WARWICK BUSINESS SCHOOL
THE UNIVERSITY OF WARWICK

Masters Programmes: Group Assignment Cover Sheet

Student Numbers: Please list numbers of all group members	**2289524, 5504970, 5516864, 5536699, 5537122, 5565439, 5579338**
Module Code:	**IB9HP0**
Module Title:	**Data Management**
Submission Deadline:	**12:00 (UK time) 20th March 2024**
Date Submitted:	**19 March 2024**
Word Count:	**2000**
Number of Pages:	**54**
Question Attempted: *(question number/title, or description of assignment)*	**4**
Have you used Artificial Intelligence (AI) in any part of this assignment?	**Yes**

Academic Integrity Declaration

We're part of an academic community at Warwick. Whether studying, teaching, or researching, we're all taking part in an expert conversation which must meet standards of academic integrity. When we all meet these standards, we can take pride in our own academic achievements, as individuals and as an academic community.

Academic integrity means committing to honesty in academic work, giving credit where we've used others' ideas and being proud of our own achievements.

In submitting my work, I confirm that:

- I have read the guidance on academic integrity provided in the Student Handbook and understand the University regulations in relation to Academic Integrity. I am aware of the potential consequences of Academic Misconduct.
- I declare that this work is being submitted on behalf of my group and is all our own, , except where I have stated otherwise.
- No substantial part(s) of the work submitted here has also been submitted by me in other credit bearing assessments courses of study (other than in certain cases of a resubmission of a piece of work), and I acknowledge that if this has been done this may lead to an appropriate sanction.
- Where a generative Artificial Intelligence such as ChatGPT has been used I confirm I have abided by both the University guidance and specific requirements as set out in the Student Handbook and the Assessment brief. I have clearly acknowledged the use of any generative Artificial Intelligence in my submission, my reasoning for using it and which generative AI (or AIs) I have used. Except where indicated the work is otherwise entirely my own.
- I understand that should this piece of work raise concerns requiring investigation in relation to any of points above, it is possible that other work I have submitted for assessment will be checked, even if marks (provisional or confirmed) have been published.
- Where a proof-reader, paid or unpaid was used, I confirm that the proof-reader was made aware of and has complied with the University's proofreading policy.

Upon electronic submission of your assessment you will be required to agree to the statements above

Data Managment Report

Simulating Real-World E-Commerce

Table of contents

1. Introduction

This comprehensive project tries to simulate a real-world e-commerce environment using end-to-end data management. It is divided into four parts: database design and implementation, data generation and administration, data pipeline development, and data analysis and reporting. The project begins by creating a thorough Entity-Relationship (E-R) diagram, which is then translated into a functional SQL database structure. Synthetic data production then occurs, assuring accurate simulations of e-commerce activity. GitHub Actions and repository setup make it easier to automate

and manage versions. For the last step, extensive data analysis with R and thorough reporting in Quarto deliver useful insights to stakeholders.

The described firm operates as an e-commerce company based in the United Kingdom, serving diverse retail categories such as "Appliances", "Beauty & Grooming", "Books", "Computing", "Entertainment", "Health & Sports", "Home & Living", "Kids & Baby", "Men's Fashion", "Mobiles & Tablets", "School & Education", "Superstore", and "Women's Fashion". The company offers subscription services to customers, comprising premium and VIP tiers, providing exclusive shipping and discount benefits. Orders are fulfilled through third-party logistics partners, with a primary focus on the entertainment and electronics sectors.

2. Database Design and Implementation

2.1. Introduction of Database Design

After an extensive review of existing ER diagrams related to e-commerce dataset and pipelines, a customized ER diagram comprising six essential entities was designed. The attributes, primary keys, relationships, and cardinalities were defined to accurately represent the dynamics of the e-commerce environment. The conceptual ER diagram was converted into logical and physical schemas, ensuring alignment with data management and database implementation principles.

2.2. E-R Diagram and Relationship Set

The e-commerce data model comprises six entities — 'buyer', 'seller', 'product', 'review', 'contact_detail' and 'category'. The relationships between them are as follows.

1. Buyer and Contact Detail: A one-to-many relationship exists between buyers and their contact details, reflecting the multiple correspondences between a buyer and their associated contact information.
2. Buyer and Buyer: One buyer can refer several other buyers to join the membership, indicating a one-to-many relationship.
3. Buyer and Review: Buyers can provide multiple reviews, establishing a one-to-many relationship between buyers and reviews.
4. Buyer and Product: Buyers can order multiple products and vice versa indicating a many to many relationship.
5. Product and Category: This represents a many-to-one relationship, signifying that a product can belong to only one category while a category can contain multiple products.
6. Product and Review: Products can receive multiple reviews, establishing a one-to-many relationship between products and reviews.
7. Seller and Product: A seller can provide multiple products while a product is typically associated with a single seller, implying a one-to-many relationship between sellers and products.

