Data Management Assignment Report

Table of contents

Introduction

There are two common workflows for data integration: ELT (Extract, Load, Transform) and ETL (Extract, Transform, Load)

In ELT process, normalization and data validation are performed after the data is ingested into the database whereas in the ETL process both transformations are done after data ingestion. In our work, we chose ETL (Extract, Transform, Load) instead of ELT workflow and used SQL and R interchangeably to perform transformations.

Part 1: Database Design and Implementation

Task 1.1: E-R Diagram

Entities

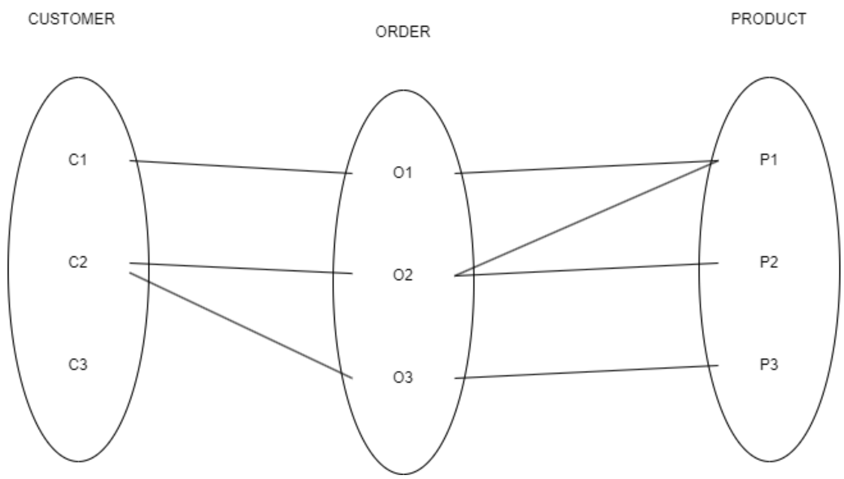
Our E-R Diagram for an E-commerce database consisted of five main entities:

1. CUSTOMER: Details of buyers, including attributes such as customer ID, customer name, contact information, and gender.
2. PRODUCT: Details of sale items, including product ID, product name, description, rating, price, and available stock.
3. ADVERTISEMENT: Details of promotional activities used to advertise products, including advertisement ID, number of times the advertisement is shown, cost of the advertisement, and advertisement placement.
4. SUPPLIER: Details of product providers including supplier ID, supplier name, and contact information.
5. CATEGORY: Represents product classifications, including category ID, name, and fee percentage.

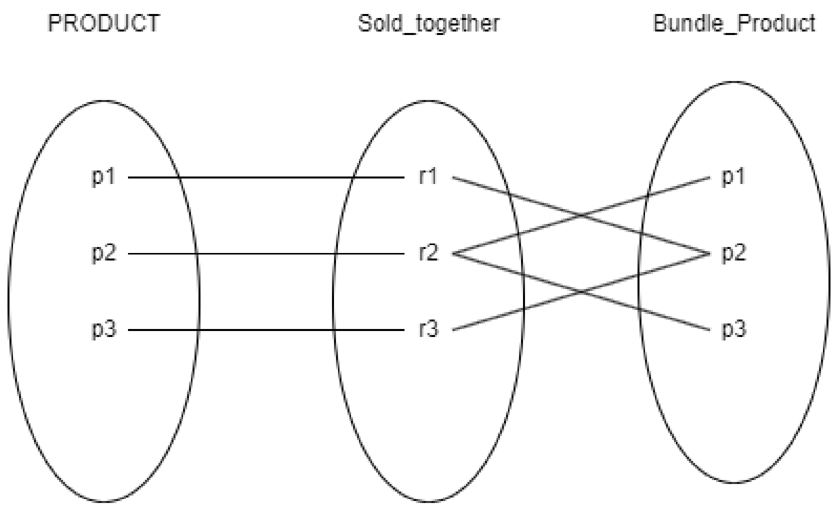
Relationships

The relationships among these entities are as follows:

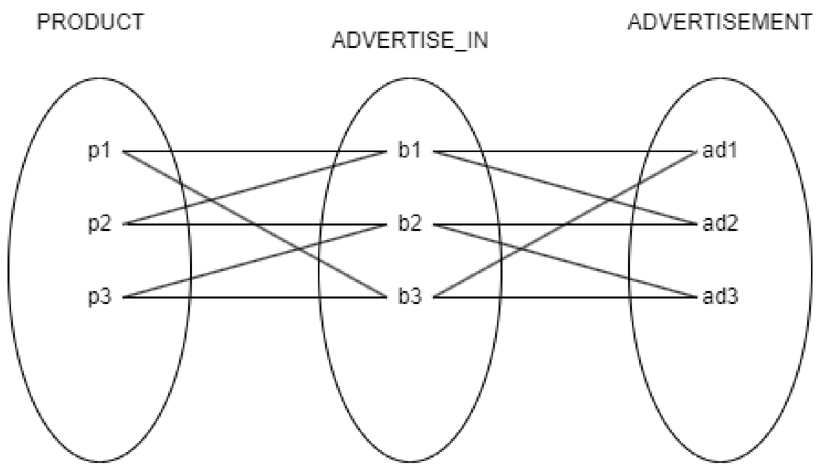
- CUSTOMER and PRODUCT entities have an M-N relationship where one customer can order many products, and one product can be ordered by many customers. This relationship is represented as ORDER, which contains the details of each order, including order ID, order date, order status, order quantity, promo code, payment method, and delivery fee.



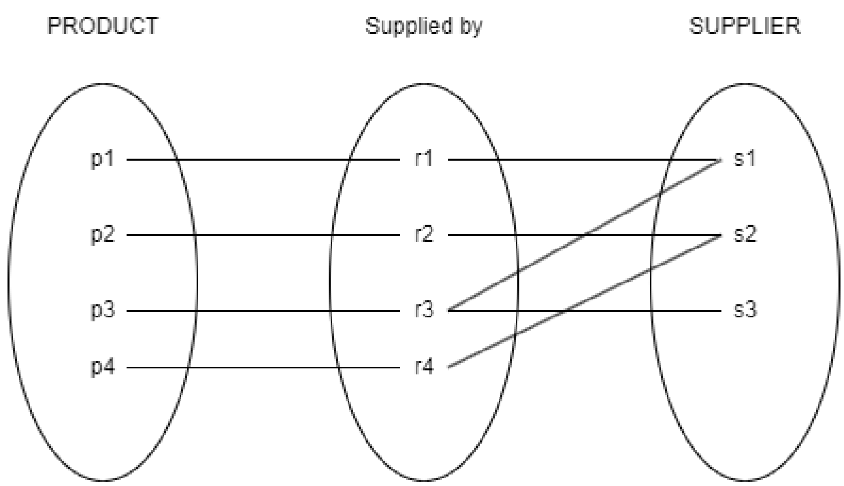
- PRODUCT entity has a self-referencing relationship Sold_Together of 1 to many. The relationship indicates that one product can be sold together with others as a bundle.



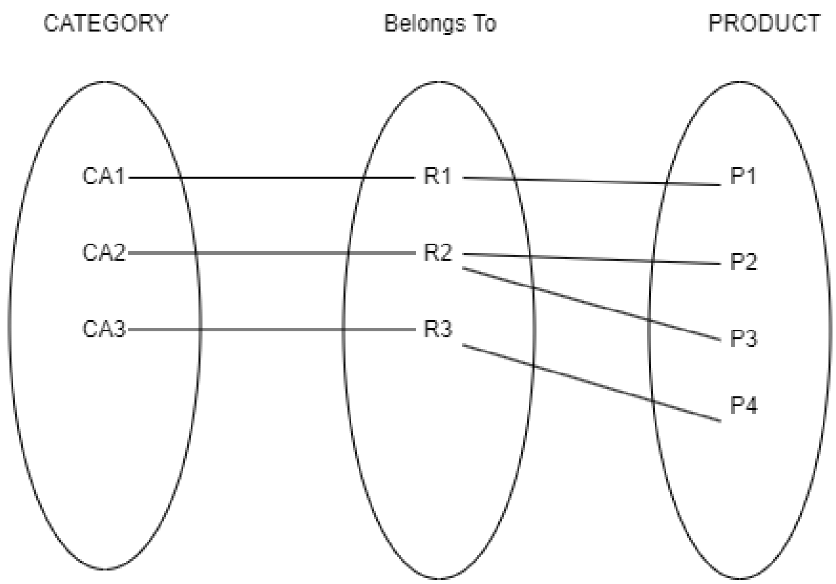
- PRODUCT and ADVERTISEMENT have an M-N relationship. ADVERTISE_IN represents this relationship where multiple advertisements can promote one product, and one advertisement can promote multiple products.



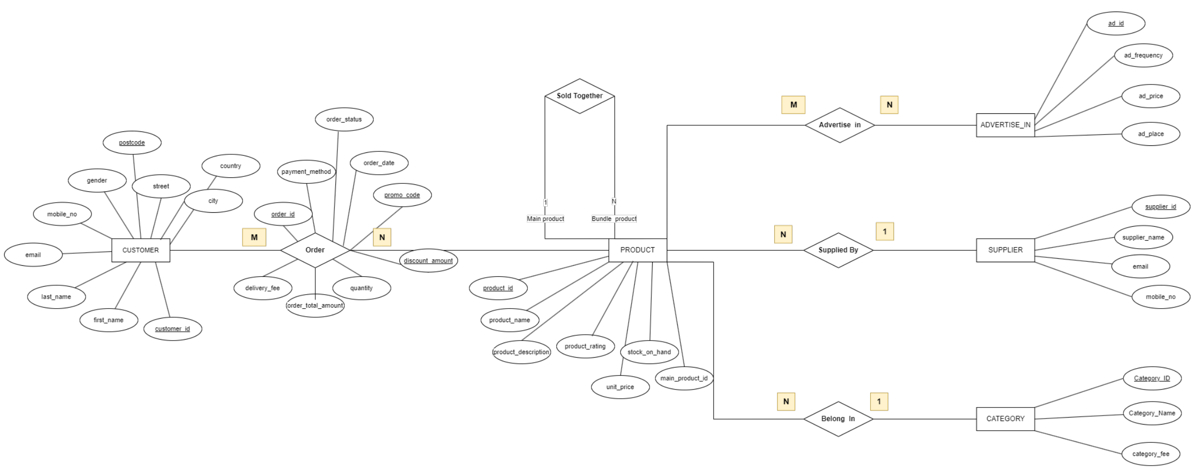
- PRODUCT and SUPPLIER have an N-1 relationship, indicating that one product can be supplied by only one supplier, but one supplier can provide multiple products.



- CATEGORY and PRODUCT have 1-N relationship, where one product belongs to one category, but one category can contain multiple products.



As a result, this is our E-R diagram.



Task 1.2: SQL Database Schema Creation

Normalization:

Our initial ER Diagram required two updates for 3NF compliance. The first adjustment was on the Order table, which initially did not adhere to 3NF.

Functional Dependency:

- {order_id, customer_id, product_id } -> { order_date , order_status, order_quantity, promo_code, discount amount, payment_method, delivery_fee}

- {promo_code} -> { discount_amount}

Assuming order_id duplication for simultaneous customer orders led to UPDATE anomalies. If there is an update on any attributes, such as payment_methods, we must alter more than one row in case there is more than one product in that order_id. Additonally, discount_amount is also transitively dependent on promo_code.

Therefore, we divided the ORDER table into three entities:

1. ORDER_DETAIL holds individual order details with attributes including product_id (primary key), customer_id and promo_code (foreign keys), order_date, order_status, and payment_method.

2. ORDER_ITEM contains product IDs and the quantities per order using order_id and product_id as foreign keys and a composite primary key, and quantity.

3. DISCOUNT stores promotional discount data, with promo_code as the primary key and discount_amount.

The next adjustment was on the CUSTOMER table, which was initially on 2NF. Based on its function dependency, street, city, and country can also be determined by postcode and customer_id.

Functional Dependency:

- {customer_id} ->{ first_name, last_name, gender, email, customer_mobile , street , city , country , postcode }
- {postcode} -> { street, city , country }

This indicated that street, city, and country were transitively dependent on postcode. As a result, a new table is created as ADDRESS, which has postcode as a primary key and stores street, city, and country as other attributes.

In summary, four new entities were created from the normalization, which were ORDER_DETAIL, ORDER_ITEM, DISCOUNT, and ADDRESS. This resulted in a total of ten entities in our schema.

