group_number_6

Group 6: 2260955, 2215107, 2230472, 2230885, 2285143, 2288495

Contents

1 Introduction

Multiple datasets was usedprovided by Olist, the largest department store in the Brazilian market. It includes customers, orders, etc.. We identified individual entities and attributes in the dataset, rationalised the relationships and cardinalities between the entities and presented them through E-R diagrams. The original datasets were then cleaned and normalised to be used for further analysis. In addition, several SQL queries were created to demonstrate how data could be retrieved, updated and analysed from the database to create workable solutions for unpredictable problems and situations.

2 Data Understanding (A1 & A2)

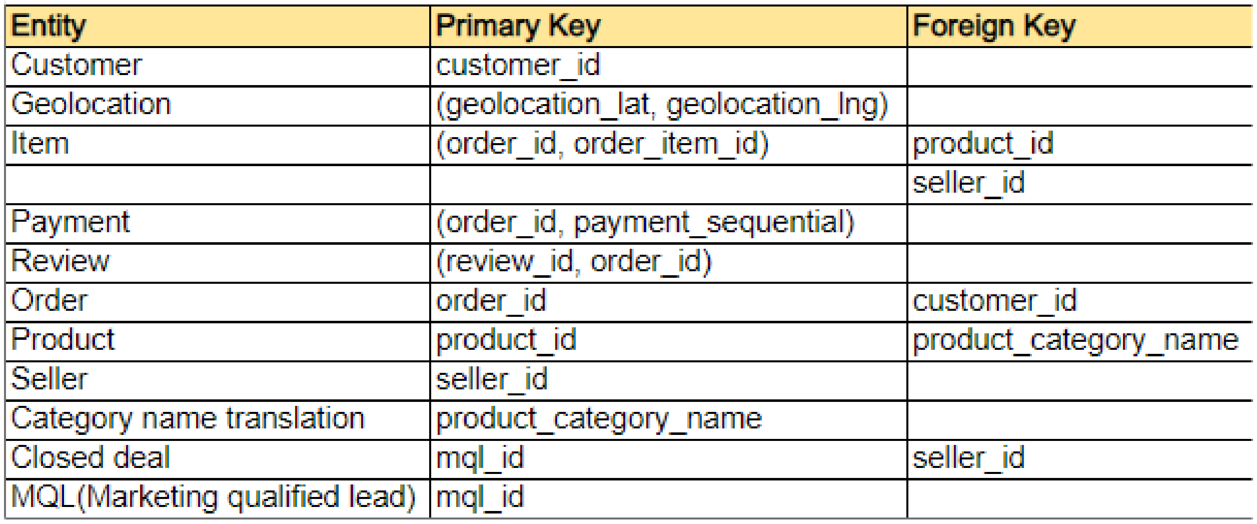
2.1 Identification of Entities, Relationships, Cardinality and Attributes

2.1.1 Assumption For the relationships and cardinalities between entities to be justified, the following assumptions are made.

For the relationships and cardinalities between entities to be justified, the following assumptions are made:

- Actual customer is represented by customer_unique_id.
- All attributes relevant to id's have a length of 32 based on the data provided.

2.1.2 Entity and Attribute Identification



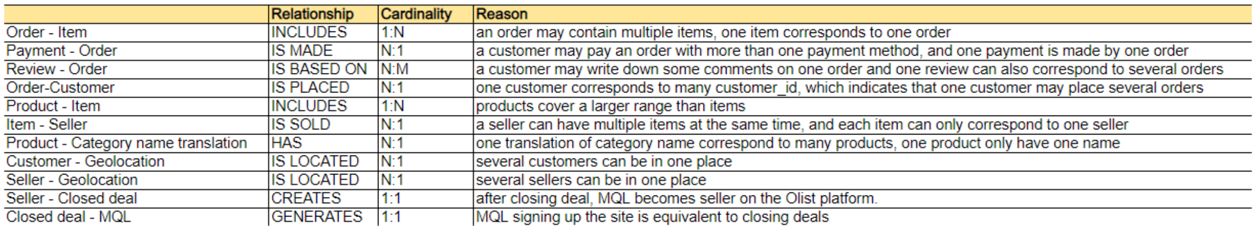
Entity	Primary Key	Foreign Key
Customer	customer_id	
Geolocation	(geolocation_lat, geolocation_lng)	
Item	(order_id, order_item_id)	product_id
		seller_id
Payment	(order_id, payment_sequential)	
Review	(review_id, order_id)	
Order	order_id	customer_id
Product	product_id	product_category_name
Seller	seller_id	
Category name translation	product_category_name	
Closed deal	mql_id	seller_id
MQL(Marketing qualified lead)	mql_id	

Figure 1: Identification of Entity

We have integrated some attributes in several entities to make them more meaningful. The details are as follows:

- review_comment (review_comment_title, review_comment_title)
- customer_geolocation (customer_zip_code_prefix, customer_city, customer_state)
- seller_geolocation (seller_zip_code_prefix, seller_city, seller_state)
- product_size (product_width_cm, product_height_cm, product_length_cm)

2.1.3 Relationship & Cardinality Identification



	Relationship	Cardinality	Reason
Order - Item	INCLUDES	1:N	an order may contain multiple items, one item corresponds to one order
Payment - Order	IS MADE	N:1	a customer may pay an order with more than one payment method, and one payment is made by one order
Review - Order	IS BASED ON	N:M	a customer may write down some comments on one order and one review can also correspond to several orders
Order-Customer	IS PLACED	N:1	one customer corresponds to many customer_id, which indicates that one customer may place several orders
Product - Item	INCLUDES	1:N	products cover a larger range than items
Item - Seller	IS SOLD	N:1	a seller can have multiple items at the same time, and each item can only correspond to one seller
Product - Category name translation	HAS	N:1	one translation of category name correspond to many products, one product only have one name
Customer - Geolocation	IS LOCATED	N:1	several customers can be in one place
Seller - Geolocation	IS LOCATED	N:1	several sellers can be in one place
Seller - Closed deal	CREATES	1:1	after closing deal, MQL becomes seller on the Olist platform.
Closed deal - MQL	GENERATES	1:1	MQL signing up the site is equivalent to closing deals

