Group 27 - Data Management (IB9HP0) Assignment

2120036, 2161200, 2215085, 2216081, 2227324, 2233220

Contents

Section 1 - Objective

This work illustrate the complete understanding of extraction, refining, and delivering insights from the Olist dataset. We begin by understanding the entities, attributes, relationships, and cardinality of the provided dataset, which leads us to design the ER diagram and provide DDL statements for the tables. The later half identifies the issues with records and normalise the schema, illustrated with an updated ER diagram. Finally, we provide complex SQL queries, their dplyr equivalents, and visualisations to answer critical business questions.

Section 2 - Dataset Overview

This section identifies entities, attributes, relationships, and cardinality. Additionally, a detailed discussion on keys is provided.

Section 2.1 Entities and Attributes

Our study starts by writing the tables to the Olist database created in the R environment using the RSQLite package (Appendix, section 1.1). The uploaded tables are analysed, and the below table (Table 1) details various entities and attributes from the database.

Table 1 - List of Entites and Attributes in the Olist dataset

Sr. No.	Entity	Attribute
1	Customers	customer_id, customer_unique_id, customer_zip_code_prefix, customer_city, customer_state
2	geolocation	geolocation_zip_code_prefix, geolocation_lat, geolocation_lng, geolocation_city, geolocation_state
3	order_items	order_id, product_id, seller_id, shipping_limit_date, price, freight_value
4	order_payments	order_id, payment_sequential, payment_type, payment_installments, payment_value
5	order_reviews	review_id, order_id, review_score, review_comment_title, review_comment_message, review_creation_date, review_answer_timestamp
6	orders	order_id, customer_id, order_status, order_purchase_timestamp, order_approved_at, order_delivered_carrier_date, order_delivered_customer_date, order_estimated_delivery_date
7	products	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
8	sellers	seller_id, seller_zip_code_prefix, seller_city, seller_state
9	products_category_name_translation	product_category_name, product_category_name_english
10	closed_deals	mql_id, seller_id, sdr_id, sr_id, won_date, business_segment, lead_type, lead_behaviour_profile, has_company, has_gtin, average_stock, business_type, declared_product_catalog_size, declared_monthly_revenue
11	marketing_qualified_leads	mql_id, first_contact_date, landing_page_id , origin

Section 2.2 Relationships

Customers place orders and provide *Order Reviews*. *Geolocation* calculates the distance between *Customers* and *Sellers*. Each *Order* has *Order Reviews*, contains several *Order Items* and is paid by different *Payment* methods. *Order Items* is the sequential number of items in each Order related to *Products*. *Sellers* sell *Products* to *Customers* for each Order, and every product has its *Product Category Name*. To aid the *Product's Name Translation* to English, a look-up table is provided that can be linked to the *Products* table. Lead becomes a seller with *Closed Deals* after a qualified lead *(MQL)* fills in a form at a landing page.

Section 2.3 Cardinality

To determine cardinality, we analysed a few records from each of the tables in the dataset. For instance, the CUSTOMERS and ORDERS table has 1:1 cardinality; identified by querying specific CUSTOMER_ID from the CUSTOMERS table and checking the number of records for this CUSTOMER_ID in the ORDERS table. Instead of checking manually, we have utilised SQL JOINS and DISTINCT keywords to provide an efficient solution (Appendix, section 1.2). The below table (Table 2) displays the cardinality among all the related tables in the dataset.

Table 2 - Cardinality in Olist dataset

	Cardinality	
CUSTOMERS	1:1	ORDERS
ORDER_ITEMS	M:N	ORDER_PAYMENTS
ORDER_ITEMS	M:N	ORDER_REVIEWS
ORDER_ITEMS	N:1	ORDERS
ORDER_ITEMS	N:1	PRODUCTS
ORDER_ITEMS	N:1	SELLERS
ORDER_ITEMS	N:1	CLOSED_DEALS
ORDER_PAYMENTS	M:N	ORDER_ITEMS
ORDER_PAYMENTS	N:1	ORDERS
ORDER_PAYMENTS	M:N	ORDER_REVIEWS
ORDER_REVIEWS	M:N	ORDER_ITEMS
ORDER_REVIEWS	M:N	ORDER_PAYMENTS
ORDER_REVIEWS	N:1	ORDERS
ORDERS	1:N	ORDER_ITEMS
ORDERS	1:N	ORDER_PAYMENTS
ORDERS	1:N	ORDER_REVIEWS
PRODUCTS	1:N	ORDER_ITEMS
PRODUCTS	N:1	PRODUCTS_CATEGORY_NAME_TRANSLATION
SELLERS	1:N	ORDER_ITEMS
SELLERS	1:1	CLOSED_DEALS
PRODUCTS_CATEGORY_NAME_TRANSLATION	1:N	PRODUCTS
CLOSED_DEALS	1:N	ORDER_ITEMS
CLOSED_DEALS	1:1	SELLERS
CLOSED_DEALS	1:1	MARKETING_QUALIFIED_LEADS
MARKETING_QUALIFIED_LEADS	1:1	CLOSED_DEALS
GEOLOCATION	M:N	CUSTOMERS
GEOLOCATION	M:N	SELLERS

Section 2.4 Keys

In the provided dataset, three sets of keys are identified, i.e., primary key, composite key, and foreign key; which are described below (Appendix, Section 1.3):

- Primary keys are identified with a simple unit test approach, i.e., if the total number of records in the column equals the total number of unique records. For instance, the number of records displayed by SQL COUNT (*) and COUNT (DISTINCT CUSTOMER_ID) from CUSTOMERS table are equal, therefore, CUSTOMER_ID becomes a primary key. Likewise, ORDER_ID, PRODUCT_ID, SELLER_ID, PRODUCT_CATEGORY_NAME, and MQL_ID are the primary keys of ORDERS, PRODUCTS, SELLERS, PRODUCT_CATEGORY_NAME_TRANSLATION, and CLOSED_DEALS table.
- Composite keys follow a similar discussion. Here, in addition to a single column, a set of two or more columns determines the uniqueness of the table. For instance, in the ORDER_ITEMS table, the combination of ORDER_ITEMS and ORDER_ITEMS_ID ensures the table's uniqueness. Similarly, ORDER_PAYMENTS and ORDER_REVIEWS table has the composite key of (ORDER_ID, PAYMENT_SEQUENTIAL) and (ORDER_ID, REVIEW_ID) respectively.
- A column is a foreign key for the table if it uniquely identifies another table. We have identified ORDER_ID as a foreign key in the ORDER_PAYMENTS and ORDER_REVIEWS table, as ORDER_ID is a primary key to the ORDERS table. Similarly, CUSTOMER_ID is a foreign key for the ORDERS table that maps to the CUSTOMERS table as a primary key.

Section 3 - The E-R Diagram

Figure 3.1 represents the detailed E-R diagram of the database. The E-R diagram follows the discussion in Section 2, which discusses entities, attributes, relationships, cardinality, and keys.

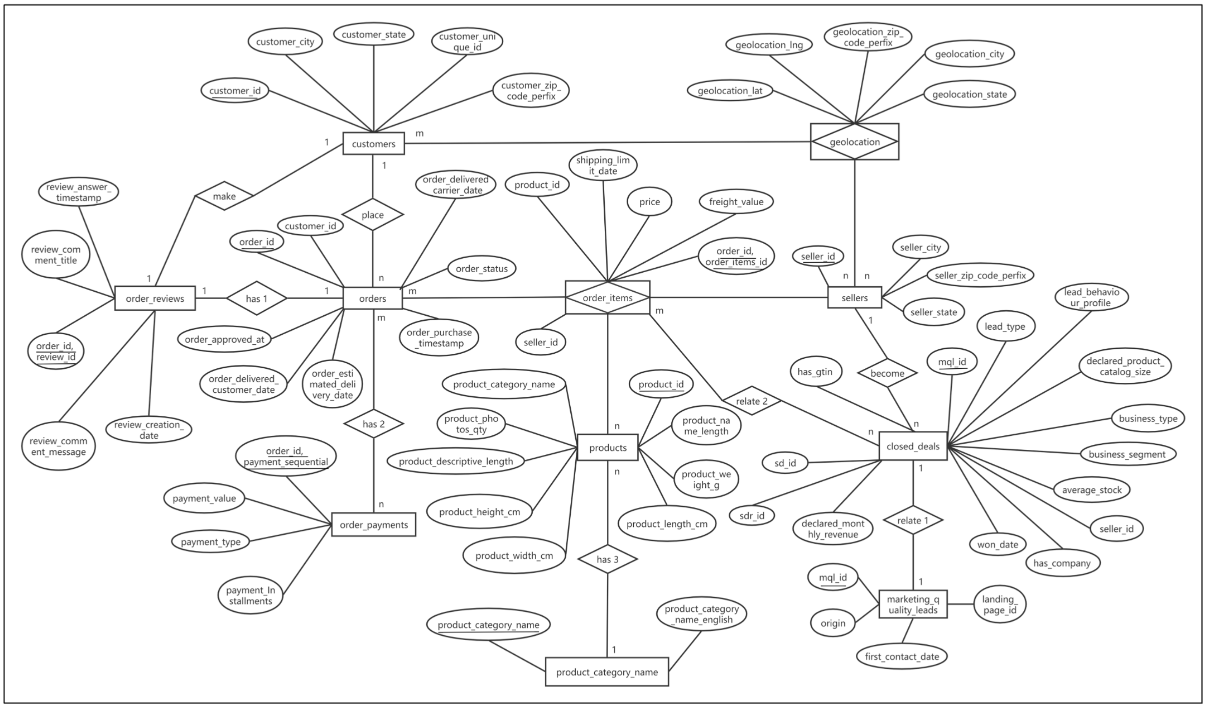

Figure 3.1 E-R diagram

Section 4 - SQL DDL

This section provides the SQL DDL statements displaying the setup of data types and key constraints.

The SQL *CREATE TABLE* consists of five inputs: table name, column names, data types, type of key(s), and constraints. The identification of keys is discussed in section 2.4. For rest, we analysed a few records from each of the tables in the dataset. For instance, in the CUSTOMERS table, CUSTOMER_ID is the primary key. Additionally, there are no missing values in other columns of this table, which necessitates the *NOT NULL* keyword as a constraint.

Data types for each table's attributes are identified by examining the records. All character type columns have *VARCHAR* as a data type. The length of *VARCHAR* is kept in accordance with the possibility of accommodating more data into the current tables, i.e., the DML *INSERT* statement shouldn't fail to execute if the length is kept less than the maximum length required to insert the data. We utilised SQL *MAX(LEN(COLUMN_NAME)* functions to determine the column's maximum length. Columns having integer or decimal values are kept as *INTEGER* and *FLOAT*, respectively. It is critical to note that the column containing *ZIP* information (present in the CUSTOMERS, GEOLOCATION, and SELLERS table) has no numerical significance. In other words, it should be of character data type.