Logical Schema

According to the E-R diagram and normalization s, we can list the logical schema as follows:

- ADDRESS(address_id, postcode, street, city, country)
- DISCOUNT(promo_code, discount_amount)
- ADVERTISEMENT(ad_id, ad_frequency, ad_place, ad_price)
- SUPPLIER(supplier_id, supplier_name, email, mobile_no)
- CATEGORY(category_id, category_name, category_fee)
- CUSTOMER(customer_id, first_name, last_name, gender, email, mobile_no, address_id)
- PRODUCT(product_id, product_name, product_description, product_rating, unit_price, stock_at_hand, main_product_id, category_id, supplier_id)
- ORDER_DETAIL(order_id, customer_id, order_date, order_status, promo_code, payment_method, delivery_fee)
- ORDER_ITEM(order_id, product_id, quantity)
- ADVERTISE_IN(product_id, ad_id)

Physical Schema

Firstly, we created a connection to our database named "database.db"

```r
connect <- dbConnect(RSQLite::SQLite(), "database.db")
```

Then, we first created parent entities, including ADDRESS, DISCOUNT, ADVERTISEMENT, SUPPLIER, and CATEGORY.

1. Create CUSTOMER entity

```sql
CREATE TABLE IF NOT EXISTS CUSTOMER (
  customer_id VARCHAR(50) PRIMARY KEY,
  first_name VARCHAR(50) NOT NULL,
  last_name VARCHAR(50) NOT NULL,
  gender VARCHAR(10),
  customer_email VARCHAR(50) NOT NULL UNIQUE,
  customer_mobile VARCHAR(15) NOT NULL UNIQUE,
  address_id VARCHAR(50) NOT NULL,
  FOREIGN KEY (address_id) REFERENCES ADDRESS (address_id)
);
```

2. Create DISCOUNT entity

```sql
CREATE TABLE IF NOT EXISTS DISCOUNT (
  promo_code VARCHAR(20) PRIMARY KEY,
  discount_percent INT NOT NULL
);
```

3. Create ADVERTISEMENT entity

```
CREATE TABLE IF NOT EXISTS ADVERTISEMENT (
  ad_id VARCHAR(50) PRIMARY KEY,
  ad_frequency INT NOT NULL,
  ad_place VARCHAR(50) NOT NULL,
  ad_price DECIMAL(10, 2) NOT NULL
);
```

4. Create SUPPLIER entity

```
CREATE TABLE IF NOT EXISTS SUPPLIER (
  supplier_id VARCHAR(50) PRIMARY KEY,
  supplier_name VARCHAR(50) NOT NULL,
  supplier_email VARCHAR(50) NOT NULL UNIQUE,
  supplier_mobile VARCHAR(20) NOT NULL UNIQUE
);
```

5. Create CATEGORY entity

```
CREATE TABLE IF NOT EXISTS CATEGORY (
  category_id VARCHAR(50) PRIMARY KEY,
  category_name VARCHAR(50) NOT NULL,
  category_fee INT NOT NULL
);
```

Then, we created entities that are children entities and entities that have referential integrity, which consist of CUSTOMER, PRODUCT, ORDER_DETAIL, ORDER_ITEM, and ADVERTISE_IN

6. Create ADDRESS entity

```
CREATE TABLE IF NOT EXISTS ADDRESS (
  address_id VARCHAR(50) PRIMARY KEY,
  postcode VARCHAR(20) NOT NULL,
  street VARCHAR(50) NOT NULL,
  city VARCHAR(100) NOT NULL,
  country VARCHAR(100) NOT NULL
  );
```

7. Create PRODUCT entity

```
CREATE TABLE IF NOT EXISTS PRODUCT (
  product_id VARCHAR(50) PRIMARY KEY,
  product_name VARCHAR(50) NOT NULL,
  product_description VARCHAR(50),
```

```sql
  product_rating DECIMAL(5,2),
  unit_price DECIMAL(10,2) NOT NULL,
  stock_on_hand INT NOT NULL,
  main_product_id VARCHAR(50),
  category_id VARCHAR(50) NOT NULL,
  supplier_id VARCHAR(50) NOT NULL,
  FOREIGN KEY (supplier_id) REFERENCES SUPPLIER (supplier_id),
  FOREIGN KEY (category_id) REFERENCES CATEGORY (category_id)
);
```

8. Create ORDER_DETAIL entity

```sql
CREATE TABLE IF NOT EXISTS ORDER_DETAIL (
  order_id VARCHAR(50) PRIMARY KEY,
  customer_id VARCHAR(50),
  order_date DATE NOT NULL,
  order_status VARCHAR(50) NOT NULL,
  promo_code VARCHAR(20),
  payment_method TEXT NOT NULL,
  delivery_fee DECIMAL(10, 2) NOT NULL,
  FOREIGN KEY (customer_id) REFERENCES CUSTOMER (customer_id),
  FOREIGN KEY (promo_code)  REFERENCES DISCOUNT (promo_code)
);
```

9. Create ORDER_ITEM entity

```sql
CREATE TABLE IF NOT EXISTS ORDER_ITEM (
  order_id VARCHAR(50),
  product_id VARCHAR(50),
  order_quantity INT NOT NULL,
  PRIMARY KEY (order_id, product_id),
  FOREIGN KEY (order_id)   REFERENCES ORDER_DETAIL (order_id)
  FOREIGN KEY (product_id) REFERENCES PRODUCT (product_id)
);
```

10. Create ADVERTISE_IN entity

```sql
CREATE TABLE IF NOT EXISTS ADVERTISE_IN (
  product_id VARCHAR(50),
  ad_id VARCHAR(50),
  PRIMARY KEY (product_id, ad_id),
  FOREIGN KEY (product_id) REFERENCES PRODUCT (product_id),
  FOREIGN KEY (ad_id)      REFERENCES ADVERTISEMENTS (ad_id)
);
```

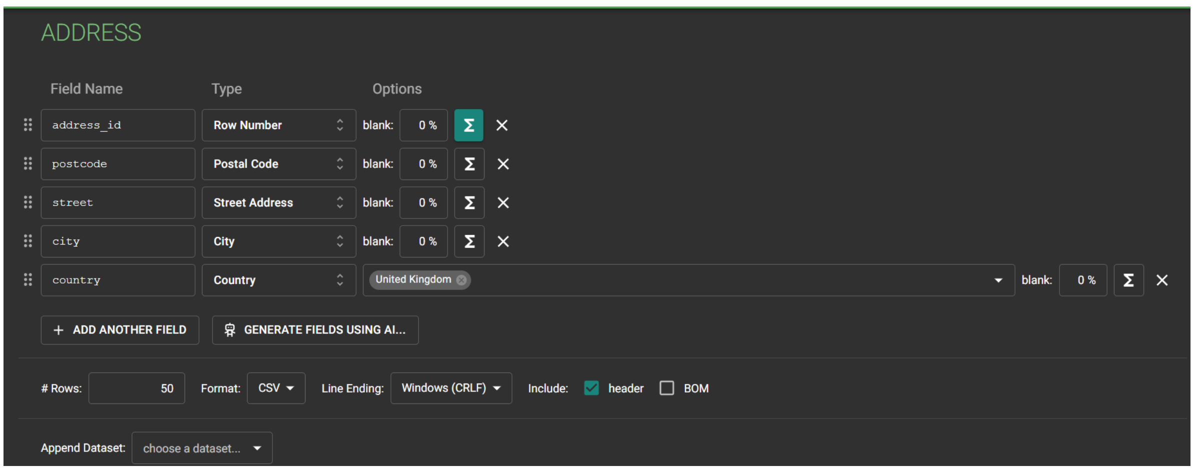
Part 2: Data Generation and Implementation

Task 2.1: Synthetic Data Generation

Synthetic data from the normalized schema was created for each table was using Mockaroo, a mock data generator platform. Data was generated for the parent entities initially to ensure that the referential integrity exists on the children entities or the entities that need a reference for foreign keys from parent entities

Data generation for parent entities

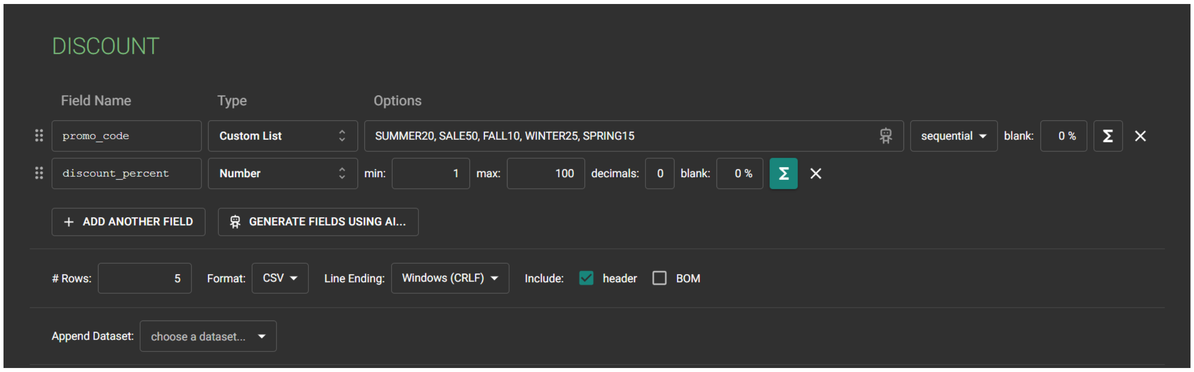
1. ADDRESS



50 observations for the ADDRESS table were created. As we would like to focus on the customers who live in the United Kingdom, the generated address information will only be in the United Kingdom.

In addition, we have set the address_id, which is the primary of the table, to start with ‘ADDR’ and followed by a number.

2. DISCOUNT

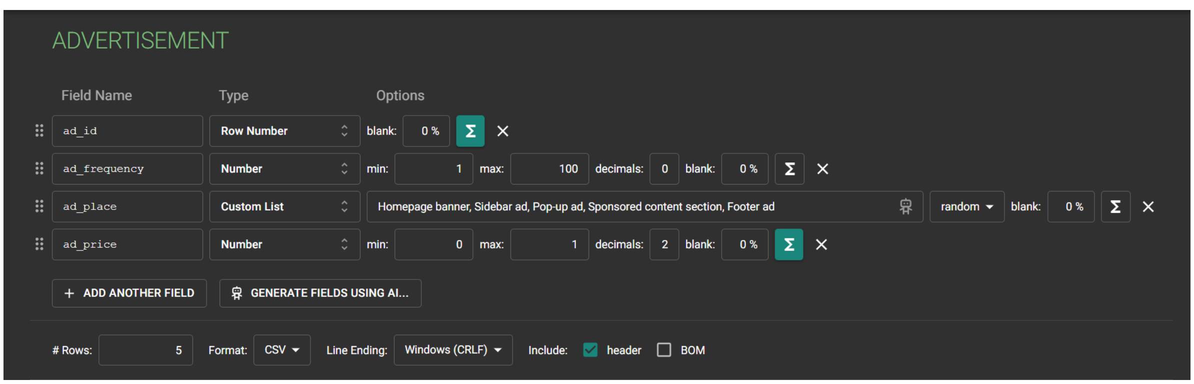


5 observations for the DISCOUNT table were created. For the promo_code, which is a primary key, the values are created using an AI generator, and the values of the discount_percent, which stores the discount amount as percentages stores, are assigned manually to particular promo_code using this formula:

Formula

```
if promo_code == 'SUMMER20' then 20
elsif promo_code == 'SALE50' then 50
elsif promo_code == 'FALL10' then 10
elsif promo_code == 'WINTER25' then 25
elsif promo_code == 'SPRING15' then 15 end
```

3. ADVERTISEMENT



5 observations for the ADVERTISEMENT table were created. We set the condition for the primary key, ad_id, to start with 'AD' followed by a number. For the ad_place attribute, we used an AI generator to generate places where advertisements can be shown. We assumed that different places would cause different amounts of ad_price, so we manually assigned ad_price values based on different ad_place with the formula attached below.

