query=text("""SET FOREIGN_KEY_CHECKS = 0;""")
conn.execute(query)
conn.commit()
```

In [32]:
```
query=text("""LOAD DATA LOCAL INFILE 'D:/Eda project on Olist dataset by Sql/C
              INTO TABLE order_payments_data
              FIELDS TERMINATED BY ','
              ENCLOSED BY '"'
              LINES TERMINATED BY '\n'
              IGNORE 1 ROWS
              (order_id,payment_sequential,payment_type,payment_installments,p
conn.execute(query)
conn.commit()
```

In [34]:
```
query=text("""SET FOREIGN_KEY_CHECKS = 1;""")
conn.execute(query)
conn.commit()
```

order_reviews_data

In [35]:
```
check_table_exists_query=text("""
                        SELECT COUNT(*) FROM information_schema.tables
                        WHERE table_schema='olist_db'
                        AND table_name='order_reviews_data';
                    """)

if conn.execute(check_table_exists_query).scalar()==1:
    conn.execute(text("""TRUNCATE TABLE order_reviews_data;"""))
    conn.commit()
    print("==========All Rows Deleted Successfully✅==========")
else:
    print("==========Table Does Not Exists❌==========")
```

```
==========Table Does Not Exists❌==========
```

In [36]:
```
check_table_exists_query=text("""SELECT COUNT(*) FROM information_schema.table
                          WHERE table_schema='olist_db'
                          AND table_name='order_reviews_data';""")

if conn.execute(check_table_exists_query).scalar()==0:
    conn.execute(text("""CREATE TABLE order_reviews_data(

                           review_id VARCHAR(255),
                           order_id VARCHAR(255),
                           review_score INTEGER,
                           review_comment_title TEXT,
                           review_comment_message TEXT,
                           review_creation_date DATETIME,
                           review_answer_timestamp DATETIME
                           );""""))
    conn.commit()
    print("==========Table Created Successfully✅==========")
else:
    print("==========Table Already Exists❌==========")
```

```
==========Table Created Successfully✅==========
```

In [38]:
```
query=text("""SET FOREIGN_KEY_CHECKS = 0;""")
conn.execute(query)
conn.commit()
```

In [39]:
```
query=text("""LOAD DATA LOCAL INFILE 'D:/Eda project on Olist dataset by Sql/C
              INTO TABLE order_reviews_data
              FIELDS TERMINATED BY ','
              ENCLOSED BY '"'
              LINES TERMINATED BY '\n'
```

```
            IGNORE 1 ROWS
            (review_id,order_id,review_score,review_comment_title,review_com
conn.execute(query)
conn.commit()
```

In [41]:
```
query=text("""SET FOREIGN_KEY_CHECKS = 1;""")
conn.execute(query)
conn.commit()
```

orders_data

In [42]:
```
check_table_exists_query=text("""
                        SELECT COUNT(*) FROM information_schema.tables
                        WHERE table_schema='olist_db'
                        AND table_name='orders_data';
                    """)

if conn.execute(check_table_exists_query).scalar()==1:
    conn.execute(text("""TRUNCATE TABLE orders_data;"""))
    conn.commit()
    print("==========All Rows Deleted Successfully✅==========")
else:
    print("==========Table Does Not Exists❌==========")
```

```
==========Table Does Not Exists❌==========
```

In [43]:
```
check_table_exists_query=text("""SELECT COUNT(*) FROM information_schema.table
                          WHERE table_schema='olist_db'
                          AND table_name='orders_data';""")

if conn.execute(check_table_exists_query).scalar()==0:
    conn.execute(text("""CREATE TABLE orders_data(
                             order_id VARCHAR(255),
                             customer_id VARCHAR(255),
                             order_status VARCHAR(255),
                             order_purchase_timestamp DATETIME,
                             order_approved_at DATETIME,
                             order_delivered_carrier_date DATETIME,
                             order_delivered_customer_date DATETIME,
                             order_estimated_delivery_date DATETIME
                             );""""))
    conn.commit()
    print("==========Table Created Successfully✅==========")
else:
    print("==========Table Already Exists❌==========")
```

```
==========Table Created Successfully✅==========
```

In [45]:
```
query=text("""SET FOREIGN_KEY_CHECKS = 0;""")
conn.execute(query)
conn.commit()
```

In [46]:
```
query=text("""LOAD DATA LOCAL INFILE 'D:/Eda project on Olist dataset by Sql/C
              INTO TABLE orders_data
              FIELDS TERMINATED BY ','
              ENCLOSED BY '"'
              LINES TERMINATED BY '\n'
              IGNORE 1 ROWS
              (order_id,customer_id,order_status,order_purchase_timestamp,orde
conn.execute(query)
conn.commit()
```

In [47]:
```
query=text("""SET FOREIGN_KEY_CHECKS = 1;""")
conn.execute(query)
conn.commit()
```

products_data

In [49]:
```
check_table_exists_query=text("""
                        SELECT COUNT(*) FROM information_schema.tables
                        WHERE table_schema='olist_db'
                        AND table_name='products_data';
                    """)

if conn.execute(check_table_exists_query).scalar()==1:
    conn.execute(text("""TRUNCATE TABLE products_data;"""))
    conn.commit()
    print("==========All Rows Deleted Successfully✅==========")
else:
    print("==========Table Does Not Exists❌==========")
```

```
==========Table Does Not Exists❌==========
```

In [50]:
```
check_table_exists_query=text("""SELECT COUNT(*) FROM information_schema.table
                          WHERE table_schema='olist_db'
                          AND table_name='products_data';""")

if conn.execute(check_table_exists_query).scalar()==0:
    conn.execute(text("""CREATE TABLE products_data(
                             product_id VARCHAR(255),
                             product_category_name VARCHAR(255),
                             product_name_lenght INTEGER,
                             product_description_lenght INTEGER,
                             product_photos_qty INTEGER,
                             product_weight_g INTEGER,
                             product_length_cm INTEGER,
                             product_height_cm INTEGER,
                             product_width_cm INTEGER
                             );""""))
    conn.commit()
    print("==========Table Created Successfully✅==========")
else:
    print("==========Table Already Exists❌==========")
```

```
==========Table Created Successfully✅==========
```

In [52]:
```
query=text("""SET FOREIGN_KEY_CHECKS = 0;""")
conn.execute(query)
conn.commit()
```

In [53]:
```
query=text("""LOAD DATA LOCAL INFILE 'D:/Eda project on Olist dataset by Sql/C
              INTO TABLE products_data
              FIELDS TERMINATED BY ','
              ENCLOSED BY '"'
              LINES TERMINATED BY '\n'
              IGNORE 1 ROWS
              (product_id,product_category_name,product_name_lenght,product_de
conn.execute(query)
conn.commit()
```

In [55]:
```
query=text("""SET FOREIGN_KEY_CHECKS = 1;""")
conn.execute(query)
conn.commit()
```

sellers_data

In [56]:
```
check_table_exists_query=text("""
                        SELECT COUNT(*) FROM information_schema.tables
                        WHERE table_schema='olist_db'
                        AND table_name='sellers_data';
                    """)

if conn.execute(check_table_exists_query).scalar()==1:
    conn.execute(text("""TRUNCATE TABLE sellers_data;"""))
    conn.commit()
    print("==========All Rows Deleted Successfully✅==========")
else:
    print("==========Table Does Not Exists❌==========")
```

```
==========Table Does Not Exists❌==========
```

In [57]:
```
check_table_exists_query=text("""SELECT COUNT(*) FROM information_schema.table
                          WHERE table_schema='olist_db'
                          AND table_name='sellers_data';""")

if conn.execute(check_table_exists_query).scalar()==0:
    conn.execute(text("""CREATE TABLE sellers_data(
                           seller_id VARCHAR(255),
                           seller_zip_code_prefix INTEGER,
                           seller_city VARCHAR(255),
                           seller_state VARCHAR(255)
                           );""""))
    conn.commit()
    print("==========Table Created Successfully✅==========")
else:
    print("==========Table Already Exists❌==========")
```

```
```

```
==========Table Created Successfully✅==========
```

In [59]:
```
query=text("""SET FOREIGN_KEY_CHECKS = 0;""")
conn.execute(query)
conn.commit()
```

In [60]:
```
query=text("""LOAD DATA LOCAL INFILE 'D:/Eda project on Olist dataset by Sql/C
              INTO TABLE sellers_data
              FIELDS TERMINATED BY ','
              ENCLOSED BY '"'
              LINES TERMINATED BY '\n'
              IGNORE 1 ROWS
              (seller_id,seller_zip_code_prefix,seller_city,seller_state);""")
conn.execute(query)
conn.commit()
```

In [62]:
```
query=text("""SET FOREIGN_KEY_CHECKS = 1;""")
conn.execute(query)
conn.commit()
```

products_categories_data

In [63]:
```
check_table_exists_query=text("""
                        SELECT COUNT(*) FROM information_schema.tables
                        WHERE table_schema='olist_db'
                        AND table_name='products_categories_data';
                    """)

if conn.execute(check_table_exists_query).scalar()==1:
    conn.execute(text("""TRUNCATE TABLE products_categories_data;"""))
    conn.commit()
    print("==========All Rows Deleted Successfully✅==========")
else:
    print("==========Table Does Not Exists❌==========")
```

```
==========Table Does Not Exists❌==========
```

In [64]:
```
check_table_exists_query=text("""SELECT COUNT(*) FROM information_schema.table
                          WHERE table_schema='olist_db'
                          AND table_name='products_categories_data';""")

if conn.execute(check_table_exists_query).scalar()==0:
    conn.execute(text("""CREATE TABLE products_categories_data(
                           product_category_name VARCHAR(255),
                           product_category_name_english VARCHAR(255)
                           );""""))
    conn.commit()
    print("==========Table Created Successfully✅==========")
else:
    print("==========Table Already Exists❌==========")
```

```
==========Table Created Successfully✅==========
```

In [66]:
```
query=text("""SET FOREIGN_KEY_CHECKS = 0;""")
conn.execute(query)
conn.commit()
```

In [67]:
```
query=text("""LOAD DATA LOCAL INFILE 'D:/Eda project on Olist dataset by Sql/C
INTO TABLE products_categories_data
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\r\n'
IGNORE 1 ROWS
(product_category_name, product_category_name_english);""")
conn.execute(query)
conn.commit()
```

In [69]:
```
query=text("""SET FOREIGN_KEY_CHECKS = 1;""")
conn.execute(query)
conn.commit()
```

Data Understanding

customers_data

Observation

- The dataset contains 99,441 rows and 5 columns.
- `customer_unique_id` represents an actual customer and may appear multiple times in the dataset.
- `customer_id` is an order-level identifier, generated at the time of purchase, and serves as the primary key in the customers table.
- A single `customer_unique_id` can be associated with multiple customer_id values due to multiple orders and/or address changes over time.
- The `customer_state` column stores state abbreviations, which are not easily interpretable without a reference mapping.
- The `customer_city` column is not stored in a consistent format.
- No NULL or blank values are present in the dataset.
- No leading or trailing whitespace is present in any string column.
- No accented characters were detected in the `customer_city` column.
- No duplicate rows were identified.
- There are `278 orphan customer_zip_code_prefix` rows in the child table(customers_data) related to parent table(geolocation_data).

In [70]:
```
query=text("""DESCRIBE customers_data;""")
pd.read_sql_query(query,conn)
```

Out[70]:

	Field	Type	Null	Key	Default	Extra
0	customer_id	varchar(255)	YES		None	
1	customer_unique_id	varchar(255)	YES		None	
2	customer_zip_code_prefix	int(11)	YES		None	
3	customer_city	varchar(255)	YES		None	
4	customer_state	varchar(255)	YES		None	

In [71]:
```
query=text("""SELECT COUNT(*) AS total_rows FROM customers_data;""")
pd.read_sql_query(query,conn)
```

Out[71]:

	total_rows
0	99441

In [72]:
```
query=text("""SELECT COUNT(*) AS total_columns FROM information_schema.columns
              WHERE table_schema='Olist_db'
              AND table_name='customers_data'""")
pd.read_sql_query(query,conn)
```

Out[72]:

	total_columns
0	5

In [73]:
```
query=text("""SELECT * FROM customers_data
             ORDER BY customer_unique_id
             LIMIT 10;""")
pd.read_sql_query(query,conn)
```

Out[73]:

	customer_id	customer_unique_id	custo
0	fadbb3709178fc513abc1b2670aa1ad2	0000366f3b9a7992bf8c76cfdf3221e2	
1	4cb282e167ae9234755102258dd52ee8	0000b849f77a49e4a4ce2b2a4ca5be3f	
2	9b3932a6253894a02c1df9d19004239f	0000f46a3911fa3c0805444483337064	
3	914991f0c02ef0843c0e7010c819d642	0000f6ccb0745a6a4b88665a16c9f078	
4	47227568b10f5f58a524a75507e6992c	0004aac84e0df4da2b147fca70cf8255	
5	4a913a170c26e3c8052ed0202849b5a8	0004bd2a26a76fe21f786e4fbd80607f	
6	d2509c13692836fc0449e88cf9eb4858	00050ab1314c0e55a6ca13cf7181fecf	
7	a81ebb9b32f102298c0c89635b4b3154	00053a61a98854899e70ed204dd4bafe	
8	3b37fb626fdf46cd99d37ec62afa88ff	0005e1862207bf6ccc02e4228effd9a0	
9	c59e8ff99836e90d8b457d4122dc34e9	0005ef4cd20d2893f0d9fbd94d3c0d97	

In [74]:
```
query=text("""SELECT customer_id,
                     COUNT(*)
              FROM customers_data
              GROUP BY customer_id
              HAVING COUNT(*)>1
              LIMIT 10;""")
pd.read_sql_query(query,conn)
```

Out[74]:

	customer_id	COUNT(*)

In [75]:
```
query=text("""SELECT customer_id,
                     customer_unique_id,
                     COUNT(*)
              FROM customers_data
              GROUP BY customer_id,
                       customer_unique_id
              HAVING COUNT(*)>1
              LIMIT 10;""")
pd.read_sql_query(query,conn)
```

Out[75]:

	customer_id	customer_unique_id	COUNT(*)

In [76]:
```
check_index_exists_query = text("""
SELECT COUNT(*)
FROM information_schema.statistics
WHERE table_schema = 'olist_db'
  AND table_name = 'customers_data'
  AND index_name = 'customer_data_customer_id_customer_unique_id_idx';
""")

if conn.execute(check_index_exists_query).scalar() == 0:
```

```
    conn.execute(text("""
        CREATE INDEX customer_data_customer_id_customer_unique_id_idx
        ON customers_data(customer_id, customer_unique_id);
    """))
    conn.commit()
    print("==========Index Created Successfully✅==========")
else:
    print("==========Index Already Exists❌==========")
```

```
==========Index Created Successfully✅==========
```

In [77]:
```
query=text("""SELECT CAST(IFNULL(SUM(duplicate_count) - COUNT(*), 0) AS SIGNED
              FROM (SELECT COUNT(*) AS duplicate_count
                    FROM customers_data
                    GROUP BY customer_id,
                             customer_unique_id
                    HAVING COUNT(*)>1) T;""")
pd.read_sql_query(query,conn)
```

Out[77]:

	total_duplicate_rows
0	0

In [78]:
```
query=text("""SELECT customer_city FROM customers_data
              WHERE customer_city REGEXP '[áàâãäéèêëíìîïóòôõöúùûüçñ]'
              LIMIT 10;""")
pd.read_sql_query(query,conn)
```

Out[78]:

	customer_city

In [79]:
```
query=text("""SELECT SUM(CASE WHEN customer_id IS NULL THEN 1 ELSE 0 END) AS c
                     SUM(CASE WHEN customer_unique_id IS NULL THEN 1 ELSE 0 EN
                     SUM(CASE WHEN customer_zip_code_prefix IS NULL THEN 1 ELS
                     SUM(CASE WHEN customer_city IS NULL THEN 1 ELSE 0 END) AS
                     SUM(CASE WHEN customer_state IS NULL THEN 1 ELSE 0 END) A
                     FROM customers_data;""")
pd.read_sql_query(query,conn)
```

Out[79]:

	customer_id_null	customer_unique_id_null	customer_zip_code_prefix_null	cus
0	0.0	0.0	0.0	

In [80]:
```
query=text("""SELECT SUM(CASE WHEN TRIM(customer_id)='' THEN 1 ELSE 0 END) AS
                     SUM(CASE WHEN TRIM(customer_unique_id)='' THEN 1 ELSE 0 E
                     SUM(CASE WHEN TRIM(customer_city)='' THEN 1 ELSE 0 END) A
                     SUM(CASE WHEN TRIM(customer_state)='' THEN 1 ELSE 0 END)
                     FROM customers_data;""")
pd.read_sql_query(query,conn)
```

Out[80]:

	customer_id_blank	customer_unique_id_blank	customer_city_blank	customer
0	0.0	0.0	0.0	

In [81]:
```
query=text("""SELECT
    SUM(CASE WHEN customer_id <> TRIM(customer_id) THEN 1 ELSE 0 END)
        AS customer_id_space_issues,
    SUM(CASE WHEN customer_unique_id <> TRIM(customer_unique_id) THEN 1 ELSE 0
        AS customer_unique_id_space_issues,
    SUM(CASE WHEN customer_city <> TRIM(customer_city) THEN 1 ELSE 0 END)
        AS city_space_issues,
    SUM(CASE WHEN customer_state <> TRIM(customer_state) THEN 1 ELSE 0 END)
        AS state_space_issues
FROM customers_data;""")
pd.read_sql_query(query,conn)
```

Out[81]:

	customer_id_space_issues	customer_unique_id_space_issues	city_space_issues
0	0.0	0.0	0.0

In [82]:
```
check_index_exists_query=text("""SELECT COUNT(*) FROM information_schema.stati
                          WHERE table_schema='olist_db'
                          AND table_name='customers_data'
                          AND index_name='customers_data_customer_zip_code_pre

if conn.execute(check_index_exists_query).scalar()==0:
    conn.execute(text("""CREATE INDEX customers_data_customer_zip_code_prefix_
    conn.commit()
    print("==========Index Created Successfully✅==========")
else:
    print("==========Index Already Exists❌==========")
```

```
==========Index Created Successfully✅==========
```

In [83]:
```
check_index_exists_query=text("""SELECT COUNT(*) FROM information_schema.stati
                          WHERE table_schema='olist_db'
                          AND table_name='geolocation_data'
                          AND index_name='geolocation_data_geolocation_zip_coo

if conn.execute(check_index_exists_query).scalar()==0:
    conn.execute(text("""CREATE INDEX geolocation_data_geolocation_zip_code_pr
    conn.commit()
    print("==========Index Created Successfully✅==========")
else:
    print("==========Index Already Exists❌==========")
```

```
==========Index Created Successfully✅==========
```

In [84]:
```
query=text("""SELECT COUNT(*) AS total_orphan_customer_zip_code_prefix
              FROM customers_data c
              LEFT JOIN geolocation_data g
```

```
            ON c.customer_zip_code_prefix = g.geolocation_zip_code_prefix
            WHERE g.geolocation_zip_code_prefix IS NULL;""")
pd.read_sql_query(query,conn)
```

Out[84]:

	total_orphan_customer_zip_code_prefix
0	278

geolocation_data

observation

- The dataset contains `1,000,163 rows` and `5 columns`.
- The dataset does not contain any column that uniquely identifies each row.
- A large number of rows `(279667)` have duplicate combinations of `geolocation_zip_code_prefix, geolocation_lat,geolocation_lng,geolocation_city, and geolocation_state`.
- No NULL or blank values were detected in the dataset.
- The `geolocation_state` column stores state abbreviations, which are not easily interpretable.
- The `geolocation_city` column contains accented characters (e.g., ã, á), and values are not stored in a consistent format.
- `One row` in the geolocation_city column contains leading or trailing whitespace.

In [85]:
```
query=text("""DESCRIBE geolocation_data;""")
pd.read_sql_query(query,conn)
```

Out[85]:

	Field	Type	Null	Key	Default	Extra
0	geolocation_zip_code_prefix	int(11)	YES	MUL	None	
1	geolocation_lat	double	YES		None	
2	geolocation_lng	double	YES		None	
3	geolocation_city	varchar(255)	YES		None	
4	geolocation_state	varchar(255)	YES		None	

In [86]:
```
query=text("""SELECT COUNT(*) AS "total_rows" FROM geolocation_data;""")
pd.read_sql_query(query,conn)
```

Out[86]:

	total_rows
0	1000163

In [87]:
```
query=text("""SELECT COUNT(*) AS "total_columns" FROM information_schema.colum
              WHERE table_schema='olist_db'
              AND table_name='geolocation_data';""")
pd.read_sql_query(query,conn)
```

Out[87]:

	total_columns
0	5

In [88]:
```
query=text("""SELECT * FROM geolocation_data
             ORDER BY geolocation_zip_code_prefix
             LIMIT 10;""")
pd.read_sql_query(query,conn)
```

Out[88]:

	geolocation_zip_code_prefix	geolocation_lat	geolocation_lng	geolocation_city
0	1001	-23.549292	-46.633559	sao paulo
1	1001	-23.550498	-46.634338	sao paulo
2	1001	-23.550642	-46.634410	sao paulo
3	1001	-23.549698	-46.633909	sao paulo
4	1001	-23.551427	-46.634074	sao paulo
5	1001	-23.550498	-46.634338	sao paulo
6	1001	-23.551337	-46.634027	sao paulo
7	1001	-23.549779	-46.633957	são paulo
8	1001	-23.551337	-46.634027	sao paulo
9	1001	-23.550498	-46.634338	sao paulo

In [89]:
```
query=text("""SELECT geolocation_zip_code_prefix,
                     COUNT(*)
              FROM geolocation_data
              GROUP BY geolocation_zip_code_prefix
              HAVING COUNT(*)>1
              LIMIT 10;""")
pd.read_sql_query(query,conn)
```

Out[89]:

	geolocation_zip_code_prefix	COUNT(*)
0	1001	26
1	1002	13
2	1003	17
3	1004	22
4	1005	25
5	1006	9
6	1007	26
7	1008	16
8	1009	41
9	1010	18

In [90]:

```
query=text("""SELECT geolocation_zip_code_prefix,
                     geolocation_state,
                     geolocation_city,
                     COUNT(*)
              FROM geolocation_data
              GROUP BY geolocation_zip_code_prefix,
                       geolocation_state,
                       geolocation_city
              HAVING COUNT(*)>1
              LIMIT 10;""")
pd.read_sql_query(query,conn)
```

Out[90]:

	geolocation_zip_code_prefix	geolocation_state	geolocation_city	COUNT(*)
0	1037	SP	sao paulo	26
1	1046	SP	sao paulo	141
2	1041	SP	sao paulo	25
3	1035	SP	sao paulo	39
4	1012	SP	são paulo	17
5	1047	SP	sao paulo	42
6	1013	SP	sao paulo	20
7	1029	SP	sao paulo	8
8	1011	SP	sao paulo	21
9	1032	SP	sao paulo	45

In [91]:

```
check_index_exists_query = text("""
SELECT COUNT(*)
```

```
FROM information_schema.statistics
WHERE table_schema = 'olist_db'
  AND table_name = 'geolocation_data'
  AND index_name = 'geolocation_data_code_lat_lng_city_state_idx';
""")

if conn.execute(check_index_exists_query).scalar() == 0:
    conn.execute(text("""
        CREATE INDEX geolocation_data_code_lat_lng_city_state_idx ON geolocati
                                            geolocation_lat,
                                            geolocation_lng,
                                            geolocation_city,
                                            geolocation_state);"""))
    conn.commit()
    print("==========Index Created Successfully✅==========")
else:
    print("==========Index Already Exists❌==========")
```

```
==========Index Created Successfully✅==========
```

In [92]:

```
query=text("""SELECT CAST(IFNULL(SUM(duplicate_count) - COUNT(*), 0) AS SIGNED
              FROM (SELECT COUNT(*) AS duplicate_count
                    FROM geolocation_data
                    GROUP BY geolocation_zip_code_prefix,
                             geolocation_lat,
                             geolocation_lng,
                             geolocation_city,
                             geolocation_state
                    HAVING COUNT(*)>1) T;""")
pd.read_sql_query(query,conn)
```

Out[92]:

	total_duplicate_rows
0	279667

In [93]:

```
query=text("""SELECT geolocation_city FROM geolocation_data
              WHERE geolocation_city REGEXP '[áàâãäéèêëíìîïóòôõöúùûüçñ]'
              LIMIT 10;""")
pd.read_sql_query(query,conn)
```

Out[93]:

	geolocation_city
0	são paulo
1	são paulo
2	são paulo
3	são paulo
4	são paulo
5	são paulo
6	são paulo
7	são paulo
8	são paulo
9	são paulo

In [94]:
```
query=text("""SELECT SUM(CASE WHEN geolocation_zip_code_prefix IS NULL THEN 1
                    SUM(CASE WHEN geolocation_lat IS NULL THEN 1 ELSE 0 END)
                    SUM(CASE WHEN geolocation_lng IS NULL THEN 1 ELSE 0 END)
                    SUM(CASE WHEN geolocation_city IS NULL THEN 1 ELSE 0 END)
                    SUM(CASE WHEN geolocation_state IS NULL THEN 1 ELSE 0 END
                    FROM geolocation_data;""")
pd.read_sql_query(query,conn)
```

Out[94]:

	geolocation_zip_code_prefix_null	geolocation_lat_null	geolocation_lng_null	ge
0	0.0	0.0	0.0	

In [95]:
```
query=text("""SELECT SUM(CASE WHEN TRIM(geolocation_city)='' THEN 1 ELSE 0 END
                    SUM(CASE WHEN TRIM(geolocation_state)='' THEN 1 ELSE 0 EN
                    FROM geolocation_data;""")
pd.read_sql_query(query,conn)
```

Out[95]:

	geolocation_city_blank	geolocation_state_blank
0	0.0	0.0

In [96]:
```
query=text("""SELECT
    SUM(CASE WHEN geolocation_city <> TRIM(geolocation_city) THEN 1 ELSE 0 END
        AS geolocation_city_space_issues,
    SUM(CASE WHEN geolocation_state <> TRIM(geolocation_state) THEN 1 ELSE 0 E
        AS geolocation_state_space_issues
FROM geolocation_data;""")
pd.read_sql_query(query,conn)
```

Out[96]:

	geolocation_city_space_issues	geolocation_state_space_issues
0	1.0	0.0

order_items_data

observation

- The dataset contains `112,650 rows` and `7 columns`.
- The dataset does not contain any column that uniquely identifies each row.
- No duplicate rows were identified across all columns.
- No NULL or blank values are present in any column.
- No leading or trailing whitespace is present in any string column.
- The order_item_id column does not represent a globally unique item identifier. it indicates the sequence number of an item within an order.
- The dataset contains a small number of `zero delivery charge values` `(approximately 383 rows)`, while no product price values equal to zero were found.
- No orphan seller_id in child table (order_items_data) related to parent table (sellers_data).
- No orphan product_id in child table (order_items_data) related to parent table (product_data).

In [97]:
```
query=text("""DESCRIBE order_items_data;""")
pd.read_sql_query(query,conn)
```

Out[97]:

	Field	Type	Null	Key	Default	Extra
0	order_id	varchar(255)	YES		None	
1	order_item_id	int(11)	YES		None	
2	product_id	varchar(255)	YES		None	
3	seller_id	varchar(255)	YES		None	
4	shipping_limit_date	datetime	YES		None	
5	price	double	YES		None	
6	freight_value	double	YES		None	

In [98]:
```
query=text("""SELECT COUNT(*) AS "total_rows" FROM order_items_data;""")
pd.read_sql_query(query,conn)
```

Out[98]:

	total_rows
0	112650

In [99]:
```
query=text("""SELECT COUNT(*) AS "total_columns" FROM information_schema.colum
              WHERE table_schema='olist_db'
```

```
                AND table_name='order_items_data';""")
pd.read_sql_query(query,conn)
```

Out[99]:

	total_columns
0	7

In [100…

```
query=text("""SELECT * FROM order_items_data
              ORDER BY order_id,
                       order_item_id
              LIMIT 10;""")
pd.read_sql_query(query,conn)
```

Out[100…

	order_id	order_item_id	pr
0	00010242fe8c5a6d1ba2dd792cb16214	1	4244733e06e7ecb4970a6e268
1	00018f77f2f0320c557190d7a144bdd3	1	e5f2d52b802189ee658865ca
2	000229ec398224ef6ca0657da4fc703e	1	c777355d18b72b67abbeef9
3	00024acbcdf0a6daa1e931b038114c75	1	7634da152a4610f1595efa32
4	00042b26cf59d7ce69dfabb4e55b4fd9	1	ac6c3623068f30de03045865e
5	00048cc3ae777c65dbb7d2a0634bc1ea	1	ef92defde845ab8450f9d70c
6	00054e8431b9d7675808bcb819fb4a32	1	8d4f2bb7e93e6710a28f34fa8
7	000576fe39319847cbb9d288c5617fa6	1	557d850972a7d6f792fd18ae1
8	0005a1a1728c9d785b8e2b08b904576c	1	310ae3c140ff94b03219ad0a
9	0005f50442cb953dcd1d21e1fb923495	1	4535b0e1091c278dfd193e5a

In [101…