Based on above discussion, the SQL DDL statements are provided in Appendix, Section 2.

Section 5 - Issues with the Data Records

Following issues are identified with the data records to improve our analysis's accuracy and make informed decisions. The SQL codes are shown in Appendix, Section 3.

- The GEOLOCATION table has duplicates. Therefore, it will cause many to many joins condition if joined based on ZIP codes to CUSTOMERS or SELLERS table.
- In PRODUCTS, there are 610 rows where the PRODUCT_CATEGORY_NAME is missing, potentially impacting any product-level analysis. This issue is identified by utilising IS NULL keyword in the SQL WHERE statement.
- Similarly, in the CLOSED_DEALS table, the major entries in the SDR_ID, SR_ID, HAS_COMPANY, HAS_GTIN, AVERAGE_STOCKS, DECLARED_PRODUCT_CATALOG_SIZE, and DECLARED_MONTHLY_REVENUE are nulls; lacking any useful insight. Likewise, there are 160 entries in the ORDERS table where ORDER_APPROVE_DATE is null.
- There is an issue with column CUSTOMER_UNIQUE_ID in the CUSTOMERS table, as duplicates are present for the unique customer id, which could mislead the analysis. The mismatch in COUNT (*) and COUNT (DISTINCT CUSTOMER_UNIQUE_ID) indicates this issue.
- There are issues with the data entered in the PRODUCTS table, where the minimum product weight (g) is recorded as 0.

Section 6 - Database Normalisation

This section, normalises the schema to the highest order. Refer to Figure 6.1 to follow the below discussion.

- The **CUSTOMERS** table is split into three tables. Firstly, we put CUSTOMER_ZIP_CODE and CUSTOMER_CITY together because they have a transitive relationship, which violates 3NF. The database meets the 2NF because the city in which a customer is located is related to the customer's primary key. The zip number, however, affects the city. There is a possibility we will update one column but not the other if a customer relocates. Thus, we split them into a new table. Secondly, we put CUSTOMER_CITY and CUSTOMER_STATE in the other table. Identical to CUSTOMER_ZIP_CITY, city and state also have a transitive relationship. It meets the 2NF, but the state affects the city, so it violates the 3NF. Finally, we make CUSTOMERS_UNIQUE_ZIP, containing CUSTOMER_ID, CUSTOMER_UNIQUE_ID, and CUSTOMER_ZIP_CODE, to link it with other tables (Appendix, Section 4).
- The **SELLERS** table follows the same pattern as described above.
- Three tables listed below, are present in the highest normal form because there are no transitive dependencies.

1. **ORDER_ITEMS**,
2. **PRODUCTS**, and
3. **PRODUCT_CATEGORY_NAME_TRANSLATION**

- In **PAYMENTS** table, we separated the PAYMENT_TYPE and created a separate key, i.e., PAYMENT_KEY, to minimise the risk an UPDATE statement may cause. For instance, if a company wants to remove the 'Voucher' payment type in the future, they will have to delete many rows, which hinders further data analysis and integrity.
- The **ORDER_REVIEW**, **ORDERS**, and **MARKETING_QUALIFIED_LEADS** table follows the above discussion, where REVIEW_SCORE, ORDER_STATUS, and ORIGIN are separated into a table with REVIEW_KEY, ORDER_KEY, and ORIGIN KEY, respectively. An UPDATE statement for review score (order status and origin) might pose risks.
- In the **CLOSED_DEALS** table, firstly, the columns with missing values are dropped. The column, LEAD_BEHAVIOUR_PROFILE, make this table violates 1NF, as values are separated by a comma. After making it into 1NF, MQL_ID is made primary key. Following the discussion on PAYMENTS table, we set up table with four keys with BUSINESS_SEGMENT, LEAD_TYPE, LEAD_BEHAVIOUR_PROFILE, and BUSINESS_TYPE to prevent SQL DML statement related issues.
- Finally, as discussed, **GEOLOCATION** table has data integrity issues caused by duplicates. Therefore, we have dropped latitude and longitude columns, to convert table in the highest normal form.

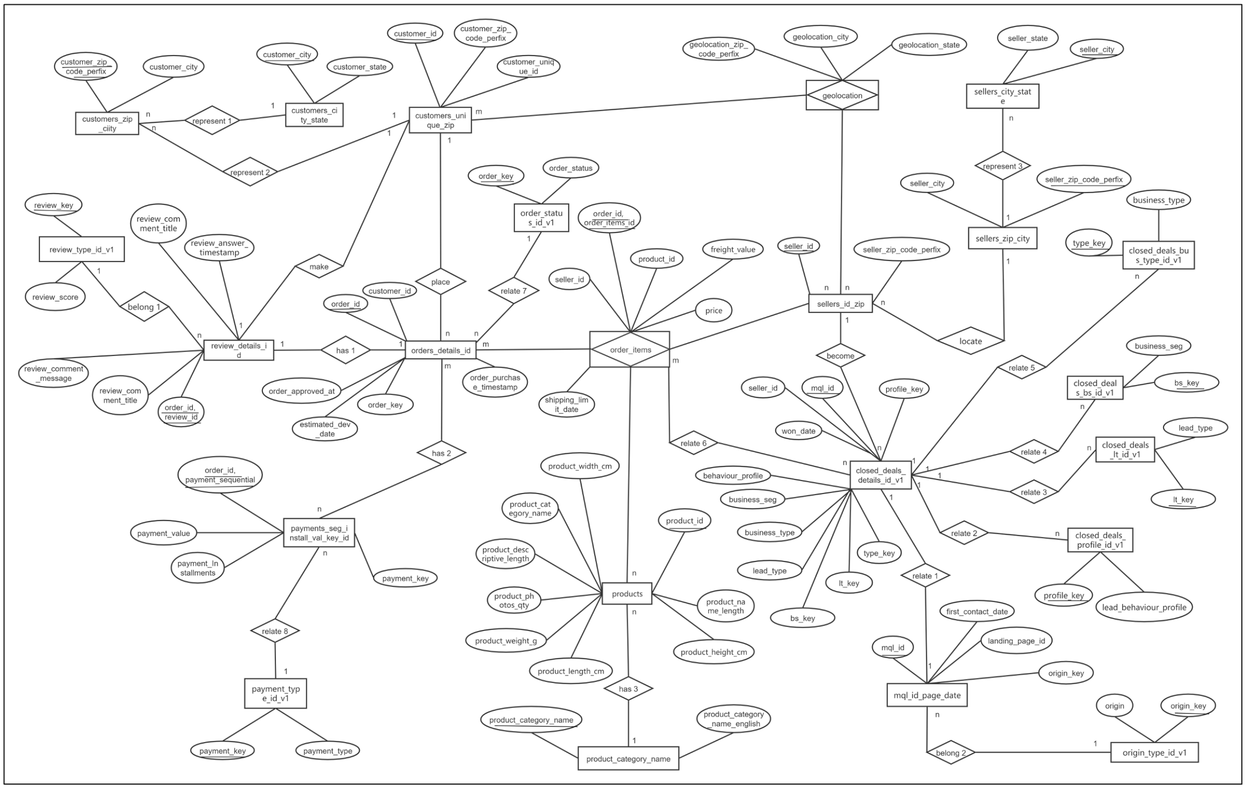

Figure 6.1 E-R diagram after normalising the schema

Section 7 - Complex SQL queries, dplyr equivalents, and ggplots

This section provides five complex SQL queries, their dplyr equivalents, and corresponding visualisations, based on the normalised schema to answer the specific business question described, in sub-sections 7.1 to 7.5. Refer to Appendix, Section 5.1 to 5.5, for a detailed code (and additional explanation).

7.1 - Distribution of sellers

To determine the distribution of sellers by the state for the top three business segments for successful deals, we have utilised the entities shown in the below Venn diagram (Figure 7.1.1). The SQL query demonstrates the understanding of SQL *joins* among four entities, and *IN* in the filter condition. Additionally, to determine the count of unique sellers and select the top three business segments, we have utilised SQL *DISTINCT* and *LIMIT* keywords in conjunction with ORDER BY.

The query mentioned above is transformed by utilising dplyr *joins*, *summarise*, *group_by*, and *arrange* functions within the ggplot to obtain the below-shown graph (Figure 7.1.2).