Figure 2: Identification of Relationship and Cardinality

2.2 E-R Diagram

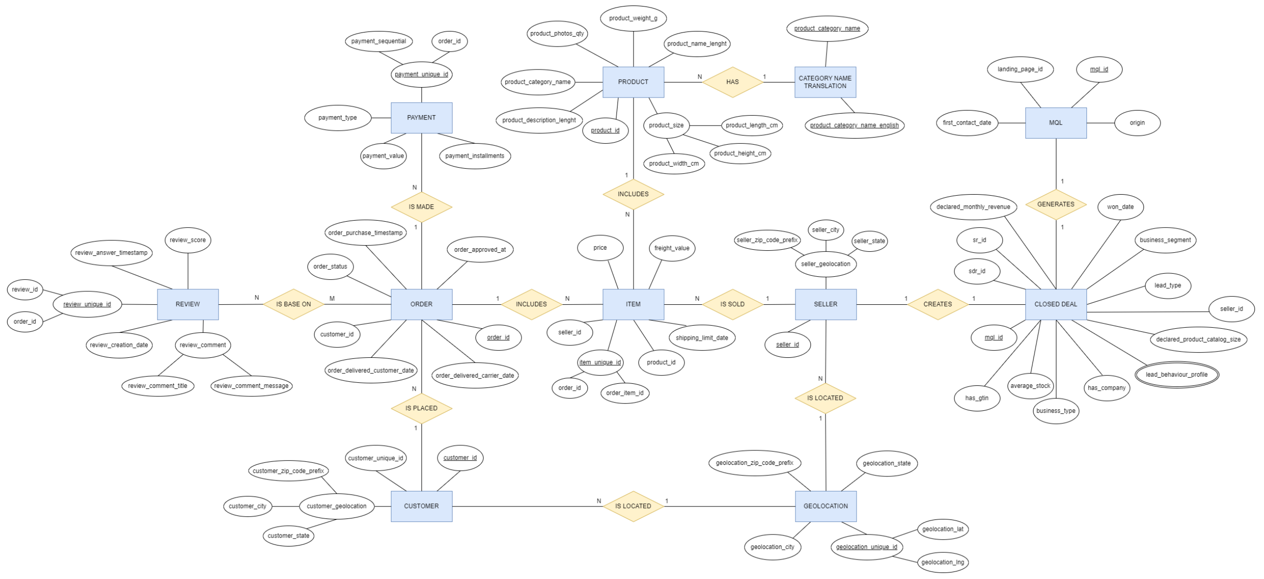


Figure 3: E-R Diagram (Pre-normalisation)

3 Data Preparation and Normalisation (A3, A4 & A5)

3.1 customers

```
-- Customers
CREATE TABLE 'customers' (
  'customer_id' VARCHAR(32) PRIMARY KEY,
  'customer_unique_id' VARCHAR(32) NOT NULL,
  'customer_zip_code_prefix' INT NOT NULL,
  'customer_city' VARCHAR NOT NULL,
  'customer_state' VARCHAR NOT NULL
) ;

dbAppendTable(connection, "customers", customers)
```

3.1.1 Problem(s)

Values in ‘customer_city’ were identified to be inconsistent. For example, there were multiple spellings of the same city. Therefore, to clean the data in the ‘customers’ table, the attributes ‘customer_city’ and ‘customer_state’ can be dropped. This is because, the city and state can be identified through the ‘customer_zip_code_prefix’ as it is already a foreign from ‘geolocation’ table which contains these details.

3.1.2 Normalisation

- Since all attributes can only be explained by a single primary key, there is no partial and transitive FDs and entity is already in 3NF.

3.2 geolocation

```
-- Geolocation
CREATE TABLE 'geolocation' (
  'geolocation_lat' VARCHAR NOT NULL,
  'geolocation_lng' VARCHAR NOT NULL,
  'geolocation_zip_code_prefix' INT NOT NULL,
  'geolocation_city' VARCHAR NOT NULL,
  'geolocation_state' VARCHAR NOT NULL,
  PRIMARY KEY ('geolocation_lat', 'geolocation_lng')
) ;

dbAppendTable(connection, "geolocation", geolocation)
```

3.2.1 Problem(s)

By identifying 'geolocation__lat' and 'geolocation__lng' as the PRIMARY KEY, it was found that it has failed the UNIQUE constraint as there are duplicates for the two attributes. Therefore, it can be suggested to clean the data in 'geolocation' table by firstly selecting unique values from 'geolocation__zip__code__prefix', 'geolocation__lat' and 'geolocation__lng' and deleting the duplicates. Next, the attributes 'geolocation__lat' and 'geolocation__lng' should be deleted and changing the PRIMARY KEY to 'geolocation__zip__code__prefix' instead. This is because, 'geolocation__lat' and 'geolocation__lng' are insignificant in identification of a location for other tables.

3.2.2 Normalisation

- City and state were stored in separate columns, hence, data is already in 1NF
- Once the data has been cleaned, 'geolocation__zip__code__prefix' has become a primary key which can explain all attributes, as such, there is no partial and transitive FDs, and data is already in 3NF.

3.3 order__items

```
-- Order_items
CREATE TABLE 'order_items' (
  'order_id' VARCHAR(32) NOT NULL,
  'order_item_id' INT NOT NULL,
  'product_id' VARCHAR(32) NOT NULL,
  'seller_id' VARCHAR(32) NOT NULL,
  'shipping_limit_date' DATETIME,
  'price' REAL,
  'freight_value' REAL,
  PRIMARY KEY ('order_id', 'order_item_id'),
  FOREIGN KEY ('product_id')
    REFERENCES products ('product_id'),
  FOREIGN KEY ('seller_id')
    REFERENCES sellers ('seller_id')
) ;

dbAppendTable(connection, "order_items", items)
```

```
SELECT
  order_id ,
  COUNT(DISTINCT shipping_limit_date) AS uniq_date,
  COUNT(DISTINCT seller_id) AS uniq_seller,
  COUNT(DISTINCT order_item_id) AS uniq_item
FROM order_items oi
GROUP BY 1
HAVING uniq_date > 1
ORDER BY 2 DESC
```

3.3.1 Normalisation

- Order_items entity is already in 1NF since all values are atomic. We further assessed whether there is any partial or transitive FD in this entity.
- We first looked at potential partial FD in 'shipping_limit_date' with 'order_id', however, there are around 336 records that contains unique date for each combination of 'order_id' and 'order_item_id'. Further investigation shows that this is caused by item shifting from different seller, hence, there is no partial FD here and data is already in 2NF.
- There is also a potential transitive FD in (order_id, order_item_id) -> product_id -> seller_id. This can be confirmed by the fact that relationship on which product sold by which seller, and vice versa, is stored in 'order_id'. This could cause DELETE anomalies where relationship between product and seller will be lost if transactions were removed.
- Hence, we propose to normalise to 3NF by creating a new table "product_seller" to store relationship as mentioned in previous point.