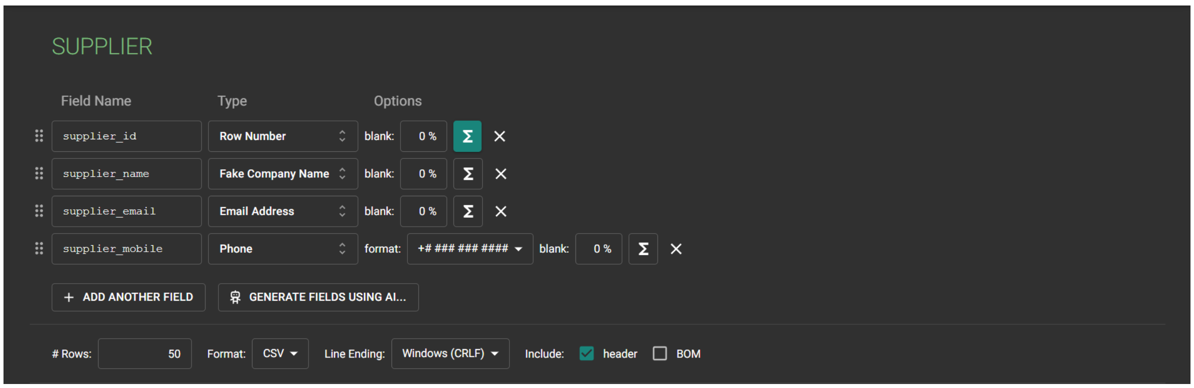
Formula

```
'AD' + this.to_s
```

Formula

```
if ad_place == 'Homepage banner' then 0.50
elsif ad_place == 'Sidebar ad' then 0.4
elsif ad_place == 'Pop-up ad' then 0.43
elsif ad_place == 'Sponsored content section' then 0.33
elsif ad_place == 'Footer ad' then 0.28
end
```

4. SUPPLIER

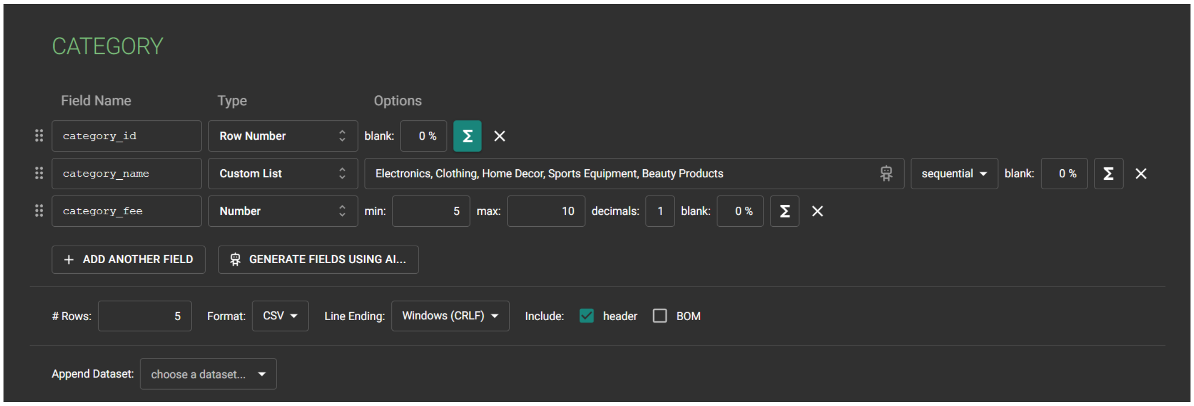


50 observations for the SUPPLIER table were created. We have set the supplier_id, which is the primary of the table, to start with 'SUP' and then follow with a number.

Formula

```
'SUP' + this.to_s
```

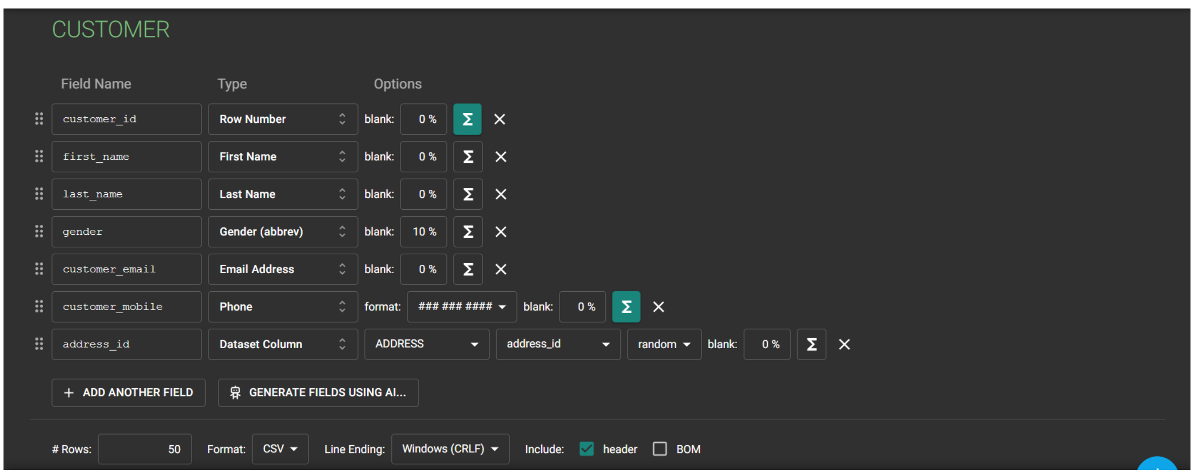
5. CATEGORY



5 observations for the CATEGORY table were created. We have set the category_id, which is the primary of the table, to start with 'CATG' and then follow with a number.

Data generation for children entities

6. CUSTOMER



50 observations for the CUSTOMER table were created. There were additional constraints that we set in the formula. Firstly, we set to condition the customer_id to start with 'CUS' and followed by a number. Moreover, we gave a condition on customer_mobile to start with +44, which is the United Kingdom country code, before generating the rest of the numbers to comply with our focus on United Kingdom customers only. The formulas for those conditions are as followed

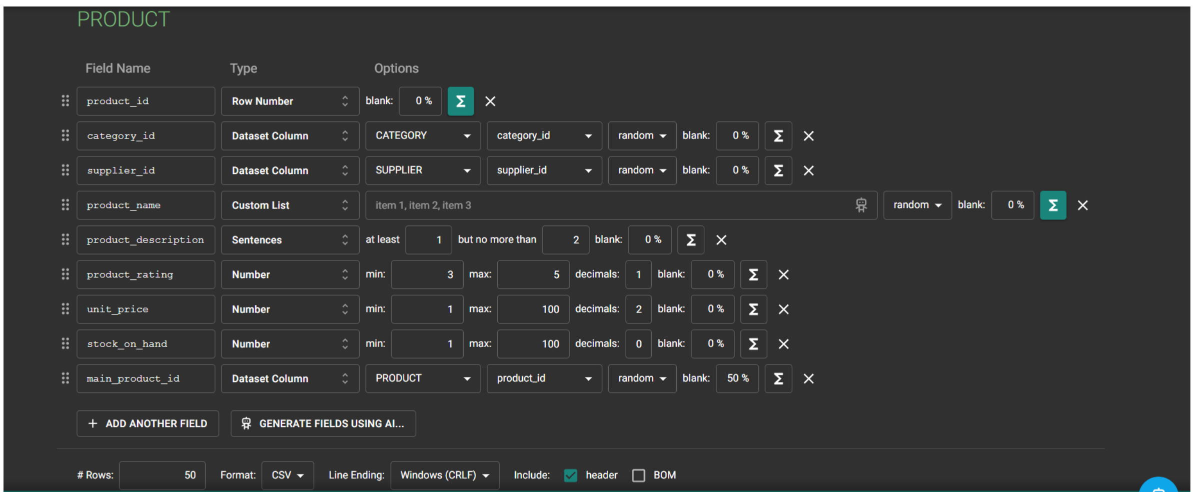
Formula

```
'CUS' + this.to_s
```

Formula

```
'+44 ' + this.to_s
```

7. PRODUCT



50 observations for the PRODUCT table were created. There were two conditions we set in this table. The first condition is on the primary key, product_id, to start the value with 'PROD' and then a number. The second condition is on product_name. To ensure consistency between the products and categories, we manually assigned the values of product_name to the categories they were supposed to belong to. These formulas were used to set those conditions.

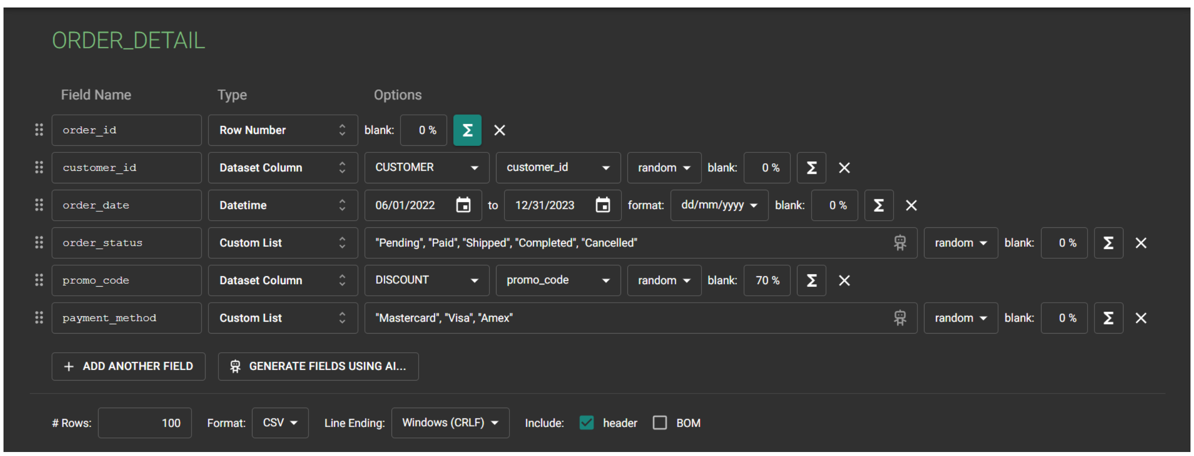
Formula

```
'PROD' + this.to_s
```

Formula

```
this =
if category_id == 'CATG1'
['QuantumX Gaming Mouse', 'Lumina Wireless Earbuds', 'NeoTech Smart Watch', 'NovaTech VR Headset', 'Infinity Bluetooth Speaker',
 'CyberX Gaming Keyboard', 'Apollo Portable Charger', 'Solaris HD Webcam', 'TitanX Noise-Canceling Headphones', 'Nebula Smart LED Bulbs',
 'FusionX Gaming Console', 'Zenith Ultra HD Monitor', 'Sparkle USB-C Hub', 'LunaTech Digital Camera', 'ZenTech Fitness Tracker',
 'Skyline Drone with HD Camera', 'Stellar Wireless Charging Pad', 'Matrix Smart Thermostat', 'Sonic Boom Portable Speaker',
 'HyperDrive External SSD'].sample
elsif category_id == 'CATG2'
['Aurora Shift Dress', 'Nova Denim Jacket', 'Celestial Leggings', 'Galaxy Hoodie', 'Eclipse Bomber Jacket', 'Lunar Maxi Skirt',
 'Solar Flare Sweater', 'Stardust T-Shirt', 'Comet Crop Top', 'Solar Eclipse Sunglasses', 'Meteorite Joggers', 'Nebula Scarf',
 'Cosmic Print Blouse', 'Constellation Dress Shirt', 'Supernova Sweatpants', 'Solar System Hooded Sweatshirt', 'Satellite Silk Tie',
 'Stellar Puffer Vest', 'Moonbeam Cardigan', 'Zenith Yoga Pants'].sample
elsif category_id == 'CATG3'
['Zen Garden Set', 'Serenity Wall Art', 'Harmony Throw Pillow', 'Tranquility Scented Candles', 'Blissful Bedding Set', 'Reflections Mirror',
 'Solace Area Rug', 'Symphony Table Lamp', 'Oasis Indoor Fountain', 'Essence Room Diffuser', 'Aura Decorative Vase', 'Zenith Wall Clock',
 'Dreamcatcher Wall Hanging', 'Enchanted Fairy Lights', 'Haven Decorative Tray', 'Radiance Window Curtains', 'Utopia Throw Blanket',
 'Sanctuary Plant Stand', 'Paradise Accent Chair', 'Euphoria Decorative Bowl'].sample
elsif category_id == 'CATG4'
['TitanX Carbon Fiber Tennis Racket', 'NovaTech Running Shoes', 'Eclipse Yoga Mat', 'Zenith Golf Club Set', 'Solaris Cycling Helmet',
 'Blaze Basketball Hoop', 'Nebula Soccer Ball', 'FusionX Fitness Bands', 'Comet CrossFit Gloves', 'Stellar Surfboard', 'Galaxy Hiking Backpack',
 'Lunar Jump Rope', 'Sonic Boom Resistance Bands', 'Celestial Climbing Harness', 'Aurora Weightlifting Gloves', 'Meteorite Mountain Bike',
 'Cosmic Compression Socks', 'Satellite Ski Poles', 'Zenith Zumba Shoes', 'Stellar Swimming Goggles'].sample
elsif category_id == 'CATG5'
['Luna Glow Facial Serum', 'Celestial Shampoo & Conditioner Set', 'Solar Flare Hair Straightener',
 'Zenith Facial Cleansing Brush', 'NovaTech Hair Dryer', 'Aurora Lip Gloss Collection', 'Nebula Nail Polish Kit', 'Stellar Eyeshadow Palette',
 'Lunar Face Mask Set', 'Harmony Body Lotion', 'Radiance Highlighter Stick', 'Cosmic Perfume', 'Eclipse Eyeliner Pen', 'Serenity Bath Bombs',
 'Stardust Makeup Brushes', 'Blissful Body Scrub', 'Tranquility Sleep Mask', 'Solaris Sunscreen Spray', 'Zenith Anti-Aging Cream',
 'Aura Lip Balm Trio'].sample
end
```

8. ORDER_DETAIL

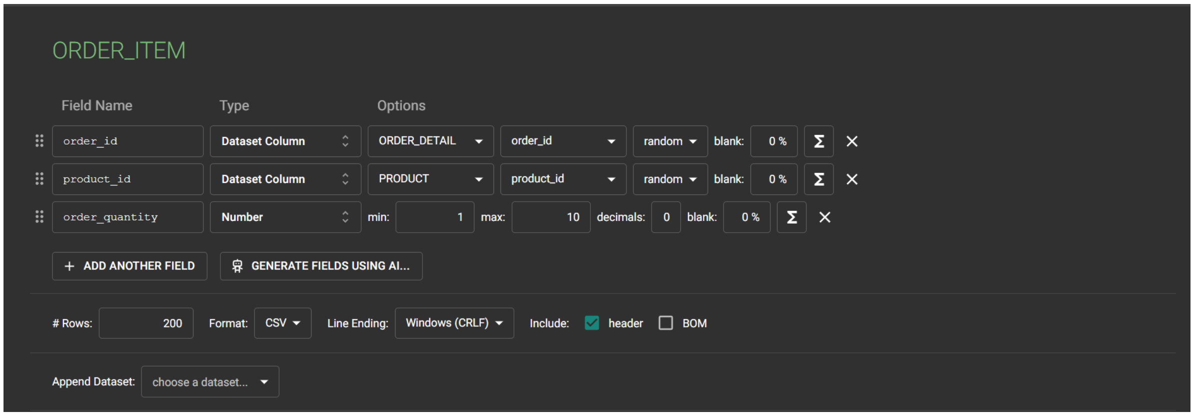


100 observations for the ORDER_DETAIL table were created. We have set the order_id, which is the primary of the table, to start with 'ORD' and then follow with a number.