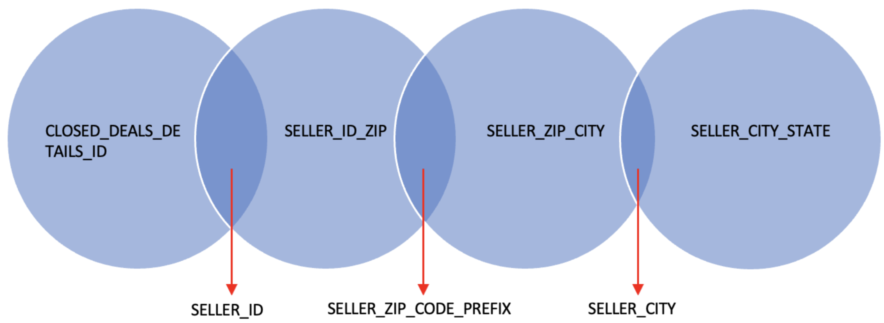

Figure 7.1.1 – Venn diagram for distribution of sellers

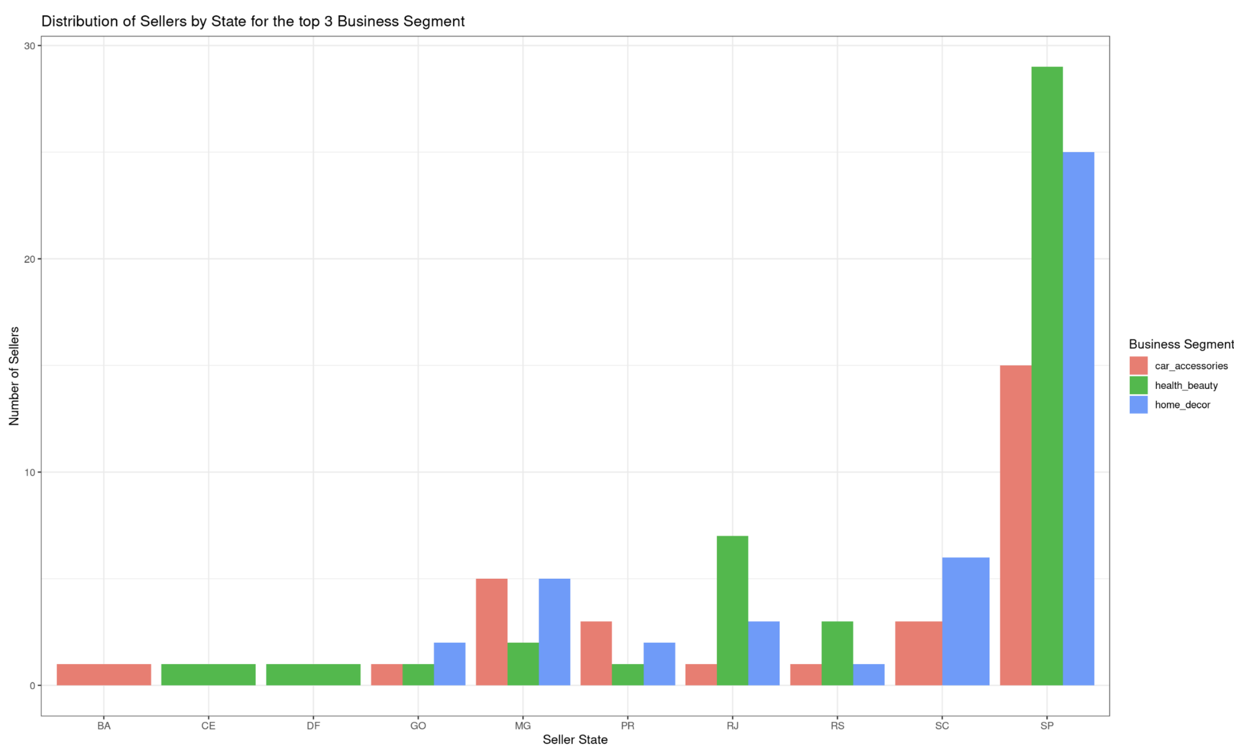

Figure 7.1.2 – Distribution of Sellers by State for the top 3 Business Segment

7.2 - Average review scores

Here we determine the average customer review scores for the top 10 products in demand, utilising the entities shown in Figure 7.2.1. The query demonstrates the utility of the *DISTINCT* and *LIMIT* keywords. In addition, three *INNER JOIN* and a *LEFT JOIN* are utilised to answer the above-mentioned business question. The query mentioned above is transformed by utilising dplyr *filter*, *joins*, *summarise*, *group_by*, and *arrange* functions inside the ggplot to obtain the below-shown graph (Figure 7.2.2).

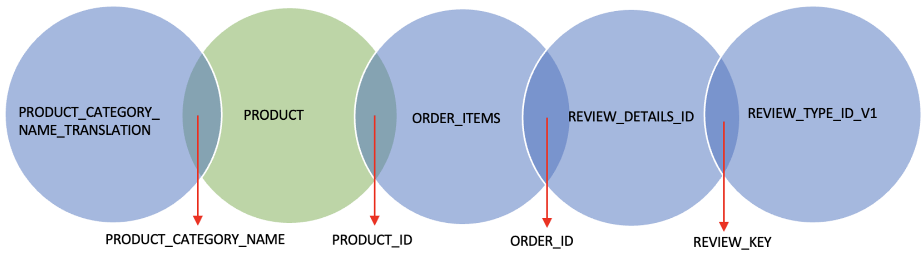

Figure 7.2.1 – Venn diagram for average review scores

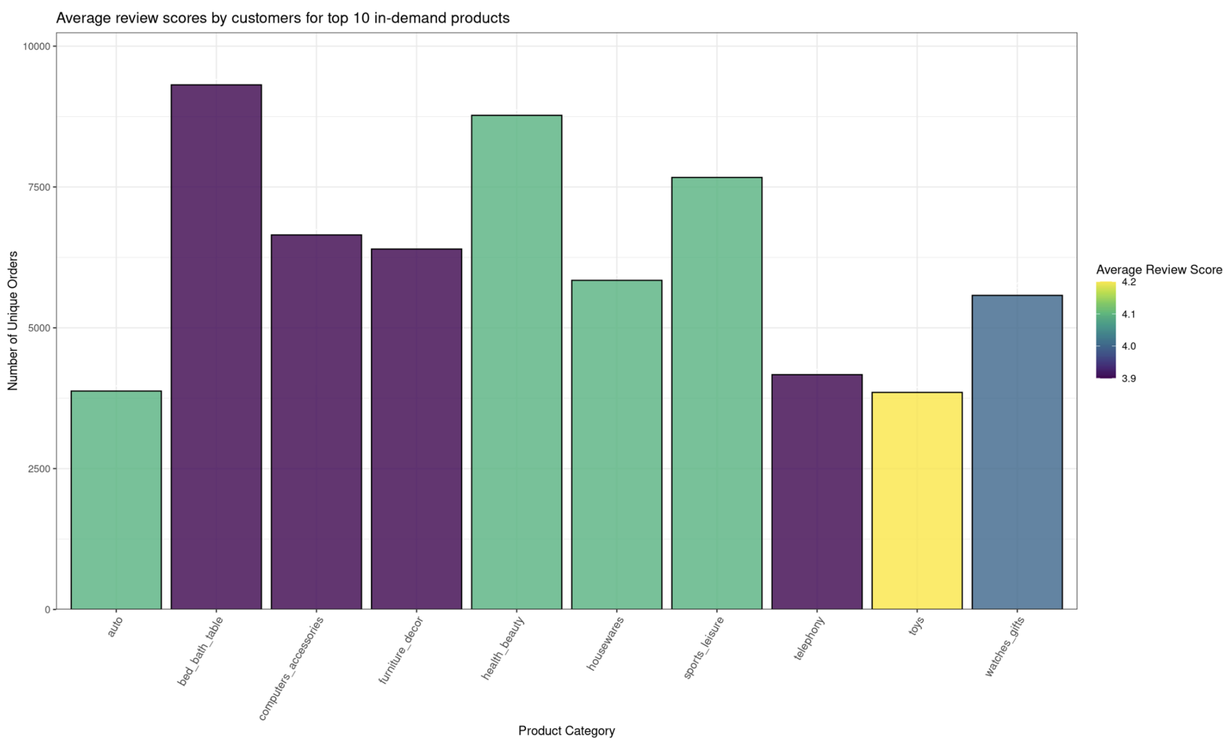

Figure 7.2.2 – Average review scores by customers for top 10 in-demand products

7.3 - Product performance

This sub-section determines the product performance of the top 10 in-demand products regarding delivery schedule with the help of entities shown in Figure 7.3.1. We have utilised SQL *CASE WHEN* statements, equivalent to *if_else* statements in the dplyr. Additionally, the use of *JOINS*, multiple *FILTER* conditions, *GROUP BY* statement, *LIMIT* and *ORDER BY* makes this query complex. The query mentioned above is transformed by utilising dplyr *joins*, *summarise*, *group_by*, *arrange*, *slice()* functions inside the ggplot to obtain the below-shown graph (Figure 7.3.2).

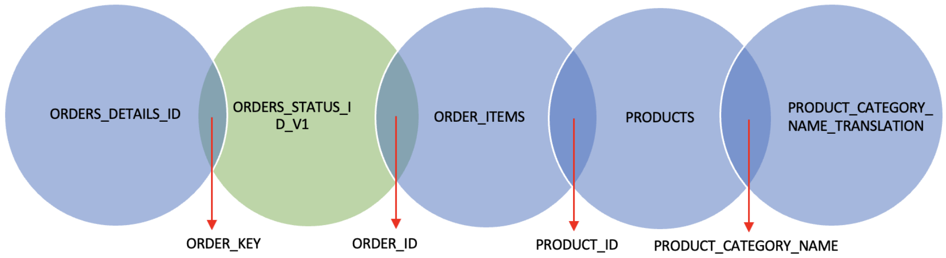

Figure 7.3.1 – Venn diagram for product performance in term of delivery

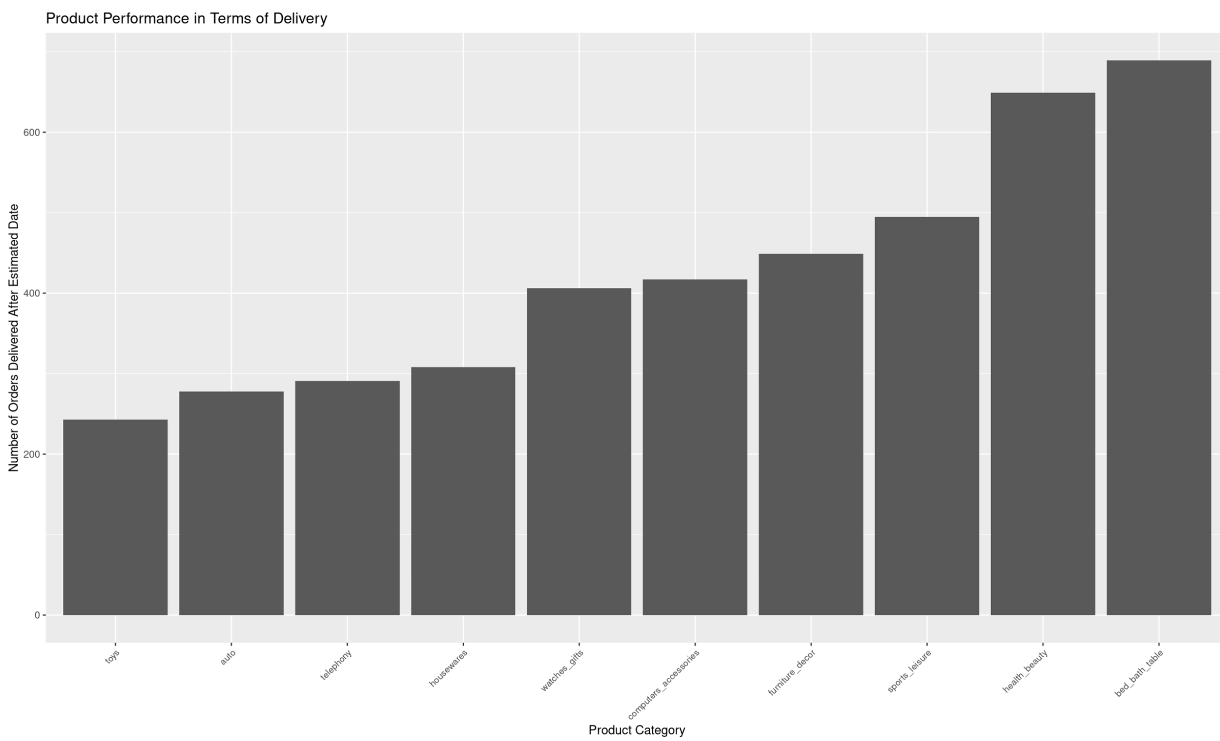

Figure 7.3.2 – Product performance in term of delivery

7.4 - Customer expenditure

To estimate the customer expenditure across months by payment type, we have two business rules to de-dupe the data from the entities shown in Figure 7.4.1. Firstly, we have taken the latest date in case multiple shipping dates are associated with the ORDER_ID. Secondly, if the order has multiple payment types associated, the payment type with maximum payment value supersedes the one with lower payment value. The query mentioned above is transformed by utilising dplyr *joins*, *summarise*, *group_by*, and *arrange* functions inside the ggplot to obtain the below-shown graph (Figure 7.4.2).