```
query=text("""SELECT order_id,
                     COUNT(*)
              FROM order_items_data
              GROUP BY order_id
              HAVING COUNT(*)>1
              LIMIT 10;""")
pd.read_sql_query(query,conn)
```

Out[101…

	order_id	COUNT(*)
0	0008288aa423d2a3f00fcb17cd7d8719	2
1	00143d0f86d6fbd9f9b38ab440ac16f5	3
2	001ab0a7578dd66cd4b0a71f5b6e1e41	3
3	001d8f0e34a38c37f7dba2a37d4eba8b	2
4	002c9def9c9b951b1bec6d50753c9891	2
5	002f98c0f7efd42638ed6100ca699b42	2
6	003324c70b19a16798817b2b3640e721	2
7	00337fe25a3780b3424d9ad7c5a4b35e	2
8	003822434f91204da0a51fe4cf2aba18	2
9	003f201cdd39cdd59b6447cff2195456	2

In [102…

```
query=text("""SELECT order_id,order_item_id,product_id,seller_id,COUNT(*) FROM
              GROUP BY order_id,
                       order_item_id,
                       product_id,
                       seller_id
              HAVING COUNT(*)>1
              LIMIT 10;""")
pd.read_sql_query(query,conn)
```

Out[102…

	order_id	order_item_id	product_id	seller_id	COUNT(*)

In [103…

```
check_index_exists_query = text("""
SELECT COUNT(*)
FROM information_schema.statistics
WHERE table_schema = 'olist_db'
  AND table_name = 'order_items_data'
  AND index_name = 'order_items_data_order_id_item_id_product_id_seller_id_idx
""")

if conn.execute(check_index_exists_query).scalar() == 0:
    conn.execute(text("""
        CREATE INDEX order_items_data_order_id_item_id_product_id_seller_id_id
                                              order_id,
                                              order_item_id,
                                              product_id,
                                              seller_id);""""))
    conn.commit()
    print("==========Index Created Successfully✅==========")
else:
    print("==========Index Already Exists❌==========")
```

```
==========Index Created Successfully✅==========
```

In [104…
```
query=text("""SELECT CAST(IFNULL(SUM(duplicate_count) - COUNT(*), 0) AS SIGNED
              FROM (SELECT COUNT(*) AS duplicate_count
                    FROM order_items_data
                    GROUP BY order_id,
                             order_item_id,
                             product_id,
                             seller_id
                    HAVING COUNT(*)>1) T;""")
pd.read_sql_query(query,conn)
```

Out[104…

	total_duplicate_rows
0	0

In [105…
```
query=text("""SELECT SUM(CASE WHEN order_id IS NULL THEN 1 ELSE 0 END) AS orde
                     SUM(CASE WHEN order_item_id IS NULL THEN 1 ELSE 0 END) AS
                     SUM(CASE WHEN product_id IS NULL THEN 1 ELSE 0 END) AS pr
                     SUM(CASE WHEN seller_id IS NULL THEN 1 ELSE 0 END) AS sel
                     SUM(CASE WHEN shipping_limit_date IS NULL THEN 1 ELSE 0 E
                     SUM(CASE WHEN price IS NULL THEN 1 ELSE 0 END) AS price_n
                     SUM(CASE WHEN freight_value IS NULL THEN 1 ELSE 0 END) AS
                     FROM order_items_data;""")
pd.read_sql_query(query,conn)
```

Out[105…

	order_id_null	order_item_id_null	product_id_null	seller_id_null	shipping_limit
0	0.0	0.0	0.0	0.0	

In [106…
```
query=text("""SELECT SUM(CASE WHEN TRIM(order_id)='' THEN 1 ELSE 0 END) AS ord
                     SUM(CASE WHEN TRIM(product_id)='' THEN 1 ELSE 0 END) AS p
                     SUM(CASE WHEN TRIM(seller_id)='' THEN 1 ELSE 0 END) AS se
                     FROM order_items_data;""")
pd.read_sql_query(query,conn)
```

Out[106…

	order_id_blank	product_id_blank	seller_id_blank
0	0.0	0.0	0.0

In [107…
```
query=text("""SELECT
    SUM(CASE WHEN order_id <> TRIM(order_id) THEN 1 ELSE 0 END)
        AS order_id_space_issues,
    SUM(CASE WHEN product_id <> TRIM(product_id) THEN 1 ELSE 0 END)
        AS product_id_space_issues,
    SUM(CASE WHEN seller_id <> TRIM(seller_id) THEN 1 ELSE 0 END)
        AS seller_id_space_issues
FROM order_items_data;""")
pd.read_sql_query(query,conn)
```

Out[107…

	order_id_space_issues	product_id_space_issues	seller_id_space_issues
0	0.0	0.0	0.0

In [108…
```
query=text("""SELECT COUNT(*) AS total_zero_delivery_charges FROM order_items_
""")
pd.read_sql_query(query,conn)
```

Out[108…

	total_zero_delivery_charges
0	383

In [109…
```
check_index_exists_query=text("""SELECT COUNT(*) FROM information_schema.stati
                          WHERE table_schema='olist_db'
                          AND table_name='order_items_data'
                          AND index_name='order_items_data_product_id_idx';"""

if conn.execute(check_index_exists_query).scalar()==0:
    conn.execute(text("""CREATE INDEX order_items_data_product_id_idx ON order
    conn.commit()
    print("==========Index Created Successfully✅==========")
else:
    print("==========Index Already Exists❌==========")
```

==========Index Created Successfully✅==========

In [110…
```
check_index_exists_query=text("""SELECT COUNT(*) FROM information_schema.stati
                          WHERE table_schema='olist_db'
                          AND table_name='products_data'
                          AND index_name='products_data_product_id_idx';""")

if conn.execute(check_index_exists_query).scalar()==0:
    conn.execute(text("""CREATE INDEX products_data_product_id_idx ON products
    conn.commit()
    print("==========Index Created Successfully✅==========")
else:
    print("==========Index Already Exists❌==========")
```

==========Index Created Successfully✅==========

In [111…
```
query=text("""SELECT COUNT(*) AS total_orphan_product_id FROM order_items_data
              LEFT JOIN products_data p
              ON ot.product_id=p.product_id
              WHERE p.product_id IS NULL;""")
pd.read_sql_query(query,conn)
```

Out[111…

	total_orphan_product_id
0	0

In [112…
```
check_index_exists_query=text("""SELECT COUNT(*) FROM information_schema.stati
                          WHERE table_schema='olist_db'
                          AND table_name='order_items_data'
                          AND index_name='order_items_data_seller_id_idx';""")
```

```
if conn.execute(check_index_exists_query).scalar()==0:
    conn.execute(text("""CREATE INDEX order_items_data_seller_id_idx ON order_
    conn.commit()
    print("==========Index Created Successfully✅==========")
else:
    print("==========Index Already Exists❌==========")
```

==========Index Created Successfully✅==========

In [113…
```
check_index_exists_query=text("""SELECT COUNT(*) FROM information_schema.stati
                         WHERE table_schema='olist_db'
                         AND table_name='sellers_data'
                         AND index_name='sellers_data_seller_id_idx';""")

if conn.execute(check_index_exists_query).scalar()==0:
    conn.execute(text("""CREATE INDEX sellers_data_seller_id_idx ON sellers_da
    conn.commit()
    print("==========Index Created Successfully✅==========")
else:
    print("==========Index Already Exists❌==========")
```

==========Index Created Successfully✅==========

In [114…
```
query=text("""SELECT COUNT(*) AS total_orphan_seller_id FROM order_items_data
              LEFT JOIN sellers_data s
              ON ot.seller_id=s.seller_id
              WHERE s.seller_id IS NULL;""")
pd.read_sql_query(query,conn)
```

Out[114…

	total_orphan_seller_id
0	0

order_payments_data

observation

- The dataset contains `103,886 rows` and `5 columns`.
- The dataset does not contain any column that uniquely identifies each row.
- No duplicate rows were identified across all columns.
- No NULL or blank values are present in any column.
- No leading or trailing whitespace is present in any string column.
- The payment_installments column contains inconsistent and semantically incorrect values depending on the payment type.
- No accented characters were detected in the payment_type column.
- The payment_type column is not stored in a consistent format.

- Data quality issues were identified in the payment_installments column based on payment-method logic:
    - For credit card payments, `payment_installments` is recorded as 0 (approximately 2 records), which is invalid because credit card payments must have at least one installment.
    - For voucher payments, `payment_installments` is recorded as 1 (approximately 5,775 records), which is misleading because voucher payments do not support installments.
- There is no orphan order_id in child table(order_payments_data) related to parent table(orders_data).

In [115…
```
query=text("""DESCRIBE order_payments_data;""")
pd.read_sql_query(query,conn)
```

Out[115…

	Field	Type	Null	Key	Default	Extra
0	order_id	varchar(255)	YES		None	
1	payment_sequential	int(11)	YES		None	
2	payment_type	varchar(255)	YES		None	
3	payment_installments	int(11)	YES		None	
4	payment_value	double	YES		None	

In [116…
```
query=text("""SELECT COUNT(*) AS "total_rows" FROM order_payments_data;""")
pd.read_sql_query(query,conn)
```

Out[116…

	total_rows
0	103886

In [117…
```
query=text("""SELECT COUNT(*) AS total_columns FROM information_schema.columns
              WHERE table_schema='olist_db'
              AND table_name='order_payments_data';""")
pd.read_sql_query(query,conn)
```

Out[117…

	total_columns
0	5

In [118…
```
query=text("""SELECT * FROM order_payments_data
              ORDER BY order_id,
                       payment_sequential
              LIMIT 10;""")
```

```
pd.read_sql_query(query,conn)
```

Out[118…

	order_id	payment_sequential	payment_type	payn
0	00010242fe8c5a6d1ba2dd792cb16214	1	credit_card	
1	00018f77f2f0320c557190d7a144bdd3	1	credit_card	
2	000229ec398224ef6ca0657da4fc703e	1	credit_card	
3	00024acbcdf0a6daa1e931b038114c75	1	credit_card	
4	00042b26cf59d7ce69dfabb4e55b4fd9	1	credit_card	
5	00048cc3ae777c65dbb7d2a0634bc1ea	1	boleto	
6	00054e8431b9d7675808bcb819fb4a32	1	credit_card	
7	000576fe39319847cbb9d288c5617fa6	1	credit_card	
8	0005a1a1728c9d785b8e2b08b904576c	1	credit_card	
9	0005f50442cb953dcd1d21e1fb923495	1	credit_card	

In [119…

```
query=text("""SELECT order_id,COUNT(*) FROM order_payments_data
              GROUP BY order_id
              HAVING COUNT(*)>1
              LIMIT 10;""")
pd.read_sql_query(query,conn)
```

Out[119…

	order_id	COUNT(*)
0	5cfd514482e22bc992e7693f0e3e8df7	2
1	b2bb080b6bc860118a246fd9b6fad6da	2
2	3689194c14ad4e2e7361ebd1df0e77b0	2
3	723e462ce1ee50e024887c0b403130f3	2
4	21b8b46679ea6482cbf911d960490048	2
5	ea9184ad433a404df1d72fa0a8764232	5
6	82ffe097d8ddbf319a523b9bbe7725d5	4
7	c61e3efc183db6b89f5556efb5aa8797	2
8	487c1451b8fd7347d0e80e5aca887e91	2
9	65c863d90026e9dd4888e28d2135c983	2

In [120…

```
check_index_exists_query = text("""
SELECT COUNT(*)
FROM information_schema.statistics
WHERE table_schema = 'olist_db'
  AND table_name = 'order_payments_data'
  AND index_name = 'order_payments_data_id_seq_type_install_value_idx';
```

```
""")

if conn.execute(check_index_exists_query).scalar() == 0:
    conn.execute(text("""
        CREATE INDEX order_payments_data_id_seq_type_install_value_idx ON orde
                                            order_id,
                                            payment_sequential,
                                            payment_type,
                                            payment_installments,
                                            payment_value);"""))
    conn.commit()
    print("==========Index Created Successfully✅==========")
else:
    print("==========Index Already Exists❌==========")
```

```
==========Index Created Successfully✅==========
```

In [121…
```
query=text("""SELECT CAST(IFNULL(SUM(duplicate_count) - COUNT(*), 0) AS SIGNED
              FROM (SELECT COUNT(*) AS duplicate_count
                    FROM order_payments_data
                    GROUP BY order_id,
                             payment_sequential,
                             payment_type,
                             payment_installments,
                             payment_value
                    HAVING COUNT(*)>1) T;""")
pd.read_sql_query(query,conn)
```

Out[121…

	total_duplicate_rows
0	0

In [122…
```
query=text("""SELECT payment_type FROM order_payments_data
              WHERE payment_type REGEXP '[áàâãäéèêëíìîïóòôõöúùûüçñ]'
              LIMIT 10;""")
pd.read_sql_query(query,conn)
```

Out[122…

	payment_type

In [123…
```
query=text("""SELECT SUM(CASE WHEN order_id IS NULL THEN 1 ELSE 0 END) AS orde
                     SUM(CASE WHEN payment_sequential IS NULL THEN 1 ELSE 0 EN
                     SUM(CASE WHEN payment_type IS NULL THEN 1 ELSE 0 END) AS
                     SUM(CASE WHEN payment_installments IS NULL THEN 1 ELSE 0
                     SUM(CASE WHEN payment_value IS NULL THEN 1 ELSE 0 END) AS
                     FROM order_payments_data;""")
pd.read_sql_query(query,conn)
```

Out[123…

	order_id_null	payment_sequential_null	payment_type_null	payment_installme
0	0.0	0.0	0.0	

In [124…
```
query=text("""SELECT SUM(CASE WHEN TRIM(order_id)='' THEN 1 ELSE 0 END) AS orc
                      SUM(CASE WHEN TRIM(payment_type)='' THEN 1 ELSE 0 END) AS
                      FROM order_payments_data;""")
pd.read_sql_query(query,conn)
```

Out[124…

	order_id_blank	payment_type_blank
0	0.0	0.0

In [125…
```
query=text("""SELECT
    SUM(CASE WHEN order_id <> TRIM(order_id) THEN 1 ELSE 0 END)
        AS order_id_space_issues,
    SUM(CASE WHEN payment_type <> TRIM(payment_type) THEN 1 ELSE 0 END)
        AS product_id_space_issues
FROM order_payments_data;""")
pd.read_sql_query(query,conn)
```

Out[125…

	order_id_space_issues	product_id_space_issues
0	0.0	0.0

In [126…
```
query=text("""SELECT * FROM order_payments_data
              WHERE payment_installments=0;
""")
pd.read_sql_query(query,conn)
```

Out[126…

	order_id	payment_sequential	payment_type	paym
0	1a57108394169c0b47d8f876acc9ba2d	2	credit_card	
1	744bade1fcf9ff3f31d860ace076d422	2	credit_card	

In [127…
```
query=text("""SELECT * FROM order_payments_data
              WHERE payment_installments=1 LIMIT 10;
""")
pd.read_sql_query(query,conn)
```

Out[127…

	order_id	payment_sequential	payment_type	payn
0	00048cc3ae777c65dbb7d2a0634bc1ea	1	boleto	
1	00054e8431b9d7675808bcb819fb4a32	1	credit_card	
2	0005f50442cb953dcd1d21e1fb923495	1	credit_card	
3	0008288aa423d2a3f00fcb17cd7d8719	1	boleto	
4	0009792311464db532ff765bf7b182ae	1	boleto	
5	000aed2e25dbad2f9ddb70584c5a2ded	1	credit_card	
6	000c3e6612759851cc3cbb4b83257986	1	boleto	
7	000e63d38ae8c00bbcb5a30573b99628	1	credit_card	
8	000f25f4d72195062c040b12dce9a18a	1	credit_card	
9	0010dedd556712d7bb69a19cb7bbd37a	1	boleto	

In [128…

```
query=text("""SELECT * FROM order_payments_data
              WHERE payment_value=0;
""")
pd.read_sql_query(query,conn)
```

Out[128…

	order_id	payment_sequential	payment_type	payn
0	00b1cb0320190ca0daa2c88b35206009	1	not_defined	
1	45ed6e85398a87c253db47c2d9f48216	3	voucher	
2	4637ca194b6387e2d538dc89b124b0ee	1	not_defined	
3	6ccb433e00daae1283ccc956189c82ae	4	voucher	
4	8bcbe01d44d147f901cd3192671144db	4	voucher	
5	b23878b3e8eb4d25a158f57d96331b18	4	voucher	
6	c8c528189310eaa44a745b8d9d26908b	1	not_defined	
7	fa65dad1b0e818e3ccc5cb0e39231352	13	voucher	
8	fa65dad1b0e818e3ccc5cb0e39231352	14	voucher	

In [129…

```
query=text("""SELECT COUNT(*) AS total_credit_card_zero_installments FROM orde
              WHERE payment_type='Credit_card' AND payment_installments=0 ;"""
pd.read_sql_query(query,conn)
```

Out[129…

	total_credit_card_zero_installments
0	2

In [130…

```
query=text("""SELECT COUNT(*) AS total_voucher_one_installments FROM order_pay
              WHERE payment_type='Voucher' AND payment_installments=1;""")
```

```
pd.read_sql_query(query,conn)
```

Out[130…

	total_voucher_one_installments
0	5775

In [131…

```
check_index_exists_query=text("""SELECT COUNT(*) FROM information_schema.stati
                           WHERE table_schema='olist_db'
                           AND table_name='order_payments_data'
                           AND index_name='order_payments_data_order_id_idx';""

if conn.execute(check_index_exists_query).scalar()==0:
    conn.execute(text("""CREATE INDEX order_payments_data_order_id_idx ON orde
    conn.commit()
    print("==========Index Created Successfully✅==========")
else:
    print("==========Index Already Exists❌==========")
```

```
==========Index Created Successfully✅==========
```

In [132…

```
check_index_exists_query=text("""SELECT COUNT(*) FROM information_schema.stati
                           WHERE table_schema='olist_db'
                           AND table_name='orders_data'
                           AND index_name='orders_data_order_id_idx';""")

if conn.execute(check_index_exists_query).scalar()==0:
    conn.execute(text("""CREATE INDEX orders_data_order_id_idx ON orders_data(
    conn.commit()
    print("==========Index Created Successfully✅==========")
else:
    print("==========Index Already Exists❌==========")
```

```
==========Index Created Successfully✅==========
```

In [133…

```
query=text("""SELECT COUNT(*) AS total_orphan_order_id FROM order_payments_dat
              LEFT JOIN orders_data o
              ON op.order_id=o.order_id
              WHERE o.order_id IS NULL;""")
pd.read_sql_query(query,conn)
```

Out[133…

	total_orphan_order_id
0	0

order_reviews_data

observation

- The dataset contains `99,223 rows` and `7 columns`.
- `The dataset does not contain any column that uniquely`

`identifies each row`.

- No duplicate rows are present in the dataset.
- No leading or trailing whitespace is present in any string column.
- The review_comment_message column contains unstructured text data, including numbers, special characters, and escape characters (e.g., \n, \t).
- The review_comment_title column contains accented or misleading characters (e.g., ã, á) and is not stored in a consistent format.
- A large number of blank values are present in `review_comment_title (87,657 rows)` and `review_comment_message (58,256 rows)`.
- No NULL values are present in any column.
- `No orders were found with review ratings equal to 0 or less than 0`.
- The `review_comment_message and review_comment_title columns were left unchanged`, as the text is unstructured and not required for the current SQL analysis scope.
- There is no orphan order_id in child table(order_reviews_data) related to parent table(orders_data).

In [134…
```
query=text("""DESCRIBE order_reviews_data;""")
pd.read_sql_query(query,conn)
```

Out[134…

	Field	Type	Null	Key	Default	Extra
0	review_id	varchar(255)	YES		None	
1	order_id	varchar(255)	YES		None	
2	review_score	int(11)	YES		None	
3	review_comment_title	text	YES		None	
4	review_comment_message	text	YES		None	
5	review_creation_date	datetime	YES		None	
6	review_answer_timestamp	datetime	YES		None	

In [135…
```
query=text("""SELECT COUNT(*) AS "total_rows" FROM order_reviews_data;""")
pd.read_sql_query(query,conn)
```

Out[135…

	total_rows
0	99223

In [136…
```
query=text("""SELECT COUNT(*) AS total_columns FROM information_schema.columns
              WHERE table_schema='olist_db'
```

```
            AND table_name='order_reviews_data';""")
pd.read_sql_query(query,conn)
```

Out[136…

	total_columns
0	7

In [137…

```
query=text("""SELECT * FROM order_reviews_data
            ORDER BY review_id,order_id
            LIMIT 10;""")
pd.read_sql_query(query,conn)
```

Out[137…

	review_id	order_id	revie
0	0001239bc1de2e33cb583967c2ca4c67	fc046d7776171871436844218f817d7d	
1	0001cc6860aeaf5b9017fe4131a52e62	d4665434b01caa9dc3e3e78b3eb3593e	
2	00020c7512a52e92212f12d3e37513c0	e28abf2eb2f1fbcbdc2dd0cd9a561671	
3	00032b0141443497c898b3093690af51	04fb47576993a3cb0c12d4b25eab6e4e	
4	00034d88989f9a4c393bdcaec301537f	5f358d797a49fe2f24352f73426215f6	
5	000359bceee1b6fe876b30d35b5f9ef2	97868e4c884bc85f4501ae0a66181cc9	
6	00041b1f51e77a663df6db3f539d694f	998b1beee068744ac833dd8b31944623	
7	00046a69550325aea5fb89f65c7387f2	9fbda7367628952bc36c3512c46d887b	
8	0005534973388c830bb858cfba83b17b	a589caa6892ceacc6bbf2f8cc30a8ad4	
9	00055e36e9608fe969231e551983a69c	f5fea26ab547eec920e6f8ecdc5c37e4	

In [138…

```
query=text("""SELECT order_id,count(*) FROM order_reviews_data
            GROUP BY order_id having count(*)>1
            LIMIT 10;""")
pd.read_sql_query(query,conn)
```

Out[138…

	order_id	count(*)
0	cf73e2cb1f4a9480ed70c154da3d954a	2
1	169d7e0fd71d624d306f132acd791cbe	2
2	1c308eca3f339414a92e518e2a2e5ee9	2
3	2002ea16e75277eaa0b5d78632048540	2
4	1d297b4800ed1a3c5b0944d84c01ee99	2
5	0176a6846bcb3b0d3aa3116a9a768597	2
6	b6e5aa946acc4e29e7069510f28a0bce	2
7	5040757d4e06a4be96d3827b860b4e7c	2
8	80446aee36e09ebf4cb79585f6c9ce61	2
9	45390ff93f092f0de1524d486a172c33	2

In [139…

```
query=text("""SELECT review_id,count(*) FROM order_reviews_data
              GROUP BY review_id having count(*)>1
              LIMIT 10;""")
pd.read_sql_query(query,conn)
```

Out[139…

	review_id	count(*)
0	28642ce6250b94cc72bc85960aec6c62	2
1	a0a641414ff718ca079b3967ef5c2495	2
2	f4d74b17cd63ee35efa82cd2567de911	2
3	ecbaf1fce7d2c09bfab46f89065afeaf	2
4	6b1de94de0f4bd84dfc4136818242faa	2
5	957011305e7a4b6c8a266eeeb8e0316d	2
6	b34f4a786bf00b7c8486141ba482783b	2
7	d433c252647c51309432ca0b763f969b	2
8	5f35bbde2b32b8617aab15b7a9a0ae24	2
9	c5976a5a98e854fb23d7e03c6754ae60	2

In [140…

```
query=text("""SELECT order_id,review_id,COUNT(*) FROM order_reviews_data
              GROUP BY order_id,review_id
              HAVING COUNT(*)>1
              LIMIT 10;""")
pd.read_sql_query(query,conn)
```

Out[140…

	order_id	review_id	COUNT(*)

In [141…
```
check_index_exists_query = text("""
SELECT COUNT(*)
FROM information_schema.statistics
WHERE table_schema = 'olist_db'
  AND table_name = 'order_reviews_data'
  AND index_name = 'order_reviews_data_order_id_review_id_idx';
""")

if conn.execute(check_index_exists_query).scalar() == 0:
    conn.execute(text("""
        CREATE INDEX order_reviews_data_order_id_review_id_idx ON order_review
                                            order_id,
                                            review_id);""""))
    conn.commit()
    print("==========Index Created Successfully✅==========")
else:
    print("==========Index Already Exists❌==========")
```

==========Index Created Successfully✅==========

In [142…
```
query=text("""SELECT CAST(IFNULL(SUM(duplicate_count) - COUNT(*), 0) AS SIGNED
              FROM (SELECT COUNT(*) AS duplicate_count
                    FROM order_reviews_data
                    GROUP BY order_id,
                             review_id
                    HAVING COUNT(*)>1) T;""")
pd.read_sql_query(query,conn)
```

Out[142…

	total_duplicate_rows
0	0

In [143…
```
query=text("""SELECT SUM(CASE WHEN review_id IS NULL THEN 1 ELSE 0 END) AS rev
                     SUM(CASE WHEN order_id IS NULL THEN 1 ELSE 0 END) AS orde
                     SUM(CASE WHEN review_score IS NULL THEN 1 ELSE 0 END) AS
                     SUM(CASE WHEN review_comment_title IS NULL THEN 1 ELSE 0
                     SUM(CASE WHEN review_comment_message IS NULL THEN 1 ELSE
                     SUM(CASE WHEN review_creation_date IS NULL THEN 1 ELSE 0
                     SUM(CASE WHEN review_answer_timestamp IS NULL THEN 1 ELSE
pd.read_sql_query(query,conn)
```

Out[143…

	review_id_null	order_id_null	review_score_null	review_comment_title_null	re
0	0.0	0.0	0.0	0.0	

In [144…
```
query=text("""SELECT SUM(CASE WHEN TRIM(review_id)='' THEN 1 ELSE 0 END) AS re
                     SUM(CASE WHEN TRIM(order_id)='' THEN 1 ELSE 0 END) AS orc
                     SUM(CASE WHEN TRIM(review_comment_title)='' THEN 1 ELSE 0
                     SUM(CASE WHEN TRIM(review_comment_message)='' THEN 1 ELSE
                     FROM order_reviews_data;""")
pd.read_sql_query(query,conn)
```

Out[144…

	review_id_blank	order_id_blank	review_comment_title_blank	review_commen
0	0.0	0.0	87657.0	

In [145…

```
query=text("""SELECT
    SUM(CASE WHEN review_id <> TRIM(review_id) THEN 1 ELSE 0 END)
        AS review_id_space_issues,
    SUM(CASE WHEN order_id <> TRIM(order_id) THEN 1 ELSE 0 END)
        AS order_id_space_issues
FROM order_reviews_data;""")
pd.read_sql_query(query,conn)
```

Out[145…

	review_id_space_issues	order_id_space_issues
0	0.0	0.0

In [146…

```
query=text("""SELECT COUNT(*) AS total_zero_review_score FROM order_reviews_da
              WHERE review_score<=0;
""")
pd.read_sql_query(query,conn)
```

Out[146…

	total_zero_review_score
0	0

In [147…

```
check_index_exists_query=text("""SELECT COUNT(*) FROM information_schema.stati
                         WHERE table_schema='olist_db'
                         AND table_name='order_reviews_data'
                         AND index_name='order_reviews_data_order_id_idx';"""

if conn.execute(check_index_exists_query).scalar()==0:
    conn.execute(text("""CREATE INDEX order_reviews_data_order_id_idx ON order
    conn.commit()
    print("==========Index Created Successfully✅==========")
else:
    print("==========Index Already Exists❌==========")
```

```
==========Index Created Successfully✅==========
```

In [148…

```
query=text("""SELECT COUNT(*) AS total_orphan_order_id FROM order_reviews_data
              LEFT JOIN orders_data o
              ON ord_rev.order_id=o.order_id
              WHERE o.order_id IS NULL;""")
pd.read_sql_query(query,conn)
```

Out[148…

	total_orphan_order_id
0	0

orders_data

observation

- The dataset contains `99,441 rows` and `8 columns`.
- `order_id` uniquely identifies each row.
- No duplicate rows are present in the dataset.
- The order_status column is not stored in a consistent format.
- No leading or trailing whitespace is present in any string column.
- No accented characters were detected in the order_status column.
- No NULL values are present in any column.
- No blank values are present in any column.
- Several datetime columns contain `zero-date placeholders (0000-00-00 00:00:00)`, which represent missing timestamps:
    - `order_approved_at has 160 zero-date values`.
    - `order_delivered_carrier_date has 1,783 zero-date values`.
    - `order_delivered_customer_date has 2,965 zero-date values`.
    - `order_purchase_timestamp and order_estimated_delivery_date do not contain any zero-date values`.
- These `zero-date placeholders indicate orders that were not yet approved, shipped, or delivered at the time of data collection`. Since this behavior is expected in real-world order data, these values were retained and not converted.
- Some delivery-related timestamp columns contained value 0000-00-00 00:00:00 instead of NULL.
- During data cleaning, these zero-date values were changed to NULL to correctly show incomplete order stages and to avoid incorrect time-based analysis.
- There is no orphan customer_id in child table(orders_data) related to parent table(customers_data).

In [149…