3.4 order_payments

```
-- Order_payments
CREATE TABLE 'order_payments' (
  'order_id' VARCHAR(32) NOT NULL,
  'payment_sequential' INT,
  'payment_type' VARCHAR NOT NULL,
  'payment_installments' INT,
  'payment_value' REAL,
  PRIMARY KEY ('order_id', 'payment_sequential')
) ;

dbAppendTable(connection, "order_payments", payments)
```

3.4.1 Normalisation

- Order_payments entity is already in 1NF since all values are atomic.
- There is a potential INSERT anomaly due to composite primary keys, however, 'order_id' and 'payment_sequential_id' will always exist together as every order in this entity must have at least 1 sequential.
- While the primary key is composite for this entity, all attributes can only be explained by both keys, hence, there is no partial and transitive FDs and entity is already in 3NF.

3.5 order_reviews

```
-- Order_reviews
CREATE TABLE 'order_reviews' (
  'review_id' VARCHAR(32) NOT NULL,
  'order_id' VARCHAR(32) NOT NULL,
  'review_score' INT NOT NULL,
  'review_comment_title' VARCHAR NOT NULL,
  'review_comment_message' VARCHAR NOT NULL,
  'review_creation_date' DATE,
  'review_answer_timestamp' DATETIME,
  PRIMARY KEY ('review_id', 'order_id')
) ;

dbAppendTable(connection, "order_reviews", reviews)
```

```
WITH dup AS(
  SELECT
    review_id,
    COUNT(DISTINCT order_id) AS order_per_review
  FROM order_reviews or2
  GROUP BY 1
)
SELECT
  or2.review_id ,
  or2.order_id,
  c.customer_id,
  c.customer_unique_id ,
  d.order_per_review,
  or2.review_score ,
  or2.review_comment_title ,
  or2.review_comment_message
FROM order_reviews or2
LEFT JOIN dup d
    ON or2.review_id = d.review_id
LEFT JOIN orders o
    ON or2.order_id = o.order_id
LEFT JOIN customers c
    ON o.customer_id = c.customer_id
WHERE order_per_review > 1
ORDER BY 3,2
```

3.5.1 Problem(s)

It was identified that there are missing values in the attributes 'review_comment_title' and 'review_comment_message'. This issue can be solved by altering the schema and allowing NULL values to be inserted for these two attributes. An assumption that can be made for this is that there will be reviews made by customers with only 'review_score'.

3.5.2 Normalisation

- Order_reviews entity is already in 1NF since all values are atomic. While comment and title are made up of multiple words, it cannot be interpreted separately, hence, the value is considered to be atomic.
- With a composite primary key, there is a potential INSERT and UPDATE anomalies caused by partial FDs. A review cannot exist without order, hence, there is no problem caused by INSERT anomalies. Every attribute can be explained by just 'review_id', hence, this can cause UPDATE anomalies where multiple rows must be updated to change contents of the review.
- With the help of SQL, we can identify that 'review_id' depends on a unique customer who can place

multiple orders. At the same time, all orders in the particular review always belong to the same unique customer.

- We proposed to normalise by creating new table “review” to store relationship between ‘review_id’ and other review attributes, while changing the original entity “order_reviews” into “customer_reviews”. This new entity will only store the relationship between ‘review_id’ and ‘customer_unique_id’, which can be obtained via joining ‘order_id’. Once completed, both new entities will be in 3NF since there is no transitive FD.

3.6 orders

```
-- Orders
CREATE TABLE 'orders' (
  'order_id' VARCHAR(32) PRIMARY KEY,
  'customer_id' VARCHAR(32) NOT NULL,
  'order_status' VARCHAR NOT NULL,
  'order_purchase_timestamp' DATETIME NOT NULL,
  'order_approved_at' DATETIME NOT NULL,
  'order_delivered_carrier_date' DATETIME NOT NULL,
  'order_delivered_customer_date' DATETIME NOT NULL,
  'order_estimated_delivery_date' DATE NOT NULL,
  FOREIGN KEY ('customer_id')
    REFERENCES customers ('customer_id')
) ;

dbAppendTable(connection, "orders", orders)
```

3.6.1 Problem(s)

Missing values were identified for the attributes ‘order_approved_at’, ‘order_delivered_carrier_date’ and ‘order_delivered_customer_date’. This can also be solved by altering the schema and allowing NULL values. This is because, the values in these three attributes are dependent on the value of ‘order_status’.

3.6.2 Normalisation

- Order entity is already in 1NF since all values are atomic. We further assessed whether there is any partial or transitive FD in this entity.
- Since all attributes can only be explained by a single primary key, there is no partial and transitive FDs and entity is already in 3NF.

3.7 products

```
-- Products
CREATE TABLE 'products' (
  'product_id' VARCHAR(32) PRIMARY KEY,
  'product_category_name' VARCHAR NOT NULL,
  'product_name_lenght' INT NOT NULL,
  'product_description_lenght' INT NOT NULL,
  'product_photos_qty' INT NOT NULL,
  'product_weight_g' INT NOT NULL,
  'product_length_cm' INT NOT NULL,
  'product_height_cm' INT NOT NULL,
  'product_width_cm' INT NOT NULL,
  FOREIGN KEY ('product_category_name')
    REFERENCES product_category_name_translation ('product_category_name')
) ;

dbAppendTable(connection, "products", products)
```

3.7.1 Problem(s)

The attributes 'product_category_name', 'product_name_length' 'product_description_length', 'product_photos_qty', 'product_weight_g', 'product_length_cm', 'product_height_cm' and 'product_width_cm' contains missing values. There should not be any missing values for these attributes. Therefore, it can be treated as data entry errors since there are minimal records affected.

3.7.2 Normalisation

- Dimension, which is a composite attribute, has been stored in 3 separate columns. Hence, product entity is already in 1NF because all values are atomic
- Since all attributes can only be explained by a single primary key, there is no partial and transitive FDs and entity is already in 3NF.