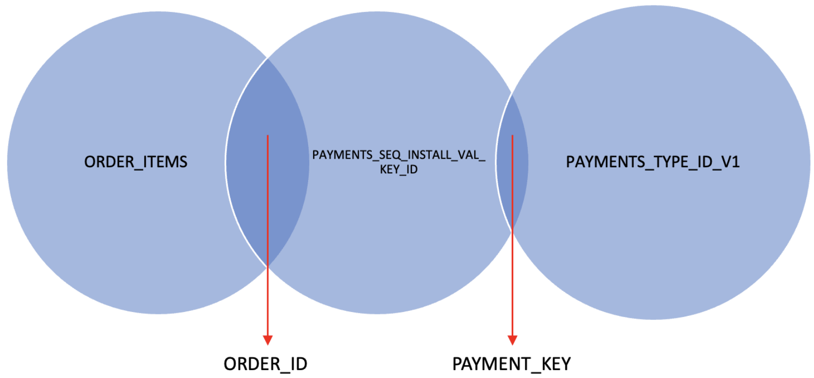

Figure 7.4.1 – Venn diagram for distribution of payments across payment types for the entire time-period

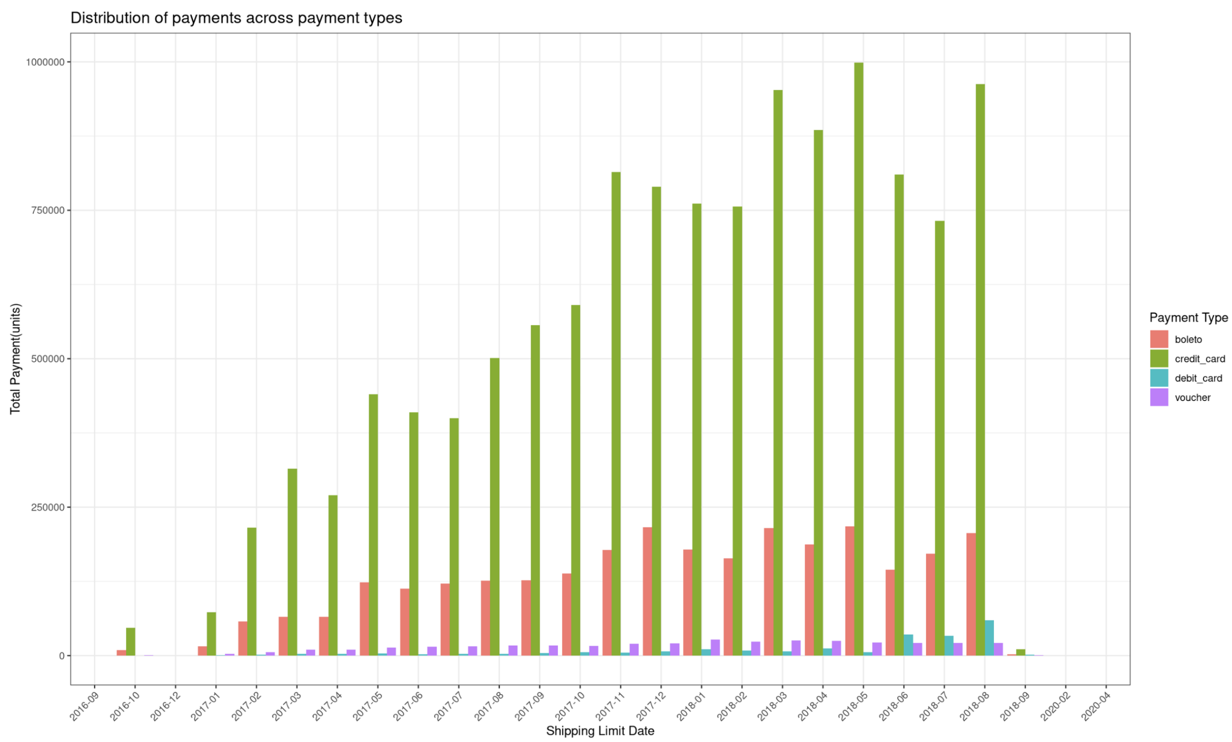

Figure 7.4.2 – Distribution of payments across payment types for the entire time-period

7.5 - Total price by business type and product name

To compare two different business types, i.e., *manufacturer* and *resellers*, for the top-performing product categories, we have utilised the entities shown in Figure 7.5.1. We first identified the top 10 products based on the logic of counting unique orders from the ORDER_ITEMS table with two inner joins with PRODUCTS and PRODUCT_CATEGORY_NAME_TRANSLATION to fetch the product's English name. Later, while restricting to the mentioned business types, we obtained an aggregated summary at the business type and product level to display the total price. The query mentioned above is transformed by utilising dplyr *joins*, *summarise*, *group_by* and multiple *filter* conditions within the ggplot to obtain the below-shown graph (Figure 7.5.2).

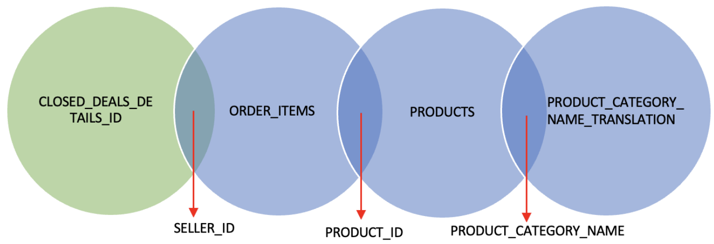

Figure 7.5.1 – Venn diagram for distribution of payments across payment types for the entire time-period

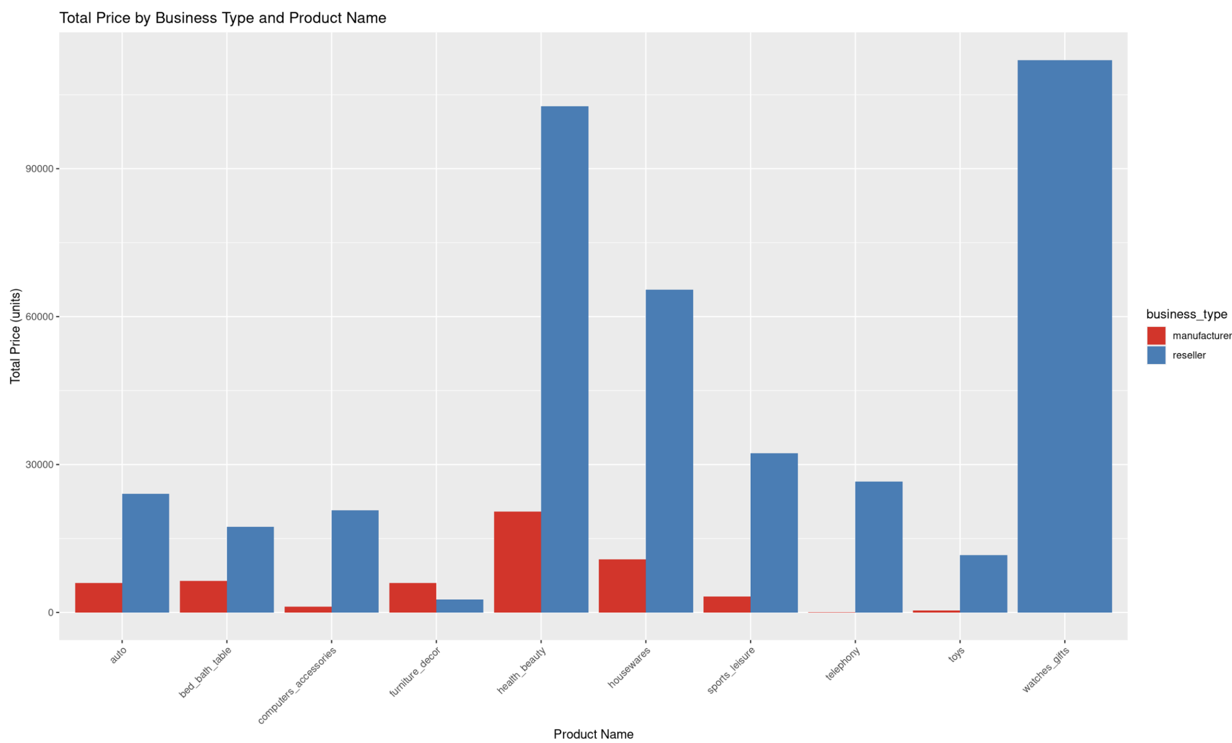

Figure 7.5.2 – Distribution of payments across payment types for the entire time-period

Section 8 - Conclusion

The presented work meets all the mentioned objectives and demonstrates a complete understanding of extraction, refining, and delivering insights through the Olist dataset.