```
query=text("""DESCRIBE orders_data;""")
pd.read_sql_query(query,conn)
```

Out[149…

	Field	Type	Null	Key	Default	Extra
0	order_id	varchar(255)	YES	MUL	None	
1	customer_id	varchar(255)	YES		None	
2	order_status	varchar(255)	YES		None	
3	order_purchase_timestamp	datetime	YES		None	
4	order_approved_at	datetime	YES		None	
5	order_delivered_carrier_date	datetime	YES		None	
6	order_delivered_customer_date	datetime	YES		None	
7	order_estimated_delivery_date	datetime	YES		None	

In [150…

```
query=text("""SELECT COUNT(*) AS "total_rows" FROM orders_data;""")
pd.read_sql_query(query,conn)
```

Out[150…

	total_rows
0	99441

In [151…

```
query=text("""SELECT COUNT(*) AS total_columns FROM information_schema.columns
            WHERE table_schema='olist_db'
            AND table_name='orders_data';""")
pd.read_sql_query(query,conn)
```

Out[151…

	total_columns
0	8

In [152…

```
query=text("""SELECT * FROM orders_data
              ORDER BY order_id,customer_id
              LIMIT 10;""")
pd.read_sql_query(query,conn)
```

Out[152…

	order_id	customer_id	order
0	00010242fe8c5a6d1ba2dd792cb16214	3ce436f183e68e07877b285a838db11a	d
1	00018f77f2f0320c557190d7a144bdd3	f6dd3ec061db4e3987629fe6b26e5cce	d
2	000229ec398224ef6ca0657da4fc703e	6489ae5e4333f3693df5ad4372dab6d3	d
3	00024acbcdf0a6daa1e931b038114c75	d4eb9395c8c0431ee92fce09860c5a06	d
4	00042b26cf59d7ce69dfabb4e55b4fd9	58dbd0b2d70206bf40e62cd34e84d795	d
5	00048cc3ae777c65dbb7d2a0634bc1ea	816cbea969fe5b689b39cfc97a506742	d
6	00054e8431b9d7675808bcb819fb4a32	32e2e6ab09e778d99bf2e0ecd4898718	d
7	000576fe39319847cbb9d288c5617fa6	9ed5e522dd9dd85b4af4a077526d8117	d
8	0005a1a1728c9d785b8e2b08b904576c	16150771dfd4776261284213b89c304e	d
9	0005f50442cb953dcd1d21e1fb923495	351d3cb2cee3c7fd0af6616c82df21d3	d

In [153…

```
query=text("""SELECT order_id,count(*) FROM orders_data
              GROUP BY order_id
              HAVING COUNT(*)>1;""")
pd.read_sql_query(query,conn)
```

Out[153…

	order_id	count(*)

In [154…

```
query=text("""SELECT order_id,customer_id,count(*) FROM orders_data
              GROUP BY order_id,customer_id
              HAVING COUNT(*)>1;""")
pd.read_sql_query(query,conn)
```

Out[154…

	order_id	customer_id	count(*)

In [155…

```
query=text("""SELECT customer_id,count(*) FROM orders_data
              GROUP BY customer_id
              HAVING COUNT(*)>1;""")
pd.read_sql_query(query,conn)
```

Out[155…

	customer_id	count(*)

In [156…
```
check_index_exists_query = text("""
SELECT COUNT(*)
FROM information_schema.statistics
WHERE table_schema = 'olist_db'
  AND table_name = 'orders_data'
  AND index_name = 'orders_data_order_id_customer_id_idx';
""")

if conn.execute(check_index_exists_query).scalar() == 0:
    conn.execute(text("""
        CREATE INDEX orders_data_order_id_customer_id_idx ON orders_data(
                                            order_id,
                                            customer_id);""""))
    conn.commit()
    print("==========Index Created Successfully✅==========")
else:
    print("==========Index Already Exists❌==========")
```

==========Index Created Successfully✅==========

In [157…
```
query=text("""SELECT CAST(IFNULL(SUM(duplicate_count) - COUNT(*), 0) AS SIGNED
              FROM (SELECT COUNT(*) AS duplicate_count
                    FROM orders_data
                    GROUP BY order_id,
                             customer_id
                    HAVING COUNT(*)>1) T;""")
pd.read_sql_query(query,conn)
```

Out[157…

	total_duplicate_rows
0	0

In [158…
```
query=text("""SELECT order_status FROM orders_data
              WHERE order_status REGEXP '[áàâãäéèêëíìîïóòôõöúùûüçñ]'
              LIMIT 10;""")
pd.read_sql_query(query,conn)
```

Out[158…

	order_status

In [159…
```
query=text("""SELECT SUM(CASE WHEN order_id IS NULL THEN 1 ELSE 0 END) AS orde
                    SUM(CASE WHEN customer_id IS NULL THEN 1 ELSE 0 END) AS c
                    SUM(CASE WHEN order_status IS NULL THEN 1 ELSE 0 END) AS
                    SUM(CASE WHEN order_purchase_timestamp IS NULL THEN 1 ELS
                    SUM(CASE WHEN order_approved_at IS NULL THEN 1 ELSE 0 END
                    SUM(CASE WHEN order_delivered_carrier_date IS NULL THEN 1
                    SUM(CASE WHEN order_delivered_customer_date IS NULL THEN
                    SUM(CASE WHEN order_estimated_delivery_date IS NULL THEN
                    FROM orders_data;""")
pd.read_sql_query(query,conn)
```

Out[159…

	order_id_null	customer_id_null	order_status_null	order_purchase_timestamp_
0	0.0	0.0	0.0	

In [160…

```
query=text("""SELECT SUM(CASE WHEN TRIM(order_id)='' THEN 1 ELSE 0 END) AS orc
                     SUM(CASE WHEN TRIM(customer_id)='' THEN 1 ELSE 0 END) AS
                     SUM(CASE WHEN TRIM(order_status)='' THEN 1 ELSE 0 END) AS
                     FROM orders_data;""")
pd.read_sql_query(query,conn)
```

Out[160…

	order_id_blank	customer_id_blank	order_status_blank
0	0.0	0.0	0.0

In [161…

```
query=text("""SELECT SUM(CASE WHEN order_purchase_timestamp = '0000-00-00 00:0
                     SUM(CASE WHEN order_approved_at = '0000-00-00 00:00:00' T
                     SUM(CASE WHEN order_delivered_carrier_date = '0000-00-00
                     SUM(CASE WHEN order_delivered_customer_date = '0000-00-00
                     SUM(CASE WHEN order_estimated_delivery_date = '0000-00-00
                     FROM orders_data;""")
pd.read_sql_query(query,conn)
```

Out[161…

	order_purchase_timestamp_zero_date	order_approved_at_zero_date	order_del
0	0.0	160.0	

In [162…

```
query=text("""SELECT
    SUM(CASE WHEN order_id <> TRIM(order_id) THEN 1 ELSE 0 END)
        AS order_id_space_issues,
    SUM(CASE WHEN customer_id <> TRIM(customer_id) THEN 1 ELSE 0 END)
        AS customerid_space_issues,
    SUM(CASE WHEN order_status <> TRIM(order_status) THEN 1 ELSE 0 END)
        AS order_status_space_issues
FROM orders_data;""")
pd.read_sql_query(query,conn)
```

Out[162…

	order_id_space_issues	customerid_space_issues	order_status_space_issues
0	0.0	0.0	0.0

In [163…

```
check_index_exists_query=text("""SELECT COUNT(*) FROM information_schema.stati
                          WHERE table_schema='olist_db'
                          AND table_name='orders_data'
                          AND index_name='orders_data_customer_id_idx';""")

if conn.execute(check_index_exists_query).scalar()==0:
    conn.execute(text("""CREATE INDEX orders_data_customer_id_idx ON orders_da
    conn.commit()
    print("==========Index Created Successfully✅==========")
else:
```

```
    print("==========Index Already Exists❌==========")
```

```
==========Index Created Successfully✅==========
```

In [164…
```
check_index_exists_query=text("""SELECT COUNT(*) FROM information_schema.stati
                          WHERE table_schema='olist_db'
                          AND table_name='customers_data'
                          AND index_name='customers_data_customer_id_idx';""")

if conn.execute(check_index_exists_query).scalar()==0:
    conn.execute(text("""CREATE INDEX customers_data_customer_id_idx ON custom
    conn.commit()
    print("==========Index Created Successfully✅==========")
else:
    print("==========Index Already Exists❌==========")
```

```
==========Index Created Successfully✅==========
```

In [165…
```
query=text("""SELECT COUNT(*) AS total_orphan_customer_id FROM orders_data o
              LEFT JOIN customers_data c
              ON o.customer_id=c.customer_id
              WHERE c.customer_id IS NULL;""")
pd.read_sql_query(query,conn)
```

Out[165…

	total_orphan_customer_id
0	0

Products_data

observation

- The dataset contains `32,951 rows` and `9 columns`.
- `product_id` uniquely identifies each row.
- No duplicate rows were identified.
- No leading or trailing whitespace is present in any string column.
- The columns product_name_lenght and product_description_lenght contain spelling errors and represent length-related attributes.
- The product_category_name column is not stored in a consistent format.
- No accented characters were detected in the product_category_name column.
- No NULL values are present in any column.
- `A large number of blank values (610 rows) are present in the product_category_name column`.
- There are `623 orphan product_category_name` in child

table(products_data) related to parent table(products_categories_data).

In [166…
```
query=text("""DESCRIBE products_data;""")
pd.read_sql_query(query,conn)
```

Out[166…

	Field	Type	Null	Key	Default	Extra
0	product_id	varchar(255)	YES	MUL	None	
1	product_category_name	varchar(255)	YES		None	
2	product_name_lenght	int(11)	YES		None	
3	product_description_lenght	int(11)	YES		None	
4	product_photos_qty	int(11)	YES		None	
5	product_weight_g	int(11)	YES		None	
6	product_length_cm	int(11)	YES		None	
7	product_height_cm	int(11)	YES		None	
8	product_width_cm	int(11)	YES		None	

In [167…
```
query=text("""SELECT COUNT(*) AS "total_rows" FROM products_data;""")
pd.read_sql_query(query,conn)
```

Out[167…

	total_rows
0	32951

In [168…
```
query=text("""SELECT COUNT(*) AS total_columns FROM information_schema.columns
              WHERE table_schema='olist_db'
              AND table_name='products_data';""")
pd.read_sql_query(query,conn)
```

Out[168…

	total_columns
0	9

In [169…
```
query=text("""SELECT * FROM products_data
             ORDER BY product_id
             LIMIT 10;""")
pd.read_sql_query(query,conn)
```

Out[169…

	product_id	product_category_name	product_name_le
0	00066f42aeeb9f3007548bb9d3f33c38	perfumaria	
1	00088930e925c41fd95ebfe695fd2655	automotivo	
2	0009406fd7479715e4bef61dd91f2462	cama_mesa_banho	
3	000b8f95fcb9e0096488278317764d19	utilidades_domesticas	
4	000d9be29b5207b54e86aa1b1ac54872	relogios_presentes	
5	0011c512eb256aa0dbbb544d8dffcf6e	automotivo	
6	00126f27c813603687e6ce486d909d01	cool_stuff	
7	001795ec6f1b187d37335e1c4704762e	consoles_games	
8	001b237c0e9bb435f2e54071129237e9	cama_mesa_banho	
9	001b72dfd63e9833e8c02742adf472e3	moveis_decoracao	

In [170…

```
query=text("""SELECT product_id,
                     COUNT(*)
              FROM products_data
              GROUP BY product_id
              HAVING COUNT(*)>1;""")
pd.read_sql_query(query,conn)
```

Out[170…

	product_id	COUNT(*)

In [171…

```
check_index_exists_query = text("""
SELECT COUNT(*)
FROM information_schema.statistics
WHERE table_schema = 'olist_db'
  AND table_name = 'products_data'
  AND index_name = 'products_data_product_id_product_category_name_idx';
""")

if conn.execute(check_index_exists_query).scalar() == 0:
    conn.execute(text("""
        CREATE INDEX products_data_product_id_product_category_name_idx ON pro
                                            product_id,
                                            product_category_name);"""))
    conn.commit()
    print("==========Index Created Successfully✅==========")
else:
    print("==========Index Already Exists❌==========")
```

```
==========Index Created Successfully✅==========
```

In [172…

```
query=text("""SELECT CAST(IFNULL(SUM(duplicate_count) - COUNT(*), 0) AS SIGNED
              FROM (SELECT COUNT(*) AS duplicate_count
                    FROM products_data
                    GROUP BY product_id,
```

```
                                product_category_name
                    HAVING COUNT(*)>1) T;""")
pd.read_sql_query(query,conn)
```

Out[172…

	total_duplicate_rows
0	0

In [173…

```
query=text("""SELECT product_category_name FROM products_data
              WHERE product_category_name REGEXP '[áàâãäéèêëíìîïóòôõöúùûüçñ]'
              LIMIT 10;""")
pd.read_sql_query(query,conn)
```

Out[173…

	product_category_name

In [174…

```
query=text("""SELECT SUM(CASE WHEN product_id IS NULL THEN 1 ELSE 0 END) AS pr
                     SUM(CASE WHEN product_category_name IS NULL THEN 1 ELSE 0
                     SUM(CASE WHEN product_name_lenght IS NULL THEN 1 ELSE 0 E
                     SUM(CASE WHEN product_description_lenght IS NULL THEN 1 E
                     SUM(CASE WHEN product_photos_qty IS NULL THEN 1 ELSE 0 EN
                     SUM(CASE WHEN product_weight_g IS NULL THEN 1 ELSE 0 END)
                     SUM(CASE WHEN product_length_cm IS NULL THEN 1 ELSE 0 END
                     SUM(CASE WHEN product_height_cm IS NULL THEN 1 ELSE 0 END
                     SUM(CASE WHEN product_width_cm IS NULL THEN 1 ELSE 0 END)
                     FROM products_data;""")
pd.read_sql_query(query,conn)
```

Out[174…

	product_id_null	product_category_name_null	product_name_lenght_null	prod
0	0.0	0.0	0.0	

In [175…

```
query=text("""SELECT SUM(CASE WHEN TRIM(product_id)='' THEN 1 ELSE 0 END) AS p
                     SUM(CASE WHEN TRIM(product_category_name)='' THEN 1 ELSE
                     FROM products_data;""")
pd.read_sql_query(query,conn)
```

Out[175…

	product_id_blank	product_category_name_blank
0	0.0	610.0

In [176…

```
query=text("""SELECT
    SUM(CASE WHEN product_id <> TRIM(product_id) THEN 1 ELSE 0 END)
        AS product_id_space_issues,
    SUM(CASE WHEN product_category_name <> TRIM(product_category_name) THEN 1
        AS product_category_name_space_issues
FROM products_data;""")
pd.read_sql_query(query,conn)
```

Out[176…

	product_id_space_issues	product_category_name_space_issues
0	0.0	0.0

In [177…

```
check_index_exists_query=text("""SELECT COUNT(*) FROM information_schema.stati
                          WHERE table_schema='olist_db'
                          AND table_name='products_data'
                          AND index_name='products_data_product_category_name_

if conn.execute(check_index_exists_query).scalar()==0:
    conn.execute(text("""CREATE INDEX products_data_product_category_name_idx
    conn.commit()
    print("==========Index Created Successfully✅==========")
else:
    print("==========Index Already Exists❌==========")
```

```
==========Index Created Successfully✅==========
```

In [178…

```
check_index_exists_query=text("""SELECT COUNT(*) FROM information_schema.stati
                          WHERE table_schema='olist_db'
                          AND table_name='products_categories_data'
                          AND index_name='products_categories_data_product_cat

if conn.execute(check_index_exists_query).scalar()==0:
    conn.execute(text("""CREATE INDEX products_categories_data_product_categor
    conn.commit()
    print("==========Index Created Successfully✅==========")
else:
    print("==========Index Already Exists❌==========")
```

```
==========Index Created Successfully✅==========
```

In [179…

```
query=text("""SELECT COUNT(*) AS total_orphan_product_category_name FROM produ
              LEFT JOIN products_categories_data pc
              ON p.product_category_name=pc.product_category_name
              WHERE pc.product_category_name IS NULL
                    AND p.product_category_name IS NOT NULL;""")
pd.read_sql_query(query,conn)
```

Out[179…

	total_orphan_product_category_name
0	623

sellers_data

observation

- The dataset contains `3,095 rows` and `4 columns`.
- `seller_id uniquely identifies each row`.

- No duplicate rows were identified.
- No leading or trailing whitespace is present in any string column.
- The seller_state column stores state abbreviations, which are not easily interpretable.
- The seller_city column is not stored in a consistent format.
- No accented characters were detected in the seller_city column.
- No NULL or blank values are present in any column.
- There are `7 orphan seller_zip_code_prefix` in the child table related(sellers_data) to parent table(geolocation_data).

In [180…
```
query=text("""DESCRIBE sellers_data;""")
pd.read_sql_query(query,conn)
```

Out[180…

	Field	Type	Null	Key	Default	Extra
0	seller_id	varchar(255)	YES	MUL	None	
1	seller_zip_code_prefix	int(11)	YES		None	
2	seller_city	varchar(255)	YES		None	
3	seller_state	varchar(255)	YES		None	

In [181…
```
query=text("""SELECT COUNT(*) AS "total_rows" FROM sellers_data;""")
pd.read_sql_query(query,conn)
```

Out[181…

	total_rows
0	3095

In [182…
```
query=text("""SELECT COUNT(*) AS total_columns FROM information_schema.columns
              WHERE table_schema='olist_db'
              AND table_name='sellers_data';""")
pd.read_sql_query(query,conn)
```

Out[182…

	total_columns
0	4

In [183…
```
query=text("""SELECT * FROM sellers_data
              ORDER BY seller_id
              LIMIT 10;""")
pd.read_sql_query(query,conn)
```

Out[183…

	seller_id	seller_zip_code_prefix	seller_city	seller_s
0	0015a82c2db000af6aaaf3ae2ecb0532	9080	santo andre	
1	001cca7ae9ae17fb1caed9dfb1094831	29156	cariacica	
2	001e6ad469a905060d959994f1b41e4f	24754	sao goncalo	
3	002100f778ceb8431b7a1020ff7ab48f	14405	franca	
4	003554e2dce176b5555353e4f3555ac8	74565	goiania	
5	004c9cd9d87a3c30c522c48c4fc07416	14940	ibitinga	
6	00720abe85ba0859807595bbf045a33b	7070	guarulhos	
7	00ab3eff1b5192e5f1a63bcecfee11c8	4164	sao paulo	
8	00d8b143d12632bad99c0ad66ad52825	30170	belo horizonte	
9	00ee68308b45bc5e2660cd833c3f81cc	3333	sao paulo	

In [184…

```
query=text("""SELECT seller_id,count(*)
              FROM sellers_data
              GROUP BY seller_id
              HAVING COUNT(*)>1;""")
pd.read_sql_query(query,conn)
```

Out[184…

	seller_id	count(*)

In [185…

```
query=text("""SELECT seller_id,
                     seller_zip_code_prefix,
                     count(*)
              FROM sellers_data
              GROUP BY seller_id,
                       seller_zip_code_prefix
              HAVING COUNT(*)>1;""")
pd.read_sql_query(query,conn)
```

Out[185…

	seller_id	seller_zip_code_prefix	count(*)

In [186…

```
check_index_exists_query = text("""
SELECT COUNT(*)
FROM information_schema.statistics
WHERE table_schema = 'olist_db'
  AND table_name = 'sellers_data'
  AND index_name = 'sellers_data_seller_id_seller_zip_code_prefix_idx';
""")

if conn.execute(check_index_exists_query).scalar() == 0:
    conn.execute(text("""
```

```
        CREATE INDEX sellers_data_seller_id_seller_zip_code_prefix_idx ON sell
                                            seller_id,
                                            seller_zip_code_prefix);"""))
    conn.commit()
    print("==========Index Created Successfully✅==========")
else:
    print("==========Index Already Exists❌==========")
```

```
==========Index Created Successfully✅==========
```

In [187…
```
query=text("""SELECT CAST(IFNULL(SUM(duplicate_count) - COUNT(*), 0) AS SIGNED
              FROM (SELECT COUNT(*) AS duplicate_count
                    FROM sellers_data
                    GROUP BY seller_id,
                             seller_zip_code_prefix
                    HAVING COUNT(*)>1) T;""")
pd.read_sql_query(query,conn)
```

Out[187…

	total_duplicate_rows
0	0

In [188…
```
query=text("""SELECT seller_city FROM sellers_data
              WHERE seller_city REGEXP '[áàâãäéèêëíìîïóòôõöúùûüçñ]'
              LIMIT 10;""")
pd.read_sql_query(query,conn)
```

Out[188…

	seller_city

In [189…
```
query=text("""SELECT SUM(CASE WHEN seller_id IS NULL THEN 1 ELSE 0 END) AS sel
                     SUM(CASE WHEN seller_zip_code_prefix IS NULL THEN 1 ELSE
                     SUM(CASE WHEN seller_city IS NULL THEN 1 ELSE 0 END) AS s
                     SUM(CASE WHEN seller_state IS NULL THEN 1 ELSE 0 END) AS
                     FROM sellers_data;""")
pd.read_sql_query(query,conn)
```

Out[189…

	seller_id_null	seller_zip_code_prefix_null	seller_city_null	seller_state_null
0	0.0	0.0	0.0	0.0

In [190…
```
query=text("""SELECT SUM(CASE WHEN TRIM(seller_id)='' THEN 1 ELSE 0 END) AS se
                     SUM(CASE WHEN TRIM(seller_city)='' THEN 1 ELSE 0 END) AS
                     SUM(CASE WHEN TRIM(seller_state)='' THEN 1 ELSE 0 END) AS
                     FROM sellers_data;""")
pd.read_sql_query(query,conn)
```

Out[190…

	seller_id_blank	seller_city_blank	seller_state_blank
0	0.0	0.0	0.0

In [191…
```
query=text("""SELECT
```

```
    SUM(CASE WHEN seller_id <> TRIM(seller_id) THEN 1 ELSE 0 END)
        AS seller_id_space_issues,
    SUM(CASE WHEN seller_city <> TRIM(seller_city) THEN 1 ELSE 0 END)
        AS seller_city_space_issues,
    SUM(CASE WHEN seller_state <> TRIM(seller_state) THEN 1 ELSE 0 END)
        AS seller_state_space_issues
FROM sellers_data;""")
pd.read_sql_query(query,conn)
```

Out[191…

	seller_id_space_issues	seller_city_space_issues	seller_state_space_issues
0	0.0	0.0	0.0

In [192…

```
check_index_exists_query=text("""SELECT COUNT(*) FROM information_schema.stati
                          WHERE table_schema='olist_db'
                          AND table_name='sellers_data'
                          AND index_name='sellers_data_seller_zip_code_prefix_

if conn.execute(check_index_exists_query).scalar()==0:
    conn.execute(text("""CREATE INDEX sellers_data_seller_zip_code_prefix_idx
    conn.commit()
    print("==========Index Created Successfully✅==========")
else:
    print("==========Index Already Exists❌==========")
```

```
==========Index Created Successfully✅==========
```

In [193…

```
query=text("""SELECT COUNT(*) AS total_orphan_seller_zip_code FROM sellers_dat
              LEFT JOIN geolocation_data g
              ON s.seller_zip_code_prefix=g.geolocation_zip_code_prefix
              WHERE g.geolocation_zip_code_prefix IS NULL;
                    """)
pd.read_sql_query(query,conn)
```

Out[193…

	total_orphan_seller_zip_code
0	7

products_categories_data

observation

- The dataset contains 71 rows and 2 columns.
- The dataset stores product category information, and each row represents one unique product category based on the category name.
- category name is unique, it is a text-based field that may change over time, so a surrogate key (category_sk) was created to ensure stable and consistent data handling.
- No duplicate records are present in the dataset.

- No leading or trailing whitespace is present in any string column.
- No accented characters were detected in the product_category_name and product_category_name_english columns.
- The product_category_name and product_category_name_english columns are not stored in a consistent format.
- No null or blank values are present in any column.

In [194…
```
query=text("""DESCRIBE products_categories_data;""")
pd.read_sql_query(query,conn)
```

Out[194…

	Field	Type	Null	Key	Default	Extra
0	product_category_name	varchar(255)	YES	MUL	None	
1	product_category_name_english	varchar(255)	YES		None	

In [195…
```
query=text("""SELECT COUNT(*) AS "total_rows" FROM products_categories_data;""
pd.read_sql_query(query,conn)
```

Out[195…

	total_rows
0	71

In [196…
```
query=text("""SELECT COUNT(*) AS total_columns FROM information_schema.columns
              WHERE table_schema='olist_db'
              AND table_name='products_categories_data';""")
pd.read_sql_query(query,conn)
```

Out[196…

	total_columns
0	2

In [197…
```
query=text("""SELECT * FROM products_categories_data
              ORDER BY product_category_name
              LIMIT 10;""")
pd.read_sql_query(query,conn)
```

Out[197…

	product_category_name	product_category_name_english
0	agro_industria_e_comercio	agro_industry_and_commerce
1	alimentos	food
2	alimentos_bebidas	food_drink
3	artes	art
4	artes_e_artesanato	arts_and_craftmanship
5	artigos_de_festas	party_supplies
6	artigos_de_natal	christmas_supplies
7	audio	audio
8	automotivo	auto
9	bebes	baby

In [198…

```
query=text("""SELECT product_category_name,count(*) FROM products_categories_d
              GROUP BY product_category_name
              HAVING COUNT(*)>1;""")
pd.read_sql_query(query,conn)
```

Out[198…

	product_category_name	count(*)

In [199…

```
check_index_exists_query = text("""
SELECT COUNT(*)
FROM information_schema.statistics
WHERE table_schema = 'olist_db'
  AND table_name = 'products_categories_data'
  AND index_name = 'products_categories_data_category_name_category_name_engli
""")

if conn.execute(check_index_exists_query).scalar() == 0:
    conn.execute(text("""
        CREATE INDEX products_categories_data_category_name_category_name_engl
                                            product_category_name,
                                            product_category_name_english );""
    conn.commit()
    print("==========Index Created Successfully✅==========")
else:
    print("==========Index Already Exists❌==========")
```

```
==========Index Created Successfully✅==========
```

In [200…

```
query=text("""SELECT CAST(IFNULL(SUM(duplicate_count) - COUNT(*), 0) AS SIGNED
              FROM (SELECT COUNT(*) AS duplicate_count
                    FROM products_categories_data
                    GROUP BY product_category_name,
                             product_category_name_english
                    HAVING COUNT(*)>1) T;""")
```

```
pd.read_sql_query(query,conn)
```

Out[200…

	total_duplicate_rows
0	0

In [201…

```
query=text("""SELECT product_category_name FROM products_categories_data
              WHERE product_category_name REGEXP '[áàâãäéèêëíìîïóòôõöúùûüçñ]'
              LIMIT 10;""")
pd.read_sql_query(query,conn)
```

Out[201…

	product_category_name

In [202…

```
query=text("""SELECT product_category_name_english FROM products_categories_da
              WHERE product_category_name_english REGEXP '[áàâãäéèêëíìîïóòôõöú
              LIMIT 10;""")
pd.read_sql_query(query,conn)
```

Out[202…

	product_category_name_english

In [203…

```
query=text("""SELECT SUM(CASE WHEN product_category_name IS NULL THEN 1 ELSE 0
                    SUM(CASE WHEN product_category_name_english IS NULL THEN
                    FROM products_categories_data;""")
pd.read_sql_query(query,conn)
```

Out[203…

	product_category_name_null	product_category_name_english_null
0	0.0	0.0

In [204…

```
query=text("""SELECT SUM(CASE WHEN TRIM(product_category_name)='' THEN 1 ELSE
                    SUM(CASE WHEN TRIM(product_category_name_english)='' THEN
                    FROM products_categories_data;""")
pd.read_sql_query(query,conn)
```

Out[204…

	product_category_name_blank	product_category_name_english_blank
0	0.0	0.0

In [205…