Formula

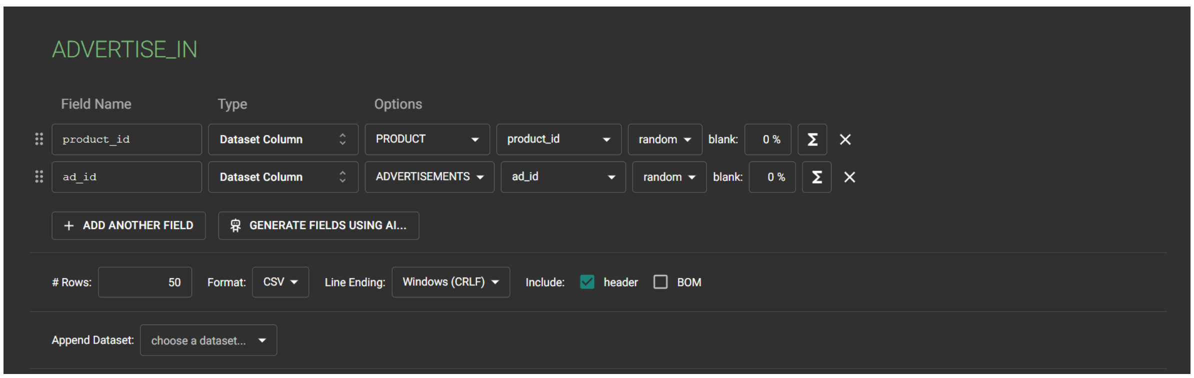
```
'ORD' + this.to_s
```

9. ORDER_ITEM



100 observations for the ORDER_ITEM table were created.

10. ADVERTISE_IN



50 observations for the ADVERTISE_IN table were created.

Task 2.2: Data Import and Quality Assurance

Data Import

Loading all generated data sets using read_csv to each entity

```
customer <- readr::read_csv("data_upload/CUSTOMER.csv")
address <- readr::read_csv("data_upload/ADDRESS.csv")
category <- readr::read_csv("data_upload/CATEGORY.csv")
supplier <- readr::read_csv("data_upload/SUPPLIER.csv")
discount <- readr::read_csv("data_upload/DISCOUNT.csv")
product <- readr::read_csv("data_upload/PRODUCT.csv")
order_item <- readr::read_csv("data_upload/ORDER_ITEM.csv")
order_detail <- readr::read_csv("data_upload/ORDER_DETAIL.csv")
```

```r
advertisement <- readr::read_csv("data_upload/ADVERTISEMENT.csv")
advertise_in <- readr::read_csv("data_upload/ADVERTISE_IN.csv")
```

Quality Assurance

The imported data sets were validated before being added to the database to ensure they aligned with the nature of the keys and data type specified in the physical schema. First, we validated parent entities including ADDRESS, DISCOUNT, ADVERTISEMENT, SUPPLIER, CATEGORY. Once all parent entities were validated, we continued validating children entities CUSTOMER, PRODUCT, ORDER_ITEM, ORDER_DETAIL, and ADVERTISE_IN.

The general validation processes that were performed to check on the data quality for all entities included:

1. Check if the values of the primary key are unique
2. Check if there is any missing data in the attributes that should not have missing values
3. Check if there are any values of a foreign key in the children table that do not exist in the parent table
4. Remove unqualified data

However, there would be some additional validation methods that were used to validate specific attribute for specific entity. The additional method will be provided on each entity if there is any.

Parent Entities Validation

1. ADDRESS

```r
#Check duplicate pk
duplicate_address_id <- address[duplicated(address$address_id),
                                "address_id"]

#Check for missing data
na_address_address_id <- address[is.na(address$address_id), "address_id"]
na_address_postcode <- address[is.na(address$postcode),
                               c("address_id", "postcode")]
na_address_street <- address[is.na(address$street),
                             c("address_id", "street")]
na_address_city <- address[is.na(address$city), c("address_id", "city")]
na_address_country <- address[is.na(address$country),
                              c("address_id", "country")]

#Remove unclean data
bad_address_record <- unique(c(duplicate_address_id$address_id,
                               na_address_address_id$address_id,
```

```
                                na_address_postcode$address_id,
                                na_address_street$address_id,
                                na_address_city$address_id,
                                na_address_country$address_id))
address <- address[!(address$address_id %in% bad_address_record), ]
```

2. DISCOUNT

   Additional validation method:

   - Check the datatype of discount_amount, which should be integers

```
#Check duplicate pk
duplicate_promo_code <- discount[duplicated(discount$promo_code),
                                 "promo_code"]

#Check percent
invalid_discount_percent <- discount[!grepl("^[0-9]+$",
                                            discount$discount_percent),
                                     c("promo_code", "discount_percent")]

#Check for missing data
na_discount_promo_code <- discount[is.na(discount$promo_code),
                                   "promo_code"]
na_discount_discount_percent <- discount[is.na(discount$discount_percent),
                                    c("promo_code", "discount_percent")]

#Remove unclean data
bad_discount_record <- unique(c(duplicate_promo_code$promo_code,
                                invalid_discount_percent$promo_code,
                                na_discount_promo_code$promo_code,
                                na_discount_discount_percent$promo_code))
discount <- discount[!(discount$promo_code %in% bad_discount_record), ]
```

3. ADVERTISEMENT

   Additional validation method:

   - Check the datatype of ad_frequency, which should be integers
   - Check if there is any negative value in the attributes that the values should only be positive

```
#Check duplicate pk
duplicate_ad_id <- advertisement[duplicated(advertisement$ad_id),
                                "ad_id"]

#Check ad frequency (can only contain integer)
invalid_ad_frequency <- advertisement[!grepl("^[0-9]+$",
                                  advertisement$ad_frequency), c
                                      ("ad_id", "ad_frequency")]

#Check for negative ad frequency
negative_ad_frequency <- advertisement[advertisement$ad_frequency < 0,
                                       c("ad_id", "ad_frequency")]

#Check for negative prices
negative_ad_prices <- advertisement[advertisement$ad_price < 0,
                                    c("ad_id", "ad_price")]

#Check for missing data
na_advertisement_ad_id <- advertisement[is.na(advertisement$ad_id),
                                        "ad_id"]
na_advertisement_ad_frequency <- advertisement[is.na(
  advertisement$ad_frequency), c("ad_id", "ad_frequency")]
na_advertisement_ad_price <- advertisement[
  is.na(advertisement$ad_price), c("ad_id", "ad_price")]
na_advertisement_ad_place <- advertisement[is.na(advertisement$ad_place),
                                           c("ad_id", "ad_place")]

#Remove unclean data
bad_advertisement_record <- unique(c(duplicate_ad_id$ad_id,
                                     invalid_ad_frequency$ad_id,
                                     negative_ad_frequency$ad_id,
                                     negative_ad_prices$ad_id,
                                     na_advertisement_ad_id$ad_id,
                                     na_advertisement_ad_frequency$ad_id,
                                     na_advertisement_ad_price$ad_id,
                                     na_advertisement_ad_place$ad_id))
advertisement <- advertisement[!(advertisement$ad_id %in%
                                   bad_advertisement_record),
                               ]
```

4. SUPPLIER

   Additional validation method:

- Check the format of suppliers' names
- Check the format of the email
- Check the format of the mobile number, which should start with a plus sign followed by 12 integer values

```
#Check duplicate pk
duplicate_supplier_id <- supplier[duplicated(supplier$supplier_id),
                                  "supplier_id"]

#Check supplier (can contain alphabets, comma, hyphen, dot)
invalid_supplier_name <- supplier[!grepl(
  "^[A-Za-z,.-]+( [A-Za-z,.-]+)*$",
      supplier$supplier_name),
  c("supplier_id", "supplier_name")]

#Check email format
invalid_supplier_email <- supplier[!grepl(
  "^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,}$",
  supplier$supplier_email), c("supplier_id", "supplier_email")]

#Check format of mobile number (+xx xxx xxx xxxx)
invalid_supplier_mobile <- supplier[!grepl(
  "^^\\+\\d{1,3}\\s[0-9]{3}\\s[0-9]{3}\\s[0-9]{4}$",
  supplier$supplier_mobile), c("supplier_id", "supplier_mobile")]

#Check for missing data
na_supplier_supplier_id <- supplier[is.na(supplier$supplier_id),
                                    "supplier_id"]
na_supplier_supplier_name <- supplier[is.na(supplier$supplier_name),
                                      c("supplier_id", "supplier_name")]
na_supplier_supplier_email <- supplier[is.na(supplier$supplier_email),
                                      c("supplier_id", "supplier_email")]
na_supplier_supplier_mobile <- supplier[is.na(supplier$supplier_mobile),
                                      c("supplier_id", "supplier_mobile")]

#Remove unclean data
bad_supplier_record <- unique(c(duplicate_supplier_id$supplier_id,
                                invalid_supplier_name$supplier_id,
                                invalid_supplier_email$supplier_id,
                                invalid_supplier_mobile$supplier_id,
                                na_supplier_supplier_id$supplier_id,
                                na_supplier_supplier_name$supplier_id,
```

```
                                  na_supplier_supplier_email$supplier_id,
                                  na_supplier_supplier_mobile$supplier_id))
supplier <- supplier[!(supplier$supplier_id %in% bad_supplier_record), ]
```

5. CATEGORY

   Additional validation method:

   - Check the datatype of categories' names

```
#Check duplicate pk
duplicate_category_id <- category[duplicated(category$category_id),
                                   "category_id"]

#Check category (can contain alphabets)
invalid_category_name <- category[!grepl(
  "^[A-Za-z]+( [A-Za-z]+)*$", category$category_name),
  c("category_id", "category_name")]

#Check for duplicate category name
duplicate_category_name <- category[duplicated(category$category_name),
                                     c("category_id", "category_name")]

#Check for negative prices
negative_category_fee <- category[category$category_fee < 0,
                                   c("category_id", "category_fee")]

#Check for missing data
na_category_category_id <- category[is.na(category$category_id),
                                     "category_id"]
na_category_category_name <- category[is.na(category$category_name),
                                       c("category_id", "category_name")]
na_category_category_fee <- category[is.na(category$category_fee),
                                      c("category_id", "category_fee")]

#Remove unclean data
bad_category_record <- unique(c(duplicate_category_id$category_id,
                                invalid_category_name$category_id,
                                duplicate_category_name$category_id,
                                negative_category_fee$category_id,
                                na_category_category_id$category_id,
                                na_category_category_name$category_id,
```

```
                                na_category_category_fee$category_id))
category <- category[!(category$category_id %in% bad_category_record), ]
```

Children Entities Validation

6. CUSTOMER

   Additional validation method:

   - Check the format of the customers' first and last names. The format is expected to be first uppercase alphabet followed by lowercase alphabet.
   - Check the format of the email
   - Check the format of the mobile number, which should start with a plus sign followed by 12 integer values

```
#Check duplicate pk
duplicate_customer_id <- customer[duplicated(customer$customer_id),
                                  "customer_id"]

#Check format of first and last name (1st alphabet is uppercase,
#rest is lowercase)
invalid_customer_firstname <- customer[!grepl(
  "^[A-Z][a-z]*$", customer$first_name),
  c("customer_id", "first_name")]
invalid_customer_lastname <- customer[!grepl(
  "^[A-Z][a-z]*$", customer$last_name),
  c("customer_id", "last_name")]

#Check email format
invalid_customer_email <- customer[!grepl(
  "^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,}$",
  customer$customer_email), c("customer_id", "customer_email")]

#Check format of mobile number (+xx xxx xxx xxxx)
invalid_customer_mobile <- customer[!grepl(
  "^\\+\\d{1,3}\\s[0-9]{3}\\s[0-9]{3}\\s[0-9]{4}$",
  customer$customer_mobile), c("customer_id", "customer_mobile")]

#Check if address_id exists in the ADDRESS table
invalid_address_fk <- customer[!customer$address_id %in%
                                address$address_id,
                              c("customer_id", "address_id")]
```

```r
#Check for missing data
na_customer_customer_id <- customer[is.na(customer$customer_id),
                                    "customer_id"]
na_customer_first_name <- customer[is.na(customer$first_name),
                                  c("customer_id", "first_name")]
na_customer_last_name <- customer[is.na(customer$last_name),
                                  c("customer_id", "last_name")]
na_customer_customer_email <- customer[is.na(customer$customer_email),
                                  c("customer_id", "customer_email")]
na_customer_customer_mobile <- customer[is.na(customer$customer_mobile),
                                  c("customer_id", "customer_mobile")]

#Remove unclean data
bad_customer_record <- unique(c(duplicate_customer_id$customer_id,
                                invalid_customer_firstname$customer_id,
                                invalid_customer_lastname$customer_id,
                                invalid_customer_email$customer_id,
                                invalid_customer_mobile$customer_id,
                                invalid_address_fk$customer_id,
                                na_customer_customer_id$customer_id,
                                na_customer_first_name$customer_id,
                                na_customer_last_name$customer_id,
                                na_customer_customer_email$customer_id,
                                na_customer_customer_mobile$customer_id))
customer <- customer[!(customer$customer_id %in% bad_customer_record), ]
```

7. PRODUCT

   Additional validation method:

   - Check the datatype of product_name
   - Check if there is any negative value in unit_price where the values should only be positive
   - Check on the validity of stock_on_hand to see if there are any missing or negative values
   - Check if the values in main_product_id are self-referential to product_id and ensure that the values differ from those of product_id.