Appendix

Section 1.1

SQL code for preliminary steps and writing tables to SQL database -

- Correcting file name - there is one file product__category__name__translation that has a different naming convention. Renaming this file to olist__product__category__name__translation__dataset to make it consistent for the processing.

```
#file.rename(
#  "product_category_name_translation.csv",
#  "olist_product_category_name_translation_dataset.csv")
```

- looping through the files to get an idea of rows and columns of each file

```
all_files <- list.files("csv_files/")

for (variable in all_files) {
  this_filepath <- paste0("csv_files/",variable)
  this_file_contents <- readr::read_csv(this_filepath, show_col_types = FALSE)

  number_of_rows <- nrow(this_file_contents)
  number_of_columns <- ncol(this_file_contents)

  print(paste0("The file: ",variable,
              " has: ",
              format(number_of_rows,big.mark = ","),
              " rows and ",
              number_of_columns," columns"))
}
```

- Performing test for 1st column if it is primary key or not:

```
for (variable in all_files) {

  this_filepath <- paste0("csv_files/",variable)
  this_file_contents <- readr::read_csv(this_filepath)
  number_of_rows <- nrow(this_file_contents)

  print(paste0("Checking for: ",variable))

  print(paste0(" is ",nrow(unique(this_file_contents[,1]))==number_of_rows))
}
```

- loading the files in SQLite database

```
# Load the library
library(RSQLite)

#setup the connection and creating an empty database

my_connection <- RSQLite::dbConnect(RSQLite::SQLite(),"olist_files.db")

for (variable in all_files) {
  this_filepath <- paste0("csv_files/",variable)
  this_file_contents <- readr::read_csv(this_filepath)
```

```
  table_name <- gsub(".csv","",variable)
  #Remove prefix and suffix
  table_name <- gsub("olist_","",table_name)
  table_name <- gsub("_dataset","",table_name)
  # table_name <- variable

  RSQLite::dbWriteTable(my_connection,table_name,this_file_contents,overwrite=TRUE)

}
```

Section 1.2

- Displaying random 5 records from the CUSTOMERS table

```sql
SELECT *
FROM CUSTOMERS
LIMIT 5;
```

- As evident from the below two queries, CUSTOMERS and ORDERS has 1:1 mapping on CUSTOMER_ID. Because the number of rows in CUSTOMERS is equal to ORDERS, which equals the number of rows in their inner join i.e. 99,441 rows.

```sql
SELECT COUNT(A.CUSTOMER_ID), COUNT(DISTINCT A.CUSTOMER_ID) AS CUST_COUNT_CUSTOMERS
FROM CUSTOMERS A;
```

Table 1: 1 records

COUNT(A.CUSTOMER_ID)	CUST_COUNT_CUSTOMERS
99441	99441

```sql
SELECT COUNT(A.CUSTOMER_ID), COUNT(DISTINCT A.CUSTOMER_ID) AS CUST_COUNT_CUSTOMERS,
COUNT(DISTINCT B.CUSTOMER_ID) AS CUST_COUNT_ORDERS
FROM CUSTOMERS A
INNER JOIN ORDERS B ON A.CUSTOMER_ID = B.CUSTOMER_ID;
```

Table 2: 1 records

COUNT(A.CUSTOMER_ID)	CUST_COUNT_CUSTOMERS	CUST_COUNT_ORDERS
99441	99441	99441

- Below set of results proves the 1:N cardinality ORDER_ITEMS and ORDERS

```sql
SELECT COUNT(A.ORDER_ID),COUNT(B.ORDER_ID),COUNT(DISTINCT A.ORDER_ID),
COUNT(DISTINCT B.ORDER_ID)
FROM ORDERS A
LEFT JOIN ORDER_ITEMS B
ON A.ORDER_ID = B.ORDER_ID;
```

Table 3: 1 records

COUNT(A.ORDER_ID)	COUNT(B.ORDER_ID)	COUNT(DISTINCT A.ORDER_ID)	COUNT(DISTINCT B.ORDER_ID)
113425	112650	99441	98666

Section 1.3

- Identification of keys:

Primary key for CUSTOEMRS table:

The number of rows is equal to the unique count of CUSTOMER_ID, therefore it is a primary key.

```
SELECT COUNT(1), COUNT(DISTINCT CUSTOMER_ID)
FROM CUSTOMERS;
```

Table 4: 1 records

COUNT(1)	COUNT(DISTINCT CUSTOMER_ID)
99441	99441

Primary key for PRODUCTS table:

```
SELECT product_id,COUNT(*) as count
FROM order_items
GROUP BY 1
HAVING COUNT > 1
ORDER BY count DESC;
```

Table 5: Displaying records 1 - 10

product_id	count
aca2eb7d00ea1a7b8ebd4e68314663af	527
99a4788cb24856965c36a24e339b6058	488
422879e10f46682990de24d770e7f83d	484
389d119b48cf3043d311335e499d9c6b	392
368c6c730842d78016ad823897a372db	388
53759a2ecddad2bb87a079a1f1519f73	373
d1c427060a0f73f6b889a5c7c61f2ac4	343
53b36df67ebb7c41585e8d54d6772e08	323
154e7e31ebfa092203795c972e5804a6	281
3dd2a17168ec895c781a9191c1e95ad7	274

Primary key for SELLERS table:

```
SELECT SELLER_ID, COUNT(*) as count
FROM SELLERS
GROUP BY 1
HAVING COUNT > 1
ORDER BY count DESC;
```

Table 6: 0 records

seller_id	count

Primary key for CLOSED_DEALS table:

```
SELECT MQL_ID, COUNT(*) as count
FROM CLOSED_DEALS
GROUP BY 1
HAVING COUNT > 1
ORDER BY count DESC;
```

Table 7: 0 records

mql_id	count

Primary key for MARKETING_QUALIFIED_LEADS table:

```
SELECT MQL_ID, COUNT(*) as count
FROM MARKETING_QUALIFIED_LEADS
GROUP BY 1
HAVING COUNT > 1
ORDER BY count DESC;
```

Table 8: 0 records

mql_id	count

Primary key for PRODUCT_CATEGORY_NAME_TRANSLATION table:

```
SELECT PRODUCT_CATEGORY_NAME, COUNT(*) as count
FROM product_category_name_translation
GROUP BY 1
HAVING COUNT > 1
ORDER BY count DESC;
```

Table 9: 0 records

product_category_name	count

Based on the below, order_id and order_items_id is a composite key for ORDER_ITEMS. As there are no duplicates at this level.

```
SELECT order_id,order_item_id , COUNT(*) as count
FROM order_items
GROUP BY 1,2
HAVING COUNT > 1
ORDER BY count DESC;
```

Table 10: 0 records

order_id	order_item_id	count

Similarly for ORDER_PAYMENTS

```
SELECT ORDER_ID,PAYMENT_SEQUENTIAL, COUNT(*) as count
FROM ORDER_PAYMENTS
GROUP BY 1,2
HAVING COUNT > 1
ORDER BY count DESC;
```

Table 11: 0 records

order_id	payment_sequential	count

Section 2

- SQL code to check the maximum length of the column

```
SELECT
MAX(LENGTH(CUSTOMER_ID)) AS CUSTOMER_ID_LEN,
MAX(LENGTH(CUSTOMER_UNIQUE_ID)) AS CUSTOMER_UNIQUE_ID_LEN,
MAX(LENGTH(CUSTOMER_ZIP_CODE_PREFIX)) AS CUSTOMER_ZIP_CODE_PREFIX_LEN,
MAX(LENGTH(CUSTOMER_CITY)) AS CUSTOMER_CITY_LEN,
MAX(LENGTH(CUSTOMER_STATE)) AS CUSTOMER_STATE_LEN
FROM CUSTOMERS;
```

Table 12: 1 records

CUSTOMER_ID_LEN	CUSTOMER_UNIQUE_ID_LEN	CUSTOMER_ZIP_CODE_PREFIX_LEN	CUSTOMER_CITY_LEN	CUSTOMER_STATE_LEN
32	32	5	32	2

- SQL DDL statement for CUSTOMERS

```
-- CUSTOMERS
CREATE TABLE CUSTOMERS (
    customer_id VARCHAR(32) PRIMARY KEY,
    customer_unique_id VARCHAR(32) NOT NULL,
    customer_zip_code_prefix VARCHAR(5) NOT NULL,
    customer_city VARCHAR(100) NOT NULL,
    customer_state CHAR(2) NOT NULL
);
```

- SQL DDL statement for GEOLOCATION

```
--GEOLOCATION
CREATE TABLE GEOLOCATION (
    geolocation_zip_code_prefix VARCHAR(5) NOT NULL,
    geolocation_lat FLOAT NOT NULL,
    geolocation_lng FLOAT NOT NULL,
    geolocation_city VARCHAR(100) NOT NULL,
```

```sql
    geolocation_state CHAR(2) NOT NULL
);
```

- SQL DDL statement for ORDER_ITEMS

```sql
-- ORDER_ITEMS
CREATE TABLE ORDER_ITEMS (
    order_id VARCHAR(32) NOT NULL,
    order_item_id INTEGER NOT NULL,
    product_id VARCHAR(32) NOT NULL,
    seller_id VARCHAR(32) NOT NULL,
    shipping_limit_date TIMESTAMP NOT NULL,
    price FLOAT NOT NULL,
    freight_value FLOAT NOT NULL,
    PRIMARY KEY (order_id, order_item_id),
    FOREIGN KEY (order_id) REFERENCES orders(order_id),
    FOREIGN KEY (product_id) REFERENCES products(product_id),
    FOREIGN KEY (seller_id) REFERENCES sellers(seller_id)
);
```

- SQL DDL statement for ORDER_PAYMENTS

```sql
--ORDER_PAYMENTS
CREATE TABLE ORDER_PAYMENTS (
    order_id VARCHAR(32) NOT NULL,
    payment_sequential INTEGER NOT NULL,
    payment_type VARCHAR(20) NOT NULL,
    payment_installments INTEGER NOT NULL,
    payment_value FLOAT NOT NULL,
    PRIMARY KEY (order_id, payment_sequential),
    FOREIGN KEY (order_id) REFERENCES orders(order_id)
);
```

- SQL DDL statement for ORDER_REVIEWS

```sql
--ORDER_REVIEWS
CREATE TABLE ORDER_REVIEWS (
    review_id VARCHAR(32) NOT NULL,
    order_id VARCHAR(32) NOT NULL,
    review_score INTEGER NOT NULL,
    review_comment_title VARCHAR(50),
    review_comment_message VARCHAR(500),
    review_creation_date TIMESTAMP NOT NULL,
    review_answer_timestamp TIMESTAMP,
    PRIMARY KEY (review_id, order_id),
    FOREIGN KEY (order_id) REFERENCES orders(order_id)
);
```

- SQL DDL statement for ORDERS

```sql
-- ORDERS
CREATE TABLE ORDERS (
    order_id VARCHAR(32) PRIMARY KEY,
    customer_id VARCHAR(32) NOT NULL,
    order_status VARCHAR(20) NOT NULL,
```

```
    order_purchase_timestamp TIMESTAMP NOT NULL,
    order_approved_at TIMESTAMP,
    order_delivered_carrier_date TIMESTAMP,
    order_delivered_customer_date TIMESTAMP,
    order_estimated_delivery_date TIMESTAMP,
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```

- SQL DDL statement for PRODUCTS

```
--PRODUCTS
CREATE TABLE PRODUCTS (
    product_id VARCHAR(32) PRIMARY KEY,
    product_category_name VARCHAR(100),
    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
    FOREIGN KEY (product_category_name) REFERENCES
    PRODUCT_CATEGORY_NAME_TRANSLATION(product_category_name)

);
```

- SQL DDL statement for SELLERS

```
-- SELLERS
CREATE TABLE SELLERS (
    seller_id VARCHAR(32) PRIMARY KEY,
    seller_zip_code_prefix VARCHAR(5) NOT NULL,
    seller_city VARCHAR(100) NOT NULL,
    seller_state CHAR(2)
);
```

- SQL DDL statement for PRODUCT_CATEGORY_NAME_TRANSLATION