```
query=text("""SELECT
    SUM(CASE WHEN product_category_name <> TRIM(product_category_name) THEN 1
        AS product_category_name_space_issues,
    SUM(CASE WHEN product_category_name_english <> TRIM(product_category_name_
        AS product_category_name_english_space_issues
FROM products_categories_data;""")
pd.read_sql_query(query,conn)
```

Out[205…

	product_category_name_space_issues	product_category_name_english_space_i
0	0.0	

Data Cleaning & Data transformation

customers_data

Action in Data transformation

- orphan rows were inserted in parent table (geolocation_data).
- customer_id was assigned as the primary key, ensuring unique identification of each row.
- NOT NULL constraints were applied to all columns, eliminating missing values.
- customer_state_full was created to store full state names alongside original abbreviations.
- customer_city_clean was created to standardize city names in a consistent title case format.
- Original columns (customer_state, customer_city) were preserved for reference.

In [206…
```
query=text("""ALTER TABLE customers_data
              MODIFY customer_id VARCHAR(255) NOT NULL,
              MODIFY customer_unique_id VARCHAR(255) NOT NULL,
              MODIFY customer_zip_code_prefix INTEGER NOT NULL,
              MODIFY customer_city VARCHAR(255) NOT NULL,
              MODIFY customer_state VARCHAR(255) NOT NULL;
""")
conn.execute(query)
conn.commit()
```

In [207…
```
check_primary_key_exists_query=text("""SHOW KEYS
                                FROM customers_data
                                WHERE Key_name = 'PRIMARY';""")
if conn.execute(check_primary_key_exists_query).fetchone()==None:
    query=text("""ALTER TABLE customers_data
                  ADD PRIMARY KEY (customer_id);""")
    conn.execute(query)
    conn.commit()
    print("==========Primary key Created Successfully✅==========")
else:
    print("==========Primary key Already Exists❌==========")
```

```
==========Primary key Created Successfully✅==========
```

In [208…
```
query=text("""INSERT INTO geolocation_data (geolocation_zip_code_prefix,geoloc
               SELECT DISTINCT c.customer_zip_code_prefix,
```

```
                        0.0 AS geolocation_lat,
                        0.0 AS geolocation_lng,
                        'Not defined' AS geolocation_city,
                        'Not defined' AS geolocation_state
                FROM customers_data c
                LEFT JOIN geolocation_data g
                  ON c.customer_zip_code_prefix = g.geolocation_zip_code_prefi
                WHERE g.geolocation_zip_code_prefix IS NULL;""")
conn.execute(query)
conn.commit()
```

In [209…

```
check_column_exists_query=text("""SELECT COUNT(*) FROM information_schema.colu
                           WHERE table_schema='olist_db'
                           AND table_name='customers_data'
                           AND column_name='customer_state_full';""")
if conn.execute(check_column_exists_query).scalar()==0:
    query=text("""ALTER TABLE customers_data
                  ADD COLUMN customer_state_full VARCHAR(255) NOT NULL;""")
    conn.execute(query)
    conn.commit()
    print("==========Column Created Successfully✅==========")
else:
    print("==========Column Already Exists❌==========")
```

```
==========Column Created Successfully✅==========
```

In [210…

```
check_index_exists_query=text("""SELECT COUNT(*) FROM information_schema.stati
                           WHERE table_schema='olist_db'
                           AND table_name='customers_data'
                           AND column_name='customer_state';""")
if conn.execute(check_index_exists_query).scalar()==0:
    query=text("""CREATE INDEX customers_data_customer_state_idx ON customers_
    conn.execute(query)
    conn.commit()
    print("==========Index Created Successfully✅==========")
else:
    print("==========Index Already Exists❌==========")
```

```
==========Index Created Successfully✅==========
```

In [211…

```
query=text("""UPDATE customers_data
SET customer_state_full =
    CASE TRIM(customer_state)
        WHEN 'AC' THEN 'Acre'
        WHEN 'AL' THEN 'Alagoas'
        WHEN 'AP' THEN 'Amapa'
        WHEN 'AM' THEN 'Amazonas'
        WHEN 'BA' THEN 'Bahia'
        WHEN 'CE' THEN 'Ceara'
        WHEN 'DF' THEN 'Distrito Federal'
        WHEN 'ES' THEN 'Espirito Santo'
        WHEN 'GO' THEN 'Goias'
```

```
        WHEN 'MA' THEN 'Maranhao'
        WHEN 'MT' THEN 'Mato Grosso'
        WHEN 'MS' THEN 'Mato Grosso do Sul'
        WHEN 'MG' THEN 'Minas Gerais'
        WHEN 'PA' THEN 'Para'
        WHEN 'PB' THEN 'Paraiba'
        WHEN 'PR' THEN 'Parana'
        WHEN 'PE' THEN 'Pernambuco'
        WHEN 'PI' THEN 'Piaui'
        WHEN 'RJ' THEN 'Rio de Janeiro'
        WHEN 'RN' THEN 'Rio Grande do Norte'
        WHEN 'RS' THEN 'Rio Grande do Sul'
        WHEN 'RO' THEN 'Rondonia'
        WHEN 'RR' THEN 'Roraima'
        WHEN 'SC' THEN 'Santa Catarina'
        WHEN 'SP' THEN 'Sao Paulo'
        WHEN 'SE' THEN 'Sergipe'
        WHEN 'TO' THEN 'Tocantins'
        ELSE 'Unknown'
    END
WHERE customer_state IS NOT NULL
  AND TRIM(customer_state) <> '';
""")
conn.execute(query)
conn.commit()
```

In [212…

```
check_column_exists_query=text("""SELECT COUNT(*) FROM information_schema.colu
                          WHERE table_schema='olist_db'
                          AND table_name='customers_data'
                          AND column_name='customer_city_clean';""")
if conn.execute(check_column_exists_query).scalar()==0:
    query=text("""ALTER TABLE customers_data
                  ADD COLUMN customer_city_clean VARCHAR(255) NOT NULL;""")
    conn.execute(query)
    conn.commit()
    print("==========Column Created Successfully✅==========")
else:
    print("==========Column Already Exists❌==========")
```

```
==========Column Created Successfully✅==========
```

In [213…

```
query=text("""UPDATE customers_data SET customer_city_clean=CONCAT(UPPER(LEFT(
             WHERE customer_city IS NOT NULL AND TRIM(customer_city) <> '';""
conn.execute(query)
conn.commit()
```

In [214…

```
query=text("""DESCRIBE customers_data;""")
pd.read_sql_query(query,conn)
```

Out[214…

	Field	Type	Null	Key	Default	Extra
0	customer_id	varchar(255)	NO	PRI	None	
1	customer_unique_id	varchar(255)	NO		None	
2	customer_zip_code_prefix	int(11)	NO	MUL	None	
3	customer_city	varchar(255)	NO		None	
4	customer_state	varchar(255)	NO	MUL	None	
5	customer_state_full	varchar(255)	NO		None	
6	customer_city_clean	varchar(255)	NO		None	

In [215…

```
query=text("""SELECT * FROM customers_data
              LIMIT 10;""")
pd.read_sql_query(query,conn)
```

Out[215…

	customer_id	customer_unique_id	custo
0	00012a2ce6f8dcda20d059ce98491703	248ffe10d632bebe4f7267f1f44844c9	
1	000161a058600d5901f007fab4c27140	b0015e09bb4b6e47c52844fab5fb6638	
2	0001fd6190edaaf884bcaf3d49edf079	94b11d37cd61cb2994a194d11f89682b	
3	0002414f95344307404f0ace7a26f1d5	4893ad4ea28b2c5b3ddf4e82e79db9e6	
4	000379cdec625522490c315e70c7a9fb	0b83f73b19c2019e182fd552c048a22c	
5	0004164d20a9e969af783496f3408652	104bdb7e6a6cdceaa88c3ea5fa6b2b93	
6	000419c5494106c306a97b5635748086	14843983d4a159080f6afe4b7f346e7c	
7	00046a560d407e99b969756e0b10f282	0b5295fc9819d831f68eb0e9a3e13ab7	
8	00050bf6e01e69d5c0fd612f1bcfb69c	e3cf594a99e810f58af53ed4820f25e5	
9	000598caf2ef4117407665ac33275130	7e0516b486e92ed3f3afdd6d1276cfbd	

In [216…

```
query=text("""SELECT COUNT(*) AS total_orphan_customers_zip_code_prefix
              FROM customers_data c
              LEFT JOIN geolocation_data g
              ON c.customer_zip_code_prefix = g.geolocation_zip_code_prefix
              WHERE g.geolocation_zip_code_prefix IS NULL;
""")
pd.read_sql_query(query,conn)
```

Out[216…

	total_orphan_customers_zip_code_prefix
0	0

geolocation_data

Action in Data cleaning

- Duplicates were removed by creating a new table geolocation_data_unique containing only unique rows.

Actions in Data Transformation

- A surrogate key geolocation_sk was created to uniquely identify rows and enforce 1NF.
- NOT NULL constraints were applied to all columns, eliminating missing values.
- The column geolocation_city_clean was planned to store accent-free and standardized city names.
- The column geolocation_state_full was planned to store full state names while preserving abbreviations.
- geolocation_city_clean was planned to standardize city names into a readable, approximate title case format for improved reporting and analytical consistency.

In [217…
```
query=text("""SET FOREIGN_KEY_CHECKS = 0;""")
conn.execute(query)
conn.commit()
```

In [218…
```
check_table_exists_query=text("""
                        SELECT COUNT(*) FROM information_schema.tables
                        WHERE table_schema='olist_db'
                        AND table_name='geolocation_data_unique';
                    """)

if conn.execute(check_table_exists_query).scalar()==1:
    conn.execute(text("""TRUNCATE TABLE geolocation_data_unique;"""))
    conn.commit()
    print("==========All Rows Deleted Successfully✅==========")
else:
    print("==========Table Does Not Exists❌==========")
```

```
==========Table Does Not Exists❌==========
```

In [219…
```
query=text("""SET FOREIGN_KEY_CHECKS = 1;""")
conn.execute(query)
conn.commit()
```

In [220…
```
query=text("""SET FOREIGN_KEY_CHECKS = 0;""")
conn.execute(query)
conn.commit()
```

In [221…
```
query = text("DROP TABLE IF EXISTS geolocation_data_unique;")
conn.execute(query)
conn.commit()
```

In [222…
```
query=text("""CREATE TABLE geolocation_data_unique AS
SELECT geolocation_zip_code_prefix, geolocation_lat, geolocation_lng, geolocat
FROM geolocation_data
GROUP BY geolocation_zip_code_prefix, geolocation_lat, geolocation_lng, geoloc
conn.execute(query)
conn.commit()
print("==========Table Created Successfully✅==========")
```

==========Table Created Successfully✅==========

In [223…
```
query=text("""SET FOREIGN_KEY_CHECKS = 1;""")
conn.execute(query)
conn.commit()
```

In [224…
```
query=text("""ALTER TABLE geolocation_data_unique
              MODIFY geolocation_zip_code_prefix INTEGER NOT NULL,
              MODIFY geolocation_lat DOUBLE NOT NULL,
              MODIFY geolocation_lng DOUBLE NOT NULL,
              MODIFY geolocation_city VARCHAR(255) NOT NULL,
              MODIFY geolocation_state VARCHAR(255) NOT NULL;
""")
conn.execute(query)
conn.commit()
```

In [225…
```
check_column_exists_query=text("""SELECT COUNT(*) FROM information_schema.colu
                          WHERE table_schema='olist_db'
                          AND table_name='geolocation_data_unique'
                          AND column_name='geolocation_sk';""")
if conn.execute(check_column_exists_query).scalar()==0:
    query=text("""ALTER TABLE geolocation_data_unique
                  ADD COLUMN geolocation_sk INTEGER AUTO_INCREMENT PRIMARY KEY
    conn.execute(query)
    conn.commit()
    print("==========Column Created Successfully✅==========")
else:
    print("==========Column Already Exists❌==========")
```

==========Column Created Successfully✅==========

In [226…
```
check_column_exists_query=text("""SELECT COUNT(*) FROM information_schema.colu
                          WHERE table_schema='olist_db'
                          AND table_name='geolocation_data_unique'
                          AND column_name='geolocation_state_full';""")
if conn.execute(check_column_exists_query).scalar()==0:
    query=text("""ALTER TABLE geolocation_data_unique
                  ADD COLUMN geolocation_state_full VARCHAR(255) NOT NULL;""")
    conn.execute(query)
    conn.commit()
```

```
    print("==========Column Created Successfully✅==========")
else:
    print("==========Column Already Exists❌==========")
```

```
==========Column Created Successfully✅==========
```

In [227…
```
check_index_exists_query=text("""SELECT COUNT(*) FROM information_schema.stati
                          WHERE table_schema='olist_db'
                          AND table_name='geolocation_data_unique'
                          AND column_name='geolocation_state';""")
if conn.execute(check_index_exists_query).scalar()==0:
    query=text("""CREATE INDEX geolocation_data_unique_geolocation_state_idx C
    conn.execute(query)
    conn.commit()
    print("==========Index Created Successfully✅==========")
else:
    print("==========Index Already Exists❌==========")
```

```
==========Index Created Successfully✅==========
```

In [228…
```
query=text("""UPDATE geolocation_data_unique SET geolocation_state_full =
                    CASE TRIM(geolocation_state)
                        WHEN 'AC' THEN 'Acre'
                        WHEN 'AL' THEN 'Alagoas'
                        WHEN 'AP' THEN 'Amapa'
                        WHEN 'AM' THEN 'Amazonas'
                        WHEN 'BA' THEN 'Bahia'
                        WHEN 'CE' THEN 'Ceara'
                        WHEN 'DF' THEN 'Distrito Federal'
                        WHEN 'ES' THEN 'Espirito Santo'
                        WHEN 'GO' THEN 'Goias'
                        WHEN 'MA' THEN 'Maranhao'
                        WHEN 'MT' THEN 'Mato Grosso'
                        WHEN 'MS' THEN 'Mato Grosso do Sul'
                        WHEN 'MG' THEN 'Minas Gerais'
                        WHEN 'PA' THEN 'Para'
                        WHEN 'PB' THEN 'Paraiba'
                        WHEN 'PR' THEN 'Parana'
                        WHEN 'PE' THEN 'Pernambuco'
                        WHEN 'PI' THEN 'Piaui'
                        WHEN 'RJ' THEN 'Rio de Janeiro'
                        WHEN 'RN' THEN 'Rio Grande do Norte'
                        WHEN 'RS' THEN 'Rio Grande do Sul'
                        WHEN 'RO' THEN 'Rondonia'
                        WHEN 'RR' THEN 'Roraima'
                        WHEN 'SC' THEN 'Santa Catarina'
                        WHEN 'SP' THEN 'Sao Paulo'
                        WHEN 'SE' THEN 'Sergipe'
                        WHEN 'TO' THEN 'Tocantins'
                        ELSE 'Unknown'
                    END
                WHERE geolocation_state IS NOT NULL
                AND TRIM(geolocation_state) <> ''
```

```
                    ;""")
conn.execute(query)
conn.commit()
```

In [229…
```
check_column_exists_query=text("""SELECT COUNT(*) FROM information_schema.colu
                          WHERE table_schema='olist_db'
                          AND table_name='geolocation_data_unique'
                          AND column_name='geolocation_city_clean';""")
if conn.execute(check_column_exists_query).scalar()==0:
    query=text("""ALTER TABLE geolocation_data_unique
                  ADD COLUMN geolocation_city_clean VARCHAR(255) NULL;""")
    conn.execute(query)
    conn.commit()
    print("==========Column Created Successfully✅==========")
else:
    print("==========Column Already Exists❌==========")
```

==========Column Created Successfully✅==========

In [230…
```
check_index_exists_query=text("""SELECT COUNT(*) FROM information_schema.stati
                          WHERE table_schema='olist_db'
                          AND table_name='geolocation_data_unique'
                          AND column_name='geolocation_city';""")
if conn.execute(check_index_exists_query).scalar()==0:
    query=text("""CREATE INDEX geolocation_data_unique_geolocation_city_idx ON
    conn.execute(query)
    conn.commit()
    print("==========Index Created Successfully✅==========")
else:
    print("==========Index Already Exists❌==========")
```

==========Index Created Successfully✅==========

In [231…
```
query=text("""UPDATE geolocation_data_unique
              SET geolocation_city_clean=CONCAT(UPPER(LEFT(TRIM(geolocation_ci
              WHERE geolocation_city IS NOT NULL AND TRIM(geolocation_city) <>
conn.execute(query)
conn.commit()
```

In [232…
```
query=text("""UPDATE geolocation_data_unique
SET geolocation_city_clean =
REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(
REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(
REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(
REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(
REPLACE(REPLACE(REPLACE(REPLACE(
TRIM(geolocation_city_clean),
'á','a'),'à','a'),'â','a'),'ã','a'),'ä','a'),
'é','e'),'è','e'),'ê','e'),'ë','e'),
'í','i'),'ì','i'),'î','i'),'ï','i'),
'ó','o'),'ò','o'),'ô','o'),'õ','o'),'ö','o'),
'ú','u'),'ù','u'),'û','u'),'ü','u'),
'ç','c'),'ñ','n');
```

```
""")
conn.execute(query)
conn.commit()
```

In [233…
```
query=text("""SELECT COUNT(*) AS "total_remaining_rows" FROM geolocation_data_
pd.read_sql_query(query,conn)
```

Out[233…

	total_remaining_rows
0	720653

In [234…
```
query=text("""DESCRIBE geolocation_data_unique;""")
pd.read_sql_query(query,conn)
```

Out[234…

	Field	Type	Null	Key	Default	Extra
0	geolocation_zip_code_prefix	int(11)	NO		None	
1	geolocation_lat	double	NO		None	
2	geolocation_lng	double	NO		None	
3	geolocation_city	varchar(255)	NO	MUL	None	
4	geolocation_state	varchar(255)	NO	MUL	None	
5	geolocation_sk	int(11)	NO	PRI	None	auto_increment
6	geolocation_state_full	varchar(255)	NO		None	
7	geolocation_city_clean	varchar(255)	YES		None	

In [235…
```
query=text("""SELECT * FROM geolocation_data_unique
              LIMIT 10;""")
pd.read_sql_query(query,conn)
```

Out[235…

	geolocation_zip_code_prefix	geolocation_lat	geolocation_lng	geolocation_city
0	1001	-23.551427	-46.634074	sao paulo
1	1001	-23.551337	-46.634027	sao paulo
2	1001	-23.550642	-46.634410	sao paulo
3	1001	-23.550498	-46.634338	sao paulo
4	1001	-23.550263	-46.634196	são paulo
5	1001	-23.549951	-46.634027	são paulo
6	1001	-23.549825	-46.633970	sao paulo
7	1001	-23.549779	-46.633957	são paulo
8	1001	-23.549698	-46.633909	sao paulo
9	1001	-23.549292	-46.633559	sao paulo

order_items_data

Action in data transformation

- A surrogate key transaction_sk was created to uniquely identify each row and to comply with First Normal Form (1NF). This column was assigned a primary key constraint.
- NOT NULL constraints were applied to all columns, eliminating missing values.
- A new column order_item_sequence was created to clearly represent the item's sequence number within each order, improving clarity and preventing misinterpretation of order_item_id.

In [236…
```
query=text("""ALTER TABLE order_items_data
              MODIFY order_id VARCHAR(255) NOT NULL,
              MODIFY order_item_id INTEGER NOT NULL,
              MODIFY product_id VARCHAR(255) NOT NULL,
              MODIFY seller_id VARCHAR(255) NOT NULL,
              MODIFY shipping_limit_date DATETIME NOT NULL,
              MODIFY price DOUBLE NOT NULL,
              MODIFY freight_value DOUBLE NOT NULL;
""")
conn.execute(query)
conn.commit()
```

In [237…
```
check_column_exists_query=text("""SELECT COUNT(*) FROM information_schema.colu
                          WHERE table_schema='olist_db'
                          AND table_name='order_items_data'
                          AND column_name='transaction_sk';""")
if conn.execute(check_column_exists_query).scalar()==0:
    query=text("""ALTER TABLE order_items_data
                  ADD COLUMN transaction_sk INTEGER AUTO_INCREMENT PRIMARY KEY
    conn.execute(query)
    conn.commit()
    print("==========Column Created Successfully✅==========")
else:
    print("==========Column Already Exists❌==========")
```

```
==========Column Created Successfully✅==========
```

In [238…
```
check_column_exists_query=text("""SELECT COUNT(*) FROM information_schema.colu
                          WHERE table_schema='olist_db'
                          AND table_name='order_items_data'
                          AND column_name='order_item_sequence';""")
if conn.execute(check_column_exists_query).scalar()==0:
    query=text("""ALTER TABLE order_items_data
                  ADD COLUMN order_item_sequence INTEGER NULL;""")
    conn.execute(query)
    conn.commit()
```

```
    print("==========Column Created Successfully✅==========")
else:
    print("==========Column Already Exists❌==========")
```

```
==========Column Created Successfully✅==========
```

In [239…
```
query=text("""UPDATE order_items_data
                SET order_item_sequence = order_item_id;""")

conn.execute(query)
conn.commit()
```

In [240…
```
query=text("""DESCRIBE order_items_data;""")
pd.read_sql_query(query,conn)
```

Out[240…

	Field	Type	Null	Key	Default	Extra
0	order_id	varchar(255)	NO	MUL	None	
1	order_item_id	int(11)	NO		None	
2	product_id	varchar(255)	NO	MUL	None	
3	seller_id	varchar(255)	NO	MUL	None	
4	shipping_limit_date	datetime	NO		None	
5	price	double	NO		None	
6	freight_value	double	NO		None	
7	transaction_sk	int(11)	NO	PRI	None	auto_increment
8	order_item_sequence	int(11)	YES		None	

In [241…
```
query=text("""SELECT * FROM order_items_data ORDER BY order_id,order_item_id L
pd.read_sql_query(query,conn)
```

Out[241…

	order_id	order_item_id	pr
0	00010242fe8c5a6d1ba2dd792cb16214	1	4244733e06e7ecb4970a6e26
1	00018f77f2f0320c557190d7a144bdd3	1	e5f2d52b802189ee658865ca
2	000229ec398224ef6ca0657da4fc703e	1	c777355d18b72b67abbeef9
3	00024acbcdf0a6daa1e931b038114c75	1	7634da152a4610f1595efa32
4	00042b26cf59d7ce69dfabb4e55b4fd9	1	ac6c3623068f30de03045865e
5	00048cc3ae777c65dbb7d2a0634bc1ea	1	ef92defde845ab8450f9d70c
6	00054e8431b9d7675808bcb819fb4a32	1	8d4f2bb7e93e6710a28f34fa8
7	000576fe39319847cbb9d288c5617fa6	1	557d850972a7d6f792fd18ae1
8	0005a1a1728c9d785b8e2b08b904576c	1	310ae3c140ff94b03219ad0a
9	0005f50442cb953dcd1d21e1fb923495	1	4535b0e1091c278dfd193e5a

order_payments_data

Action in data transformation

- A surrogate key order_payment_sk was created to uniquely identify each row and to comply with First Normal Form (1NF). This column was assigned a primary key constraint.
- NOT NULL constraints were applied to all columns, eliminating missing values.
- A new column payment_installments_clean was created to standardize installment values:
- Credit card payments with payment_installments = 0 were converted to 1.
- Voucher payments with payment_installments = 1 were converted to 0.
- A new column payment_type_clean was created to store standardized, readable payment types in approximate title case for improved reporting and analytical consistency.

In [242…

```
query=text("""ALTER TABLE order_payments_data
              MODIFY order_id VARCHAR(255) NOT NULL,
              MODIFY payment_sequential INTEGER NOT NULL,
              MODIFY payment_type VARCHAR(255) NOT NULL,
              MODIFY payment_installments INTEGER NOT NULL,
              MODIFY payment_value DOUBLE NOT NULL;
""")
conn.execute(query)
conn.commit()
```

In [243…
```
check_column_exists_query=text("""SELECT COUNT(*) FROM information_schema.colu
                           WHERE table_schema='olist_db'
                           AND table_name='order_payments_data'
                           AND column_name='order_payment_sk';""")
if conn.execute(check_column_exists_query).scalar()==0:
    query=text("""ALTER TABLE order_payments_data
                  ADD COLUMN order_payment_sk INTEGER AUTO_INCREMENT PRIMARY K
    conn.execute(query)
    conn.commit()
    print("==========Column Created Successfully✅==========")
else:
    print("==========Column Already Exists❌==========")
```

==========Column Created Successfully✅==========

In [244…
```
check_column_exists_query=text("""SELECT COUNT(*) FROM information_schema.colu
                           WHERE table_schema='olist_db'
                           AND table_name='order_payments_data'
                           AND column_name='payment_installments_clean';""")
if conn.execute(check_column_exists_query).scalar()==0:
    query=text("""ALTER TABLE order_payments_data
                  ADD column payment_installments_clean INTEGER NULL;""")
    conn.execute(query)
    conn.commit()
    print("==========Column Created Successfully✅==========")
else:
    print("==========Column Already Exists❌==========")
```

==========Column Created Successfully✅==========

In [245…
```
check_column_exists_query=text("""SELECT COUNT(*) FROM information_schema.colu
                           WHERE table_schema='olist_db'
                           AND table_name='order_payments_data'
                           AND column_name='payment_type_clean';""")
if conn.execute(check_column_exists_query).scalar()==0:
    query=text("""ALTER TABLE order_payments_data
                  ADD column payment_type_clean VARCHAR(255) NULL;""")
    conn.execute(query)
    conn.commit()
    print("==========Column Created Successfully✅==========")
else:
    print("==========Column Already Exists❌==========")
```

==========Column Created Successfully✅==========

In [246…
```
query=text("""UPDATE order_payments_data SET payment_installments_clean=paymer
conn.execute(query)
conn.commit()
```

In [247…
```
query=text("""UPDATE order_payments_data SET payment_installments_clean=1
              WHERE TRIM(payment_type)='Credit_card'
```

```
                AND payment_installments = 0;""")
conn.execute(query)
conn.commit()
```

In [248…
```
query=text("""UPDATE order_payments_data
              SET payment_installments_clean = 0
              WHERE TRIM(payment_type) = 'Voucher'
              AND payment_installments = 1;""")
conn.execute(query)
conn.commit()
```

In [249…
```
query=text("""UPDATE order_payments_data
              SET payment_type_clean=CONCAT(UPPER(LEFT(TRIM(payment_type),1)),
              WHERE payment_type IS NOT NULL AND TRIM(payment_type) <> '';""")
conn.execute(query)
conn.commit()
```

In [250…
```
query=text("""DESCRIBE order_payments_data;""")
pd.read_sql_query(query,conn)
```

Out[250…

	Field	Type	Null	Key	Default	Extra
0	order_id	varchar(255)	NO	MUL	None	
1	payment_sequential	int(11)	NO		None	
2	payment_type	varchar(255)	NO		None	
3	payment_installments	int(11)	NO		None	
4	payment_value	double	NO		None	
5	order_payment_sk	int(11)	NO	PRI	None	auto_increment
6	payment_installments_clean	int(11)	YES		None	
7	payment_type_clean	varchar(255)	YES		None	

In [251…
```
query=text("""SELECT * FROM order_payments_data
              ORDER BY order_id,payment_sequential
              LIMIT 10;""")
pd.read_sql_query(query,conn)
```

Out[251…

	order_id	payment_sequential	payment_type	payn
0	00010242fe8c5a6d1ba2dd792cb16214	1	credit_card	
1	00018f77f2f0320c557190d7a144bdd3	1	credit_card	
2	000229ec398224ef6ca0657da4fc703e	1	credit_card	
3	00024acbcdf0a6daa1e931b038114c75	1	credit_card	
4	00042b26cf59d7ce69dfabb4e55b4fd9	1	credit_card	
5	00048cc3ae777c65dbb7d2a0634bc1ea	1	boleto	
6	00054e8431b9d7675808bcb819fb4a32	1	credit_card	
7	000576fe39319847cbb9d288c5617fa6	1	credit_card	
8	0005a1a1728c9d785b8e2b08b904576c	1	credit_card	
9	0005f50442cb953dcd1d21e1fb923495	1	credit_card	

order_reviews_data

Action in data transformation

- NOT NULL constraints were applied to all columns except review_comment_title and review_comment_message.
- A surrogate key order_review_sk was created to uniquely identify each row and to comply with First Normal Form (1NF). This column was assigned a primary key constraint.

In [252…

```
query=text("""ALTER TABLE order_reviews_data
              MODIFY review_id VARCHAR(255) NOT NULL,
              MODIFY order_id VARCHAR(255) NOT NULL,
              MODIFY review_score INTEGER NOT NULL,
              MODIFY review_comment_title TEXT,
              MODIFY review_comment_message TEXT,
              MODIFY review_creation_date DATETIME NOT NULL,
              MODIFY review_answer_timestamp DATETIME NOT NULL;
""")
conn.execute(query)
conn.commit()
```

In [253…