   ```r
   #Check duplicate pk
   duplicate_product_id <- product[duplicated(product$product_id),
                                   "product_id"]
   ```

```
#Check product name (can contain alphabets, comma, hyphen, dot)
invalid_product_name <- product[!grepl(
  "^[A-Za-z,.-]+( [A-Za-z,.-]+)*$", product$product_name),
  c("product_id", "product_name")]

#Check for negative prices
negative_unit_prices <- product[product$unit_price < 0,
                                c("product_id", "unit_price")]

#Check invalid stock
invalid_stock <- product[!grepl("^[0-9]+$", product$stock_on_hand),
                         c("product_id", "stock_on_hand")]
negative_stock <- product[product$stock_on_hand < 0,
                          c("product_id", "stock_on_hand")]

#Check if supplier_id exists in the SUPPLIER table
invalid_supplier_fk <- product[!product$supplier_id %in%
                                 supplier$supplier_id,
                               c("product_id", "supplier_id")]

#Check if category_id exists in the CATEGORY table
invalid_category_fk <- product[!product$category_id %in%
                                category$category_id,
                               c("product_id", "category_id")]

#Check if main product is self referential and
#it cannot be the same as product_id
invalid_main_product_ids <- product[!is.na(product$main_product_id) &
          !(product$main_product_id %in% product$product_id &
            product$main_product_id != product$product_id),
          c("product_id", "main_product_id")]

#Check for missing data
na_product_product_id <- product[is.na(product$product_id),
                                 "product_id"]
na_product_category_id <- product[is.na(product$category_id),
                                  c("product_id", "category_id")]
na_product_supplier_id <- product[is.na(product$supplier_id),
                                  c("product_id", "supplier_id")]
na_product_product_name <- product[is.na(product$product_name),
                                  c("product_id", "product_name")]
na_product_unit_price <- product[is.na(product$unit_price),
```

```
                                 c("product_id", "unit_price")]
na_product_stock_on_hand <- product[is.na(product$stock_on_hand),
                                 c("product_id", "stock_on_hand")]

#Remove unclean data
bad_product_record <- unique(c(duplicate_product_id$product_id,
                               invalid_product_name$product_id,
                               negative_unit_prices$product_id,
                               invalid_stock$product_id,
                               negative_stock$product_id,
                               invalid_supplier_fk$product_id,
                               invalid_category_fk$product_id,
                               invalid_main_product_ids$product_id,
                               na_product_product_id$product_id,
                               na_product_category_id$product_id,
                               na_product_supplier_id$product_id,
                               na_product_product_name$product_id,
                               na_product_unit_price$product_id,
                               na_product_stock_on_hand$product_id))
product <- product[!(product$product_id %in% bad_product_record), ]
```

8. ORDER_ITEM

   Additional validation method:

   - Check on the values of order_status. The values of this attribute are expected to be Pending, Paid, Shipped, Complete, or Cancelled
   - Check on the values of payment_method. The values of this attribute are expected to be Mastercard, Visa, or Amex
   - Check on the date format
   - Check if there is any negative value in delivery_fee where the values should only be positive

```
#Check duplicate pk
duplicate_order_id <- order_detail[duplicated(order_detail$order_id),
                                   "order_id"]

#Check if customer_id exists in the CUSTOMER table
invalid_customer_fk <- order_detail[!order_detail$customer_id %in%
                                      customer$customer_id, "order_id"]

#Check if promo_code exists in the DISCOUNT table
```

```r
discounted_order <- order_detail[!is.na(order_detail$promo_code), ]
invalid_promo_fk <- discounted_order[!discounted_order$promo_code %in%
                                       discount$promo_code,
                                     c("order_id", "promo_code")]

#Check order status
status <- c("Pending", "Paid", "Shipped", "Completed", "Cancelled")
invalid_order_status <- order_detail[!order_detail$order_status %in%
                                       status, c("order_id",
                                                 "order_status")]

#Check payment method
payment <- c("Mastercard", "Visa", "Amex")
invalid_payment_method <- order_detail[!order_detail$payment_method %in%
                                         payment, c("order_id",
                                                    "payment_method")]

#Check date format (dd/mm/yyyy)
invalid_order_date <- order_detail[!grepl("^\\d{2}/\\d{2}/\\d{4}$",
                                          order_detail$order_date),
                                      c("order_id", "order_date")]

#Check for negative delivery fee
negative_delivery_fee <- order_detail[order_detail$delivery_fee < 0,
                                      c("order_id", "delivery_fee")]

#Check for missing data
na_order_order_id <- order_detail[is.na(order_detail$order_id),
                                  "order_id"]
na_order_customer_id <- order_detail[is.na(order_detail$customer_id),
                                    c("order_id", "customer_id")]
na_order_order_status <- order_detail[is.na(order_detail$order_status),
                                    c("order_id", "order_status")]
na_order_order_date <- order_detail[is.na(order_detail$order_date),
                                    c("order_id", "order_date")]
na_order_payment_method <- order_detail[is.na(order_detail$payment_method),
                                    c("order_id", "payment_method")]
na_order_delivery_fee <- order_detail[is.na(order_detail$delivery_fee),
                                    c("order_id", "delivery_fee")]

#Remove unclean data
bad_order_record <- unique(c(duplicate_order_id$order_id,
```

```
                              invalid_customer_fk$order_id,
                              invalid_promo_fk$order_id,
                              invalid_order_status$order_id,
                              invalid_payment_method$order_id,
                              invalid_order_date$order_id,
                              negative_delivery_fee$order_id,
                              na_order_order_id$order_id,
                              na_order_customer_id$order_id,
                              na_order_order_status$order_id,
                              na_order_order_date$order_id,
                              na_order_payment_method$order_id,
                              na_order_delivery_fee$order_id))
order_detail <- order_detail[!(order_detail$order_id %in%
                                  bad_order_record), ]
```

9. ORDER_DETAIL

   Additional validation method:Check if the values of the primary key are unique

   - Check on the validity of order_quantity to see if there are any missing or negative values

```
#Check duplicate for composite primary key
order_item_composite <- paste(order_item$order_id, order_item$product_id)
duplicate_order_item_composite <- order_item[duplicated(
  order_item_composite), c("order_id", "product_id")]

#Check if order_id exists in the ORDER_ITEM table
invalid_order_fk <- order_item[!order_item$order_id %in%
                                  order_detail$order_id,
                                c("order_id", "product_id")]

#Check if product_id exists in the PRODUCT table
invalid_product_fk <- order_item[!order_item$product_id %in%
                                    product$product_id,
                                  c("order_id", "product_id")]

#Check invalid order quantity
invalid_quantity <- order_item[!grepl("^[0-9]+$",
                                      order_item$order_quantity),
                                c("order_id", "order_quantity")]
negative_zero_quantity <- order_item[order_item$order_quantity < 1,
                                 c("order_id", "order_quantity")]
```

```
#Check for missing data
na_order_item_order_id <- order_item[is.na(order_item$order_id),
                                     "order_id"]
na_order_item_product_id <- order_item[is.na(order_item$order_id),
                                      c("order_id", "product_id")]
na_order_item_order_quantity <- order_item[is.na(
                                      order_item$order_quantity),
                                      c("order_id", "order_quantity")]

#Remove unclean data
bad_order_item_record <- unique(c())
order_item <- order_item[!(order_item$order_id %in%
                             bad_order_item_record), ]

#Remove unclean data
# Combine all unclean data
bad_order_item_record <- rbind(duplicate_order_item_composite,
                               invalid_order_fk,
                               invalid_product_fk,
                               invalid_quantity,
                               negative_zero_quantity,
                               na_order_item_order_id,
                               na_order_item_product_id,
                               na_order_item_order_quantity)

# Remove duplicates from bad records
bad_order_item_record <- unique(bad_order_item_record)

# Remove unclean data based on composite key
order_item <- order_item[!paste(order_item$order_id,
                                order_item$product_id)
                         %in% paste(bad_order_item_record$order_id,
                                    bad_order_item_record$product_id), ]
```

10. ADVERTISE_IN

```
#Check duplicate for composite primary key
advertise_in_composite <- paste(advertise_in$ad_id,
                                advertise_in$product_id)
duplicate_advertise_in_composite <- advertise_in[duplicated(
  advertise_in_composite), c("ad_id", "product_id")]
```

```
#Check if ad_id exists in the ADVERTISEMENT table
invalid_advertisement_fk <- advertise_in[!advertise_in$ad_id %in%
                                           advertisement$ad_id,
                                         c("ad_id", "product_id")]

#Check if product_id exists in the PRODUCT table
invalid_product_fk <- advertise_in[!advertise_in$product_id %in%
                                     product$product_id,
                                   c("ad_id", "product_id")]

#Check for missing data
na_advertise_in_ad_id <- advertise_in[is.na(advertise_in$ad_id), "ad_id"]
na_advertise_in_product_id <- advertise_in[is.na(advertise_in$product_id),
                                           c("ad_id", "product_id")]

#Remove unclean data
# Combine all unclean data
bad_advertise_in_record <- rbind(duplicate_advertise_in_composite,
                                 invalid_advertisement_fk,
                                 invalid_product_fk,
                                 na_advertise_in_ad_id,
                                 na_advertise_in_product_id)

# Remove duplicates from bad records
bad_advertise_in_record <- unique(bad_advertise_in_record)

# Remove unclean data based on composite key
advertise_in <- advertise_in[!paste(advertise_in$ad_id,
                                    advertise_in$product_id) %in%
                               paste(bad_advertise_in_record$ad_id, bad_advertise_in_rec
```

Once the dataset was validated, it was imported to the database, where the schema is already created.

```
#add new records into table
connect <- dbConnect(RSQLite::SQLite(), "database.db")

tables <- c("CUSTOMER", "ADDRESS", "CATEGORY", "SUPPLIER", "PRODUCT",
            "DISCOUNT", "ORDER_ITEM", "ORDER_DETAIL", "ADVERTISEMENT",
            "ADVERTISE_IN")

table_new <- list(customer, address, category, supplier, product, discount,
```

```
                  order_item, order_detail, advertisement, advertise_in)

# Loop through each table
for (i in seq_along(tables)) {
  table <- tables[i]
  new_records <- table_new[[i]]

  # Read existing records from the table
  existing <- dbGetQuery(connect, paste("SELECT * FROM", table))

  # Convert data types if needed (e.g., order_date column)
  if ("order_date" %in% colnames(existing)) {
    existing$order_date <- as.character(existing$order_date)
    new_records$order_date <- as.character(new_records$order_date)
  }

  # Find new records not present in existing table
  new <- anti_join(new_records, existing)

  # Append new records to the table
  if (nrow(new) > 0) {
    dbWriteTable(connect, table, new, append = TRUE)
  }}
```

Part 3: Data Pipeline Generation

Task 3.1: GitHub Repository and Workflow Setup

Repository named **DM_Group3** was createdon GitHub with public access and following files were created:

1. **README:** Includes all student ID numbers of our group.
2. **database_schema.R :** R Script to create a database and build the database schema.
3. **data_validation_and_load.R:** R Script for data validation and data ingestion into the database.
4. **data_analysis.R:** R script to perform data analysis for necessary data visualizations
5. **data_upload and images:** Folders to store 10 datasets generated by Mockaroo, and

images to store E-R diagrams.