3.8 sellers

```
-- Sellers
CREATE TABLE 'sellers' (
  'seller_id' VARCHAR(32) PRIMARY KEY,
  'seller_zip_code_prefix' INT,
  'seller_city' VARCHAR NOT NULL,
  'seller_state' VARCHAR NOT NULL
) ;

dbAppendTable(connection, "sellers", sellers)
```

3.8.1 Problem(s)

Values in 'seller_city' were also identified to be inconsistent. There were multiple spellings of the same city similarly to 'customers' table. The attributes 'seller_city' and 'seller_state' can be dropped as the city and state can alsp be identified through the 'seller_zip_code_prefix' which is a foreign from 'geolocation' table that contains these details.

3.8.2 Normalisation

- Since all attributes can only be explained by a single primary key, there is no partial and transitive FDs and entity is already in 3NF.

3.9 product_category_name_translation

```
-- Product_category_name_translation
CREATE TABLE 'product_category_name_translation' (
  'product_category_name' VARCHAR PRIMARY KEY,
  'product_category_name_english' VARCHAR NOT NULL
) ;
```

```
dbAppendTable(connection, "product_category_name_translation", category)
```

3.9.1 Normalisation

- Data only contains 1 attribute and a single primary key, as such, data is already in 3NF.

3.10 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' DATETIME,
  'business_segment' VARCHAR NOT NULL,
  'lead_type' VARCHAR NOT NULL,
  'lead_behaviour_profile' VARCHAR NOT NULL,
  'has_company' VARCHAR NOT NULL,
  'has_gtin' VARCHAR NOT NULL,
  'average_stock' VARCHAR NOT NULL,
  'business_type' VARCHAR NOT NULL,
  'declared_product_catalog_size' INT NOT NULL,
  'declared_monthly_revenue' REAL,
  FOREIGN KEY ('seller_id')
    REFERENCES sellers ('seller_id')
) ;

dbAppendTable(connection, "closed_deals", closed_deals)
```

3.10.1 Problem(s)

There are also missing values identified in the attributes 'business_segment', 'lead_type', 'lead_behaviour_profile', 'has_company', 'has_gtin', 'average_stock', 'business_type' and 'declared_product_catalog_size'. It can be solved by altering the schema and allowing NULL values. This is because, these values in these attributes are only identified upon contact with the seller.

Another issue identified is the values in 'average_stock' contains multiple data types. However, with the small number of records that was affected, it is treated as data entry error.

3.10.2 Normalisation

- 'lead_behavior_profile' is a multivalue attribute which describes up to 2 behaviours. As such, the data us still not in 1NF. We propose to split into 2 attributes 'lead_behavior_profile_1' and 'lead_behavior_profile_2' to store them separately. The business team could also utilise it by storing dominant behaviour in 1 and the other in 2.
- Since all attributes can only be explained by a single primary key, there is no partial and transitive FDs and entity is already in 3NF.

3.11 marketing_qualified_leads

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

dbAppendTable(connection, "marketing_qualified_leads", mql)
```

3.11.1 Problem(s)

Values in ‘origin’ contains missing values when there should not be. This is because, where the lead was acquired should be known when ‘landing_page_id’ is present. Therefore, this can be treated as data entry error with the minimal records affected.

3.11.2 Normalisation

- Since all attributes can only be explained by a single primary key, there is no partial and transitive FDs and entity is already in 3NF.

3.12 E-R Diagram (Post-normalisation)

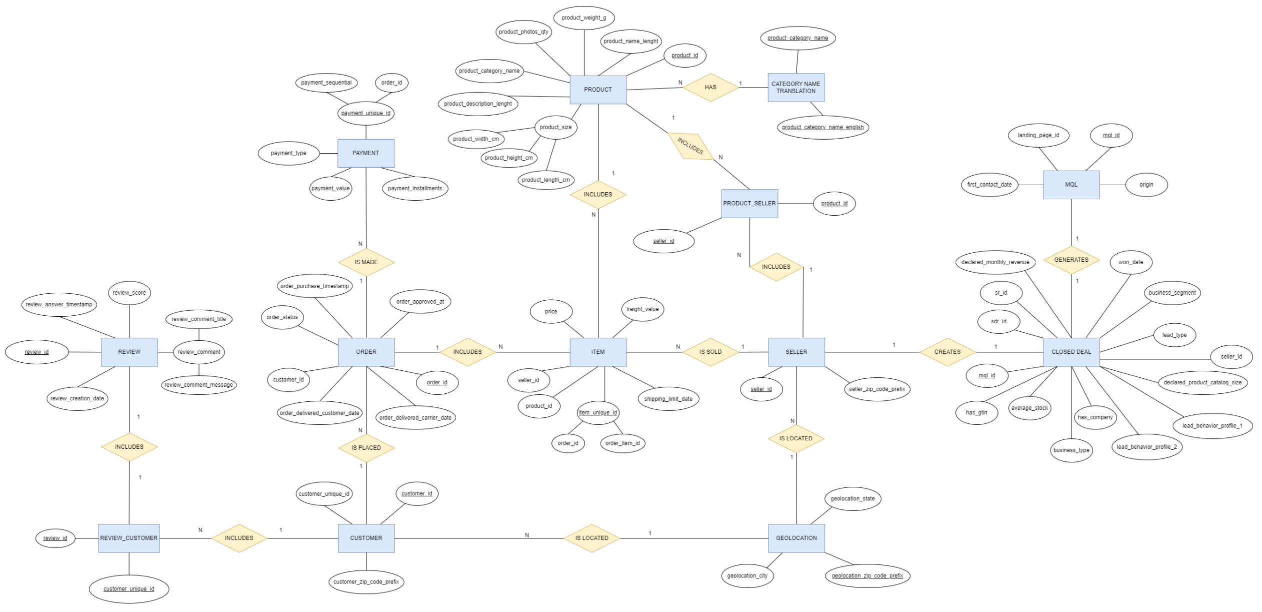


Figure 4: E-R Diagram (Post-normalisation)