```
check_column_exists_query=text("""SELECT COUNT(*) FROM information_schema.colu
                          WHERE table_schema='olist_db'
                          AND table_name='order_reviews_data'
                          AND column_name='order_review_sk';""")
if conn.execute(check_column_exists_query).scalar()==0:
    query=text("""ALTER TABLE order_reviews_data
                  ADD COLUMN order_review_sk INTEGER AUTO_INCREMENT PRIMARY KE
    conn.execute(query)
    conn.commit()
```

```
    print("==========Column Created Successfully✅==========")
else:
    print("==========Column Already Exists❌==========")
```

```
==========Column Created Successfully✅==========
```

In [254…
```
query=text("""DESCRIBE order_reviews_data;""")
pd.read_sql_query(query,conn)
```

Out[254…

	Field	Type	Null	Key	Default	Extra
0	review_id	varchar(255)	NO		None	
1	order_id	varchar(255)	NO	MUL	None	
2	review_score	int(11)	NO		None	
3	review_comment_title	text	YES		None	
4	review_comment_message	text	YES		None	
5	review_creation_date	datetime	NO		None	
6	review_answer_timestamp	datetime	NO		None	
7	order_review_sk	int(11)	NO	PRI	None	auto_increment

In [255…
```
query=text("""SELECT * FROM order_reviews_data ORDER BY review_id,order_id LIM
pd.read_sql_query(query,conn)
```

Out[255…

	review_id	order_id	revie
0	0001239bc1de2e33cb583967c2ca4c67	fc046d7776171871436844218f817d7d	
1	0001cc6860aeaf5b9017fe4131a52e62	d4665434b01caa9dc3e3e78b3eb3593e	
2	00020c7512a52e92212f12d3e37513c0	e28abf2eb2f1fbcbdc2dd0cd9a561671	
3	00032b0141443497c898b3093690af51	04fb47576993a3cb0c12d4b25eab6e4e	
4	00034d88989f9a4c393bdcaec301537f	5f358d797a49fe2f24352f73426215f6	
5	000359bceee1b6fe876b30d35b5f9ef2	97868e4c884bc85f4501ae0a66181cc9	
6	00041b1f51e77a663df6db3f539d694f	998b1beee068744ac833dd8b31944623	
7	00046a69550325aea5fb89f65c7387f2	9fbda7367628952bc36c3512c46d887b	
8	0005534973388c830bb858cfba83b17b	a589caa6892ceacc6bbf2f8cc30a8ad4	
9	00055e36e9608fe969231e551983a69c	f5fea26ab547eec920e6f8ecdc5c37e4	

orders_data

Action in data cleaning

- order_id was assigned as the primary key.
- Zero dates like '0000-00-00 00:00:00' were replaced with NULL to clearly indicate that, at the time of data collection, the orders had not yet been approved, shipped, or delivered. This ensures accuracy in analytics and reporting

Actions in Data Transformation

- NOT NULL constraints were applied only to mandatory columns, while delivery-related timestamp columns were intentionally left nullable to preserve real-world order lifecycle behavior.
- A new column order_status_clean was created to store standardized, readable order status values in approximate title case for improved reporting and analytical consistency.

In [256…
```
query=text("""ALTER TABLE orders_data
              MODIFY order_id VARCHAR(255) NOT NULL,
              MODIFY customer_id VARCHAR(255) NOT NULL,
              MODIFY order_status VARCHAR(255) NOT NULL,
              MODIFY order_purchase_timestamp DATETIME,
              MODIFY order_estimated_delivery_date DATETIME,
              MODIFY order_approved_at DATETIME,
              MODIFY order_delivered_carrier_date DATETIME,
              MODIFY order_delivered_customer_date DATETIME;
""")
conn.execute(query)
conn.commit()
```

In [257…
```
query=text("""UPDATE orders_data
SET order_approved_at = NULL
WHERE order_approved_at = '0000-00-00 00:00:00';""")
conn.execute(query)
conn.commit()

query=text("""UPDATE orders_data
SET order_delivered_carrier_date = NULL
WHERE order_delivered_carrier_date = '0000-00-00 00:00:00';""")
conn.execute(query)
conn.commit()

query=text("""UPDATE orders_data
SET order_delivered_customer_date = NULL
WHERE order_delivered_customer_date = '0000-00-00 00:00:00';""")
conn.execute(query)
conn.commit()
```

In [258…
```
check_primary_key_exists_query=text("""SHOW KEYS FROM orders_data
                                    WHERE key_name='PRIMARY';""")
if conn.execute(check_primary_key_exists_query).fetchone()==None:
    query=text("""ALTER TABLE orders_data
                  ADD PRIMARY KEY (order_id);""")
    conn.execute(query)
    conn.commit()
    print("==========Primary key Created Successfully✅==========")
else:
    print("==========Primary key Already Exists❌==========")
```

==========Primary key Created Successfully✅==========

In [259…
```
query=text("""SELECT customer_id,COUNT(*) FROM orders_data GROUP BY customer_i
pd.read_sql_query(query,conn)
```

Out[259…

	customer_id	COUNT(*)

In [260…
```
check_column_exists_query=text("""SELECT COUNT(*) FROM information_schema.colu
                           WHERE table_schema='olist_db'
                           AND table_name='orders_data'
                           AND column_name='order_status_clean';""")
if conn.execute(check_column_exists_query).scalar()==0:
    query=text("""ALTER TABLE orders_data
                  ADD COLUMN order_status_clean VARCHAR(255) NULL;""")
    conn.execute(query)
    conn.commit()
    print("==========Column Created Successfully✅==========")
else:
    print("==========Column Already Exists❌==========")
```

==========Column Created Successfully✅==========

In [261…
```
query=text("""UPDATE orders_data SET order_status_clean=CONCAT(UPPER(LEFT(TRIM
conn.execute(query)
conn.commit()
```

In [262…
```
query=text("""DESCRIBE orders_data;""")
pd.read_sql_query(query,conn)
```

Out[262…

	Field	Type	Null	Key	Default	Extra
0	order_id	varchar(255)	NO	PRI	None	
1	customer_id	varchar(255)	NO	MUL	None	
2	order_status	varchar(255)	NO		None	
3	order_purchase_timestamp	datetime	YES		None	
4	order_approved_at	datetime	YES		None	
5	order_delivered_carrier_date	datetime	YES		None	
6	order_delivered_customer_date	datetime	YES		None	
7	order_estimated_delivery_date	datetime	YES		None	
8	order_status_clean	varchar(255)	YES		None	

In [263…

```
query=text("""SELECT * FROM orders_data ORDER BY order_id,customer_id LIMIT 10
pd.read_sql_query(query,conn)
```

Out[263…

	order_id	customer_id	order
0	00010242fe8c5a6d1ba2dd792cb16214	3ce436f183e68e07877b285a838db11a	d
1	00018f77f2f0320c557190d7a144bdd3	f6dd3ec061db4e3987629fe6b26e5cce	d
2	000229ec398224ef6ca0657da4fc703e	6489ae5e4333f3693df5ad4372dab6d3	d
3	00024acbcdf0a6daa1e931b038114c75	d4eb9395c8c0431ee92fce09860c5a06	d
4	00042b26cf59d7ce69dfabb4e55b4fd9	58dbd0b2d70206bf40e62cd34e84d795	d
5	00048cc3ae777c65dbb7d2a0634bc1ea	816cbea969fe5b689b39cfc97a506742	d
6	00054e8431b9d7675808bcb819fb4a32	32e2e6ab09e778d99bf2e0ecd4898718	d
7	000576fe39319847cbb9d288c5617fa6	9ed5e522dd9dd85b4af4a077526d8117	d
8	0005a1a1728c9d785b8e2b08b904576c	16150771dfd4776261284213b89c304e	d
9	0005f50442cb953dcd1d21e1fb923495	351d3cb2cee3c7fd0af6616c82df21d3	d

In [264…

```
query=text("""SELECT SUM(CASE WHEN order_purchase_timestamp = '0000-00-00 00:0
                      SUM(CASE WHEN order_approved_at = '0000-00-00 00:00:00' T
                      SUM(CASE WHEN order_delivered_carrier_date = '0000-00-00
                      SUM(CASE WHEN order_delivered_customer_date = '0000-00-00
```

```
                    SUM(CASE WHEN order_estimated_delivery_date = '0000-00-00
                    FROM orders_data;""")
pd.read_sql_query(query,conn)
```

Out[264…

	order_purchase_timestamp_zero_date	order_approved_at_zero_date	order_del
0	0.0	0.0	

products_data

Action in data transformation

- orphan rows were inserted in parent table(products_categories_data).
- A new column(category_sk) was added to the product_data_unique table, which will store the foreign key referencing the products_categories_data table.
- NOT NULL constraints were applied to all columns except product_category_name, allowing optional category information.
- product_id was assigned as the primary key.
- New corrected columns were created to fix spelling issues:
    - product_name_length
    - product_description_length
    - These columns replaced the misspelled length-related columns while preserving the original columns for traceability.
- A new column product_category_name_clean was created to store standardized, readable category names in approximate title case for improved reporting and analytical consistency.

In [265…

```
query=text("""SET FOREIGN_KEY_CHECKS = 0;""")
conn.execute(query)
conn.commit()
```

In [266…

```
check_table_exists_query=text("""
                        SELECT COUNT(*) FROM information_schema.tables
                        WHERE table_schema='olist_db'
                        AND table_name='product_data_unique';
                    """)

if conn.execute(check_table_exists_query).scalar()==1:
    conn.execute(text("""TRUNCATE TABLE product_data_unique;"""))
    conn.commit()
    print("==========All Rows Deleted Successfully✅==========")
else:
    print("==========Table Does Not Exists❌==========")
```

```
==========Table Does Not Exists❌==========
```

In [267…
```
query=text("""SET FOREIGN_KEY_CHECKS = 1;""")
conn.execute(query)
conn.commit()
```

In [268…
```
query=text("""SET FOREIGN_KEY_CHECKS = 0;""")
conn.execute(query)
conn.commit()
```

In [269…
```
query = text("DROP TABLE IF EXISTS product_data_unique;")
conn.execute(query)
conn.commit()
```

In [270…
```
query=text("""SET FOREIGN_KEY_CHECKS = 1;""")
conn.execute(query)
conn.commit()
```

In [271…
```
query=text("""UPDATE products_data
SET product_category_name= 'Not defined'
WHERE TRIM(product_category_name) = '';""")
conn.execute(query)
conn.commit()
```

In [272…
```
query=text("""INSERT INTO products_categories_data (product_category_name,proc
              SELECT DISTINCT p.product_category_name,'Not defined' AS product
              LEFT JOIN products_categories_data pc
              ON p.product_category_name=pc.product_category_name
              WHERE pc.product_category_name IS NULL
              AND p.product_category_name IS NOT NULL;""")
conn.execute(query)
conn.commit()
```

In [273…
```
query=text("""SET FOREIGN_KEY_CHECKS = 0;""")
conn.execute(query)
conn.commit()
```

In [274…
```
query = text("""
CREATE TABLE product_data_unique AS
SELECT
    product_id,
    product_category_name,
    product_name_lenght,
    product_description_lenght,
    product_photos_qty,
    product_weight_g,
    product_length_cm,
    product_height_cm,
    product_width_cm
FROM products_data
GROUP BY
    product_id,
    product_category_name;
```

```
""")

conn.execute(query)
conn.commit()
print("==========Table Created Successfully✅==========")
```

```
==========Table Created Successfully✅==========
```

In [275…
```
query=text("""SET FOREIGN_KEY_CHECKS = 1;""")
conn.execute(query)
conn.commit()
```

In [276…
```
query=text("""ALTER TABLE product_data_unique
              MODIFY product_id VARCHAR(255) NOT NULL,
              MODIFY product_category_name VARCHAR(255),
              MODIFY product_name_lenght INTEGER NOT NULL,
              MODIFY product_description_lenght INTEGER NOT NULL,
              MODIFY product_photos_qty INTEGER NOT NULL,
              MODIFY product_weight_g INTEGER NOT NULL,
              MODIFY product_length_cm INTEGER NOT NULL,
              MODIFY product_height_cm INTEGER NOT NULL,
              MODIFY product_width_cm INTEGER NOT NULL;
""")
conn.execute(query)
conn.commit()
```

In [277…
```
check_primary_key_exists_query=text("""SHOW KEYS FROM product_data_unique
                                WHERE key_name='PRIMARY';""")
if conn.execute(check_primary_key_exists_query).fetchone()==None:
    query=text("""ALTER TABLE product_data_unique
                  ADD PRIMARY KEY (product_id);""")
    conn.execute(query)
    conn.commit()
    print("==========Primary key Created Successfully✅==========")
else:
    print("==========Primary key Already Exists❌==========")
```

```
==========Primary key Created Successfully✅==========
```

In [278…
```
check_column_exists_query=text("""SELECT COUNT(*) FROM information_schema.colu
                          WHERE table_schema='olist_db'
                          AND table_name='product_data_unique'
                          AND column_name='product_category_name_clean';""")
if conn.execute(check_column_exists_query).scalar()==0:
    query=text("""ALTER TABLE product_data_unique
                  ADD column product_category_name_clean VARCHAR(255) NULL;"""
    conn.execute(query)
    conn.commit()
    print("==========Column Created Successfully✅==========")
else:
    print("==========Column Already Exists❌==========")
```

```
==========Column Created Successfully✅==========
```

In [279…
```
check_column_exists_query=text("""SELECT COUNT(*) FROM information_schema.colu
                          WHERE table_schema='olist_db'
                          AND table_name='product_data_unique'
                          AND column_name='category_sk';""")
if conn.execute(check_column_exists_query).scalar()==0:
    query=text("""ALTER TABLE product_data_unique
                  ADD COLUMN category_sk INT;""")
    conn.execute(query)
    conn.commit()
    print("==========Column Created Successfully✅==========")
else:
    print("==========Column Already Exists❌==========")
```

```
==========Column Created Successfully✅==========
```

In [280…
```
query=text("""UPDATE product_data_unique
              SET product_category_name_clean=TRIM(product_category_name)
              WHERE product_category_name IS NOT NULL
              AND TRIM(product_category_name) <> '';""")
conn.execute(query)
conn.commit()
```

In [281…
```
query=text("""UPDATE product_data_unique
              SET product_category_name_clean = 'Not defined'
              WHERE product_category_name_clean IS NULL
              OR TRIM(product_category_name_clean) = '';
""")
conn.execute(query)
conn.commit()
```

In [282…
```
query=text("""UPDATE product_data_unique
              SET product_category_name_clean=CONCAT(UPPER(LEFT(TRIM(product_c
              WHERE product_category_name IS NOT NULL AND TRIM(product_categor
conn.execute(query)
conn.commit()
```

In [283…
```
query=text("""DESCRIBE product_data_unique;""")
pd.read_sql_query(query,conn)
```

Out[283…

	Field	Type	Null	Key	Default	Extra
0	product_id	varchar(255)	NO	PRI	None	
1	product_category_name	varchar(255)	YES		None	
2	product_name_lenght	int(11)	NO		None	
3	product_description_lenght	int(11)	NO		None	
4	product_photos_qty	int(11)	NO		None	
5	product_weight_g	int(11)	NO		None	
6	product_length_cm	int(11)	NO		None	
7	product_height_cm	int(11)	NO		None	
8	product_width_cm	int(11)	NO		None	
9	product_category_name_clean	varchar(255)	YES		None	
10	category_sk	int(11)	YES		None	

In [284…

```
query=text("""SELECT * FROM product_data_unique
              ORDER BY product_id
              LIMIT 10;""")
pd.read_sql_query(query,conn)
```

Out[284…

	product_id	product_category_name	product_name_le
0	00066f42aeeb9f3007548bb9d3f33c38	perfumaria	
1	00088930e925c41fd95ebfe695fd2655	automotivo	
2	0009406fd7479715e4bef61dd91f2462	cama_mesa_banho	
3	000b8f95fcb9e0096488278317764d19	utilidades_domesticas	
4	000d9be29b5207b54e86aa1b1ac54872	relogios_presentes	
5	0011c512eb256aa0dbbb544d8dffcf6e	automotivo	
6	00126f27c813603687e6ce486d909d01	cool_stuff	
7	001795ec6f1b187d37335e1c4704762e	consoles_games	
8	001b237c0e9bb435f2e54071129237e9	cama_mesa_banho	
9	001b72dfd63e9833e8c02742adf472e3	moveis_decoracao	

In [285…

```
query=text("""SELECT COUNT(*) AS total_orphan_product_category_name FROM produ
              LEFT JOIN products_categories_data pc
              ON p.product_category_name=pc.product_category_name
              WHERE pc.product_category_name IS NULL
                    AND p.product_category_name IS NOT NULL;""")
pd.read_sql_query(query,conn)
```

Out[285…

	total_orphan_product_category_name
0	0

sellers_data

Action in data cleaning

- 799 duplicate rows were removed to ensure unique seller records.

Actions in Data Transformation

- orphan rows were inserted in parent table(geolocation_data)
- NOT NULL constraints were applied to all columns to eliminate missing values in mandatory fields.
- seller_id was assigned as the primary key.
- A new column seller_state_full was created to store full state names while preserving the original seller_state abbreviations.
- A new column seller_city_clean was created to store standardized, readable city names in approximate title case for improved reporting and analytical consistency.

In [286…

```
query=text("""SET FOREIGN_KEY_CHECKS = 0;""")
conn.execute(query)
conn.commit()
```

In [287…

```
check_table_exists_query=text("""
                        SELECT COUNT(*) FROM information_schema.tables
                        WHERE table_schema='olist_db'
                        AND table_name='seller_data_unique';
                    """)

if conn.execute(check_table_exists_query).scalar()==1:
    conn.execute(text("""TRUNCATE TABLE seller_data_unique;"""))
    conn.commit()
    print("==========All Rows Deleted Successfully✅==========")
else:
    print("==========Table Does Not Exists❌==========")
```

```
==========Table Does Not Exists❌==========
```

In [288…

```
query=text("""SET FOREIGN_KEY_CHECKS = 1;""")
conn.execute(query)
conn.commit()
```

In [289…

```
query=text("""INSERT INTO geolocation_data (geolocation_zip_code_prefix,geoloc
              SELECT   s.seller_zip_code_prefix,
                       0.0 AS geolocation_lat,
```

```
                      0.0 AS geolocation_lng,
                      'Not defined' AS geolocation_city,
                      'Not defined' AS geolocation_state FROM sellers_data s
               LEFT JOIN geolocation_data g
               ON s.seller_zip_code_prefix=g.geolocation_zip_code_prefix
               WHERE g.geolocation_zip_code_prefix IS NULL;""")
conn.execute(query)
conn.commit()
```

In [290…
```
query=text("""SET FOREIGN_KEY_CHECKS = 0;""")
conn.execute(query)
conn.commit()
```

In [291…
```
query = text("DROP TABLE IF EXISTS seller_data_unique;")
conn.execute(query)
conn.commit()
```

In [292…
```
query = text("""
CREATE TABLE seller_data_unique AS
SELECT
    seller_id,
    seller_zip_code_prefix,
    seller_city,
    seller_state
FROM sellers_data
GROUP BY
    seller_id,
    seller_zip_code_prefix;
""")

conn.execute(query)
conn.commit()
print("==========Table Created Successfully✅==========")
```

```
==========Table Created Successfully✅==========
```

In [293…
```
query=text("""SET FOREIGN_KEY_CHECKS = 1;""")
conn.execute(query)
conn.commit()
```

In [294…
```
query=text("""ALTER TABLE seller_data_unique
               MODIFY seller_id VARCHAR(255) NOT NULL,
               MODIFY seller_zip_code_prefix INTEGER NOT NULL,
               MODIFY seller_city VARCHAR(255) NOT NULL,
               MODIFY seller_state VARCHAR(255) NOT NULL;
""")
conn.execute(query)
conn.commit()
```

In [295…
```
check_primary_key_exists_query=text("""SHOW KEYS FROM seller_data_unique
                                WHERE key_name='PRIMARY';""")
if conn.execute(check_primary_key_exists_query).fetchone()==None:
```

```
    query=text("""ALTER TABLE seller_data_unique
                  ADD PRIMARY KEY(seller_id);""")
    conn.execute(query)
    conn.commit()
    print("==========Primary key Created Successfully✅==========")
else:
    print("==========Primary key Already Exists❌==========")
```

```
==========Primary key Created Successfully✅==========
```

In [296…
```
check_column_exists_query=text("""SELECT COUNT(*) FROM information_schema.colu
                           WHERE table_schema='olist_db'
                           AND table_name='seller_data_unique'
                           AND column_name='seller_state_full';""")
if conn.execute(check_column_exists_query).scalar()==0:
    query=text("""ALTER TABLE seller_data_unique
                   ADD COLUMN seller_state_full VARCHAR(255) NULL;""")
    conn.execute(query)
    conn.commit()
    print("==========Column Created Successfully✅==========")
else:
    print("==========Column Already Exists❌==========")
```

```
==========Column Created Successfully✅==========
```

In [297…
```
check_index_exists_query=text("""SELECT COUNT(*) FROM information_schema.stati
                           WHERE table_schema='olist_db'
                           AND table_name='seller_data_unique'
                           AND column_name='seller_state'""")
if conn.execute(check_index_exists_query).scalar()==0:
    query=text("""CREATE INDEX seller_data_unique_seller_state_idx ON seller_d
    conn.execute(query)
    conn.commit()
    print("==========Index Created Successfully✅==========")
else:
    print("==========Index Already Exists❌==========")
```

```
==========Index Created Successfully✅==========
```

In [298…
```
query=text("""UPDATE seller_data_unique SET seller_state_full=
                        CASE TRIM(seller_state)
                    WHEN 'AC' THEN 'Acre'
                    WHEN 'AL' THEN 'Alagoas'
                    WHEN 'AP' THEN 'Amapa'
                    WHEN 'AM' THEN 'Amazonas'
                    WHEN 'BA' THEN 'Bahia'
                    WHEN 'CE' THEN 'Ceara'
                    WHEN 'DF' THEN 'Distrito Federal'
                    WHEN 'ES' THEN 'Espirito Santo'
                    WHEN 'GO' THEN 'Goias'
                    WHEN 'MA' THEN 'Maranhao'
                    WHEN 'MT' THEN 'Mato Grosso'
                    WHEN 'MS' THEN 'Mato Grosso do Sul'
```

```
                    WHEN 'MG' THEN 'Minas Gerais'
                    WHEN 'PA' THEN 'Para'
                    WHEN 'PB' THEN 'Paraiba'
                    WHEN 'PR' THEN 'Parana'
                    WHEN 'PE' THEN 'Pernambuco'
                    WHEN 'PI' THEN 'Piaui'
                    WHEN 'RJ' THEN 'Rio de Janeiro'
                    WHEN 'RN' THEN 'Rio Grande do Norte'
                    WHEN 'RS' THEN 'Rio Grande do Sul'
                    WHEN 'RO' THEN 'Rondonia'
                    WHEN 'RR' THEN 'Roraima'
                    WHEN 'SC' THEN 'Santa Catarina'
                    WHEN 'SP' THEN 'Sao Paulo'
                    WHEN 'SE' THEN 'Sergipe'
                    WHEN 'TO' THEN 'Tocantins'
                    ELSE 'Unknown'
                END
                WHERE seller_state IS NOT NULL
                AND TRIM(seller_state) <> ''
               ;""")
conn.execute(query)
conn.commit()
```

In [299…

```
check_column_exists_query=text("""SELECT COUNT(*) FROM information_schema.colu
                          WHERE table_schema='olist_db'
                          AND table_name='seller_data_unique'
                          AND column_name='seller_city_clean';""")
if conn.execute(check_column_exists_query).scalar()==0:
    query=text("""ALTER TABLE seller_data_unique
                  ADD COLUMN seller_city_clean VARCHAR(255) NULL;""")
    conn.execute(query)
    conn.commit()
    print("==========Column Created Successfully✅==========")
else:
    print("==========Column Already Exists❌==========")
```

```
==========Column Created Successfully✅==========
```

In [300…

```
query=text("""UPDATE seller_data_unique SET seller_city_clean=CONCAT(UPPER(LEF
conn.execute(query)
conn.commit()
```

In [301…

```
query=text("""DESCRIBE seller_data_unique;""")
pd.read_sql_query(query,conn)
```

Out[301…

	Field	Type	Null	Key	Default	Extra
0	seller_id	varchar(255)	NO	PRI	None	
1	seller_zip_code_prefix	int(11)	NO		None	
2	seller_city	varchar(255)	NO		None	
3	seller_state	varchar(255)	NO	MUL	None	
4	seller_state_full	varchar(255)	YES		None	
5	seller_city_clean	varchar(255)	YES		None	

In [302…

```
query=text("""SELECT * FROM seller_data_unique ORDER BY seller_id LIMIT 10;"""
pd.read_sql_query(query,conn)
```

Out[302…

	seller_id	seller_zip_code_prefix	seller_city	seller_s
0	0015a82c2db000af6aaaf3ae2ecb0532	9080	santo andre	
1	001cca7ae9ae17fb1caed9dfb1094831	29156	cariacica	
2	001e6ad469a905060d959994f1b41e4f	24754	sao goncalo	
3	002100f778ceb8431b7a1020ff7ab48f	14405	franca	
4	003554e2dce176b5555353e4f3555ac8	74565	goiania	
5	004c9cd9d87a3c30c522c48c4fc07416	14940	ibitinga	
6	00720abe85ba0859807595bbf045a33b	7070	guarulhos	
7	00ab3eff1b5192e5f1a63bcecfee11c8	4164	sao paulo	
8	00d8b143d12632bad99c0ad66ad52825	30170	belo horizonte	
9	00ee68308b45bc5e2660cd833c3f81cc	3333	sao paulo	

In [303…

```
query=text("""SELECT COUNT(*) AS total_orphan_seller_zip_code FROM sellers_dat
              LEFT JOIN geolocation_data g
              ON s.seller_zip_code_prefix=g.geolocation_zip_code_prefix
              WHERE g.geolocation_zip_code_prefix IS NULL;
                    """)
pd.read_sql_query(query,conn)
```

Out[303…

	total_orphan_seller_zip_code
0	0

products_categories_data

Action in data transformation

- category_id was assigned as the primary key
- A surrogate key category_sk was created to uniquely identify each row and to comply with First Normal Form (1NF). This column was assigned a primary key constraint.
- New columns product_category_name_clean and product_category_name_english_clean were created to store standardized, readable category names in approximate title case, while preserving the original columns.

In [304…
```
check_column_exists_query=text("""SELECT COUNT(*) FROM information_schema.colu
                          WHERE table_schema='olist_db'
                          AND table_name='products_categories_data'
                          AND column_name='category_sk';""")
if conn.execute(check_column_exists_query).scalar()==0:
    query=text("""ALTER TABLE products_categories_data
                  ADD COLUMN category_sk INTEGER AUTO_INCREMENT PRIMARY KEY;""
    conn.execute(query)
    conn.commit()
    print("==========Column Created Successfully✅==========")
else:
    print("==========Column Already Exists❌==========")
```

```
==========Column Created Successfully✅==========
```

In [305…
```
query=text("""UPDATE product_data_unique p
JOIN products_categories_data c
  ON p.product_category_name = c.product_category_name
SET p.category_sk = c.category_sk;""")
conn.execute(query)
conn.commit()
```

In [306…
```
check_column_exists_query=text("""SELECT COUNT(*) FROM information_schema.colu
                          WHERE table_schema='olist_db'
                          AND table_name='products_categories_data'
                          AND column_name='product_category_name_clean';""")
if conn.execute(check_column_exists_query).scalar()==0:
    query=text("""ALTER TABLE products_categories_data
                  ADD COLUMN product_category_name_clean VARCHAR(255) NULL;"""
    conn.execute(query)
    conn.commit()
    print("==========Column Created Successfully✅==========")
else:
    print("==========Column Already Exists❌==========")
```

```
==========Column Created Successfully✅==========
```

In [307…
```
query=text("""UPDATE products_categories_data SET product_category_name_clean=
conn.execute(query)
conn.commit()
```

In [308…
```
check_column_exists_query=text("""SELECT COUNT(*) FROM information_schema.colu
                          WHERE table_schema='olist_db'
                          AND table_name='products_categories_data'
                          AND column_name='product_category_name_english_clear
if conn.execute(check_column_exists_query).scalar()==0:
    query=text("""ALTER TABLE products_categories_data
                  ADD COLUMN product_category_name_english_clean VARCHAR(255)
    conn.execute(query)
    conn.commit()
    print("==========Column Created Successfully✅==========")
else:
    print("==========Column Already Exists❌==========")
```