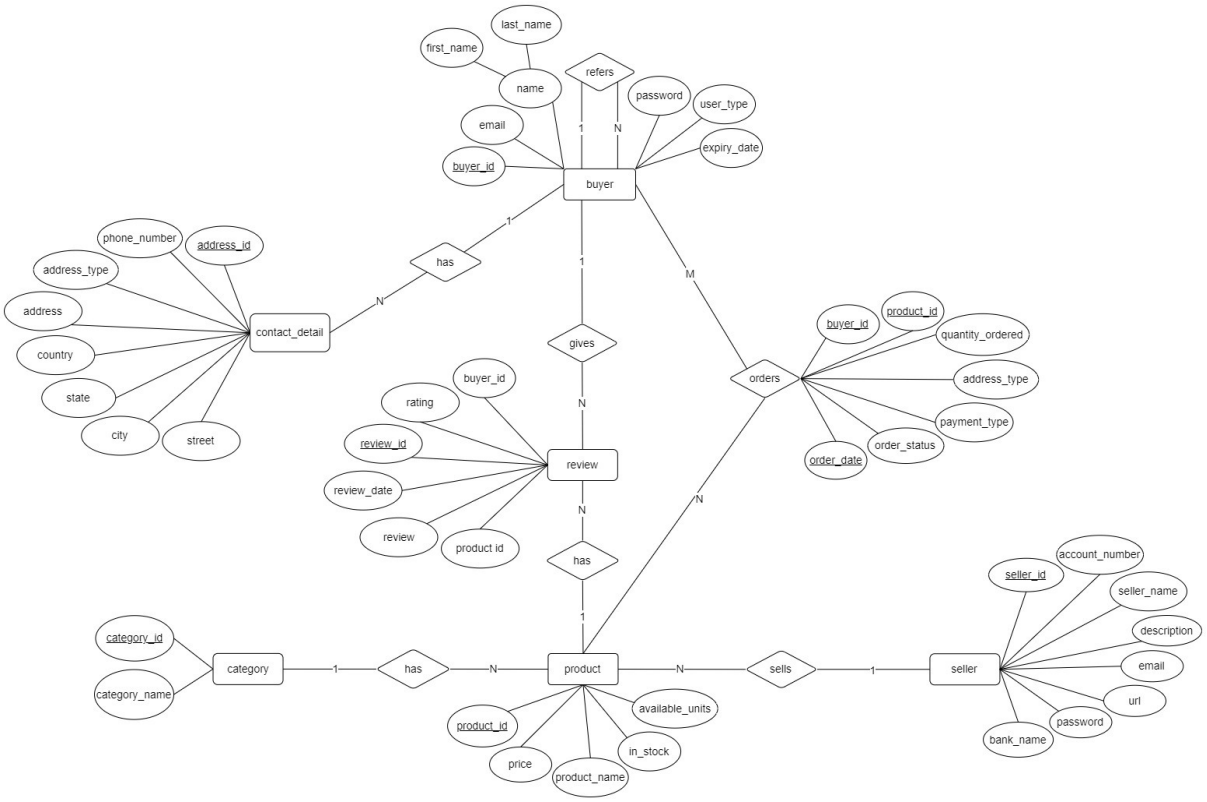

Figure 1: E-R Diagram

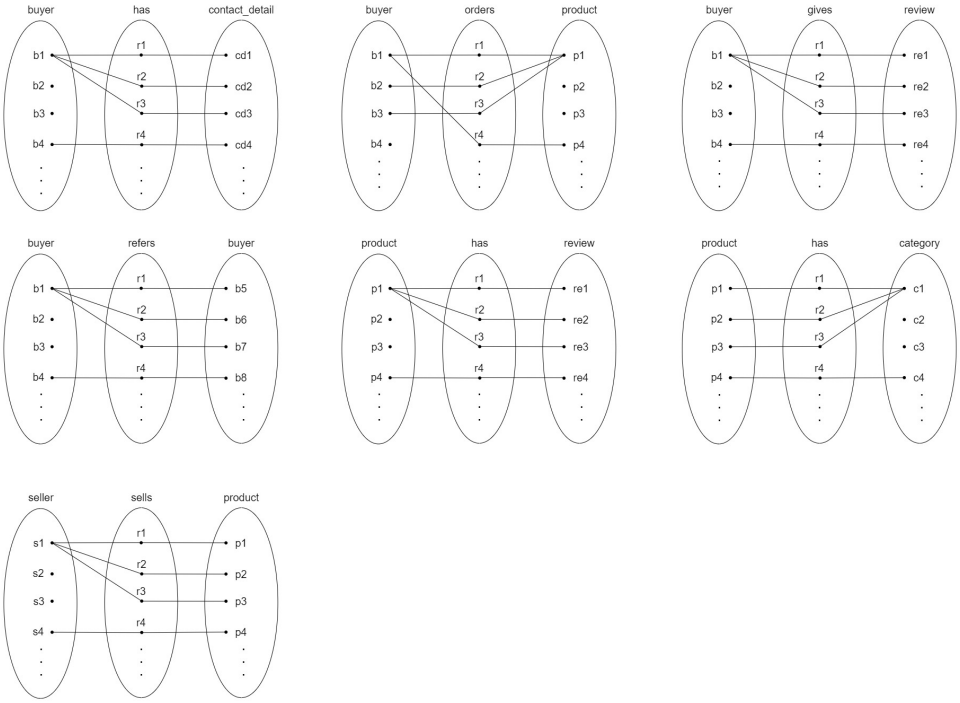

Figure 2: Relationship Sets

2.3. Logical Schema

The following is the logical schema of entities, primary keys are underlined and foreign keys are provided in italics.

- **buyer** (buyer_id, first_name, last_name, email, password, user_type, expiry_date)
- **product** (product_id, product_name, in_stock, available_units, price, *seller_id*, *category_id*)
- **seller** (seller_id, seller_name, description, email, url, password, account_number, bank_name)
- **category** (category_id, category_name)
- **contact_detail** (address_id, address, country, state, city, street, phone_number, address_type, *buyer_id*)
- **review** (review_id, rating, review, review_date, *product_id*, *buyer_id*)

The logical schema of relationship tables can be seen as below. The key attributes are provided in italics.

- reference (*buyer_id*, referred_by)
- order (*product_id*, quantity_ordered, *order_date*, order_status, payment_type, address_type, *buyer_id*)

2.4. Physical Schema

The following physical schema creates the table according to the ER diagram and decides on suitable data types for each attribute in the tables based on the nature of the data they will store.

```
# Load necessary libraries
library(readr)
library(RSQLite)
library(dplyr)
library(tidyverse)
library(sf)
library(rnaturalearth)
library(ggplot2)
library(rnaturalearthdata)
```

```
# Create a database connection
connection <- RSQLite::dbConnect(RSQLite::SQLite(),"database/group32.db")

## Products Table
dbExecute(connection, "
  CREATE TABLE IF NOT EXISTS 'project_products' (
  'product_id' VARCHAR(255) PRIMARY KEY,
```

```
    'seller_id' VARCHAR(10) NOT NULL,
  'category_id' VARCHAR(10) NOT NULL,
  'product_name' TEXT NOT NULL,