```
-- PRODUCT_CATEGORY_NAME_TRANSLATION
CREATE TABLE PRODUCT_CATEGORY_NAME_TRANSLATION (
    product_category_name VARCHAR(100) PRIMARY KEY,
    product_category_name_english VARCHAR(100)
);
```

- SQL DDL statement for CLOSED_DEALS

```
-- CLOSED_DEALS
CREATE TABLE CLOSED_DEALS (
             mql_id VARCHAR(32) PRIMARY KEY,
             seller_id VARCHAR(32) NOT NULL,
             sdr_id VARCHAR(32) NOT NULL,
             sr_id VARCHAR(32) NOT NULL,
             won_date TIMESTAMP NOT NULL,
             business_segment VARCHAR(32),
             lead_type VARCHAR(32),
```

```
            lead_behaviour_profile VARCHAR(32),
            has_company VARCHAR(32),
            has_gtin VARCHAR(32),
            average_stock VARCHAR(32),
            business_type VARCHAR(32),
            declared_product_catalog_size INTEGER,
            declared_monthly_revenue INTEGER NOT NULL,
            FOREIGN KEY (seller_id) REFERENCES sellers(seller_id)
               );
```

- SQL DDL statement for MARKETING_QUALIFIED_DEALS

```
-- MARKETING_QUALIFIED_DEALS
CREATE TABLE MARKETING_QUALIFIED_DEALS (
            mql_id VARCHAR(32) PRIMARY KEY,
            first_contact_date DATE NOT NULL,
            landing_page_id VARCHAR(32) NOT NULL,
            origin VARCHAR(32)
           );
```

Section 3

- Duplicates in GEOLOCATION table:

```sql
SELECT geolocation_zip_code_prefix, geolocation_lat,
geolocation_lng, geolocation_city,
geolocation_state,
COUNT(*) as count
FROM geolocation
GROUP BY 1,2,3,4,5
HAVING COUNT > 1
ORDER BY count DESC
Limit 10;
```

Table 13: Displaying records 1 - 10

geolocation_zip_code_prefix	geolocation_lat	geolocation_lng	geolocation_city	geolocation_state	count
88220	-27.10210	-48.62961	itapema	SC	314
06414	-23.49590	-46.87469	barueri	SP	189
06414	-23.49062	-46.86900	barueri	SP	127
05145	-23.50605	-46.71738	sao paulo	SP	126
22620	-23.00551	-43.37596	rio de janeiro	RJ	102
22640	-23.00458	-43.31990	rio de janeiro	RJ	89
22775	-22.96591	-43.39000	rio de janeiro	RJ	89
06401	-23.50924	-46.88667	barueri	SP	81
71936	-15.84145	-48.02403	brasilia	DF	80
30240	-19.92417	-43.91648	belo horizonte	MG	79

- Missing PRODUCT_CATEGORY_NAME in the PRODUCTS table

```sql
SELECT COUNT(*)
FROM PRODUCTS
WHERE PRODUCT_CATEGORY_NAME IS NULL;
```

Table 14: 1 records

COUNT(*)
610

- Issue with data entry in PRODUCTS table

```sql
SELECT COUNT(*)
FROM ORDERS
WHERE ORDER_APPROVED_AT IS NULL;
```

Table 15: 1 records

COUNT(*)
160

- Issue with CUSTOMER_UNIQUE_ID

```sql
SELECT COUNT(*), COUNT(DISTINCT CUSTOMER_UNIQUE_ID)
FROM CUSTOMERS;
```

Table 16: 1 records

COUNT(*)	COUNT(DISTINCT CUSTOMER_UNIQUE_ID)
99441	96096

- Issue with data entry in PRODUCTS table

```sql
SELECT MIN(PRODUCT_WEIGHT_G), MAX(PRODUCT_WEIGHT_G)
FROM PRODUCTS;
```

Table 17: 1 records

MIN(PRODUCT_WEIGHT_G)	MAX(PRODUCT_WEIGHT_G)
0	40425

Section 4

- SQL code for CUSTOMERS normalisation

```sql
CREATE TABLE CUSTOMERS_ZIP_CITY AS
SELECT DISTINCT CUSTOMER_ZIP_CODE_PREFIX, CUSTOMER_CITY
FROM CUSTOMERS;
```

```sql
CREATE TABLE CUSTOMERS_CITY_STATE AS
SELECT DISTINCT CUSTOMER_CITY, CUSTOMER_STATE
FROM CUSTOMERS;
```

```sql
CREATE TABLE CUSTOMERS_UNIQUE_ZIP AS
SELECT DISTINCT CUSTOMER_ID,CUSTOMER_UNIQUE_ID, CUSTOMER_ZIP_CODE_PREFIX
FROM CUSTOMERS;
```

Section 5.1

- The SQL code for Section 7.1

```sql

SELECT A.BUSINESS_SEGMENT,D.SELLER_STATE, COUNT(DISTINCT B.SELLER_ID) AS SELLERS
FROM CLOSED_DEALS_DETAILS_ID A
INNER JOIN SELLERS_ID_ZIP B ON A.SELLER_ID = B.SELLER_ID
INNER JOIN SELLERS_ZIP_CITY C ON B.SELLER_ZIP_CODE_PREFIX = C.SELLER_ZIP_CODE_PREFIX
INNER JOIN SELLERS_CITY_STATE D ON C.SELLER_CITY = D.SELLER_CITY
WHERE A.BUSINESS_SEGMENT IN (
SELECT DISTINCT BUSINESS_SEGMENT FROM (
SELECT BUSINESS_SEGMENT, COUNT(DISTINCT SELLER_ID) AS SELLERS_FREQ
FROM CLOSED_DEALS_DETAILS_ID
GROUP BY 1
ORDER BY 2 DESC
LIMIT 3)
```

```
)
GROUP BY 1,2
ORDER BY 1,2,3;
```

- The dplyr code for Section 7.1

```
my_connection <- RSQLite::dbConnect(RSQLite::SQLite(),"olist_files.db")

CLOSED_DEALS_DETAILS_ID <- as.data.frame(dbGetQuery(my_connection,
                                    "SELECT * FROM CLOSED_DEALS_DETAILS_ID"))

SELLERS_ID_ZIP <- as.data.frame(dbGetQuery(my_connection,
                                            "SELECT * FROM SELLERS_ID_ZIP"))

SELLERS_ZIP_CITY <- as.data.frame(dbGetQuery(my_connection,
                                            "SELECT * FROM SELLERS_ZIP_CITY"))

SELLERS_CITY_STATE <- as.data.frame(dbGetQuery(my_connection,
                                     "SELECT * FROM SELLERS_CITY_STATE"))




# First sub-query to get the top 3 business segments by seller frequency

top_business_segments <-
  CLOSED_DEALS_DETAILS_ID %>%
  group_by(business_segment) %>%
  summarize(SELLERS_FREQ = n_distinct(seller_id)) %>%
  arrange(desc(SELLERS_FREQ)) %>%
  slice(1:3) %>%
  select(business_segment)

# Main query to join tables and get count of distinct sellers




result <-
  CLOSED_DEALS_DETAILS_ID %>%
  inner_join(SELLERS_ID_ZIP, by = "seller_id") %>%
  inner_join(SELLERS_ZIP_CITY, by = c(
    "seller_zip_code_prefix" = "seller_zip_code_prefix")) %>%
  inner_join(SELLERS_CITY_STATE, by = c("seller_city" = "seller_city")) %>%
  filter(business_segment %in% top_business_segments$business_segment) %>%
  group_by(business_segment, seller_state) %>%
  summarize(SELLERS = n_distinct(seller_id)) %>%
  arrange(business_segment, seller_state, SELLERS)
```

- The ggplot code for Section 7.1

```
ggplot(

  CLOSED_DEALS_DETAILS_ID %>%
  inner_join(SELLERS_ID_ZIP, by = "seller_id") %>%
```

```
  inner_join(SELLERS_ZIP_CITY, by = c(
    "seller_zip_code_prefix" = "seller_zip_code_prefix")) %>%
  inner_join(SELLERS_CITY_STATE, by = c("seller_city" = "seller_city")) %>%

  filter(business_segment %in% top_business_segments$business_segment) %>%

  group_by(business_segment, seller_state) %>%
  summarize(SELLERS = n_distinct(seller_id)) %>%
  arrange(business_segment, seller_state, SELLERS),

  aes(x = seller_state, y = SELLERS, fill = business_segment)) +

  geom_bar(stat = "identity", position = "dodge") +

  labs(x = "Seller State", y = "Number of Sellers", fill = "Business Segment",
  title = "Distribution of Sellers by State for the top 3 Business Segment") +

  theme_bw()
```

Section 5.2

- The SQL code for Section 7.2

```sql
SELECT A.*
FROM (
  SELECT E.PRODUCT_CATEGORY_NAME_ENGLISH,
  ROUND(AVG(D.REVIEW_SCORE),1) AS AVG_REVIEW_SCORE,
  COUNT(DISTINCT C.ORDER_ID) AS UNIQUE_ORDERS
  FROM PRODUCTS A
  LEFT JOIN ORDER_ITEMS B ON A.PRODUCT_ID = B.PRODUCT_ID
  INNER JOIN REVIEW_DETAILS_ID C ON B.ORDER_ID = C.ORDER_ID
  INNER JOIN REVIEW_TYPE_ID_V1 D ON C.REVIEW_KEY = D.REVIEW_KEY
  INNER JOIN PRODUCT_CATEGORY_NAME_TRANSLATION E
    ON A.PRODUCT_CATEGORY_NAME = E.PRODUCT_CATEGORY_NAME
  WHERE A.PRODUCT_CATEGORY_NAME IS NOT NULL
  GROUP BY A.PRODUCT_CATEGORY_NAME
) A
ORDER BY 3 DESC
LIMIT 10;
```

Table 18: Displaying records 1 - 10

PRODUCT_CATEGORY_NAME_ENGLISH	AVG_REVIEW_SCORE	UNIQUE_ORDERS
bed_bath_table	3.9	9313
health_beauty	4.1	8771
sports_leisure	4.1	7669
computers_accessories	3.9	6649
furniture_decor	3.9	6398
housewares	4.1	5843
watches_gifts	4.0	5576
telephony	3.9	4168
auto	4.1	3877
toys	4.2	3853

- The dplyr code for Section 7.2

```
my_connection <- RSQLite::dbConnect(RSQLite::SQLite(),"olist_files.db")