```
==========Column Created Successfully✅==========
```

In [309…
```
query=text("""UPDATE products_categories_data SET product_category_name_englis
conn.execute(query)
conn.commit()
```

In [310…
```
query=text("""DESCRIBE products_categories_data;""")
pd.read_sql_query(query,conn)
```

Out[310…

	Field	Type	Null	Key	Default	E
0	product_category_name	varchar(255)	YES	MUL	None	
1	product_category_name_english	varchar(255)	YES		None	
2	category_sk	int(11)	NO	PRI	None	auto_increr
3	product_category_name_clean	varchar(255)	YES		None	
4	product_category_name_english_clean	varchar(255)	YES		None	

In [311…
```
query=text("""SELECT * FROM products_categories_data ORDER BY product_category
pd.read_sql_query(query,conn)
```

Out[311…

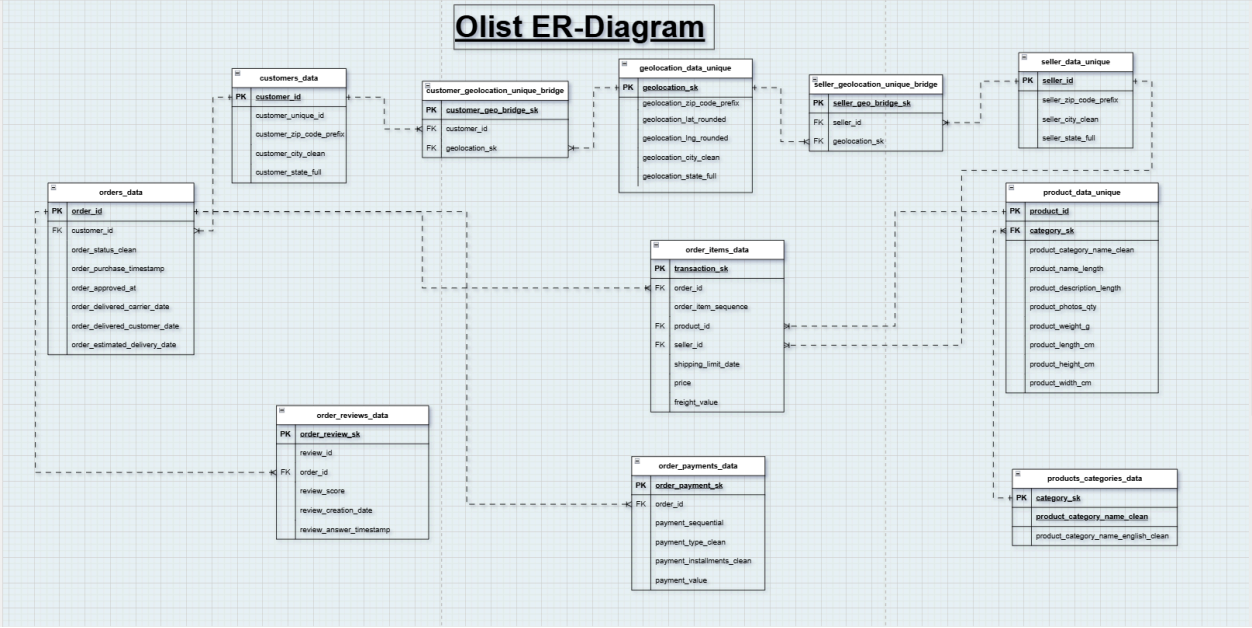
	product_category_name	product_category_name_english	category_sk	produc
0	agro_industria_e_comercio	agro_industry_and_commerce	38	
1	alimentos	food	46	
2	alimentos_bebidas	food_drink	11	
3	artes	art	47	
4	artes_e_artesanato	arts_and_craftmanship	68	
5	artigos_de_festas	party_supplies	35	
6	artigos_de_natal	christmas_supplies	55	
7	audio	audio	21	
8	automotivo	auto	3	
9	bebes	baby	12	

Schema Design

1. Designed a partially normalized (3NF) ER diagram for the Olist e-commerce dataset, defining clear primary and foreign key relationships across customers, orders, products, sellers, payments, reviews, and geolocation entities.
2. Implemented one-to-many relationships (Customers → Orders, Orders → Order Items, Orders → Payments/Reviews, Products/Sellers → Order Items) to enforce referential integrity.
3. Introduced surrogate keys for dimension and fact tables (transaction_sk, order_payment_sk, order_review_sk, category_sk) to ensure referential integrity, simplify joins, and support scalable analytics.
4. Modeled geolocation via bridge tables to eliminate redundancy and handle many-to-many relationships between customers/sellers and locations.
5. Standardized column naming conventions and cleaned categorical fields to maintain a professional, consistent schema.

ER Diagram

In [59]:
```
from IPython.display import Image,display
display(Image(filename=r"D:/Eda project on Olist dataset by Sql/Olist ER-diagr
```



Foreign key

In [313…
```
query=text("""SET FOREIGN_KEY_CHECKS = 0;""")
conn.execute(query)
conn.commit()
```

In [314…
```
check_table_exists_query=text("""
                        SELECT COUNT(*) FROM information_schema.tables
                        WHERE table_schema='olist_db'
                        AND table_name='customer_geolocation_unique_bridge';
                    """)

if conn.execute(check_table_exists_query).scalar()==1:
    conn.execute(text("""TRUNCATE TABLE customer_geolocation_unique_bridge;"""
    conn.commit()
    print("==========All Rows Deleted Successfully✅==========")
else:
    print("==========Table Does Not Exists❌==========")
```

```
==========Table Does Not Exists❌==========
```

In [315…
```
query=text("""SET FOREIGN_KEY_CHECKS = 1;""")
conn.execute(query)
conn.commit()
```

In [316…
```
check_table_exists_query=text("""SELECT COUNT(*) FROM information_schema.table
                            WHERE table_schema='olist_db'
```

```
                          AND table_name='customer_geolocation_unique_bridge'
                          ;""")
if conn.execute(check_table_exists_query).scalar()==0:
    query=text("""CREATE TABLE customer_geolocation_unique_bridge(customer_geo
                                                        customer_id VARCHAR(255
                                                        geolocation_sk INTEGER
                                                        CONSTRAINT customer_geo
                                                        ON DELETE CASCADE
                                                        ON UPDATE CASCADE,
                                                        CONSTRAINT customer_geo
                                                        ON DELETE CASCADE
                                                        ON UPDATE CASCADE,
                                                        CONSTRAINT customer_geo
                                                        );""")
    conn.execute(query)
    conn.commit()
    print("==========Table Created Successfully✅==========")
else:
    print("==========Table Already Exists❌==========")
```

```
==========Table Created Successfully✅==========
```

In [317…

```
check_index_exists_query=text("""SELECT COUNT(*) FROM information_schema.stati
                          WHERE table_schema='olist_db'
                          AND table_name='geolocation_data_unique'
                          AND column_name='geolocation_zip_code_prefix';""")
if conn.execute(check_index_exists_query).scalar()==0:
    query=text("""CREATE INDEX geolocation_data_unique_geolocation_zip_code_pr
    conn.execute(query)
    conn.commit()
    print("==========Index Created Successfully✅==========")
else:
    print("==========Index Already Exists❌==========")
```

```
==========Index Created Successfully✅==========
```

In [318…

```
query=text("""INSERT IGNORE INTO customer_geolocation_unique_bridge (customer_
SELECT
    c.customer_id,
    g.geolocation_sk
FROM customers_data c
JOIN (
    SELECT
        geolocation_zip_code_prefix,
        MIN(geolocation_sk) AS geolocation_sk
    FROM geolocation_data_unique
    GROUP BY geolocation_zip_code_prefix
) g
ON c.customer_zip_code_prefix = g.geolocation_zip_code_prefix;""")
conn.execute(query)
conn.commit()
```

In [319…

```
query=text("""SET FOREIGN_KEY_CHECKS = 0;""")
conn.execute(query)
```

```
conn.commit()
```

In [320…
```
check_table_exists_query=text("""
                         SELECT COUNT(*) FROM information_schema.tables
                         WHERE table_schema='olist_db'
                         AND table_name='seller_geolocation_unique_bridge';
                       """)

if conn.execute(check_table_exists_query).scalar()==1:
    conn.execute(text("""TRUNCATE TABLE seller_geolocation_unique_bridge;"""))
    conn.commit()
    print("==========All Rows Deleted Successfully✅==========")
else:
    print("==========Table Does Not Exists❌==========")
```

==========Table Does Not Exists❌==========

In [321…
```
query=text("""SET FOREIGN_KEY_CHECKS = 1;""")
conn.execute(query)
conn.commit()
```

In [322…
```
check_table_exists_query=text("""SELECT COUNT(*) FROM information_schema.table
                         WHERE table_schema='olist_db'
                         AND table_name='seller_geolocation_unique_bridge'
                         ;""")
if conn.execute(check_table_exists_query).scalar()==0:
    query=text("""CREATE TABLE seller_geolocation_unique_bridge(seller_geo_bri
                                                    seller_id VARCHAR(255) NO
                                                    geolocation_sk INTEGER NO
                                                    CONSTRAINT seller_geoloca
                                                    ON DELETE CASCADE
                                                    ON UPDATE CASCADE,
                                                    CONSTRAINT seller_geoloca
                                                    ON DELETE CASCADE
                                                    ON UPDATE CASCADE,
                                                    CONSTRAINT seller_geoloca
                                                    );""")
    conn.execute(query)
    conn.commit()
    print("==========Table Created Successfully✅==========")
else:
    print("==========Table Already Exists❌==========")
```

==========Table Created Successfully✅==========

In [323…
```
check_index_exists_query=text("""SELECT COUNT(*) FROM information_schema.stati
                         WHERE table_schema='olist_db'
                         AND table_name='seller_data_unique'
                         AND column_name='seller_zip_code_prefix';""")
if conn.execute(check_index_exists_query).scalar()==0:
    query=text("""CREATE INDEX seller_data_unique_seller_zip_code_prefix_idx C
    conn.execute(query)
    conn.commit()
```

```
    print("==========Index Created Successfully✅==========")
else:
    print("==========Index Already Exists❌==========")
```

```
==========Index Created Successfully✅==========
```

In [324…

```
query=text("""INSERT IGNORE INTO seller_geolocation_unique_bridge (seller_id,
SELECT
    s.seller_id,
    g.geolocation_sk
FROM sellers_data s
JOIN (
    SELECT
        geolocation_zip_code_prefix,
        MIN(geolocation_sk) AS geolocation_sk
    FROM geolocation_data_unique
    GROUP BY geolocation_zip_code_prefix
) g
ON s.seller_zip_code_prefix = g.geolocation_zip_code_prefix;""")
conn.execute(query)
conn.commit()
```

In [325…

```
check_fk_exists_query = text("""
SELECT COUNT(*)
FROM information_schema.table_constraints
WHERE table_schema = 'olist_db'
AND table_name = 'orders_data'
AND constraint_name = 'orders_data_customer_id_fk';
""")

if conn.execute(check_fk_exists_query).scalar() == 0:
    query = text("""
    ALTER TABLE orders_data
    ADD CONSTRAINT orders_data_customer_id_fk
    FOREIGN KEY(customer_id) REFERENCES customers_data(customer_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE;
    """)
    conn.execute(query)
    conn.commit()
    print("==========Foreign key Created Successfully✅==========")
else:
    print("==========Foreign key Already Exists❌==========")
```

```
==========Foreign key Created Successfully✅==========
```

In [326…

```
check_fk_exists_query = text("""
SELECT COUNT(*)
FROM information_schema.table_constraints
WHERE table_schema = 'olist_db'
AND table_name = 'order_reviews_data'
AND constraint_name = 'order_reviews_data_order_id_fk';
""")
```

```
if conn.execute(check_fk_exists_query).scalar() == 0:
    query = text("""
    ALTER TABLE order_reviews_data
    ADD CONSTRAINT order_reviews_data_order_id_fk FOREIGN KEY(order_id) REFERE
    ON DELETE CASCADE
    ON UPDATE CASCADE;
    """)
    conn.execute(query)
    conn.commit()
    print("==========Foreign key Created Successfully✅==========")
else:
    print("==========Foreign key Already Exists❌==========")
```

==========Foreign key Created Successfully✅==========

In [327…
```
check_fk_exists_query = text("""
SELECT COUNT(*)
FROM information_schema.table_constraints
WHERE table_schema = 'olist_db'
AND table_name = 'order_payments_data'
AND constraint_name = 'order_payments_data_order_id_fk';
""")

if conn.execute(check_fk_exists_query).scalar() == 0:
    query = text("""
    ALTER TABLE order_payments_data
    ADD CONSTRAINT order_payments_data_order_id_fk FOREIGN KEY(order_id) REFER
    ON DELETE CASCADE
    ON UPDATE CASCADE;
    """)
    conn.execute(query)
    conn.commit()
    print("==========Foreign key Created Successfully✅==========")
else:
    print("==========Foreign key Already Exists❌==========")
```

==========Foreign key Created Successfully✅==========

In [328…
```
check_index_exists_query=text("""SELECT COUNT(*) FROM information_schema.stati
                          WHERE table_schema='olist_db'
                          AND table_name='product_data_unique'
                          AND column_name='product_category_name';""")
if conn.execute(check_index_exists_query).scalar()==0:
    query=text("""CREATE INDEX product_data_unique_product_category_name_idx C
    conn.execute(query)
    conn.commit()
    print("==========Index Created Successfully✅==========")
else:
    print("==========Index Already Exists❌==========")
```

==========Index Created Successfully✅==========

In [329…
```
check_fk_exists_query = text("""
SELECT COUNT(*)
FROM information_schema.table_constraints
```

```
WHERE table_schema = 'olist_db'
AND table_name = 'product_data_unique'
AND constraint_name = 'product_data_unique_category_sk_fk';
""")

if conn.execute(check_fk_exists_query).scalar() == 0:
    query = text("""
    ALTER TABLE product_data_unique
    ADD CONSTRAINT product_data_unique_category_sk_fk
    FOREIGN KEY (category_sk)
    REFERENCES products_categories_data(category_sk)
    ON DELETE RESTRICT
    ON UPDATE CASCADE;
    """)
    conn.execute(query)
    conn.commit()
    print("==========Foreign key Created Successfully✅==========")
else:
    print("==========Foreign key Already Exists❌==========")
```

==========Foreign key Created Successfully✅==========

In [330…

```
check_index_exists_query=text("""SELECT COUNT(*) FROM information_schema.stati
                          WHERE table_schema='olist_db'
                          AND table_name='order_items_data'
                          AND column_name='order_id';""")
if conn.execute(check_index_exists_query).scalar()==0:
    query=text("""CREATE INDEX order_items_data_order_id_idx ON order_items_da
    conn.execute(query)
    conn.commit()
    print("==========Index Created Successfully✅==========")
else:
    print("==========Index Already Exists❌==========")
```

==========Index Already Exists❌==========

In [331…

```
check_fk_exists_query = text("""
SELECT COUNT(*)
FROM information_schema.table_constraints
WHERE table_schema = 'olist_db'
AND table_name = 'order_items_data'
AND constraint_name = 'order_items_data_order_id_fk';
""")

if conn.execute(check_fk_exists_query).scalar() == 0:
    query = text("""
    ALTER TABLE order_items_data
    ADD CONSTRAINT order_items_data_order_id_fk FOREIGN KEY(order_id) REFERENC
    ON DELETE CASCADE
    ON UPDATE CASCADE;
    """)
    conn.execute(query)
    conn.commit()
    print("==========Foreign key Created Successfully✅==========")
```

```
else:
    print("==========Foreign key Already Exists❌==========")
```

```
==========Foreign key Created Successfully✅==========
```

In [332…

```
check_fk_exists_query = text("""
SELECT COUNT(*)
FROM information_schema.table_constraints
WHERE table_schema = 'olist_db'
AND table_name = 'order_items_data'
AND constraint_name = 'order_items_data_product_id_fk';
""")

if conn.execute(check_fk_exists_query).scalar() == 0:
    query = text("""
    ALTER TABLE order_items_data
    ADD CONSTRAINT order_items_data_product_id_fk FOREIGN KEY(product_id) REFE
    ON DELETE CASCADE
    ON UPDATE CASCADE;
    """)
    conn.execute(query)
    conn.commit()
    print("==========Foreign key Created Successfully✅==========")
else:
    print("==========Foreign key Already Exists❌==========")
```

```
==========Foreign key Created Successfully✅==========
```

In [333…

```
check_fk_exists_query = text("""
SELECT COUNT(*)
FROM information_schema.table_constraints
WHERE table_schema = 'olist_db'
AND table_name = 'order_items_data'
AND constraint_name = 'order_items_data_seller_id_fk';
""")

if conn.execute(check_fk_exists_query).scalar() == 0:
    query = text("""
    ALTER TABLE order_items_data
    ADD CONSTRAINT order_items_data_seller_id_fk FOREIGN KEY(seller_id) REFERE
    ON DELETE CASCADE
    ON UPDATE CASCADE;
    """)
    conn.execute(query)
    conn.commit()
    print("==========Foreign key Created Successfully✅==========")
else:
    print("==========Foreign key Already Exists❌==========")
```

```
==========Foreign key Created Successfully✅==========
```

Bussiness problem questions

1. What is the total revenue generated by Olist, and how has it changed over time?
2. How many orders were placed on Olist, and how does this vary by month or season?
3. What are the most popular product categories on Olist, and how do their sales volumes compare to each other?
4. What is the average order value (AOV) on Olist, and how does this vary by product category or payment method?
5. How many sellers are active on Olist, and how does this number change over time?
6. What is the distribution of seller ratings on Olist, and how does this impact sales performance?
7. How many customers have made repeated purchases on Olist, and what percentage of total sales do they account for?
8. What is the average customer rating for products sold on Olist, and how does this impact sales performance?
9. What is the average order cancellation rate on Olist, and how does this impact seller performance?
10. What are the top-selling products on Olist, and how has their sales volume changed over time?
11. Which payment methods are most commonly used by Olist customers, and how does this vary by product category or geographic region?
12. Which product categories have the highest profit margins on Olist, and how can the company increase profitability across different categories?
13. Geolocation having high customer density. Calculate customer retention rate according to geolocations.

Sql Analysis

Question No 1

1. What is the total revenue generated by Olist, and how has it changed over time?

Key Insights

1. Olist generated approximately R$15.24M in total revenue from delivered orders.
2. 2017 shows the highest revenue, as it is the only year with complete data coverage.
3. Revenue steadily increased month-over-month during 2017, with noticeable peaks toward the year end, indicating strong seasonal demand.
4. 2016 and 2018 revenue figures are lower due to partial-year data availability, so direct comparison with 2017 should be avoided.

`Overall, Olist demonstrates a strong upward revenue trend, reflecting business growth over time.`

In [12]:
```
# Total revenue
query=text("""SELECT ROUND(SUM(op.payment_value), 2) AS total_revenue
              FROM order_payments_data op
              JOIN orders_data o
                 ON op.order_id = o.order_id
              WHERE o.order_status = 'delivered';
              """)
pd.read_sql_query(query,conn)
```

Out[12]:

	total_revenue
0	15422461.77

In [10]:
```
# Revenue trends over months
query=text("""SELECT DATE_FORMAT(order_purchase_timestamp,'%Y-%m') AS month,
                     ROUND(SUM(op.payment_value), 2) AS total_revenue
              FROM order_payments_data op
              JOIN orders_data o
                 ON op.order_id = o.order_id
              WHERE o.order_status = 'delivered'
```

```
            AND DATE_FORMAT(order_purchase_timestamp,'%Y-%m') IS NOT NULL
            GROUP BY DATE_FORMAT(order_purchase_timestamp,'%Y-%m')
            ORDER BY month;
            """)
pd.read_sql_query(query,conn)
```

Out[10]:

	month	total_revenue
0	2016-10	46566.71
1	2016-12	19.62
2	2017-01	127545.67
3	2017-02	271298.65
4	2017-03	414369.39
5	2017-04	390952.18
6	2017-05	567066.73
7	2017-06	490225.60
8	2017-07	566403.93
9	2017-08	646000.61
10	2017-09	701169.99
11	2017-10	751140.27
12	2017-11	1153528.05
13	2017-12	843199.17
14	2018-01	1078606.86
15	2018-02	966510.88
16	2018-03	1120678.00
17	2018-04	1132933.95
18	2018-05	1128836.69
19	2018-06	1012090.68
20	2018-07	1027903.86
21	2018-08	985414.28

In [11]:

```
# Revenue trends over years
query=text("""SELECT DATE_FORMAT(order_purchase_timestamp,'%Y') AS year,
                     ROUND(SUM(op.payment_value), 2) AS total_revenue
              FROM order_payments_data op
              JOIN orders_data o
                 ON op.order_id = o.order_id
              WHERE o.order_status = 'delivered'
              AND DATE_FORMAT(order_purchase_timestamp,'%Y') IS NOT NULL
```

```
            GROUP BY DATE_FORMAT(order_purchase_timestamp,'%Y')
            ORDER BY year;""")
pd.read_sql_query(query,conn)
```

Out[11]:

	year	total_revenue
0	2016	46586.33
1	2017	6922900.24
2	2018	8452975.20

Question No 2

2. How many orders were placed on Olist, and how does this vary by month or season?

Key Insights

1. `Around 96.4K` total orders were placed on Olist (include only delivered orders).
2. Order volume increased significantly in 2017, which represents the first full year of data.
3. 2016 shows low order counts as it contains only partial-year data.
4. 2018 also has lower counts because data is available for only part of the year.
5. Orders are fairly evenly distributed across seasons, showing stable demand throughout the year.

In [9]:
```
# Total orders placed
query=text("""SELECT COUNT(DISTINCT order_id) AS total_orders_placed
            FROM orders_data
            WHERE order_status = 'delivered';""")
pd.read_sql_query(query,conn)
```

Out[9]:

	total_orders_placed
0	96478

In [10]:
```
# Orders vary over month
query=text("""SELECT
                DATE_FORMAT(order_purchase_timestamp,'%Y-%m') AS month,
                COUNT(DISTINCT order_id) AS total_orders_placed
```

```
            FROM orders_data
            WHERE order_status = 'delivered'
            GROUP BY DATE_FORMAT(order_purchase_timestamp,'%Y-%m')
            ORDER BY month;""")
pd.read_sql_query(query,conn)
```

Out[10]:

	month	total_orders_placed
0	2016-09	1
1	2016-10	265
2	2016-12	1
3	2017-01	750
4	2017-02	1653
5	2017-03	2546
6	2017-04	2303
7	2017-05	3546
8	2017-06	3135
9	2017-07	3872
10	2017-08	4193
11	2017-09	4150
12	2017-10	4478
13	2017-11	7289
14	2017-12	5513
15	2018-01	7069
16	2018-02	6555
17	2018-03	7003
18	2018-04	6798
19	2018-05	6749
20	2018-06	6099
21	2018-07	6159
22	2018-08	6351

In [11]:

```
# Orders vary over season
query=text("""SELECT CASE
                        WHEN MONTH(order_purchase_timestamp) IN (12,1,2) THEN
                        WHEN MONTH(order_purchase_timestamp) IN (3,4,5) THEN '
                        WHEN MONTH(order_purchase_timestamp) IN (6,7,8) THEN '
```

```
                       WHEN MONTH(order_purchase_timestamp) IN (9,10,11) THEN
                   END AS 'Season',
                   COUNT(DISTINCT order_id) AS total_orders_placed
            FROM orders_data
            WHERE order_status = 'delivered'
            GROUP BY Season
            ORDER BY Season;""")
pd.read_sql_query(query,conn)
```

Out[11]:

	Season	total_orders_placed
0	Autumn	28945
1	Spring	16183
2	Summer	21541
3	Winter	29809

Question No 3

3. What are the most popular product categories on Olist, and how do their sales volumes compare to each other?

Key Insights

1. Health & Beauty is the highest-selling category on Olist, generating `approximately R$1.25M`, reflecting strong customer demand for personal care products.

2. Watches & Gifts `(≈R$1.20M)`,Bed/Bath/Table `(≈R$1.03M)`,Sports_leisure `(≈R$0.9M)` and Computers_accessories `(≈R$0.9M)` also perform well, showing that customers spend on gifting ,home essentials,sports and computer products.

3. The top five categories account for the majority of total sales, while the remaining categories contribute moderately.

In [44]:
```
# Sales compare across top 20 different categories
query=text("""SELECT pc.product_category_name_english_clean,
                            COUNT(oi.order_item_sequence) AS total_units_sold
                            SUM(oi.price) AS total_sales
            FROM order_items_data oi
            JOIN product_data_unique p
                ON oi.product_id=p.product_id
            JOIN products_categories_data pc
```

```
                    ON p.category_sk=pc.category_sk
                GROUP BY pc.product_category_name_english_clean
                ORDER BY total_units_sold DESC
                LIMIT 20;
                """)
pd.read_sql_query(query,conn)
```

Out[44]:

	product_category_name_english_clean	total_units_sold	total_sales
0	Bed_bath_table	11115	1036988.68
1	Health_beauty	9670	1258681.34
2	Sports_leisure	8641	988048.97
3	Furniture_decor	8334	729762.49
4	Computers_accessories	7827	911954.32
5	Housewares	6964	632248.66
6	Watches_gifts	5991	1205005.68
7	Telephony	4545	323667.53
8	Garden_tools	4347	485256.46
9	Auto	4235	592720.11
10	Toys	4117	483946.60
11	Cool_stuff	3796	635290.85
12	Perfumery	3419	399124.87
13	Baby	3065	411764.89
14	Electronics	2767	160246.74
15	Stationery	2517	230943.23
16	Fashion_bags_accessories	2031	152823.54
17	Pet_shop	1947	214315.41
18	Office_furniture	1691	273960.70
19	Not defined	1627	185049.76

Question No 4

4. What is the average order value (AOV) on Olist, and how does this vary by product category or payment method?

Key Insights

1. Average Order Value (AOV) on Olist is `approximately R$159`.
2. Highest AOV categories: Computers, Fixed Telephony, Small Appliances/ Home Oven/Coffee.
3. Lowest AOV categories: CDs/DVDs/Musicals, Fashion/Children's Clothes, Technical Books, Flowers, Home Comfort.
    - This shows that customers spend more on electronics and home essentials, and less on everyday or gift items.
4. Payment types with highest AOV: Credit Card, Debit Card, and Boleto, followed by Voucher.
    - Suggests that these payment methods are associated with larger purchases.

In [22]:
```
# Average order value
query=text("""SELECT ROUND((SUM(op.payment_value)/COUNT(DISTINCT o.order_id)),

               FROM orders_data o
               JOIN order_payments_data op
                  ON o.order_id=op.order_id
               WHERE o.order_status = 'delivered';""")
pd.read_sql_query(query,conn)
```

Out[22]:

	average_order_value
0	159.86

In [23]:
```
# Average Order value vary over product_category
query=text("""SELECT pc.product_category_name_english_clean,
                     ROUND((SUM(op.payment_value)/COUNT(DISTINCT o.order_id)),

               FROM orders_data o
               JOIN order_payments_data op
                  ON o.order_id=op.order_id
               JOIN order_items_data oi
                  ON o.order_id=oi.order_id
```

```
                JOIN product_data_unique p
                   ON oi.product_id=p.product_id
                JOIN products_categories_data pc
                   ON p.category_sk=pc.category_sk
                WHERE payment_type <> 'not_defined'
                AND o.order_status = 'delivered'
                GROUP BY pc.product_category_name_english_clean
                ORDER BY average_order_value DESC
                ;""")
pd.read_sql_query(query,conn)
```

Out[23]:

	product_category_name_english_clean	average_order_value
0	Computers	1551.82
1	Fixed_telephony	907.26
2	Small_appliances_home_oven_and_coffee	698.36
3	Agro_industry_and_commerce	653.29
4	Home_appliances_2	523.59
5	Office_furniture	507.28
6	Signaling_and_security	494.66
7	Construction_tools_safety	383.22
8	Musical_instruments	361.90
9	Air_conditioning	361.18
10	Small_appliances	356.36
11	Furniture_bedroom	333.97
12	Furniture_living_room	324.39
13	Construction_tools_construction	323.20
14	Kitchen_dining_laundry_garden_furniture	304.59
15	Construction_tools_lights	290.67
16	Home_construction	278.71
17	Watches_gifts	252.48
18	Drinks	240.10
19	Industry_commerce_and_business	238.24
20	Computers_accessories	237.27
21	Garden_tools	235.10
22	La_cuisine	224.12
23	Furniture_decor	221.10
24	Auto	218.83
25	Costruction_tools_garden	218.46
26	Costruction_tools_tools	217.21
27	Home_confort	213.51
28	Cool_stuff	209.23
29	Health_beauty	187.45
30	Baby	187.10

	product_category_name_english_clean	average_order_value
31	Housewares	186.28
32	Bed_bath_table	182.56
33	Pet_shop	182.30
34	Music	181.62
35	Luggage_accessories	181.60
36	Consoles_games	180.55
37	Sports_leisure	179.21
38	Not defined	175.08
39	Audio	172.93
40	Diapers_and_hygiene	165.05
41	Market_place	162.83
42	Security_and_services	162.26
43	Perfumery	160.81
44	Furniture_mattress_and_upholstery	159.59
45	Toys	158.67
46	Fashion_male_clothing	157.77
47	Party_supplies	156.28
48	Christmas_supplies	151.15
49	Cine_photo	146.71
50	Art	146.15
51	Fashion_sport	138.69
52	Stationery	136.34
53	Fashio_female_clothing	136.19
54	Fashion_shoes	135.13
55	Books_imported	128.86
56	Tablets_printing_image	127.13
57	Books_general_interest	126.65
58	Home_appliances	125.63
59	Food_drink	120.40
60	Fashion_bags_accessories	117.32
61	Telephony	114.80

	product_category_name_english_clean	average_order_value
62	Dvds_blu_ray	114.59
63	Fashion_underwear_beach	106.13
64	Food	102.81
65	Fashion_childrens_clothes	102.71
66	Arts_and_craftmanship	101.14
67	Cds_dvds_musicals	99.95
68	Electronics	97.68
69	Books_technical	95.46
70	Flowers	76.31
71	Home_comfort_2	71.27

In [24]:
```
# Average Order value vary over payment type
query=text("""SELECT op.payment_type_clean,
                     ROUND((SUM(op.payment_value)/COUNT(DISTINCT o.order_id)),