4 Data Exploration (A6 & A7)

4.1 SQL Queries

The query is designed to show the performance of each product category by year from 2016 - 2018. It is also designed to order by year then Gross Merchandise Value (GMV) from the highest to lowest.

```
--Query 1: GMV value of each product category by year
SELECT
    pcnt .product_category_name_english ,
    strftime('%Y', DATE(o.order_purchase_timestamp , 'unixepoch')) AS purchase_year,
    COUNT(DISTINCT o.order_id) AS number_of_order,
    ROUND(SUM(oi.price + oi.freight_value),2) AS gross_merchandise_value,
    ROUND(AVG(oi.price),2) AS avg_price
FROM orders o
LEFT JOIN order_items oi
    ON o.order_id = oi.order_id
LEFT JOIN products p
    ON oi.product_id = p.product_id
LEFT JOIN product_category_name_translation pcnt
    ON p.product_category_name = pcnt.product_category_name
WHERE o.order_status = "delivered"
GROUP BY pcnt .product_category_name_english, purchase_year
ORDER BY purchase_year, gross_merchandise_value DESC
```

```
query1 <- orders %>%
  filter(order_status == "delivered") %>%
  left_join(order_items, by = c("order_id" = "order_id"),multiple="all") %>%
  left_join(products, by = c("product_id" = "product_id")) %>%
  left_join(product_category_name_translation, by = c("product_category_name" =
  ↪  "product_category_name")) %>%
  select(product_category_name_english, order_id, order_purchase_timestamp, price,
  ↪  freight_value) %>%
  mutate(purchase_year = format(as.POSIXct(order_purchase_timestamp, origin =
  ↪  "1970-01-01"), "%Y")) %>%
  group_by(product_category_name_english, purchase_year) %>%
  summarize(number_of_order = n_distinct(order_id),
            gross_merchandise_value = round(sum(price + freight_value), 2),
            avg_price = round(mean(price), 2)) %>%
  arrange(purchase_year, desc(gross_merchandise_value))
```

The query is designed to identify if there is any product category that is prone to be cancelled.

```
--Query 2: Numbers of canceled order for each product categories
SELECT
    pcnt.product_category_name_english ,
    CASE WHEN o.order_status = 'canceled'THEN 'canceled' ELSE 'not cancel' END AS
    ↪  status_group,
    COUNT(DISTINCT oi.order_id) AS number_order
FROM order_items oi
LEFT JOIN orders o
    ON oi.order_id = o.order_id
LEFT JOIN products p
    ON oi.product_id = p.product_id
LEFT JOIN product_category_name_translation pcnt
    ON p.product_category_name = pcnt.product_category_name
GROUP BY p.product_category_name,status_group
ORDER BY p.product_category_name,status_group
```

```
query2 <- orders %>%
  left_join(order_items, by = "order_id",multiple="all") %>%
  left_join(products, by = "product_id") %>%
  left_join(product_category_name_translation, by = "product_category_name") %>%
  group_by(product_category_name_english, status_group = ifelse(order_status ==
  ↪  "canceled", "canceled", "not cancel")) %>%
  summarize(number_order = n_distinct(order_id)) %>%
  arrange(product_category_name_english, status_group)
```

The query is designed to explore how each state performs in term of estimate deliver date vs. actual deliver date. To have a valid comparison, only 'delivered' order is filtered.

```
--Query 3: Average deliverday days of each state
SELECT
    g.geolocation_state,
    ROUND(AVG(JULIANDAY(DATE(o.order_estimated_delivery_date , 'unixepoch')) -
    ↪  JULIANDAY(DATE(o.order_purchase_timestamp , 'unixepoch'))),0) AS
    ↪  avg_estimate_delivery_days,
    ROUND(AVG(JULIANDAY(DATE(o.order_delivered_customer_date , 'unixepoch')) -
    ↪  JULIANDAY(DATE(o.order_purchase_timestamp , 'unixepoch'))),0) AS
    ↪  avg_actual_delivery_days,
    ROUND(AVG(JULIANDAY(DATE(o.order_delivered_customer_date , 'unixepoch')) -
    ↪  JULIANDAY(DATE(o.order_estimated_delivery_date , 'unixepoch'))),0) AS days_diff
FROM orders o
LEFT JOIN customers c
    ON o.customer_id = c.customer_id
LEFT JOIN geolocation g
    ON c.customer_zip_code_prefix = g.geolocation_zip_code_prefix
WHERE o.order_status = 'delivered'
GROUP BY g.geolocation_state
ORDER BY days_diff ASC
```

```
query3 <- orders %>%
  filter(order_status == "delivered") %>%
  left_join(customers, by = "customer_id") %>%
  left_join(geolocation, by = c("customer_zip_code_prefix" =
  ↪  "geolocation_zip_code_prefix"),multiple="all") %>%
  group_by(geolocation_state) %>%
  summarize(avg_estimate_delivery_days =
  ↪  round(mean(as.numeric(as.Date(order_estimated_delivery_date, origin =
  ↪  "1970-01-01")) - as.numeric(as.Date(order_purchase_timestamp, origin =
  ↪  "1970-01-01"))), 0),
            avg_actual_delivery_days =
            ↪  round(mean(as.numeric(as.Date(order_delivered_customer_date, origin =
            ↪  "1970-01-01")) - as.numeric(as.Date(order_purchase_timestamp, origin =
            ↪  "1970-01-01"))), 0),
            days_diff = round(mean(as.numeric(as.Date(order_delivered_customer_date,
            ↪  origin = "1970-01-01")) -
            ↪  as.numeric(as.Date(order_estimated_delivery_date, origin =
            ↪  "1970-01-01"))), 0)) %>%
  arrange(days_diff)
```

The query is designed to see Olist's performance in each state with different metrics such as order quantity, value and average value. To have a valid comparison, only 'delivered' order is filtered.

```sql
--Query 4: Performance of each state in term of customer
SELECT
    g.geolocation_state ,
    COUNT(DISTINCT o.order_id) AS number_order,
    SUM(oi.price+oi.freight_value) AS order_value,
    SUM(oi.price+oi.freight_value)/COUNT(DISTINCT o.order_id) AS avg_order_value
FROM orders o
LEFT JOIN order_items oi
    ON o.order_id = oi.order_id
LEFT JOIN customers c
    ON o.customer_id = c.customer_id
LEFT JOIN geolocation g
    ON c.customer_zip_code_prefix = g.geolocation_zip_code_prefix
WHERE o.order_status = "delivered"
GROUP BY g.geolocation_state
ORDER BY order_value DESC
```

```r
query4<- orders %>%
  filter(order_status == "delivered") %>%
  left_join(order_items, by = "order_id",multiple="all") %>%
  left_join(customers, by = "customer_id") %>%
  left_join(geolocation, by = c("customer_zip_code_prefix" =
  ↪  "geolocation_zip_code_prefix"),multiple="all") %>%
  group_by(geolocation_state) %>%
  summarize(number_order = n_distinct(order_id),
            order_value = sum(price + freight_value),
            avg_order_value = order_value/number_order) %>%
  arrange(desc(order_value))
```

The query is designed to see if there is any difference between product category in term of average delivery days and freight value.