6. **Report:** Quarto Markdown file to report the execution of tasks.

New project was created in RStudio for each user. Github was used for version control to track changes, create branches, merge codes,etc.

Github actions were utilized to automate the entire ETL workflow process such as running tests, building, and deploying tasks after pushing code, which can improve development efficiency and ensure code stability

Workflow file **etl.yaml** was created in the **.github/workflows** directory in **DM_Group3** repository.

Task 3.2: GitHub Actions for Continuous Integration

Workflow was designed to execute a task named "ETL workflow for group 3", after a trigger by pushing events to the 'main' branch. Sequence of steps were executed on the latest version of Ubuntu:

1. Use GitHub Actions checkout action to check out code into the working directory.
2. Set up R environment and specify the R version as '4.3.3'.
3. Cache R packages to avoid reinstalling them on each run.
4. Install required packages if the cache misses.
5. Run database_schema.Rmd to render database schema.
6. Run data_validation_and_load.Rmd to render validation and load data into database.
7. Run data_analysis.Rmd to render data analysis.
8. Utilize an authentication token to push changes to the 'main' branch.

Part 4: Data Analysis and Reporting with Quarto in R

Task 4.1: Advanced Data Analysis in R

```r
# Connecting to the database
connect <- dbConnect(RSQLite::SQLite(), "database.db")
```

```r
# Report 1

# Top 10 products based on the Profit Generated

# Getting the tables from the database
product <- RSQLite::dbGetQuery(connect,'SELECT * FROM PRODUCT')
order_item <- RSQLite::dbGetQuery(connect,'SELECT * FROM ORDER_ITEM')
order_detail <- RSQLite::dbGetQuery(connect,'SELECT * FROM ORDER_DETAIL')
discount <- RSQLite::dbGetQuery(connect,'SELECT * FROM DISCOUNT')
# Creating a profit table that includes a total_profit column for each product
#incorporating the discount percent
#filtering out all "cancelled" orders.

profit_data <- order_item %>%
  inner_join(order_detail, by = "order_id") %>%
  filter(order_status != "Cancelled") %>%
  inner_join(product, by = "product_id") %>%
  left_join(discount, by = c("promo_code" = "promo_code")) %>%
  mutate(
    discount_percentage = ifelse(is.na(discount_percent), 0, discount_percent),
    total_profit = (order_quantity * unit_price) *
      (1 - discount_percentage / 100)
  ) %>%
  group_by(product_id, product_name) %>%
  summarise(
    total_profit = sum(total_profit),
    .groups = 'drop'
  ) %>%
  arrange(desc(total_profit))

# Selecting the top 10 profitable products
top_10_profit_data <- head(profit_data, 10)
```

```r
# Visualizing the results
ggplot(top_10_profit_data, aes(x = reorder(product_id, total_profit),
                               y = total_profit,
                               fill = product_name)) +
  geom_bar(stat = "identity") +
  scale_fill_discrete(name = "product Name") +
  labs(title = "Top 10 Most Profitable Products", x = "Product ID",
       y = "Total Profit", caption="Figure 1") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1),
        legend.title = element_text(size = 12),
        legend.text = element_text(size = 10),
        plot.title = element_text(hjust = 0.5))
```

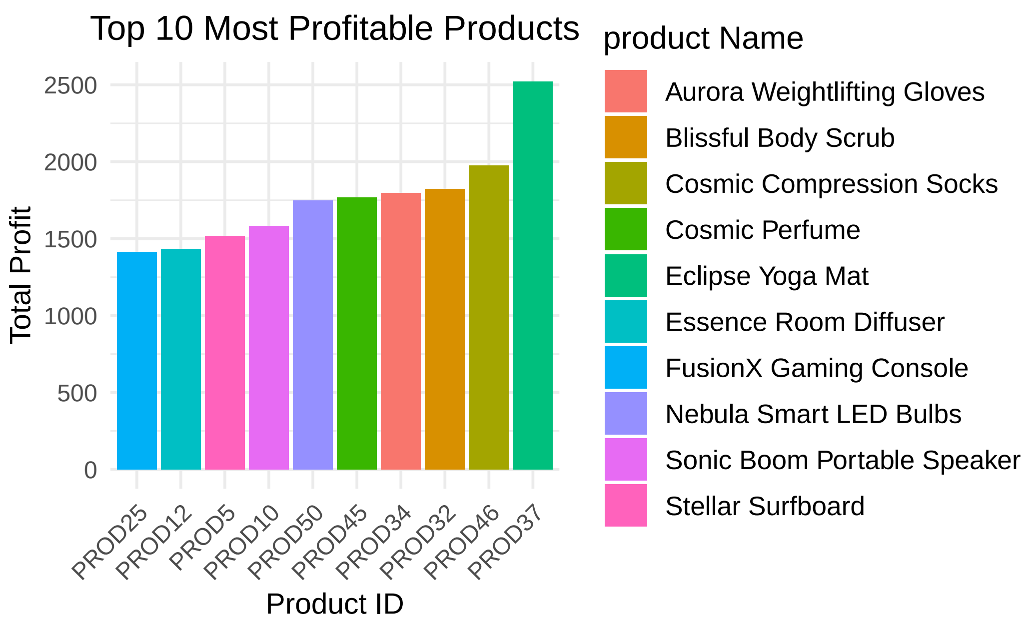


Figure 1

```r
# Top Most Selling Products

# Creating a sales that that includes the total quantity sold for each product
# selecting the top 10.
sales_data <- order_item %>%
  group_by(product_id) %>%
  summarise(Total_Sales = sum(order_quantity, na.rm = TRUE)) %>%
  arrange(desc(Total_Sales))
```

```r
top_selling_products <- sales_data %>%
  left_join(product, by = "product_id") %>%
  select(product_id, product_name, Total_Sales) %>%
  top_n(10, Total_Sales)

# Visualizing the results
ggplot(top_selling_products, aes(x = reorder(product_id, Total_Sales),
                                 y = Total_Sales, fill = product_name)) +
  geom_bar(stat = "identity") +
  scale_fill_discrete(name = "product Name") +
  labs(title = "Top Most Selling Products", x = "Product ID",
       y = "Total Sales", caption="Figure 2") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1),
        legend.title = element_text(size = 12),
        legend.text = element_text(size = 10),
        plot.title = element_text(hjust = 0.5))
```

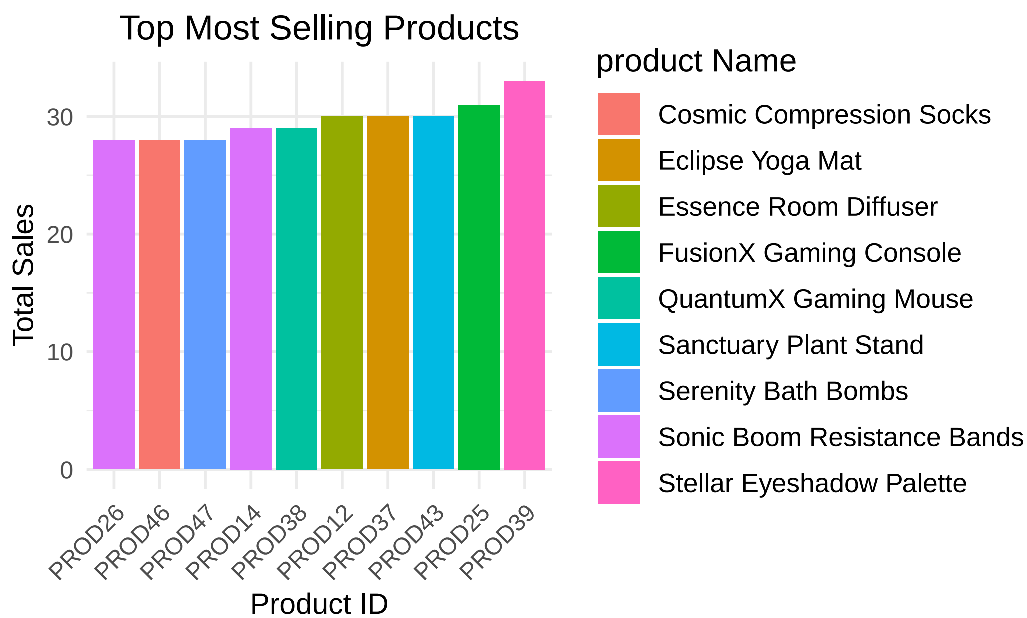


Figure 2

Figure 1 represents the profitability of individual products. The “Eclipse Yoga Mat” generates the highest profit for the company indicating a higher margin or a premium pricing strategy. Conversely, Figure 2 illustrates sales volume, where items like the “Stellar Eyeshadow Palette” and “Fusion X Gaming Console” lead, suggesting they are popular among customers. Some products are both popular and profitable implying a successful product strategy. However, there are also noticeable discrepancies. For example, "QuantumX Gaming Mouse" appears as a top seller, yet it's absent from the most profitable items, implying a lower profit margin. In contrast, "Nebula Smart LED Bulbs" are among the most profitable but not the top sellers, suggesting a high margin compensating for lesser sales.

The e-commerce company could focus on marketing strategies for high-margin products to boost their volume of sales, increasing overall profitability. For products that sell well but are less profitable, the company may need to assess whether they can improve margins through better supplier negotiations.

Products that are both top sellers and highly profitable should be kept in optimum stock to avoid lost sales opportunities. For less profitable items, the company may consider keeping lower stock levels or discontinuing them if they do not contribute significantly to overall profits.

```
#Report 2

# Bundles vs Individual products Comparison by Sales Volume

# Join product with order item
product_order_item <- order_item %>%
  inner_join(product, by = "product_id")

# Calculate total quantity sold for bundled products
bundled_sales <- product_order_item %>%
  filter(!is.na(main_product_id)) %>%
  group_by(main_product_id) %>%
  summarise(Total_Quantity_Sold = sum(order_quantity, na.rm = TRUE)) %>%
  ungroup() %>%
  mutate(Product_Type = "Bundled")

# Calculate total quantity sold for individual products
individual_sales <- product_order_item %>%
  filter(is.na(main_product_id)) %>%
  group_by(product_id) %>%
  summarise(Total_Quantity_Sold = sum(order_quantity, na.rm = TRUE)) %>%
  ungroup() %>%
  mutate(Product_Type = "Individual")

# Combining the results
total_sales_by_type <- bind_rows(bundled_sales, individual_sales) %>%
  group_by(Product_Type) %>%
  summarise(Total_Quantity_Sold = sum(Total_Quantity_Sold, na.rm = TRUE)) %>%
```

```
  ungroup()

print(total_sales_by_type)
```

```
# A tibble: 2 x 2
  Product_Type Total_Quantity_Sold
  <chr>                      <int>
1 Bundled                      548
2 Individual                   520
```

```
# Visualizing the results
ggplot(total_sales_by_type, aes(x = Product_Type, y = Total_Quantity_Sold,
                                fill = Product_Type)) +
  geom_bar(stat = "identity", width = 0.5) +  # Adjust width here
  scale_fill_manual(values = c("Bundled" = "skyblue",
                               "Individual" = "salmon")) +
  labs(title = "Total Quantity Sold by Product Type",
       x = "Product Type",
       y = "Total Quantity Sold") +
  theme_minimal() +
  theme(legend.position = "none")
```

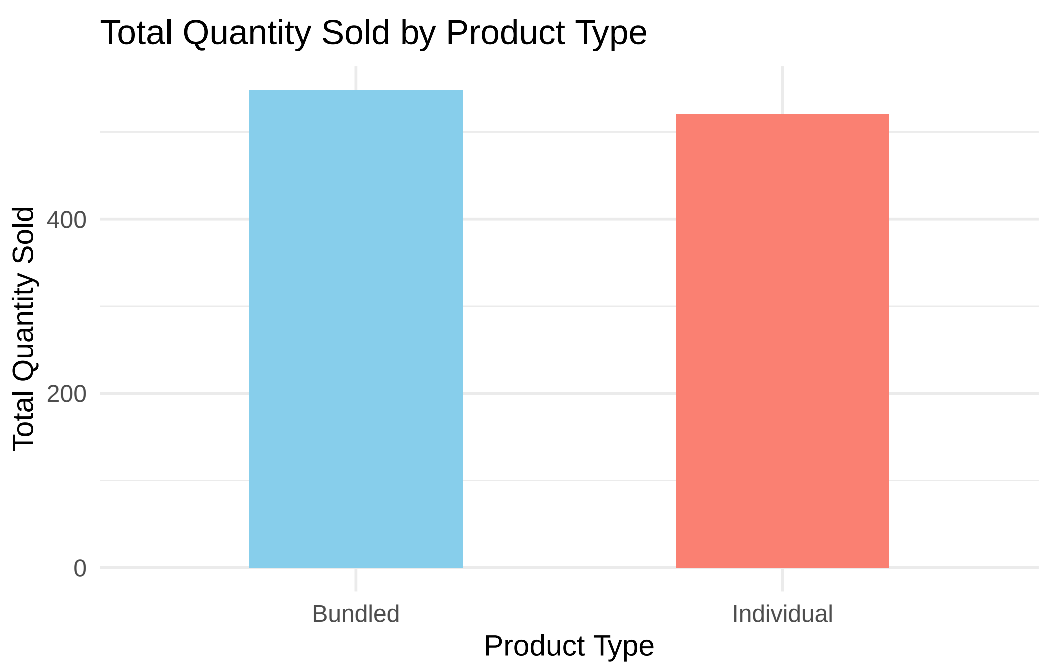


Figure 3

As the table shows, the total sales for individual products (products sold without bundles) and bundled products yield nearly equal quantities sold. This suggests that the e-commerce company's bundling strategy to boost sales was not as effective. However, it is difficult to conclude the efficacy of this deal based solely on the volume of sales.