               FROM orders_data o
               JOIN order_payments_data op
                  ON o.order_id=op.order_id
               WHERE payment_type <> 'not_defined'
               AND o.order_status = 'delivered'
               GROUP BY op.payment_type_clean
               ;""")
pd.read_sql_query(query,conn)
```

Out[24]:

	payment_type_clean	average_order_value
0	Boleto	144.33
1	Credit_card	162.86
2	Debit_card	140.35
3	Voucher	93.24

Question No 5

5. How many sellers are active on Olist, and how does this number change over time?

Key Insights

1. Around `3,095` sellers are active on the Olist platform overall.
2. Seller activity was low in 2016, showing the platform was in its early stage.
3. In 2017, the number of active sellers increased steadily each month, showing strong platform growth.
4. In 2018, seller activity continued to increase, but September shows very low seller count, likely due to missing or incomplete data.

`Overall, seller activity was much higher in 2017 and 2018 compared to 2016.`

In [58]:
```
# Active sellers
query=text("""SELECT COUNT(DISTINCT s.seller_id) AS active_seller
              FROM sellers_data s
              JOIN order_items_data oi
                ON s.seller_id=oi.seller_id
              ;""")
pd.read_sql_query(query,conn)
```

Out[58]:

	active_seller
0	3095

In [26]:
```
# Active sellers vary over month
query=text("""SELECT DATE_FORMAT(o.order_purchase_timestamp,'%Y-%m') AS month,
                     COUNT(DISTINCT s.seller_id) AS active_seller
              FROM sellers_data s
              JOIN order_items_data oi
                 ON s.seller_id=oi.seller_id
              JOIN orders_data o
                 ON oi.order_id=o.order_id
              GROUP BY DATE_FORMAT(o.order_purchase_timestamp,'%Y-%m');
              """)
pd.read_sql_query(query,conn)
```

Out[26]:

	month	active_seller
0	2016-09	3
1	2016-10	143
2	2016-12	1
3	2017-01	227
4	2017-02	427
5	2017-03	499
6	2017-04	506
7	2017-05	583
8	2017-06	539
9	2017-07	606
10	2017-08	708
11	2017-09	731
12	2017-10	776
13	2017-11	965
14	2017-12	861
15	2018-01	970
16	2018-02	947
17	2018-03	996
18	2018-04	1123
19	2018-05	1115
20	2018-06	1175
21	2018-07	1261
22	2018-08	1278
23	2018-09	1

In [29]:

```
# Active sellers changes over year
query=text("""SELECT DATE_FORMAT(o.order_purchase_timestamp,'%Y') AS year,
                     COUNT(DISTINCT s.seller_id) AS active_seller
              FROM sellers_data s
              JOIN order_items_data oi
                  ON s.seller_id=oi.seller_id
              JOIN orders_data o
                  ON oi.order_id=o.order_id
              GROUP BY DATE_FORMAT(o.order_purchase_timestamp,'%Y');
              """)
```

```
pd.read_sql_query(query,conn)
```

Out[29]:

	year	active_seller
0	2016	145
1	2017	1784
2	2018	2383

Question No 6

6. What is the distribution of seller ratings on Olist, and how does this impact sales performance?

Key Insights

1. Most sellers on Olist have high ratings.

   Sellers with higher ratings generate more sales, while sellers with low ratings contribute very little to total sales.

`This shows that good customer reviews are linked to better sales performance.`

In [17]:
```
# Distribution of seller rating and sales performance across different ratings
query=text("""WITH seller_sales AS (
    SELECT
        seller_id,
        SUM(price) AS total_sales
    FROM order_items_data
    GROUP BY seller_id
),
seller_ratings AS (
    SELECT
        oi.seller_id,
        ROUND(AVG(r.review_score), 2) AS avg_rating
    FROM order_items_data oi
    JOIN order_reviews_data r
        ON oi.order_id = r.order_id
    GROUP BY oi.seller_id
),
seller_metrics AS (
    SELECT
        s.seller_id,
        sr.avg_rating,
        ss.total_sales
    FROM seller_data_unique s
    LEFT JOIN seller_sales ss
```

```
        ON s.seller_id = ss.seller_id
    LEFT JOIN seller_ratings sr
        ON s.seller_id = sr.seller_id
),
rating_bucket AS (
    SELECT
        CASE
            WHEN avg_rating IS NULL THEN 'No Rating'
            WHEN avg_rating >= 1 AND avg_rating < 2 THEN '1–2 Poor'
            WHEN avg_rating >= 2 AND avg_rating < 3 THEN '2–3 Average'
            WHEN avg_rating >= 3 AND avg_rating < 4 THEN '3–4 Good'
            WHEN avg_rating >= 4 AND avg_rating <= 5 THEN '4–5 Excellent'
        END AS rating_group,
        COUNT(seller_id) AS seller_count,
        SUM(COALESCE(total_sales,0)) AS total_sales
    FROM seller_metrics
    GROUP BY rating_group
)
SELECT
    rating_group,
    seller_count,
    ROUND(
        seller_count * 100.0 / SUM(seller_count) OVER (),
        2
    ) AS seller_percentage,
    ROUND(total_sales, 2) AS total_sales
FROM rating_bucket
ORDER BY
    CASE rating_group
        WHEN 'No Rating' THEN 1
        WHEN '1–2 Poor' THEN 2
        WHEN '2–3 Average' THEN 3
        WHEN '3–4 Good' THEN 4
        WHEN '4–5 Excellent' THEN 5
    END;
""")
pd.read_sql_query(query,conn)
```

Out[17]:

	rating_group	seller_count	seller_percentage	total_sales
0	No Rating	5	0.16	7657.80
1	1–2 Poor	171	5.53	109609.63
2	2–3 Average	171	5.53	288539.21
3	3–4 Good	754	24.36	4250843.81
4	4–5 Excellent	1994	64.43	8934993.25

Question No 7

7. How many customers have made repeated purchases on Olist, and what percentage of total sales do they account for?

Key Insights

1. Around 2,800 customers made repeat purchases on Olist.
2. These repeat customers generated about 5.55% of total sales, `showing that most sales come from one-time customers.`

In [25]:
```
query = text("""
                WITH customer_orders AS (
                                SELECT
                                    c.customer_unique_id,
                                    COUNT(DISTINCT o.order_id) AS total_orders
                                FROM customers_data c
                                JOIN orders_data o
                                    ON c.customer_id = o.customer_id
                                WHERE o.order_status = 'delivered'
                                GROUP BY c.customer_unique_id
                ),

                    repeat_customers AS (
                                SELECT customer_unique_id
                                FROM customer_orders
                                WHERE total_orders > 1
                    ),

                    repeat_customer_sales AS (
                                SELECT
                                    SUM(order_total) AS repeat_sales
                                FROM (
                                    SELECT
                                        SUM(oi.price) + MAX(oi.freight_value)
                                    FROM repeat_customers rc
                                    JOIN customers_data c
                                        ON rc.customer_unique_id = c.customer_
                                    JOIN orders_data o
                                        ON c.customer_id = o.customer_id
                                    JOIN order_items_data oi
                                        ON o.order_id = oi.order_id
                                    WHERE o.order_status = 'delivered'
                                    GROUP BY oi.order_id
                                ) t
                    ),
```

```
                    total_sales AS (
                                SELECT SUM(order_total) AS total_sales
                                FROM (
                                    SELECT SUM(oi.price) + MAX(oi.freight_valu
                                    FROM order_items_data oi
                                    JOIN orders_data o
                                        ON oi.order_id = o.order_id
                                    WHERE o.order_status = 'delivered'
                                    GROUP BY oi.order_id
                                ) t
                    )

            SELECT
                (SELECT COUNT(*) FROM repeat_customers) AS repeat_customer_cou
                rcs.repeat_sales,
                ROUND((rcs.repeat_sales / ts.total_sales) * 100, 2) AS repeat_
            FROM repeat_customer_sales rcs
            CROSS JOIN total_sales ts;
""")
pd.read_sql_query(query, conn)
```

Out[25]:

	repeat_customer_count	repeat_sales	repeat_customer_sales_percentage
0	2801	842653.21	5.55

Question No 8

8. What is the average customer rating for products sold on Olist, and how does this impact sales performance?

Key Insights

1. Olist ratings are order-level, so all products in an order share the same rating, which can reduce product-level accuracy.
2. The overall average customer rating for products on Olist is `around 4.03`, which shows generally positive customer feedback.
3. Products with mid-to-high ratings (3–4.4) generate the highest average sales per product, while extremely high-rated products (4.5–5) tend to be lower-priced and contribute less per product despite having more items.

In [5]:

```
# Overall Average customer Rating for products
query=text("""SELECT
                ROUND(AVG(r.review_score), 2) AS overall_avg_product_rating
```

```
            FROM products_data p
            JOIN order_items_data oi
               ON p.product_id = oi.product_id
            JOIN order_reviews_data r
               ON oi.order_id = r.order_id;

""")
pd.read_sql_query(query,conn)
```

Out[5]:

	overall_avg_product_rating
0	4.03

In [7]:

```
# Impact Sales Performance
query=text("""WITH products_rating AS (
                                SELECT
                                    oi.product_id,
                                    AVG(r.review_score) AS avg_rating,
                                    SUM(oi.price) AS total_sales
                                FROM order_items_data oi
                                JOIN order_reviews_data r
                                    ON oi.order_id = r.order_id
                                GROUP BY oi.product_id
                            )

            SELECT
                CASE
                    WHEN avg_rating < 2 THEN '1.0–1.9 Very Poor'
                    WHEN avg_rating < 3 THEN '2.0–2.9 Poor'
                    WHEN avg_rating < 4 THEN '3.0–3.9 Average'
                    WHEN avg_rating < 4.5 THEN '4.0–4.4 Good'
                    ELSE '4.5–5.0 Excellent'
                END AS rating_bucket,
                COUNT(DISTINCT product_id) AS total_products,
                ROUND(AVG(total_sales), 2) AS avg_sales_per_product
            FROM products_rating
            GROUP BY rating_bucket
            ORDER BY MIN(avg_rating);
              """)
pd.read_sql_query(query,conn)
```

Out[7]:

	rating_bucket	total_products	avg_sales_per_product
0	1.0–1.9 Very Poor	2842	237.92
1	2.0–2.9 Poor	1579	425.91
2	3.0–3.9 Average	4917	638.37
3	4.0–4.4 Good	6544	653.58
4	4.5–5.0 Excellent	16907	281.68

Question No 9

What is the average order cancellation rate on Olist, and how does this impact seller performance?

Key Insights

1. The overall order cancellation rate on Olist is `around 0.63%`, which is very low.
2. Sellers with 0% or very low cancellation rates (0–1%) generate higher sales.
3. Sellers with higher cancellation rates (above 5%) generate much lower sales.

`This shows that lower cancellation rates are linked to better seller performance.`

In [24]:
```
# Overall cancellation rate in percentage
query=text("""SELECT ROUND(
                              (SUM(
                                  CASE WHEN order_status_clean='canceled' THEN 1
                                  )/COUNT(*))*100,2) AS cancellation_rate_percer
              FROM orders_data;""")
pd.read_sql_query(query,conn)
```

Out[24]:

	cancellation_rate_percent
0	0.63

In [17]:
```
# Impact cancellation rate on seller sales performance
query=text("""
              WITH order_level_data AS (
    SELECT
        o.order_id,
        oi.seller_id,
        o.order_status_clean,
        SUM(oi.price) AS order_item_total,
        MAX(oi.freight_value) AS order_freight_value

    FROM orders_data o
    JOIN order_items_data oi
        ON o.order_id = oi.order_id
    GROUP BY o.order_id, oi.seller_id, o.order_status_clean
),
```

```
seller_level_metrics AS (
    SELECT
        seller_id,
        ROUND(
            SUM(
                CASE
                    WHEN order_status_clean = 'canceled' THEN 1
                    ELSE 0
                END
            ) * 100.0 / COUNT(*),
        2) AS cancellation_rate,

        ROUND(
            SUM(
                CASE
                    WHEN order_status_clean <> 'canceled'
                    THEN (order_item_total + order_freight_value)
                    ELSE 0
                END
            ),
        2) AS total_sales

    FROM order_level_data
    GROUP BY seller_id
)

SELECT
    CASE
        WHEN cancellation_rate = 0 THEN '0%'
        WHEN cancellation_rate <= 1 THEN '0-1%'
        WHEN cancellation_rate <= 5 THEN '1-5%'
        ELSE '>5%'
    END AS cancellation_bucket,

    COUNT(DISTINCT seller_id) AS seller_count,
    ROUND(AVG(total_sales), 2) AS avg_sales_per_seller,
    ROUND(SUM(total_sales), 2) AS total_sales

FROM seller_level_metrics
GROUP BY cancellation_bucket

ORDER BY
    CASE cancellation_bucket
        WHEN '0%' THEN 1
        WHEN '0-1%' THEN 2
        WHEN '1-5%' THEN 3
        ELSE 4
    END;

          """)
pd.read_sql_query(query,conn)
```

Out[17]:

	cancellation_bucket	seller_count	avg_sales_per_seller	total_sales
0	0%	2766	3813.96	10549426.27
1	0-1%	62	57109.83	3540809.69
2	1-5%	100	11979.97	1197996.60
3	>5%	167	1337.36	223339.26

Question No 10

10. What are the top-selling products on Olist, and how have their sales volume changed over time?

Key Insights

1. Sales of top 20 products increase steadily over time, especially during 2017–2018, indicating growing customer adoption.
2. Some product categories show seasonal sales spikes, suggesting higher demand during specific months.

In [67]:

```
query=text("""WITH top_20_products AS (
                            SELECT oi.product_id
                            FROM order_items_data oi
                            GROUP BY oi.product_id
                            ORDER BY COUNT(*) DESC
                            LIMIT 20
                        )
                        SELECT
                            DATE_FORMAT(o.order_purchase_timestamp, '%Y-%m') A
                            pc.product_category_name_english_clean,
                            COUNT(*) AS monthly_quantity,
                            ROUND(SUM(oi.price), 2) AS monthly_sales
                        FROM top_20_products t
                        JOIN order_items_data oi
                            ON t.product_id = oi.product_id
                        JOIN orders_data o
                            ON oi.order_id = o.order_id
                        JOIN product_data_unique p
                            ON oi.product_id = p.product_id
                        JOIN products_categories_data pc
                        ON p.category_sk=pc.category_sk
                        GROUP BY month,pc.product_category_name_english_clean
                        ORDER BY month;

    """)
pd.read_sql_query(query,conn)
```

Out[67]:

	month	product_category_name_english_clean	monthly_quantity	monthly_
0	2017-01	Health_beauty	1	
1	2017-02	Bed_bath_table	2	1
2	2017-02	Computers_accessories	3	4
3	2017-02	Health_beauty	16	5
4	2017-02	Home_confort	1	
5	2017-03	Bed_bath_table	22	19
6	2017-03	Computers_accessories	3	4
7	2017-03	Health_beauty	24	13
8	2017-03	Home_confort	4	3
9	2017-03	Watches_gifts	5	12
10	2017-04	Bed_bath_table	34	30
11	2017-04	Computers_accessories	5	6
12	2017-04	Garden_tools	3	1
13	2017-04	Health_beauty	22	8
14	2017-04	Home_confort	4	3
15	2017-04	Watches_gifts	3	7
16	2017-05	Bed_bath_table	58	52
17	2017-05	Garden_tools	14	8
18	2017-05	Health_beauty	38	18
19	2017-05	Home_confort	12	10
20	2017-05	Housewares	14	8
21	2017-06	Bed_bath_table	51	45
22	2017-06	Computers_accessories	3	4
23	2017-06	Garden_tools	8	4
24	2017-06	Health_beauty	51	24
25	2017-06	Home_confort	6	5
26	2017-06	Housewares	73	43
27	2017-06	Watches_gifts	9	15
28	2017-07	Bed_bath_table	60	53
29	2017-07	Computers_accessories	6	8
30	2017-07	Furniture_decor	6	4

	month	product_category_name_english_clean	monthly_quantity	monthly_
31	2017-07	Garden_tools	65	38
32	2017-07	Health_beauty	61	29
33	2017-07	Home_confort	19	17
34	2017-07	Housewares	2	1
35	2017-07	Watches_gifts	21	37
36	2017-08	Bed_bath_table	37	33
37	2017-08	Computers_accessories	7	10
38	2017-08	Furniture_decor	21	12
39	2017-08	Garden_tools	110	65
40	2017-08	Health_beauty	74	55
41	2017-08	Home_confort	11	9
42	2017-08	Watches_gifts	28	47
43	2017-09	Bed_bath_table	23	20
44	2017-09	Computers_accessories	11	16
45	2017-09	Furniture_decor	12	4
46	2017-09	Garden_tools	90	53
47	2017-09	Health_beauty	81	75
48	2017-09	Home_confort	12	10
49	2017-09	Watches_gifts	38	60
50	2017-10	Bed_bath_table	14	12
51	2017-10	Computers_accessories	17	26
52	2017-10	Furniture_decor	30	13
53	2017-10	Garden_tools	125	74
54	2017-10	Health_beauty	72	45
55	2017-10	Home_confort	8	7
56	2017-10	Housewares	16	9
57	2017-10	Watches_gifts	5	10
58	2017-11	Bed_bath_table	40	35
59	2017-11	Computers_accessories	19	28
60	2017-11	Furniture_decor	72	43
61	2017-11	Garden_tools	329	175

	month	product_category_name_english_clean	monthly_quantity	monthly_
62	2017-11	Health_beauty	90	83
63	2017-11	Home_confort	21	18
64	2017-11	Housewares	40	23
65	2017-11	Not defined	13	15
66	2017-11	Watches_gifts	46	65
67	2017-12	Bed_bath_table	18	15
68	2017-12	Computers_accessories	35	47
69	2017-12	Furniture_decor	47	35
70	2017-12	Garden_tools	167	93
71	2017-12	Health_beauty	53	46
72	2017-12	Home_confort	5	4
73	2017-12	Housewares	15	8
74	2017-12	Not defined	32	39
75	2017-12	Watches_gifts	22	29
76	2018-01	Bed_bath_table	17	14
77	2018-01	Computers_accessories	96	131
78	2018-01	Furniture_decor	124	86
79	2018-01	Garden_tools	105	52
80	2018-01	Health_beauty	55	58
81	2018-01	Home_confort	5	4
82	2018-01	Housewares	9	5
83	2018-01	Not defined	42	51
84	2018-01	Watches_gifts	46	51
85	2018-02	Bed_bath_table	23	19
86	2018-02	Computers_accessories	171	205
87	2018-02	Furniture_decor	53	34
88	2018-02	Garden_tools	168	85
89	2018-02	Health_beauty	69	102
90	2018-02	Home_confort	7	6
91	2018-02	Housewares	10	5
92	2018-02	Not defined	43	52

	month	product_category_name_english_clean	monthly_quantity	monthly_
93	2018-02	Watches_gifts	46	55
94	2018-03	Bed_bath_table	17	14
95	2018-03	Computers_accessories	122	158
96	2018-03	Furniture_decor	82	51
97	2018-03	Garden_tools	117	63
98	2018-03	Health_beauty	62	112
99	2018-03	Home_confort	11	9
100	2018-03	Housewares	1	
101	2018-03	Not defined	48	59
102	2018-03	Watches_gifts	84	107
103	2018-04	Bed_bath_table	6	5
104	2018-04	Computers_accessories	104	130
105	2018-04	Furniture_decor	100	63
106	2018-04	Garden_tools	107	57
107	2018-04	Health_beauty	46	86
108	2018-04	Home_confort	8	6
109	2018-04	Housewares	3	1
110	2018-04	Not defined	15	18
111	2018-04	Watches_gifts	115	125
112	2018-05	Bed_bath_table	26	21
113	2018-05	Computers_accessories	88	116
114	2018-05	Furniture_decor	103	68
115	2018-05	Garden_tools	127	67
116	2018-05	Health_beauty	46	79
117	2018-05	Home_confort	13	11
118	2018-05	Watches_gifts	183	199
119	2018-06	Bed_bath_table	23	18
120	2018-06	Computers_accessories	64	79
121	2018-06	Furniture_decor	15	6
122	2018-06	Garden_tools	41	20
123	2018-06	Health_beauty	38	67

	month	product_category_name_english_clean	monthly_quantity	monthly_
124	2018-06	Home_confort	4	3
125	2018-06	Watches_gifts	52	60
126	2018-07	Bed_bath_table	10	7
127	2018-07	Computers_accessories	29	35
128	2018-07	Furniture_decor	12	5
129	2018-07	Garden_tools	32	15
130	2018-07	Health_beauty	28	52
131	2018-07	Home_confort	11	8
132	2018-07	Watches_gifts	21	24
133	2018-08	Bed_bath_table	7	5
134	2018-08	Computers_accessories	17	21
135	2018-08	Furniture_decor	19	8
136	2018-08	Garden_tools	29	14
137	2018-08	Health_beauty	40	100
138	2018-08	Home_confort	3	2
139	2018-08	Not defined	4	4
140	2018-08	Watches_gifts	19	24

Question No 11

11. Which payment methods are most commonly used by Olist customers, and how does this vary by product category or geographic region?

Key Insights

1. Credit card is the most commonly used payment method by Olist customers across all product categories.
2. Boleto is the second most popular payment method, especially for low to mid-priced products.
3. Voucher and debit card payments are used much less compared to credit cards and boleto.

4. Payment method usage is mostly consistent across regions, with credit cards dominating in every region.

In [44]:
```
# Most common payment type
query=text("""SELECT payment_type,COUNT(*) AS payment_counts
              FROM order_payments_data
              GROUP BY payment_type
              ORDER BY payment_counts DESC
              LIMIT 2;
              """)
pd.read_sql_query(query,conn)
```

Out[44]:

	payment_type	payment_counts
0	credit_card	76795
1	boleto	19784

In [6]:
```
# Most common payment type vary across product categories
query=text("""WITH top_payment_types AS (
                              SELECT
                                     payment_type_clean,
                                     COUNT(*) AS payment_counts
                              FROM order_payments_data
                              GROUP BY payment_type_clean
                              ORDER BY payment_counts DESC
                              LIMIT 2
              )
              SELECT p.product_category_name_clean,
                     pc.product_category_name_english_clean,
                     op.payment_type_clean,
                     COUNT(*) AS payment_counts
              FROM top_payment_types tpt
              JOIN order_payments_data op
                 ON tpt.payment_type_clean=op.payment_type_clean
              JOIN order_items_data oi
                 ON op.order_id=oi.order_id
              JOIN product_data_unique p
                 ON oi.product_id=p.product_id
              JOIN products_categories_data pc
                 ON p.category_sk=pc.category_sk
              GROUP BY p.product_category_name_clean,
                       pc.product_category_name_english_clean,
                       op.payment_type_clean
              ORDER BY op.payment_type_clean,
                       payment_counts DESC;
                       """)
pd.read_sql_query(query,conn)
```

Out[6]:

	product_category_name_clean	product_category_name_english
0	Informatica_acessorios	Computers_acce
1	Cama_mesa_banho	Bed_bat
2	Beleza_saude	Health_
3	Esporte_lazer	Sports_
4	Moveis_decoracao	Furniture
5	Utilidades_domesticas	Hous
6	Ferramentas_jardim	Garde
7	Relogios_presentes	Watche
8	Telefonia	Tel
9	Automotivo	
10	Brinquedos	
11	Cool_stuff	Co
12	Eletronicos	Elec
13	Perfumaria	Per
14	Bebes	
15	Moveis_escritorio	Office_fu
16	Papelaria	Sta
17	Fashion_bolsas_e_acessorios	Fashion_bags_acce
18	Pet_shop	Pe
19	Not defined	Not
20	Consoles_games	Consoles_
21	Construcao_ferramentas_construcao	Construction_tools_const
22	Malas_acessorios	Luggage_acce
23	Instrumentos_musicais	Musical_instru
24	Eletrodomesticos	Home_app
25	Livros_interesse_geral	Books_general_i
26	Eletroportateis	Small_app
27	Casa_construcao	Home_const
28	Alimentos	
29	Moveis_sala	Furniture_livin
30	Audio	

	product_category_name_clean	product_category_name_english
31	Market_place	Marke
32	Alimentos_bebidas	Foo
33	Casa_conforto	Home_
34	Construcao_ferramentas_iluminacao	Construction_tools
35	Telefonia_fixa	Fixed_tel
36	Climatizacao	Air_condi
37	Livros_tecnicos	Books_te
38	Bebidas	
39	Sinalizacao_e_seguranca	Signaling_and_s
40	Agro_industria_e_comercio	Agro_industry_and_com
41	Industria_comercio_e_negocios	Industry_commerce_and_bu
42	Moveis_cozinha_area_de_servico_jantar_e_jardim	Kitchen_dining_laundry_garden_fu
43	Artes	
44	Construcao_ferramentas_jardim	Costruction_tools_
45	Eletrodomesticos_2	Home_applia
46	Construcao_ferramentas_seguranca	Construction_tools_
47	Fashion_calcados	Fashion
48	Pcs	Com
49	Artigos_de_natal	Christmas_s
50	Moveis_quarto	Furniture_be
51	Fashion_underwear_e_moda_praia	Fashion_underwear_
52	Fashion_roupa_masculina	Fashion_male_c
53	Cine_foto	Cine
54	Construcao_ferramentas_ferramentas	Costruction_tool
55	Tablets_impressao_imagem	Tablets_printing_
56	Dvds_blu_ray	Dvds_
57	Livros_importados	Books_im
58	Fashion_roupa_feminina	Fashio_female_c
59	Fashion_esporte	Fashio
60	Artigos_de_festas	Party_s
61	Musica	

	product_category_name_clean	product_category_name_english
62	Portateis_casa_forno_e_cafe	Small_appliances_home_oven_and_
63	Fraldas_higiene	Diapers_and_h
64	Flores	F
65	Artes_e_artesanato	Arts_and_craftm
66	Casa_conforto_2	Home_cor
67	Moveis_colchao_e_estofado	Furniture_mattress_and_uph
68	Portateis_cozinha_e_preparadores_de_alimentos	Not c
69	Cds_dvds_musicais	Cds_dvds_m
70	Fashion_roupa_infanto_juvenil	Fashion_childrens_
71	Pc_gamer	Not c
72	La_cuisine	La_
73	Seguros_e_servicos	Security_and_s
74	Cama_mesa_banho	Bed_batl
75	Beleza_saude	Health_
76	Esporte_lazer	Sports_
77	Moveis_decoracao	Furniture
78	Informatica_acessorios	Computers_acce
79	Utilidades_domesticas	Hous
80	Relogios_presentes	Watche
81	Telefonia	Tele
82	Brinquedos	
83	Automotivo	
84	Ferramentas_jardim	Garde
85	Cool_stuff	Co
86	Perfumaria	Per
87	Bebes	
88	Eletronicos	Elec
89	Papelaria	Sta
90	Fashion_bolsas_e_acessorios	Fashion_bags_acce
91	Pet_shop	Pe
92	Not defined	Not c

	product_category_name_clean	product_category_name_english
93	Moveis_escritorio	Office_fu
94	Malas_acessorios	Luggage_acce
95	Consoles_games	Consoles_
96	Construcao_ferramentas_construcao	Construction_tools_const
97	Eletrodomesticos	Home_app
98	Eletroportateis	Small_app
99	Instrumentos_musicais	Musical_instru
100	Casa_construcao	Home_const
101	Livros_interesse_geral	Books_general_i
102	Moveis_sala	Furniture_livin
103	Alimentos	
104	Casa_conforto	Home_
105	Bebidas	
106	Audio	
107	Climatizacao	Air_condi
108	Market_place	Marke
109	Moveis_cozinha_area_de_servico_jantar_e_jardim	Kitchen_dining_laundry_garden_fu
110	Fashion_calcados	Fashion
111	Construcao_ferramentas_iluminacao	Construction_tools
112	Industria_comercio_e_negocios	Industry_commerce_and_bu
113	Livros_tecnicos	Books_te
114	Alimentos_bebidas	Food
115	Construcao_ferramentas_jardim	Costruction_tools_
116	Telefonia_fixa	Fixed_tele
117	Eletrodomesticos_2	Home_applia
118	Pcs	Com
119	Artes	
120	Agro_industria_e_comercio	Agro_industry_and_com
121	Construcao_ferramentas_seguranca	Construction_tools_
122	Sinalizacao_e_seguranca	Signaling_and_s
123	Artigos_de_natal	Christmas_s

	product_category_name_clean	product_category_name_english
124	Fashion_roupa_masculina	Fashion_male_c
125	Fashion_underwear_e_moda_praia	Fashion_underwear
126	Construcao_ferramentas_ferramentas	Costruction_tool
127	Moveis_quarto	Furniture_be
128	Tablets_impressao_imagem	Tablets_printing
129	Portateis_casa_forno_e_cafe	Small_appliances_home_oven_and
130	Dvds_blu_ray	Dvds_
131	Livros_importados	Books_im
132	Cine_foto	Cine
133	Fashion_roupa_feminina	Fashio_female_c
134	Moveis_colchao_e_estofado	Furniture_mattress_and_uph
135	Fraldas_higiene	Diapers_and_h
136	Artigos_de_festas	Party_s
137	Musica	
138	Flores	F
139	Casa_conforto_2	Home_cor
140	Fashion_esporte	Fashio
141	Artes_e_artesanato	Arts_and_craftm
142	La_cuisine	La_
143	Cds_dvds_musicais	Cds_dvds_m
144	Portateis_cozinha_e_preparadores_de_alimentos	Not c
145	Pc_gamer	Not c
146	Fashion_roupa_infanto_juvenil	Fashion_childrens_
147	Seguros_e_servicos	Security_and_s

In [7]:
```
# Most common payment type vary across geographic region
query=text("""WITH top_payment_types AS (
                                SELECT
                                       payment_type_clean,
                                       COUNT(*) AS payment_counts
                                FROM order_payments_data
                                GROUP BY payment_type_clean
                                ORDER BY payment_counts DESC
                                LIMIT 2
                )
```