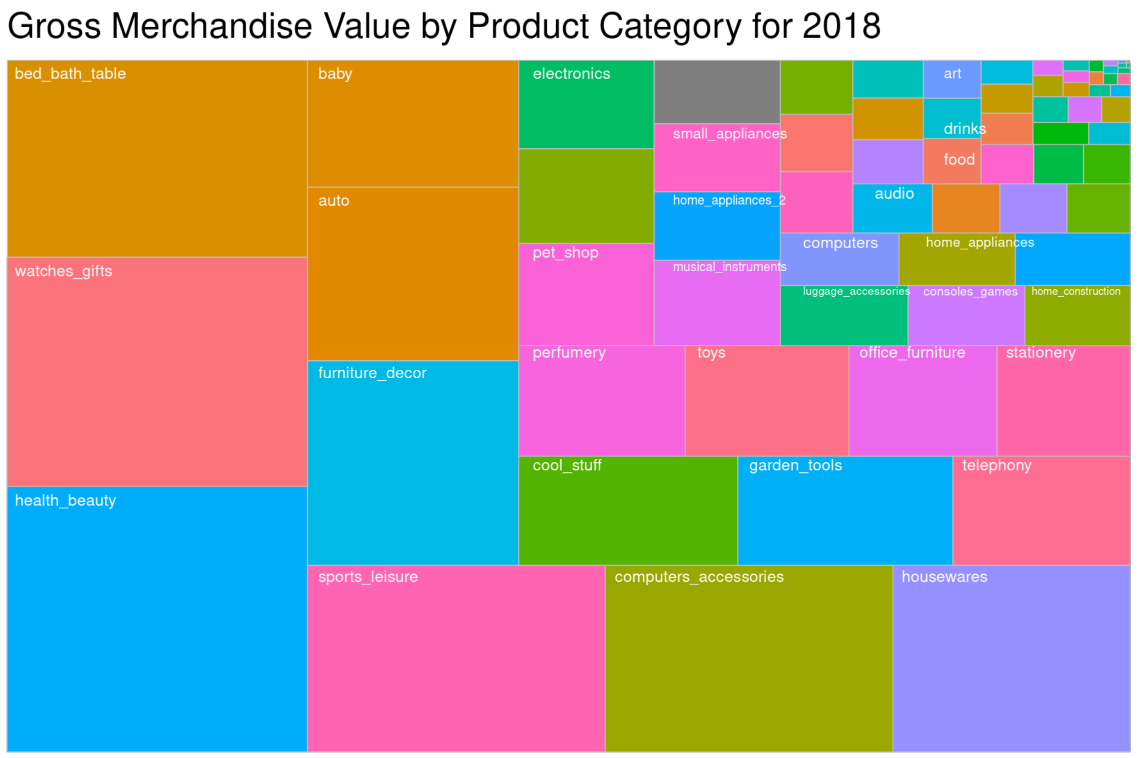
```sql
--Query 5: Average delivery days and freight cost for each product categories
SELECT
    pcnt.product_category_name_english ,
    ROUND(AVG(JULIANDAY(DATE(o.order_delivered_customer_date , 'unixepoch')) -
    ↪  JULIANDAY(DATE(o.order_purchase_timestamp , 'unixepoch'))),0) AS
    ↪  avg_actual_delivery_days,
    ROUND(AVG(oi.freight_value),2) AS avg_freight_cost
FROM orders o
LEFT JOIN order_items oi
    ON o.order_id = oi.order_id
LEFT JOIN products p
    ON oi.product_id = p.product_id
LEFT JOIN product_category_name_translation pcnt
    ON p.product_category_name = pcnt.product_category_name
WHERE o.order_status = "delivered"
GROUP BY pcnt.product_category_name_english
```

```r
query5<- orders %>%
  filter(order_status == "delivered") %>%
  left_join(order_items, by = "order_id",multiple="all") %>%
  left_join(products, by = "product_id") %>%
  left_join(product_category_name_translation, by = "product_category_name") %>%
  group_by(product_category_name_english) %>%
  summarize(avg_actual_delivery_days =
  ↪  round(mean(as.numeric(as.Date(order_delivered_customer_date, origin =
  ↪  "1970-01-01")) - as.numeric(as.Date(order_purchase_timestamp, origin =
  ↪  "1970-01-01"))), 0),
            avg_freight_cost = round(mean(freight_value), 2)) %>%
  arrange(avg_actual_delivery_days)
```