PRODUCTS <- as.data.frame(dbGetQuery(my_connection,
                                    "SELECT * FROM PRODUCTS"))

ORDER_ITEMS <- as.data.frame(dbGetQuery(my_connection,
                                    "SELECT * FROM ORDER_ITEMS"))

REVIEW_DETAILS_ID <- as.data.frame(dbGetQuery(my_connection,
                                          "SELECT * FROM REVIEW_DETAILS_ID"))

REVIEW_TYPE_ID_V1 <- as.data.frame(dbGetQuery(my_connection,
                                          "SELECT * FROM REVIEW_TYPE_ID_V1"))

PRODUCT_TRANSLATION <- as.data.frame(dbGetQuery(my_connection,
                      "SELECT * FROM PRODUCT_CATEGORY_NAME_TRANSLATION"))


result_b <- PRODUCTS %>%
  filter(!is.na(product_category_name)) %>%
  left_join(ORDER_ITEMS, by = "product_id") %>%
  inner_join(REVIEW_DETAILS_ID, by = "order_id") %>%
  inner_join(REVIEW_TYPE_ID_V1, by = "REVIEW_KEY") %>%
  inner_join(PRODUCT_TRANSLATION,
             by = c("product_category_name" = "product_category_name")) %>%
  group_by(product_category_name) %>%
  summarize(AVG_REVIEW_SCORE = round(mean(review_score), 1),
            UNIQUE_ORDERS = n_distinct(order_id)) %>%
  arrange(desc(UNIQUE_ORDERS)) %>%
  head(10)
```

- The ggplot code for Section 7.2

```
ggplot(
  PRODUCTS %>%
  left_join(ORDER_ITEMS, by = "product_id") %>%
  inner_join(ORDER_REVIEWS, by = "order_id") %>%
  inner_join(PRODUCT_TRANSLATION, by = c(
    "product_category_name" = "product_category_name")) %>%
  group_by(product_category_name_english) %>%
  summarize(
    AVG_REVIEW_SCORE = round(mean(review_score), 1),
    UNIQUE_ORDERS = n_distinct(order_id)
  ) %>%
  filter(!is.na(product_category_name_english)) %>%
  select(product_category_name_english, AVG_REVIEW_SCORE, UNIQUE_ORDERS) %>%
  arrange(desc(UNIQUE_ORDERS)) %>%
  head(10),
  aes(x = product_category_name_english, y = UNIQUE_ORDERS,
                 fill = AVG_REVIEW_SCORE)) +
  geom_bar(stat = "identity", color = "black", alpha = 0.8) +
  scale_fill_gradientn(colors = viridis(10)) +
  labs(x = "Product Category", y = "Number of Unique Orders",
```

```
      fill = "Average Review Score") +
  ggtitle("Average review scores by customers for top 10 in-demand products") +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 60, hjust = 1, size = 10)) +
  guides(fill = guide_colorbar(title.position = "top", title.hjust = 0.5)) +
  scale_y_continuous(expand = c(0, 0), limits = c(0, max(result$UNIQUE_ORDERS) * 1.1)) +
  geom_text(aes(label = AVG_REVIEW_SCORE), size = 4, vjust = -0.5, color = "white")
```

Section 5.3

- The SQL code for Section 7.3

In the below code, we have utilised the basic concept of CTE in SQL to answer the business question. The motivation behind utilising the CTE is make the query more efficient, as it increases the readability of the code and minimizes the over-use of nested sub-queries.

```
WITH DATES AS
(
SELECT A.order_id,
strftime('%Y-%m-%d',
datetime(A.order_delivered_customer_date, 'unixepoch')) as delivered_date,
strftime('%Y-%m-%d',
datetime(A.order_estimated_delivery_date,'unixepoch')) as estimated_date
FROM ORDERS_DETAILS_ID A
INNER JOIN ORDERS_STATUS_ID_V1 B ON A.ORDER_KEY = B.ORDER_KEY
WHERE B.ORDER_STATUS = "delivered"
AND A.order_delivered_customer_date IS NOT NULL
AND A.order_estimated_delivery_date IS NOT NULL
),
DELIVERED_CATEGORY AS
(
SELECT A.order_id,
D.PRODUCT_CATEGORY_NAME_ENGLISH AS PRODUCT_NAME,
CASE WHEN delivered_date <= estimated_date THEN 'BEFORE_OR_ON_SCHEDULE'
WHEN delivered_date > estimated_date THEN 'AFTER_SCHEDULE' END AS DELIVERED_CATEGORY
FROM DATES A
INNER JOIN ORDER_ITEMS B ON A.ORDER_ID = B.ORDER_ID
LEFT JOIN PRODUCTS C ON B.PRODUCT_ID = C.PRODUCT_ID
LEFT JOIN PRODUCT_CATEGORY_NAME_TRANSLATION D ON
C.PRODUCT_CATEGORY_NAME = D.PRODUCT_CATEGORY_NAME
WHERE PRODUCT_CATEGORY_NAME_ENGLISH IS NOT NULL
),
TOTAL_ORDERS AS
(
SELECT PRODUCT_NAME,DELIVERED_CATEGORY,COUNT(DISTINCT ORDER_ID) AS TOTAL_ORDERS
FROM DELIVERED_CATEGORY
GROUP BY 1,2
)
SELECT PRODUCT_NAME,SUM(TOTAL_ORDERS) AS UNQIUE_ORDERS,
SUM(CASE WHEN DELIVERED_CATEGORY = "AFTER_SCHEDULE"
              THEN TOTAL_ORDERS END) AS AFTER_SCHEDULE,
SUM(CASE WHEN DELIVERED_CATEGORY = "BEFORE_OR_ON_SCHEDULE"
THEN TOTAL_ORDERS END) AS BEFORE_OR_ON_SCHEDULE
FROM TOTAL_ORDERS
```

```
GROUP BY 1
ORDER BY UNQIUE_ORDERS DESC
LIMIT 10;
```

Table 19: Displaying records 1 - 10

PRODUCT_NAME	UNQIUE_ORDERS	AFTER_SCHEDULE	BEFORE_OR_ON_SCHEDULE
bed_bath_table	9272	689	8583
health_beauty	8647	649	7998
sports_leisure	7529	495	7034
computers_accessories	6529	417	6112
furniture_decor	6307	449	5858
housewares	5743	308	5435
watches_gifts	5493	406	5087
telephony	4093	291	3802
auto	3809	278	3531
toys	3803	243	3560

- The dplyr code for Section 7.3

```
my_connection <- RSQLite::dbConnect(RSQLite::SQLite(),"olist_files.db")

ORDERS_DETAILS_ID <- as.data.frame(dbGetQuery(my_connection,
                                          "SELECT * FROM ORDERS_DETAILS_ID"))

ORDERS_STATUS_ID_V1 <- as.data.frame(dbGetQuery(my_connection,
                                          "SELECT * FROM ORDERS_STATUS_ID_V1"))


ORDER_ITEMS <- as.data.frame(dbGetQuery(my_connection,
                            "SELECT * FROM ORDER_ITEMS"))

PRODUCTS <- as.data.frame(dbGetQuery(my_connection, "SELECT * FROM PRODUCTS"))

PRODUCT_TRANSLATION <- as.data.frame(dbGetQuery(my_connection,
                          "SELECT * FROM PRODUCT_CATEGORY_NAME_TRANSLATION"))

library(base)


DATES <- ORDERS_DETAILS_ID %>%
  inner_join(ORDERS_STATUS_ID_V1, by = "ORDER_KEY") %>%
  filter(order_status == "delivered"&
           !is.na(order_delivered_customer_date)&
           !is.na(order_estimated_delivery_date)) %>%
  summarise(order_id,
        delivered_date = format(as.POSIXct(order_delivered_customer_date,
                                           origin = "1970-01-01"), "%Y-%m-%d"),
        estimated_date= format(as.POSIXct(order_estimated_delivery_date,
                                          origin = "1970-01-01"), "%Y-%m-%d"));


# create DELIVERED_CATEGORY table
```

```r
DELIVERED_CATEGORY <- DATES %>%
  inner_join(ORDER_ITEMS, by = "order_id") %>%
  left_join(PRODUCTS, by = "product_id") %>%
  left_join(PRODUCT_TRANSLATION, by = c("product_category_name" =
                                        "product_category_name")) %>%
  filter(!is.na(product_category_name_english)) %>%
  mutate(DELIVERED_CATEGORY = if_else(delivered_date <= estimated_date,
                             "BEFORE_OR_ON_SCHEDULE", "AFTER_SCHEDULE"))

# create TOTAL_ORDERS table
TOTAL_ORDERS <- DELIVERED_CATEGORY %>%
  group_by(product_category_name_english, DELIVERED_CATEGORY) %>%
  summarise(TOTAL_ORDERS = n_distinct(order_id))

# final query
RESULT <- TOTAL_ORDERS %>%
  group_by(product_category_name_english) %>%
  summarise(UNQIUE_ORDERS = sum(TOTAL_ORDERS),
            AFTER_SCHEDULE = sum(if_else(DELIVERED_CATEGORY == "AFTER_SCHEDULE",
                                         TOTAL_ORDERS, 0)),
            BEFORE_OR_ON_SCHEDULE = sum(if_else(
          DELIVERED_CATEGORY == "BEFORE_OR_ON_SCHEDULE", TOTAL_ORDERS, 0))) %>%
  arrange(desc(UNQIUE_ORDERS)) %>%
  slice(1:10)
```

- The ggplot code for Section 7.3

```r
ggplot(TOTAL_ORDERS %>%
  group_by(product_category_name_english) %>%
  summarise(UNQIUE_ORDERS = sum(TOTAL_ORDERS),
            AFTER_SCHEDULE = sum(if_else(DELIVERED_CATEGORY == "AFTER_SCHEDULE",
                                         TOTAL_ORDERS, 0)),
            BEFORE_OR_ON_SCHEDULE = sum(if_else(
          DELIVERED_CATEGORY == "BEFORE_OR_ON_SCHEDULE", TOTAL_ORDERS, 0))) %>%
  arrange(desc(UNQIUE_ORDERS)) %>%
  slice(1:10), aes(x = reorder(product_category_name_english, AFTER_SCHEDULE)
                    , y = AFTER_SCHEDULE)) +
  geom_bar(stat = "identity") +
  labs(title = "Product Performance in Terms of Delivery",
       x = "Product Category",
       y = "Number of Orders Delivered After Estimated Date") +
  theme(axis.text.x = element_text(size = 8, angle = 45, hjust = 1))
```

Section 5.4

- The SQL code for Section 7.4

Motivation behind utilising the ROW_NUMBER() function - To answer the real-world business questions, we occasionally need to partition data based on certain business rules. For example, if a order_id has multiple payment types, the business would want to consider the payment type having maximum payment value for that order_id, therefore, in such scenarios, ROW_NUMBER() function becomes useful.