```
                SELECT g.geolocation_zip_code_prefix,
                       g.geolocation_city_clean,
                       g.geolocation_state_full,
                       op.payment_type,
                       COUNT(*) AS payment_counts
                FROM top_payment_types tpt
                JOIN order_payments_data op
                   ON tpt.payment_type_clean=op.payment_type_clean
                JOIN orders_data o
                   ON op.order_id=o.order_id
                JOIN customers_data c
                   ON o.customer_id=c.customer_id
                JOIN customer_geolocation_unique_bridge cgub
                   ON c.customer_id=cgub.customer_id
                JOIN geolocation_data_unique g
                   ON cgub.geolocation_sk=g.geolocation_sk
                GROUP BY g.geolocation_zip_code_prefix,
                         g.geolocation_city_clean,
                         g.geolocation_state_full,
                         op.payment_type_clean
                ORDER BY op.payment_type_clean,
                         payment_counts DESC
                LIMIT 100 OFFSET 0;
                         """)
pd.read_sql_query(query,conn)
```

Out[7]:

	geolocation_zip_code_prefix	geolocation_city_clean	geolocation_state_full	p
0	36570	Vicosa	Minas Gerais	
1	22790	Rio de janeiro	Rio de Janeiro	
2	11065	Santos	Sao Paulo	
3	22793	Rio de janeiro	Rio de Janeiro	
4	22631	Rio de janeiro	Rio de Janeiro	
5	24220	Niteroi	Rio de Janeiro	
6	37200	Lavras	Minas Gerais	
7	11740	Itanhaem	Sao Paulo	
8	20550	Rio de janeiro	Rio de Janeiro	
9	11680	Ubatuba	Sao Paulo	
10	13214	Jundiai	Sao Paulo	
11	35162	Ipatinga	Minas Gerais	
12	38400	Uberlandia	Minas Gerais	
13	13060	Campinas	Sao Paulo	
14	13040	Campinas	Sao Paulo	
15	35500	Divinopolis	Minas Gerais	
16	13050	Campinas	Sao Paulo	
17	7600	Mairipora	Sao Paulo	
18	22775	Rio de janeiro	Rio de Janeiro	
19	13820	Jaguariuna	Sao Paulo	
20	13070	Campinas	Sao Paulo	
21	29101	Vila velha	Espirito Santo	
22	13212	Jundiai	Sao Paulo	
23	24230	Niteroi	Rio de Janeiro	
24	80030	Curitiba	Parana	
25	9371	Maua	Sao Paulo	
26	37701	Pocos de caldas	Minas Gerais	
27	12243	Sao jose dos campos	Sao Paulo	
28	24210	Niteroi	Rio de Janeiro	
29	88330	Balneario camboriu	Santa Catarina	
30	18550	Boituva	Sao Paulo	

	geolocation_zip_code_prefix	geolocation_city_clean	geolocation_state_full	p
31	11070	Santos	Sao Paulo	
32	29102	Vila velha	Espirito Santo	
33	11025	Santos	Sao Paulo	
34	51020	Recife	Pernambuco	
35	13480	Limeira	Sao Paulo	
36	30140	Belo horizonte	Minas Gerais	
37	13560	Sao carlos	Sao Paulo	
38	97015	Santa maria	Rio Grande do Sul	
39	20520	Rio de janeiro	Rio de Janeiro	
40	13175	Sumare	Sao Paulo	
41	39400	Montes claros	Minas Gerais	
42	11015	Santos	Sao Paulo	
43	35400	Ouro preto	Minas Gerais	
44	28970	Araruama	Rio de Janeiro	
45	90010	Porto alegre	Rio Grande do Sul	
46	13087	Campinas	Sao Paulo	
47	11702	Praia grande	Sao Paulo	
48	21321	Rio de janeiro	Rio de Janeiro	
49	13348	Indaiatuba	Sao Paulo	
50	14020	Ribeirao preto	Sao Paulo	
51	36900	Manhuacu	Minas Gerais	
52	11900	Registro	Sao Paulo	
53	47850	Luis eduardo magalhaes	Bahia	
54	11730	Mongagua	Sao Paulo	
55	35300	Caratinga	Minas Gerais	
56	13085	Campinas	Sao Paulo	
57	37550	Pouso alegre	Minas Gerais	
58	18900	Santa cruz do rio pardo	Sao Paulo	
59	98700	Ijui	Rio Grande do Sul	
60	13084	Campinas	Sao Paulo	
61	22451	Rio de janeiro	Rio de Janeiro	

	geolocation_zip_code_prefix	geolocation_city_clean	geolocation_state_full	p
62	13920	Pedreira	Sao Paulo	
63	35900	Itabira	Minas Gerais	
64	11030	Santos	Sao Paulo	
65	21330	Rio de janeiro	Rio de Janeiro	
66	12233	Sao jose dos campos	Sao Paulo	
67	36400	Conselheiro lafaiete	Minas Gerais	
68	37640	Extrema	Minas Gerais	
69	24020	Niteroi	Rio de Janeiro	
70	9725	Sao bernardo do campo	Sao Paulo	
71	30130	Belo horizonte	Minas Gerais	
72	13484	Limeira	Sao Paulo	
73	11035	Santos	Sao Paulo	
74	20230	Rio de janeiro	Rio de Janeiro	
75	30190	Belo horizonte	Minas Gerais	
76	23970	Parati	Rio de Janeiro	
77	13347	Indaiatuba	Sao Paulo	
78	20040	Rio de janeiro	Rio de Janeiro	
79	89900	Sao miguel do oeste	Santa Catarina	
80	99700	Erechim	Rio Grande do Sul	
81	8290	Sao paulo	Sao Paulo	
82	11310	Sao vicente	Sao Paulo	
83	13211	Jundiai	Sao Paulo	
84	13140	Paulinia	Sao Paulo	
85	11045	Santos	Sao Paulo	
86	30160	Belo horizonte	Minas Gerais	
87	37540	Santa rita do sapucai	Minas Gerais	
88	33400	Lagoa santa	Minas Gerais	
89	35164	Ipatinga	Minas Gerais	
90	18530	Tiete	Sao Paulo	
91	22250	Rio de janeiro	Rio de Janeiro	
92	35700	Sete lagoas	Minas Gerais	

	geolocation_zip_code_prefix	geolocation_city_clean	geolocation_state_full	p
93	11060	Santos	Sao Paulo	
94	11250	Bertioga	Sao Paulo	
95	13083	Campinas	Sao Paulo	
96	99010	Passo fundo	Rio Grande do Sul	
97	89801	Chapeco	Santa Catarina	
98	93700	Campo bom	Rio Grande do Sul	
99	96010	Pelotas	Rio Grande do Sul	

Question No 12

12. Which product categories have the highest profit margins on Olist, and how can the company increase profitability across different categories?

Key Insights

1. Small and lightweight product categories have the highest profit margins on Olist because their freight cost is low compared to product price.
2. Categories such as technology accessories, fashion items, and small home products perform better in terms of profit margin than bulky goods.
3. Heavy and bulky categories like furniture, construction materials, and large appliances have lower profit margins due to high shipping costs.

`Overall, freight cost is the key factor affecting profit margins across all product categories.`

How Olist Can Increase Profitability

- Focus marketing and promotions on high-margin, lightweight categories.
- Reduce delivery costs for heavy products using nearby sellers or optimized logistics.

In [16]:
```
query=text("""
CREATE TABLE payment_per_order AS
                    SELECT
                        order_id,
                        SUM(payment_value) AS total_payment
                    FROM order_payments_data
                    GROUP BY order_id;
 """)
conn.execute(query)
conn.commit()
```

In [17]:
```
query=text("""
CREATE INDEX payment_per_order_order_id_idx ON payment_per_order(order_id);
 """)
conn.execute(query)
conn.commit()
```

In [18]:
```
query=text("""
CREATE TABLE order_item_total AS
                SELECT
                     order_id,
                     SUM(price + freight_value) AS order_total_cost
                FROM order_items_data
                GROUP BY order_id;
 """)
conn.execute(query)
conn.commit()
```

In [19]:
```
query=text("""
CREATE INDEX order_item_total_order_id_idx ON order_item_total(order_id);
 """)
conn.execute(query)
conn.commit()
```

In [22]:
```
query=text("""
CREATE TABLE item_allocated AS (
                    SELECT
                        oi.order_id,
                        oi.product_id,
                        oi.price,
                        oi.freight_value,
                        p.category_sk,
                        ((oi.price + oi.freight_value) / oit.order_total_cost
                    FROM order_items_data oi
                    JOIN order_item_total oit
                        ON oi.order_id = oit.order_id
                    JOIN payment_per_order ppo
                        ON oi.order_id = ppo.order_id
                    JOIN product_data_unique p
                        ON oi.product_id = p.product_id);
 """)
conn.execute(query)
```

```
conn.commit()
```

In [5]:

```
query=text("""
SELECT
    pc.product_category_name_english_clean AS category,
    ROUND(SUM(item_revenue),2) AS total_item_revenue,
    ROUND(SUM(price + freight_value),2) AS cost,
    ROUND((SUM(item_revenue) - SUM(price + freight_value)),2) as proxy_profit_
    ROUND(
        ((SUM(item_revenue) - SUM(price + freight_value))
        / SUM(item_revenue)) * 100,
        2
    ) AS proxy_profit_margin_percent
FROM item_allocated ia
JOIN products_categories_data pc
    ON ia.category_sk = pc.category_sk
GROUP BY pc.product_category_name_english_clean
ORDER BY proxy_profit_margin_percent DESC;
 """)
pd.read_sql_query(query,conn)
```

Out[5]:

	category	revenue	cost	proxy_profit_m
0	Drinks	28202.46	28169.95	
1	Tablets_printing_image	8764.92	8754.61	
2	Construction_tools_construction	165516.23	165328.00	1
3	Books_general_interest	56109.50	56052.40	
4	Construction_tools_lights	48709.21	48663.80	
5	Small_appliances_home_oven_and_coffee	50216.42	50193.57	
6	Home_confort	67109.11	67073.27	
7	Pet_shop	254001.97	253876.65	1
8	Food	36681.85	36664.44	
9	Telephony	395038.46	394883.32	1
10	Air_conditioning	61801.17	61774.19	
11	Electronics	206913.28	206825.06	
12	Furniture_living_room	86919.56	86884.73	
13	Bed_bath_table	1242220.09	1241681.72	5
14	Perfumery	453467.62	453338.71	1
15	Toys	561503.42	561372.55	1
16	Cool_stuff	719459.32	719329.95	1
17	Luggage_accessories	170911.27	170875.21	
18	Not defined	213695.09	213662.90	
19	Dvds_blu_ray	7289.36	7288.13	
20	Small_appliances	206711.49	206668.83	
21	Consoles_games	177333.88	177293.24	
22	Furniture_decor	902729.94	902511.79	2
23	Stationery	277807.92	277741.71	
24	Office_furniture	342559.51	342532.65	
25	Housewares	778486.23	778397.77	
26	Fixed_telephony	64227.86	64220.81	
27	Market_place	33838.49	33834.53	
28	Books_technical	23381.37	23379.12	
29	Health_beauty	1441197.90	1441104.61	
30	Garden_tools	584282.97	584219.21	

	category	revenue	cost	proxy_profit_m
31	Computers_accessories	1059395.10	1059272.40	1
32	Watches_gifts	1305637.22	1305541.61	
33	Home_appliances	95002.11	94990.43	
34	Baby	480164.33	480118.00	
35	Sports_leisure	1156757.08	1156656.48	1
36	Agro_industry_and_commerce	78379.19	78374.07	
37	Furniture_mattress_and_upholstery	5998.54	5998.54	
38	Art	28247.81	28247.81	
39	Christmas_supplies	12030.10	12030.12	
40	Home_comfort_2	1170.58	1170.58	
41	Security_and_services	324.51	324.51	
42	Furniture_bedroom	24661.01	24661.01	
43	Fashio_female_clothing	3425.39	3425.39	
44	Home_construction	96923.01	96920.36	
45	Home_appliances_2	123917.92	123917.92	
46	Cds_dvds_musicals	954.99	954.99	
47	Fashion_bags_accessories	184264.83	184273.54	
48	Musical_instruments	210147.09	210137.37	
49	Construction_tools_safety	44463.62	44463.62	
50	Books_imported	5409.70	5409.70	
51	Auto	685400.43	685384.32	
52	Food_drink	19687.46	19687.47	
53	Kitchen_dining_laundry_garden_furniture	58327.80	58327.80	
54	Party_supplies	5313.15	5313.15	
55	Fashion_underwear_beach	11457.74	11457.74	
56	Flowers	1598.90	1598.91	
57	Costruction_tools_garden	31027.29	31027.29	
58	Fashion_sport	2697.64	2697.64	
59	Arts_and_craftmanship	2184.14	2184.14	
60	Audio	56398.94	56398.94	
61	Fashion_male_clothing	12950.22	12950.23	

	category	revenue	cost	proxy_profit_m
62	Costruction_tools_tools	17934.17	17934.17	
63	Industry_commerce_and_business	47554.30	47554.29	
64	Signaling_and_security	28017.04	28017.05	
65	La_cuisine	2388.54	2388.54	
66	Diapers_and_hygiene	2141.27	2141.27	
67	Fashion_shoes	28481.64	28481.64	
68	Computers	232799.41	232799.43	
69	Music	6724.86	6724.86	
70	Fashion_childrens_clothes	665.36	665.36	
71	Cine_photo	8167.85	8189.66	-

Question No 13

13. Which geolocations (cities, and states) have the highest number of active customers on Olist?

Key Insights

1. Most customers are in big cities: Sao Paulo, Rio de Janeiro, Belo Horizonte.
2. These cities are the key markets for sales and marketing.
3. Smaller cities have fewer customers, but they are good opportunities to grow.
4. Focus on top cities first, then expand to others gradually.

In [4]:
```
query=text("""SELECT
                    g.geolocation_city_clean,
                    g.geolocation_state_full,
                    COUNT(c.customer_unique_id) as customers_count
               FROM geolocation_data_unique g
               JOIN customer_geolocation_unique_bridge cgb
                  ON g.geolocation_sk=cgb.geolocation_sk
               JOIN customers_data c
                  ON cgb.customer_id=c.customer_id
               JOIN orders_data o
                  ON c.customer_id=o.customer_id
```

```
                GROUP BY
                        g.geolocation_city_clean,
                        g.geolocation_state_full
                ORDER BY customers_count DESC
                LIMIT 100 OFFSET 0;""")
pd.read_sql_query(query,conn)
```

Out[4]:

	geolocation_city_clean	geolocation_state_full	customers_count
0	Sao paulo	Sao Paulo	15545
1	Rio de janeiro	Rio de Janeiro	6899
2	Belo horizonte	Minas Gerais	2767
3	Brasilia	Distrito Federal	1924
4	Curitiba	Parana	1507
5	Campinas	Sao Paulo	1444
6	Porto alegre	Rio Grande do Sul	1379
7	Salvador	Bahia	1241
8	Guarulhos	Sao Paulo	1189
9	Sao bernardo do campo	Sao Paulo	938
10	Niteroi	Rio de Janeiro	849
11	Santo andre	Sao Paulo	796
12	Osasco	Sao Paulo	746
13	Santos	Sao Paulo	713
14	Goiania	Goias	692
15	Sao jose dos campos	Sao Paulo	691
16	Fortaleza	Ceara	654
17	Sorocaba	Sao Paulo	633
18	Recife	Pernambuco	613
19	Florianopolis	Santa Catarina	570
20	Jundiai	Sao Paulo	565
21	Ribeirao preto	Sao Paulo	528
22	Belem	Para	443
23	Nova iguacu	Rio de Janeiro	442
24	Barueri	Sao Paulo	433
25	Juiz de fora	Minas Gerais	427
26	Contagem	Minas Gerais	426
27	Sao goncalo	Rio de Janeiro	409
28	Mogi das cruzes	Sao Paulo	383
29	Vitoria	Espirito Santo	379
30	Uberlandia	Minas Gerais	374

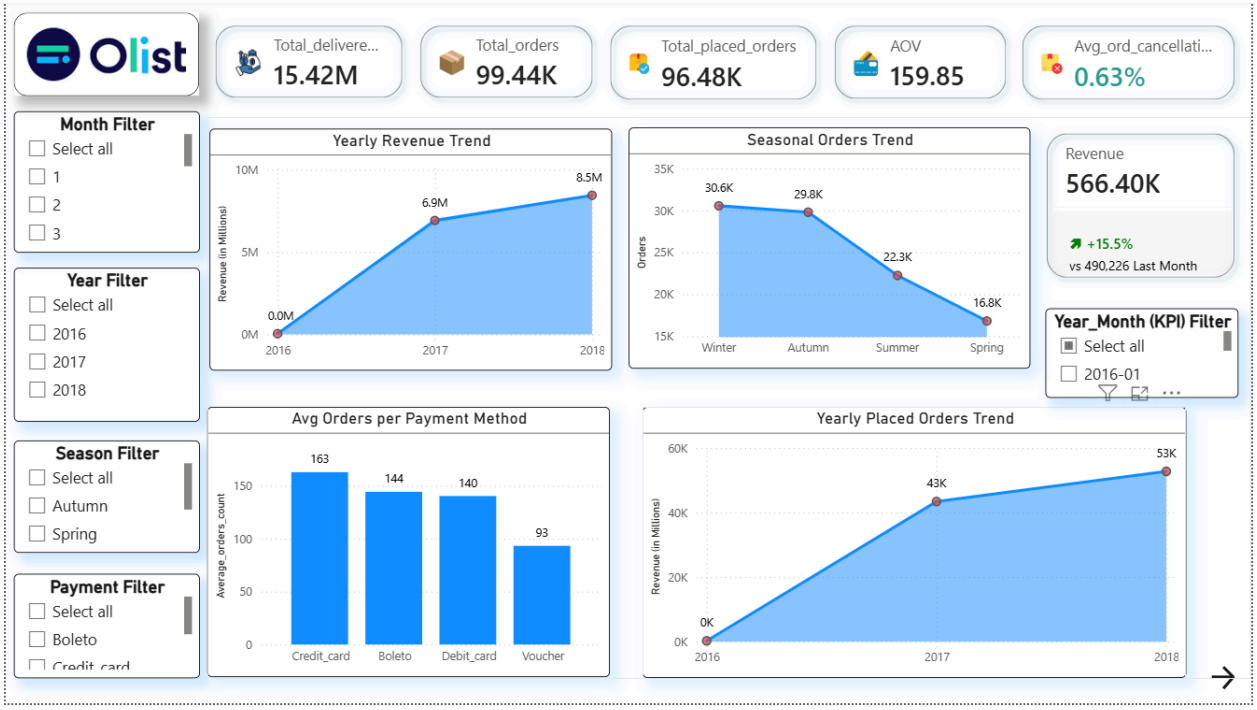
	geolocation_city_clean	geolocation_state_full	customers_count
31	Piracicaba	Sao Paulo	369
32	Sao luis	Maranhao	353
33	Vila velha	Espirito Santo	336
34	Sao jose do rio preto	Sao Paulo	335
35	Carapicuiba	Sao Paulo	328
36	Maua	Sao Paulo	323
37	Campo grande	Mato Grosso do Sul	320
38	Praia grande	Sao Paulo	311
39	Londrina	Parana	306
40	Taboao da serra	Sao Paulo	296
41	Diadema	Sao Paulo	286
42	Serra	Espirito Santo	285
43	Indaiatuba	Sao Paulo	280
44	Teresina	Piaui	279
45	Not defined	Unknown	278
46	Sao caetano do sul	Sao Paulo	277
47	Bauru	Sao Paulo	273
48	Maringa	Parana	271
49	Taubate	Sao Paulo	270
50	Duque de caxias	Rio de Janeiro	266
51	Joinville	Santa Catarina	264
52	Joao pessoa	Paraiba	256
53	Cotia	Sao Paulo	250
54	Cuiaba	Mato Grosso	248
55	Maceio	Alagoas	247
56	Petropolis	Rio de Janeiro	240
57	Campos dos goytacazes	Rio de Janeiro	239
58	Macae	Rio de Janeiro	239
59	Sao carlos	Sao Paulo	236
60	Suzano	Sao Paulo	234
61	Volta redonda	Rio de Janeiro	232

	geolocation_city_clean	geolocation_state_full	customers_count
62	Guaruja	Sao Paulo	226
63	Aracaju	Sergipe	225
64	Caxias do sul	Rio Grande do Sul	224
65	Americana	Sao Paulo	224
66	Montes claros	Minas Gerais	211
67	Canoas	Rio Grande do Sul	210
68	Natal	Rio Grande do Norte	207
69	Betim	Minas Gerais	203
70	Marilia	Sao Paulo	196
71	Santa maria	Rio Grande do Sul	194
72	Jacarei	Sao Paulo	193
73	Uberaba	Minas Gerais	191
74	Limeira	Sao Paulo	190
75	Blumenau	Santa Catarina	188
76	Pelotas	Rio Grande do Sul	186
77	Feira de santana	Bahia	185
78	Santana de parnaiba	Sao Paulo	185
79	Valinhos	Sao Paulo	183
80	Sumare	Sao Paulo	183
81	Itaquaquecetuba	Sao Paulo	172
82	Ipatinga	Minas Gerais	171
83	Itapevi	Sao Paulo	170
84	Sao jose	Santa Catarina	169
85	Sao vicente	Sao Paulo	163
86	Franca	Sao Paulo	161
87	Presidente prudente	Sao Paulo	161
88	Atibaia	Sao Paulo	159
89	Embu das artes	Sao Paulo	158
90	Rio claro	Sao Paulo	155
91	Araraquara	Sao Paulo	155
92	Nova friburgo	Rio de Janeiro	151

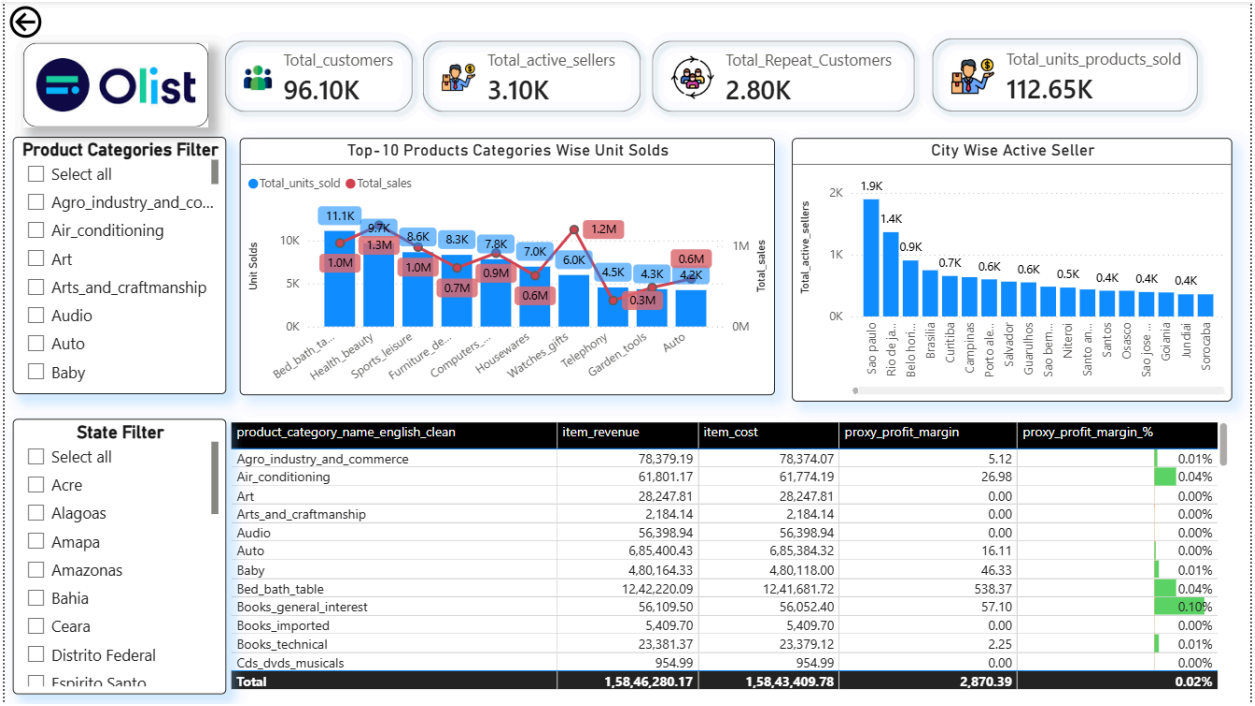
	geolocation_city_clean	geolocation_state_full	customers_count
93	Cascavel	Parana	148
94	Braganca paulista	Sao Paulo	146
95	Hortolandia	Sao Paulo	145
96	Ponta grossa	Parana	143
97	Governador valadares	Minas Gerais	141
98	Aracatuba	Sao Paulo	141
99	Manaus	Amazonas	140

Dashboard

In [8]:
```
from IPython.display import Image,display
display(Image(filename=r"D:/Eda project on Olist dataset by Sql/Dashboard_imag
```



In [9]:
```
display(Image(filename=r"D:/Eda project on Olist dataset by Sql/Dashboard_imag
```



product_category_name_english_clean	item_revenue	item_cost	proxy_profit_margin	proxy_profit_margin_%
Agro_industry_and_commerce	78,379.19	78,374.07	5.12	0.01%
Air_conditioning	61,801.17	61,774.19	26.98	0.04%
Art	28,247.81	28,247.81	0.00	0.00%
Arts_and_craftmanship	2,184.14	2,184.14	0.00	0.00%
Audio	56,398.94	56,398.94	0.00	0.00%
Auto	6,85,400.43	6,85,384.32	16.11	0.00%
Baby	4,80,164.33	4,80,118.00	46.33	0.01%
Bed_bath_table	12,42,220.09	12,41,681.72	538.37	0.04%
Books_general_interest	56,109.50	56,052.40	57.10	0.10%
Books_imported	5,409.70	5,409.70	0.00	0.00%
Books_technical	23,381.37	23,379.12	2.25	0.01%
Cds_dvds_musicals	954.99	954.99	0.00	0.00%
Total	**1,58,46,280.17**	**1,58,43,409.78**	**2,870.39**	**0.02%**

In [4]:
```
conn.close()
engine.dispose()
```