4.2 Visualisation

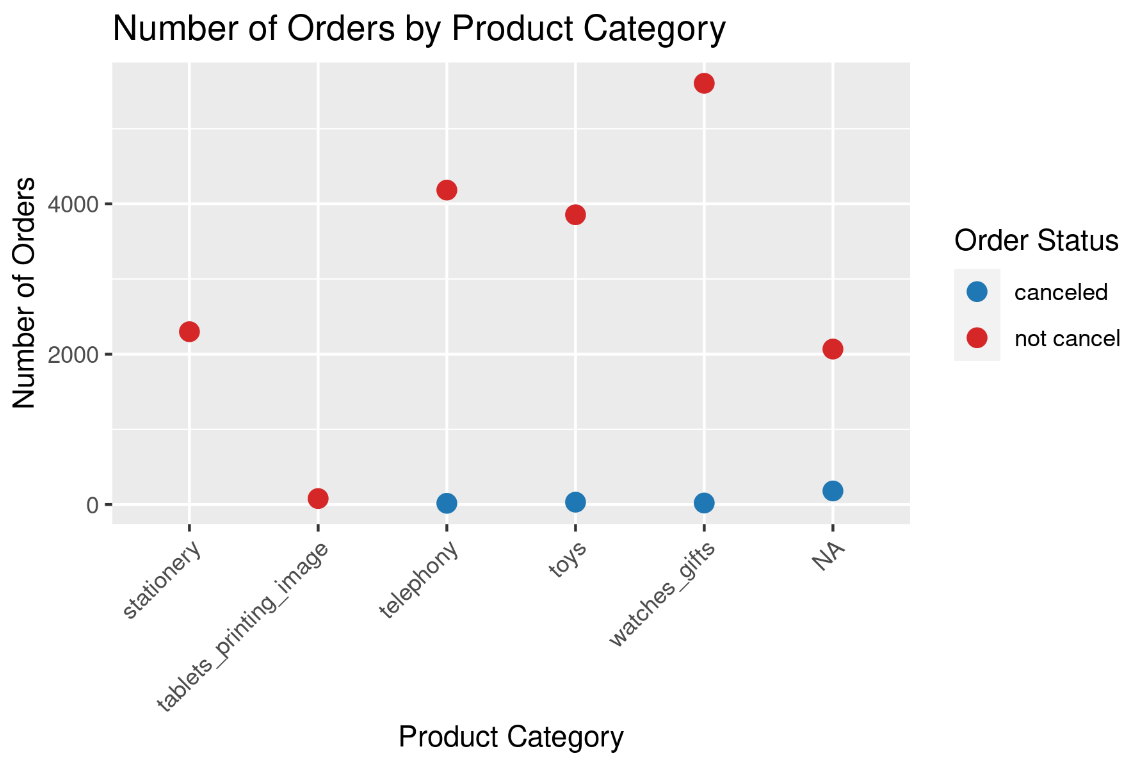
```
query1_2018 <- query1 %>% filter(purchase_year == "2018")

tree <- ggplot(query1_2018, aes(area = gross_merchandise_value, fill =
↪  product_category_name_english, label = product_category_name_english)) +
  geom_treemap() +
  labs(title = "Gross Merchandise Value by Product Category for 2018 ",
       fill = "Product Category") +
  theme_minimal() +
  guides(fill = FALSE) +
  geom_treemap_text(size = 6, color = "white")
```

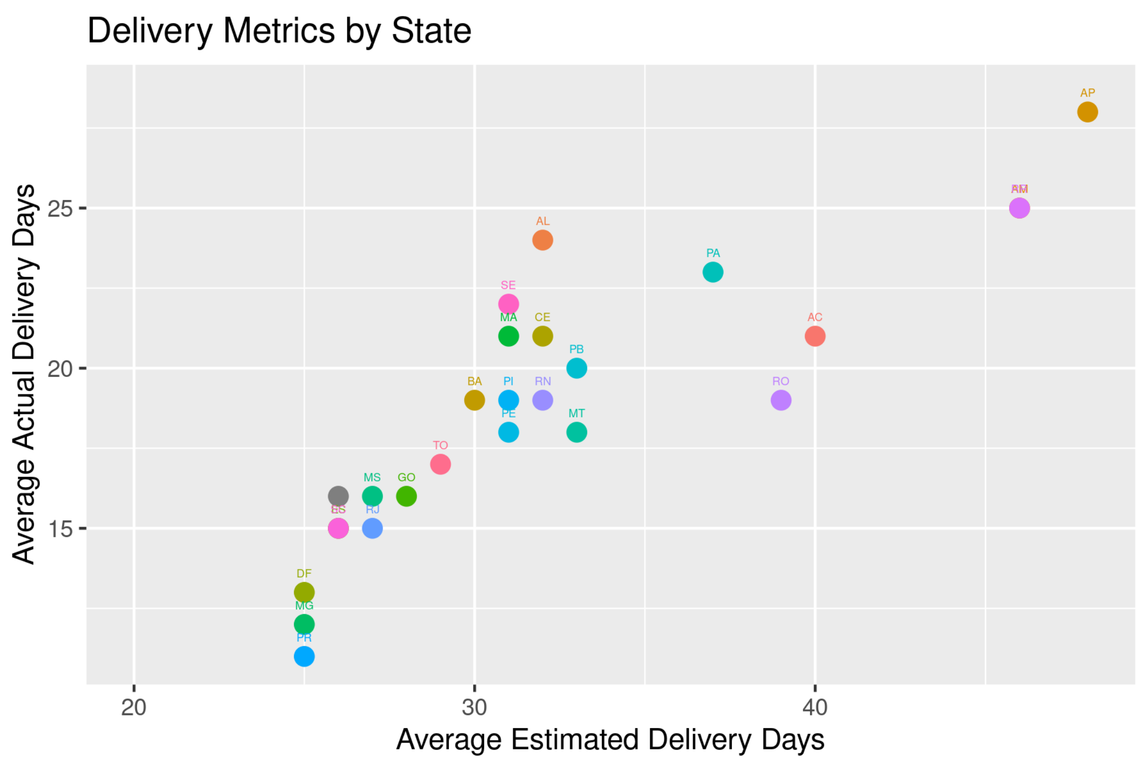


```
plot2<-ggplot(query2[(nrow(query2)-9):nrow(query2), ], aes(x =
↪  product_category_name_english, y = number_order, color = status_group)) +
  geom_point(size = 3) +
  scale_color_manual(values = c("#1f77b4", "#d62728")) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  labs(title = "Number of Orders by Product Category", x = "Product Category", y =
  ↪  "Number of Orders", color = "Order Status")
```