Optimizing bundle offerings by analyzing and aligning them with customer preferences based on product ratings can significantly enhance their appeal. Experimenting with different bundle types and conducting a cost-benefit analysis might uncover ways to refine the strategy. The results (Figure 3) suggest a market demand for both types of products, indicating that customers prefer flexibility in their buying options. Rather than discontinuing the bundle products, the company should look deeper into the performance metrics of this deal, considering the inventory turnover and profit margins.

```
#Report 3

#SQL Query to get the sales over a time period

time_period_sales<- RSQLite::dbGetQuery(connect,'SELECT ORDITM.order_id,
                                        order_quantity,order_date FROM
                                        ORDER_DETAIL ORDDET INNER JOIN
                                        ORDER_ITEM ORDITM ON ORDITM.order_id =
                                        ORDDET.order_id')
#test

#Conversion of order_date field to appropiate format
time_period_sales$order_date <- as.Date(time_period_sales$order_date,
                                        format="%d/%m/%Y")

time_period_sales$order_date_mnth_yr <- format(
  time_period_sales$order_date,'%Y/%m')

#Conversion of order_date field to factors with levels

time_period_sales$order_date_mnth_yr <- factor(
  time_period_sales$order_date_mnth_yr, levels=c('2022/06',
'2022/07','2022/08',
'2022/09','2022/10','2022/11','2022/12',
'2023/01','2023/02','2023/03','2023/04',
'2023/05','2023/06','2023/07','2023/08'
,'2023/09','2023/10','2023/11','2023/12'))

#Group by sales for each year
```

```
time_period_sales_group_by<- time_period_sales %>%
group_by(order_date_mnth_yr)%>%
summarise(quantity=sum(order_quantity))

# Time series graph to show the quantity sold over a given time period

(ggplot(time_period_sales_group_by, aes(x = order_date_mnth_yr,
y = quantity,group=1)) +
geom_point()+geom_line()+
xlab('Time Period(year/month)')+ylab('Quantity Sold'))+
theme(axis.text.x=element_text(angle=45,hjust = 1))
```

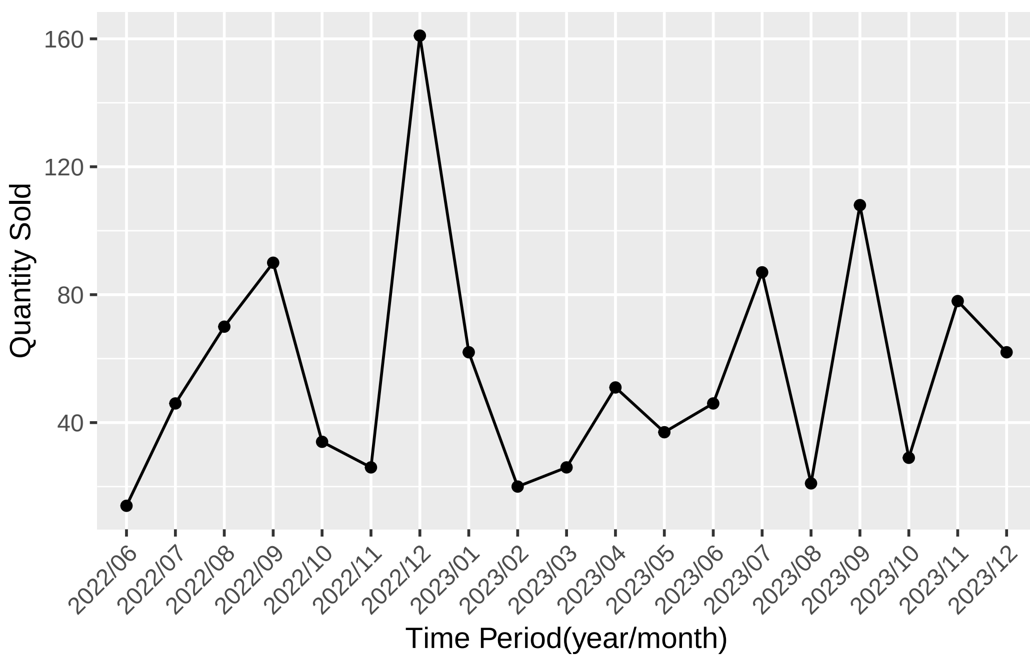


Figure 4: Time Series Graph to depict the seasonal trends in sales

Figure 4 helps us understand the purchase patterns of the customers over different periods of time. Purchase pattern analysis is significant for an e-commerce company as different strategies could be deployed to maximize sales and revenue. Figure 1 illustrates that the sales were maximum in September and December months over the time period from 2022 to 2023. Hence, appropriate pricing strategies could be deployed to increase revenue from top-selling products. Also, a sharp decline in sales was identified in October, which was later followed by an increase in succeeding months. This study could help maintain efficient inventory management for perishable products. Overall, this variation in sales across different seasons provides important insights for better management decisions.

```
#Report 4:

#SQL query to join customer and order tables to get the sales for each city

cust_order_join<- RSQLite::dbGetQuery(connect,
                                      'SELECT city,ORDITM.order_quantity
                                      FROM ADDRESS ADR
                                      INNER JOIN CUSTOMER CUST
                                      ON ADR.address_id= CUST.address_id
                                      INNER JOIN ORDER_DETAIL ORDDET
                                      ON ORDDET.customer_id = CUST.customer_id
                                      INNER JOIN ORDER_ITEM ORDITM
                                      ON ORDITM.order_id = ORDDET.order_id')

#cust_order_join<- inner_join(Customer,Order,by="customer_id")

#Grouping by city to get the overall sales per city

sales_per_region<- cust_order_join %>% group_by(city)%>%
  summarise(quantity=sum(order_quantity))

#Get the top 10 cities for sales
top_10_sales_per_region <- sales_per_region[order(-sales_per_region$quantity),]

top_10_sales_per_region <- head(top_10_sales_per_region,10)

#Bar graph to show the sales per region

ggplot1 <- ggplot(top_10_sales_per_region,
aes(x = reorder(city,-quantity), y = quantity,fill=quantity)) +
geom_bar(stat='identity') +
scale_fill_gradient(low = "lightblue", high = "darkblue")+
xlab('Top 10 cities where sales is maximum')+ylab('Quantity Sold')+
theme(axis.text.x=element_text(angle=45,
hjust = 1))

#Table to get the revenue per city

cust_order_product_join <-RSQLite::dbGetQuery(connect,
'SELECT city,ORDITM.order_quantity,unit_price
FROM ADDRESS ADR
INNER JOIN
```

```
CUSTOMER CUST ON ADR.address_id= CUST.address_id
INNER JOIN
ORDER_DETAIL ORDDET ON ORDDET.customer_id = CUST.customer_id
INNER JOIN
ORDER_ITEM ORDITM ON ORDITM.order_id = ORDDET.order_id
INNER JOIN
PRODUCT PRD ON PRD.product_id = ORDITM.product_id')

#SQL Query to get the top 5 products

top_5_products <-RSQLite::dbGetQuery(connect,
'SELECT city,ORDITM.order_quantity,unit_price,
PRD.product_id,PRD.product_name FROM
ADDRESS ADR INNER JOIN
CUSTOMER CUST ON ADR.address_id= CUST.address_id
INNER JOIN
ORDER_DETAIL ORDDET ON ORDDET.customer_id = CUST.customer_id
INNER JOIN
ORDER_ITEM ORDITM ON ORDITM.order_id = ORDDET.order_id
INNER JOIN
PRODUCT PRD ON PRD.product_id = ORDITM.product_id')

#Revenue calculation

cust_order_product_join$revenue<- cust_order_product_join$order_quantity*
cust_order_product_join$unit_price

#Grouping by the get the revenue for each city
revenue_per_region <- cust_order_product_join %>% group_by(city) %>%
  summarise(revenue= sum(revenue))

#Get the top 10 cities for revenue

top_5_revenue_per_region <-
revenue_per_region[order(-revenue_per_region$revenue),]

top_5_revenue_per_region <- head(top_5_revenue_per_region,10)

#Bar graph to show the revenue per region

ggplot2 <- ggplot(top_5_revenue_per_region,aes(x = reorder(city, -revenue),
y=revenue,fill=revenue))+
```

```r
geom_bar(stat='identity')+
scale_fill_gradient(low = "lightblue", high = "darkblue")+
xlab('Top 10 cities with the maximum revenue')+ylab('Revenue')+
theme(axis.text.x=element_text(angle=45,hjust = 1))



#Revenue calculation to show the top 5 products for each city
#with maximum revenue

top_5_products$revenue <-
top_5_products$order_quantity*top_5_products$unit_price

top_5_products_region <- top_5_products %>% group_by(city,product_name) %>%
summarise(rev= sum(revenue))

#Table to show the products that are
#sold for the top 2 revenue producing cities
(top_5_products_region <- top_5_products_region %>%
filter(city %in% head(top_5_revenue_per_region$city,2))
%>% select(product_name))
```

```
# A tibble: 24 x 2
# Groups:   city [2]
   city       product_name
   <chr>      <chr>
 1 Birmingham Cosmic Compression Socks
 2 Birmingham Essence Room Diffuser
 3 Birmingham FusionX Gaming Console
 4 Birmingham Moonbeam Cardigan
 5 Birmingham Nebula Scarf
 6 Birmingham Solar Flare Sweater
 7 Birmingham Sonic Boom Portable Speaker
 8 Birmingham Sonic Boom Resistance Bands
 9 Birmingham Stellar Eyeshadow Palette
10 Birmingham Tranquility Scented Candles
# i 14 more rows
```

```r
grid.arrange(ggplot1,ggplot2)
```