```sql
WITH ORDER_SHIPPING_DATE AS
(
SELECT ORDER_ID, MAX(strftime('%Y-%m',
```

```
datetime(SHIPPING_LIMIT_DATE, 'unixepoch'))) AS SHIPPING_LIMIT_DATE
FROM ORDER_ITEMS
GROUP BY ORDER_ID
HAVING COUNT(DISTINCT shipping_limit_date) > 1
UNION ALL
SELECT ORDER_ID, strftime('%Y-%m',
datetime(SHIPPING_LIMIT_DATE, 'unixepoch')) AS SHIPPING_LIMIT_DATE
FROM ORDER_ITEMS
GROUP BY ORDER_ID
HAVING COUNT(DISTINCT shipping_limit_date) = 1
),
ORDER_PAYMENT AS
(
SELECT ORDER_ID, PAYMENT_TYPE, PAYMENT_VALUE
FROM (
  SELECT ORDER_ID, PAYMENT_TYPE, PAYMENT_VALUE,
         ROW_NUMBER() OVER (PARTITION BY ORDER_ID ORDER BY PAYMENT_VALUE DESC) as rn
  FROM PAYMENTS_SEQ_INSTALL_VAL_KEY_ID A
  INNER JOIN PAYMENTS_TYPE_ID_V1 B ON A.PAYMENT_KEY = B.PAYMENT_KEY
  WHERE PAYMENT_TYPE <> "not_defined"
) A
WHERE rn = 1
)
SELECT A.SHIPPING_LIMIT_DATE, PAYMENT_TYPE ,SUM(B.PAYMENT_VALUE) AS TOTAL_PAYMENT
FROM ORDER_SHIPPING_DATE A
INNER JOIN ORDER_PAYMENT B
ON A.ORDER_ID = B.ORDER_ID
WHERE CAST(REPLACE(SHIPPING_LIMIT_DATE, '-', '') AS INTEGER) BETWEEN 201609 AND 202004
GROUP BY 1,2
ORDER BY 1;
```

Table 20: Displaying records 1 - 10

SHIPPING_LIMIT_DATE	PAYMENT_TYPE	TOTAL_PAYMENT
2016-09	credit_card	75.06
2016-10	boleto	9076.14
2016-10	credit_card	46674.03
2016-10	debit_card	241.73
2016-10	voucher	571.27
2016-12	credit_card	19.62
2017-01	boleto	15418.97
2017-01	credit_card	72926.42
2017-01	debit_card	517.14
2017-01	voucher	2116.96

- The dplyr code for Section 7.4

```
my_connection <- RSQLite::dbConnect(RSQLite::SQLite(),"olist_files.db")

PAYMENTS_SEQ_INSTALL_VAL_KEY_ID <- as.data.frame(dbGetQuery(my_connection,
                              "SELECT * FROM PAYMENTS_SEQ_INSTALL_VAL_KEY_ID"))

PAYMENTS_TYPE_ID_V1 <- as.data.frame(dbGetQuery(my_connection,
```

```
                                    "SELECT * FROM PAYMENTS_TYPE_ID_V1"))


ORDER_ITEMS <- as.data.frame(dbGetQuery(my_connection,
                                "SELECT * FROM ORDER_ITEMS"))


setDT(ORDER_ITEMS)
order_shipping_date <- ORDER_ITEMS[, .(
  SHIPPING_LIMIT_DATE = ifelse(
    uniqueN(shipping_limit_date) > 1,
    max(format(as.POSIXct(shipping_limit_date, origin = "1970-01-01"), "%Y-%m")),
    format(as.POSIXct(shipping_limit_date, origin = "1970-01-01"), "%Y-%m")
  )
), by = order_id]


order_payment <- PAYMENTS_SEQ_INSTALL_VAL_KEY_ID %>%
  inner_join(PAYMENTS_TYPE_ID_V1, by = "PAYMENT_KEY") %>%
  filter(payment_type != "not_defined") %>%
  group_by(order_id) %>%
  slice_max(payment_type, n = 1)

result <- inner_join(order_shipping_date, order_payment, by = "order_id") %>%
  filter(as.integer(str_replace(SHIPPING_LIMIT_DATE, "-", "")) %>%
           between(201609, 202004)) %>%
  group_by(SHIPPING_LIMIT_DATE, payment_type) %>%
  summarize(TOTAL_PAYMENT = sum(payment_value)) %>%
  arrange(SHIPPING_LIMIT_DATE)
```

- The ggplot code for Section 7.4

```
ggplot(inner_join(order_shipping_date, order_payment, by = "order_id") %>%
  filter(as.integer(str_replace(SHIPPING_LIMIT_DATE, "-", "")) %>%
           between(201609, 202004)) %>%
  group_by(SHIPPING_LIMIT_DATE, payment_type) %>%
  summarize(TOTAL_PAYMENT = sum(payment_value)) %>%
  arrange(SHIPPING_LIMIT_DATE)
  , aes(x = SHIPPING_LIMIT_DATE, y = TOTAL_PAYMENT, fill = payment_type)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(x = "Shipping Limit Date", y = "Total Payment(units)", fill = "Payment Type") +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1),
        plot.title = element_text(size = 14)) +
  ggtitle("Distribution of payments across payment types")
```

Section 5.5

- The SQL code for Section 7.5

```
WITH TOP_10_PORD AS
(
SELECT C.PRODUCT_CATEGORY_NAME_ENGLISH, COUNT(DISTINCT ORDER_ID) AS TOTAL_ORDERS
FROM ORDER_ITEMS A
```

```
INNER JOIN PRODUCTS B ON A.PRODUCT_ID = B.PRODUCT_ID
INNER JOIN PRODUCT_CATEGORY_NAME_TRANSLATION C ON
B.PRODUCT_CATEGORY_NAME = C.PRODUCT_CATEGORY_NAME
GROUP BY 1
ORDER BY 2 DESC
LIMIT 10
)
SELECT A.BUSINESS_TYPE, D.PRODUCT_CATEGORY_NAME_ENGLISH, SUM(B.PRICE + B.FREIGHT_VALUE)
FROM CLOSED_DEALS_DETAILS_ID A
INNER JOIN ORDER_ITEMS B ON A.SELLER_ID = B.SELLER_ID
INNER JOIN PRODUCTS C ON B.PRODUCT_ID = C.PRODUCT_ID
INNER JOIN PRODUCT_CATEGORY_NAME_TRANSLATION D ON
C.PRODUCT_CATEGORY_NAME = D.PRODUCT_CATEGORY_NAME
WHERE A.BUSINESS_TYPE IN ("manufacturer","reseller")
AND D.PRODUCT_CATEGORY_NAME_ENGLISH IN (
  SELECT DISTINCT PRODUCT_CATEGORY_NAME_ENGLISH FROM TOP_10_PORD
)
GROUP BY A.BUSINESS_TYPE, D.PRODUCT_CATEGORY_NAME_ENGLISH;
```

Table 21: Displaying records 1 - 10

business_type	product_category_name_english	SUM(B.PRICE + B.FREIGHT_VALUE)
manufacturer	auto	5953.13
manufacturer	bed_bath_table	1973.20
manufacturer	computers_accessories	1160.38
manufacturer	furniture_decor	2839.07
manufacturer	health_beauty	12067.77
manufacturer	housewares	7121.88
manufacturer	sports_leisure	1179.68
manufacturer	telephony	58.52
manufacturer	toys	402.56
reseller	auto	20568.69

- The dplyr code for Section 7.5

```
my_connection <- RSQLite::dbConnect(RSQLite::SQLite(),"olist_files.db")

PRODUCTS <- as.data.frame(dbGetQuery(my_connection, "SELECT * FROM PRODUCTS"))

ORDER_ITEMS <- as.data.frame(dbGetQuery(my_connection, "SELECT * FROM ORDER_ITEMS"))

PRODUCT_TRANSLATION <- as.data.frame(dbGetQuery(my_connection,
                            "SELECT * FROM PRODUCT_CATEGORY_NAME_TRANSLATION"))

CLOSED_DEALS_BS_ID_V1 <- as.data.frame(dbGetQuery(my_connection,
                                        "SELECT * FROM CLOSED_DEALS_BS_ID_V1"))


CLOSED_DEALS_DETAILS_ID <- as.data.frame(dbGetQuery(my_connection,
                                    "SELECT * FROM CLOSED_DEALS_DETAILS_ID"))
```

```r
# top 10 products by order count
top_10_prod <-
  ORDER_ITEMS %>%
  inner_join(PRODUCTS, by = "product_id") %>%
  inner_join(PRODUCT_TRANSLATION, by = c(
    "product_category_name" = "product_category_name")) %>%
  group_by(product_category_name_english) %>%
  summarise(TOTAL_ORDERS = n_distinct(order_id)) %>%
  arrange(desc(TOTAL_ORDERS)) %>%
  slice_head(n = 10)

# closed deals by business type and top 10 products
total_price_by_business_type <- CLOSED_DEALS_DETAILS_ID %>%
  inner_join(ORDER_ITEMS, by = "seller_id") %>%
  inner_join(PRODUCTS, by = "product_id") %>%
  inner_join(PRODUCT_TRANSLATION, by = c(
    "product_category_name" = "product_category_name")) %>%
  filter(business_type %in% c("manufacturer", "reseller")) %>%
  filter(product_category_name_english %in%
           top_10_prod$product_category_name_english) %>%
  group_by(business_type, product_category_name_english) %>%
  summarise(TOTAL_PRICE = sum(price + freight_value))
```

- The ggplot code for Section 7.5

```r
ggplot(CLOSED_DEALS_DETAILS_ID %>%
  inner_join(ORDER_ITEMS, by = "seller_id") %>%
  inner_join(PRODUCTS, by = "product_id") %>%
  inner_join(PRODUCT_TRANSLATION, by = c(
    "product_category_name" = "product_category_name")) %>%
  filter(business_type %in% c("manufacturer", "reseller")) %>%
  filter(product_category_name_english %in%
           top_10_prod$product_category_name_english) %>%
  group_by(business_type, product_category_name_english) %>%
  summarise(TOTAL_PRICE = sum(price + freight_value)),
  aes(x = product_category_name_english,
                                    y = TOTAL_PRICE, fill = business_type)) +
  geom_bar(stat = "identity", position = "dodge") +
  xlab("Product Name") +
  ylab("Total Price (units)") +
  ggtitle("Total Price by Business Type and Product Name") +
  scale_fill_manual(values = c("#e41a1c", "#377eb8")) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```