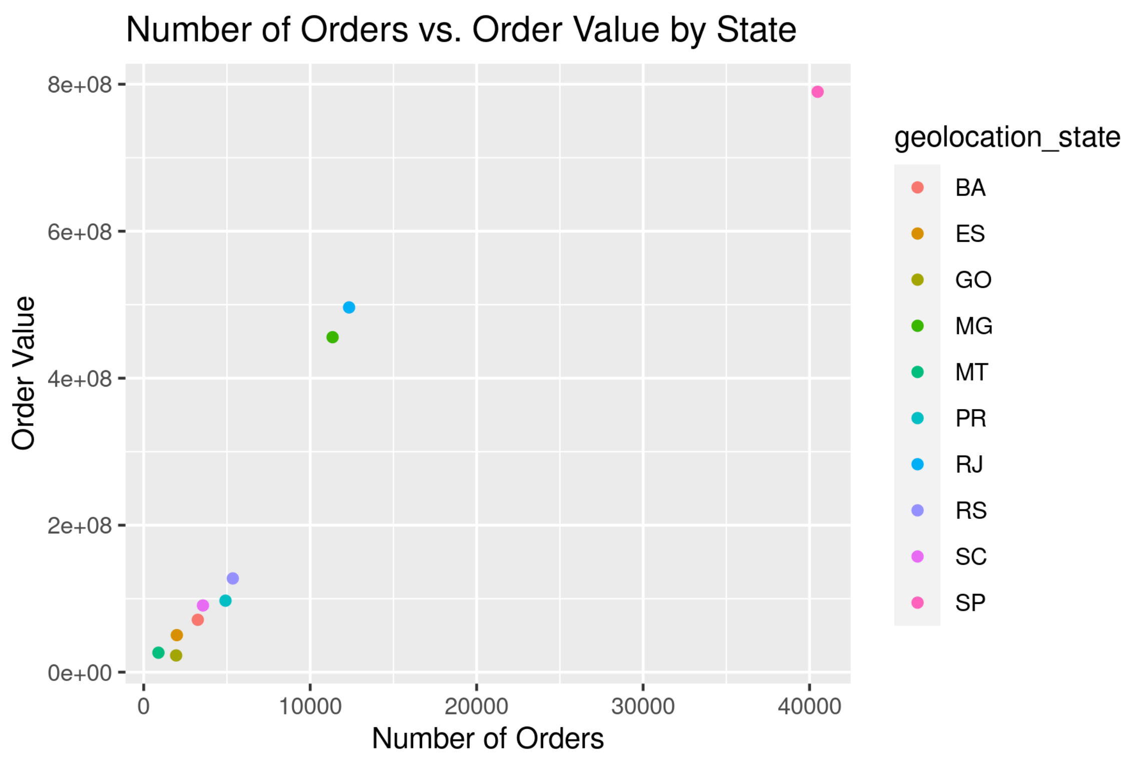
```r
ggsave(file = "plot_2.png", plot = plot2, width = 6, height = 4, dpi = 300)
```



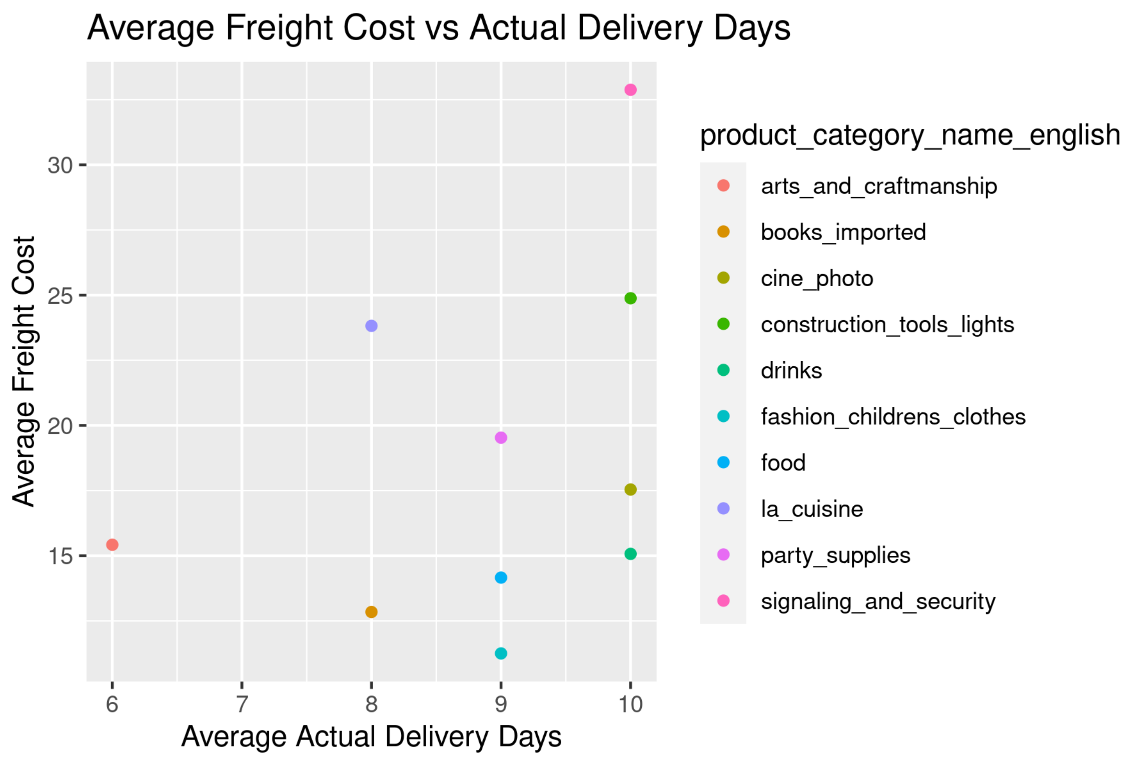
```r
plot3<-ggplot(query3, aes(x = avg_estimate_delivery_days, y = avg_actual_delivery_days,
↪  color = geolocation_state)) +
  geom_point(size = 3) +geom_text(aes(label = geolocation_state), nudge_y = 0.6, size =
  ↪  1.5)+
  labs(title = "Delivery Metrics by State", x = "Average Estimated Delivery Days", y =
  ↪  "Average Actual Delivery Days", color = "State")+theme(legend.position = "none")
ggsave(file = "plot_3.png", plot = plot3, width = 6, height = 4, dpi = 300)
```



```r
plot4<-ggplot(query4[1:10, ], aes(x = number_order, y =order_value,colour=geolocation_state)) +geom_point() +labs(title = "Number of Orders vs. Order Value by State", x = "Number of Orders", y = "Order Value")
 ggsave(file = "plot_4.png", plot = plot4, width = 6, height = 4, dpi = 300)
```



```
plot5<-ggplot(query5[1:10, ], aes(x = avg_actual_delivery_days, y =
↪  avg_freight_cost,colour=product_category_name_english)) +
  geom_point() +
  labs(title = "Average Freight Cost vs Actual Delivery Days",
       x = "Average Actual Delivery Days",
       y = "Average Freight Cost")
```


Average Freight Cost vs Actual Delivery Days
Average Freight Cost
Average Actual Delivery Days
product_category_name_english
arts_and_craftmanship
books_imported
cine_photo
construction_tools_lights
drinks
fashion_childrens_clothes
food
la_cuisine
party_supplies
signaling_and_security