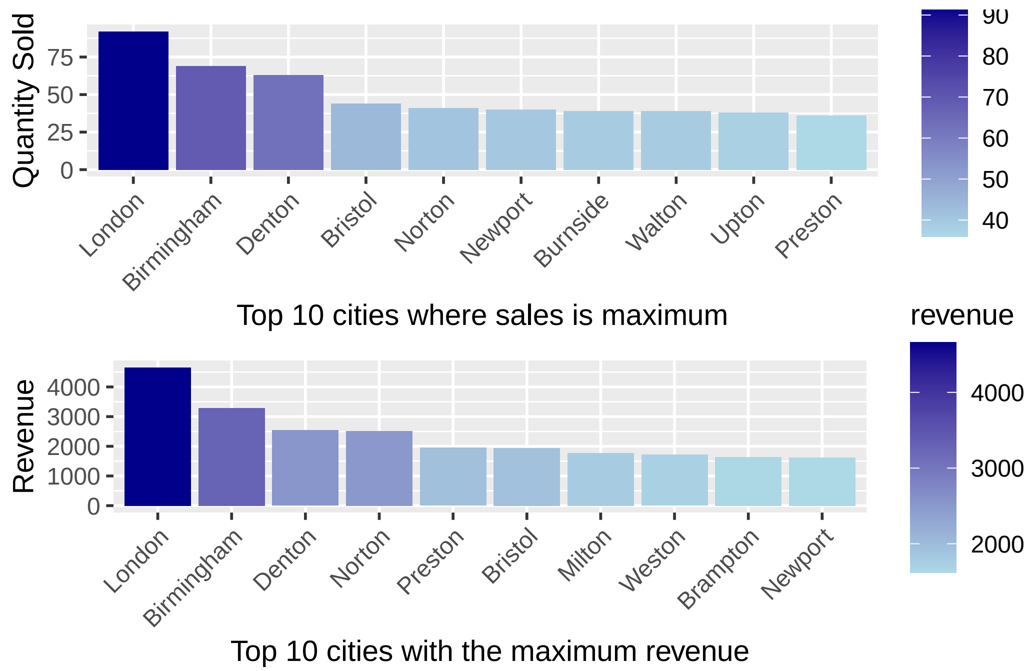


Figure 5: Top 10 cities with maximum revenue and sales

Figure 5 gives a crucial insight into cities that generated significant revenue with a lesser quantity of products sold. Top Products from these cities could be identified as shown in Table 1, as their sales could primarily improve revenue. Also, cities like Ford, Manchester, and Milton generated outstanding revenue despite limited sales. This indicates that customers in this region had a good willingness to pay, which could be capitalised by targeted marketing.

```
#Report 5
# Joining Product with advertise_in and advertisement to get advertisement
#details
advertise_in <- RSQLite::dbGetQuery(connect,'SELECT * FROM ADVERTISE_IN')
advertisement <- RSQLite::dbGetQuery(connect,'SELECT * FROM ADVERTISEMENT')

advertisement_data <- product %>%
  inner_join(advertise_in, by = "product_id") %>%
  inner_join(advertisement, by = "ad_id") %>%
  group_by(product_id, product_name, ad_place) %>%
  summarise(
    total_frequency = sum(ad_frequency),
    .groups = 'drop' )


# Joining product with order to get the number of sales per product
```

```r
number_of_sales <- product %>%
  inner_join(order_item, by = "product_id") %>%
  group_by(product_id, product_name) %>%
  summarise(sales_count = n(), .groups = 'drop')



merged_data <- merge(number_of_sales, advertisement_data, by =
c("product_id", "product_name"))



# Analyzing which ad place is most effective by calculating a
#ratio of total sales to total ad frequency
effective_ad_type <- merged_data %>%
  group_by(ad_place) %>%
  summarise( total_sales = sum(sales_count),
             total_frequency = sum(total_frequency), .groups = 'drop') %>%
  mutate( effectiveness = total_sales / total_frequency) %>%
  arrange(desc(effectiveness))




ggplot(effective_ad_type, aes(x = ad_place, y = effectiveness,
fill = ad_place))+geom_col(show.legend = FALSE, width = 0.5) +
# Adjust bar width here
scale_fill_brewer(palette = "Paired") +
# Use a more appealing color palette
labs(title = "Effectiveness of Advertisement Types",
x = "Ad Place",
y = "Total Sales / Total Frequency") +
theme_minimal(base_size = 14) +
# Increase base text size for better readability
theme(plot.title = element_text(hjust = 0.5, size = 20),
# Center and style title
axis.title = element_text(size = 16),
# Style axis titles
axis.text = element_text(size = 12),
# Style axis texts
panel.grid.major = element_blank(),
# Remove major grid lines
```

```
panel.grid.minor = element_blank(),
# Remove minor grid lines
panel.background = element_rect(fill = "white", colour = "grey50"))
```

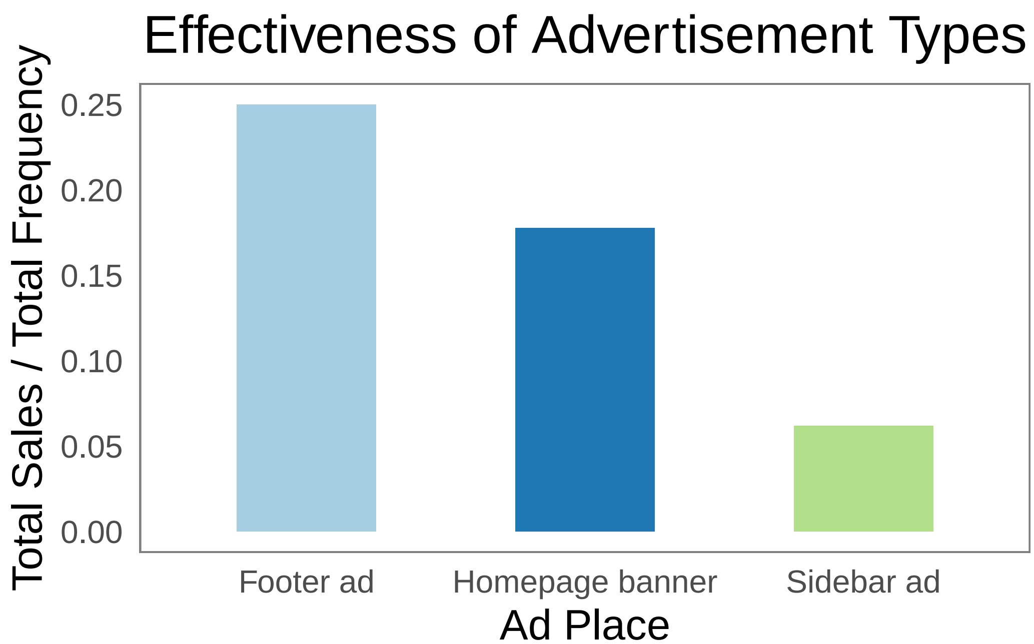


Figure 6: Sales of products per 1 advertisement of each type

We analysed 3 types of advertisements on the Marketplace to find that Products that are advertised with the Footer ads, on average, have higher sales in units. The intention was to compare the number of units sold depending on the type of advertisement this product was in. This finding can be used to assign a price for ads to be charged from suppliers.

```
#Report 6

## Calculating Average Review for each category

category <- RSQLite::dbGetQuery(connect,'SELECT * FROM CATEGORY')

# Join product_df with category_df to include category names
product_with_category <- product %>%
  inner_join(category, by = "category_id")


avg_rating_by_category <- product_with_category %>%
```

```
  group_by(category_id, category_name) %>%
  summarise(Average_Rating = round(mean(product_rating, na.rm = TRUE),2),
            .groups = 'drop') %>%
  arrange(category_id)


#Calculating Marketplace fee
merged_product_fee <- product %>%
  inner_join(select(category, category_id, category_fee, category_name),
             by = "category_id")
category_fee <- order_item %>%
  inner_join(select(order_detail, order_id, order_status), by = "order_id") %>%
  inner_join(select(merged_product_fee, product_id, product_name, unit_price,
                    category_fee, category_id, category_name),
             by = "product_id") %>%
  filter(order_status == "Completed") %>%
  mutate(marketplace_fee = order_quantity * unit_price * category_fee/100) %>%
  mutate(cat_total_sales = order_quantity * unit_price) %>%
  group_by(category_id, category_name) %>%
  summarise(total_fee = sum(marketplace_fee),
            total_sales = sum(cat_total_sales),
            total_sales_unit = sum(order_quantity)) %>%
  inner_join(select(avg_rating_by_category, category_id, Average_Rating),
               by= 'category_id')


#Visualise


#graph to compare category sales vs category fee
ggplot(category_fee, aes(x = reorder(category_name, -total_fee))) +
  geom_bar(aes(y = total_sales, fill = "Total Sales"), stat = "identity",
           position = position_dodge(width = 0.9), alpha = 0.7) +
  geom_bar(aes(y = total_fee, fill = "marketplace fee"), stat = "identity",
           position = position_dodge(width = 0.9)) +
  scale_fill_manual(name = "category", values =
                      c("marketplace fee" = "#3182bd",
                        "Total Sales" = "#31a354")) +
  labs(title = "Marketplace Fee vs Total Sales by Category", x = "Category",
       y = "Amount, GBP") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

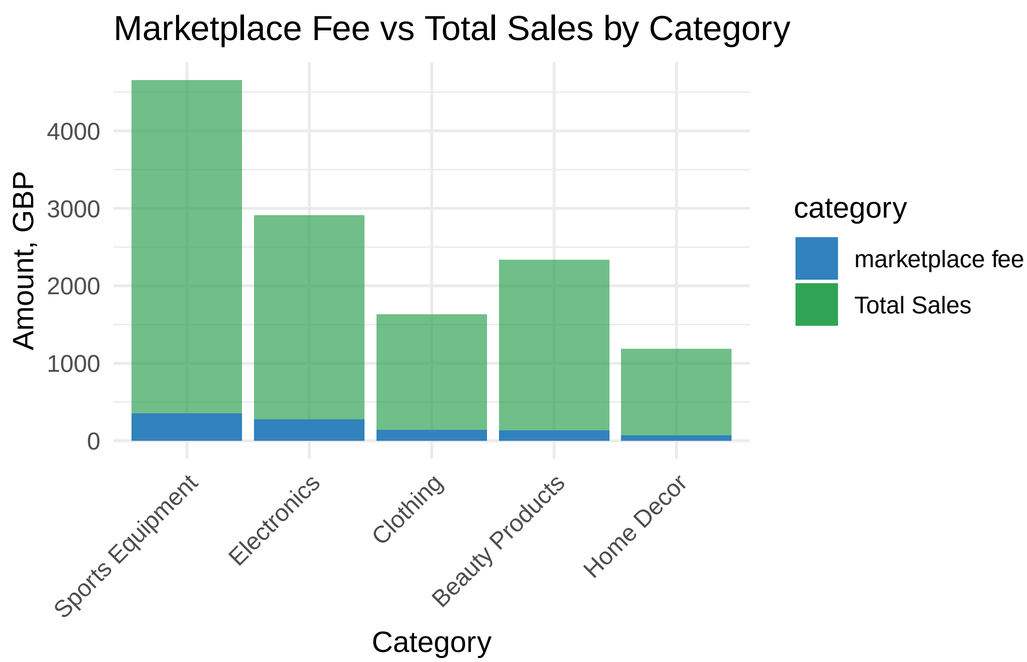


Figure 7: Comparing Marketplace fees to the sales in each Category of products

High sales in the category relate to a higher Marketplace fee. Increasing fees for the categories with higher sales and decreasing fees for the categories with low sales will maximise Marketplace profit by selling more in less popular categories and getting higher fees for already popular categories. Sports Equipment has the highest sales in terms of money. However, as we will see later, most units are sold in the Electronics category.

```
#Report 7
# Graph to compare Category sales VS Category average rating
ggplot(category_fee, aes(x = reorder(category_name, -Average_Rating))) +
  geom_bar(aes(y = total_sales_unit, fill = "Total Sales in Units"),
           stat = "identity", position = position_dodge(width = 0.9),
           alpha = 0.6) +
  geom_bar(aes(y = (Average_Rating-3)*20, fill = "Average Rating"),
           stat = "identity", position = position_dodge(width = 0.9),
           alpha = 1) +
  geom_text(aes(label = sprintf("%.2f", Average_Rating),
                y = Average_Rating, x = category_name),
            color = "black", size = 4, hjust = 0.5) +
  scale_fill_manual(name = "Category", values =
                      c("Average Rating" = "#FFCC80",
                        "Total Sales in Units" = "#31a354")) +
  labs(title = "Average Rating vs Total Sales in Units by Category",
```

```
      x = "Category", y = "Total Sales in Units") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  scale_y_continuous(name = "Total Sales in Units")
```

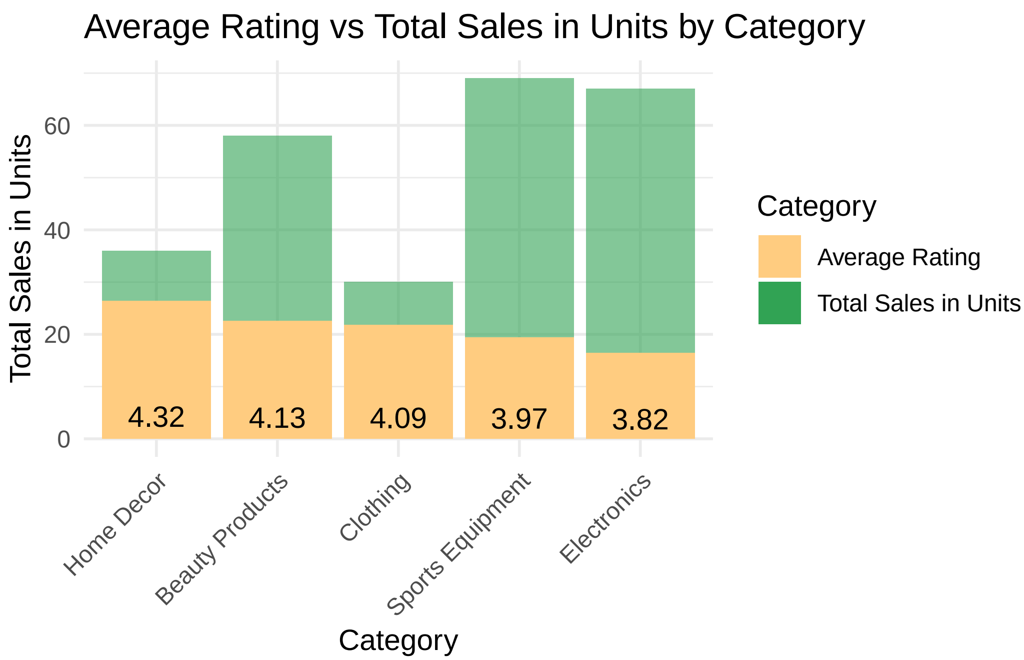


Figure 8: Average customers' rating for the category and category's sales in units

Categories that have products with higher ratings have lower sales in units. This means that with the increase in sales, the number of negative reviews increases at a higher rate. Marketplace may want to reconsider suppliers for the categories with lower average ratings.