  'in_stock' BIT NOT NULL DEFAULT 0,
  'available_units' INT NOT NULL DEFAULT 0,
  'price' MONEY NOT NULL CHECK (price > 0),
  FOREIGN KEY ('seller_id') REFERENCES seller('seller_id'),
  FOREIGN KEY ('category_id') REFERENCES category('category_id')
);
")
```

We create the product table and assign product_id as the primary key. seller_id and category_id are foreign keys that reference the seller table and category table. All the columns in the product table should not contain any null value since these will all be necessary information for the e-commerce system operation. Also, the in_stock columns will be binary to indicate whether a product currently has stock.

```
## Create Seller Table
dbExecute(connection, "
CREATE TABLE IF NOT EXISTS 'project_seller' (
'seller_id' VARCHAR(10) PRIMARY KEY,
'seller_name' TEXT NOT NULL,
'url' VARCHAR(255),
'description' TEXT,
'email' VARCHAR(255) UNIQUE,
'password' VARCHAR(255),
'account_number' VARCHAR(255),
'bank_name' TEXT
);
")
```

The seller table has seller_id as the primary key. The email address column has UNIQUE constraints. This prevents duplicate email addresses for different sellers.

```
## Create category Table
dbExecute(connection, "
CREATE TABLE IF NOT EXISTS 'project_categories' (
'category_id' VARCHAR(255) PRIMARY KEY,
'category_name' TEXT NOT NULL
);
")
```

The category table contains category_id as the primary key.

```
## Create buyer Table
dbExecute(connection, "
  CREATE TABLE IF NOT EXISTS 'project_buyer' (
  'buyer_id' VARCHAR PRIMARY KEY,
  'first_name' TEXT NOT NULL,
  'last_name' TEXT,
  'email' VARCHAR NOT NULL UNIQUE,
  'password' VARCHAR NOT NULL,
  'user_type' TEXT,
  'expiry_date' DATE
);
")
```

The buyer table has buyer_id as the primary key. The expiry date should be in the DATE type.

```
## Create contact details Table
dbExecute(connection, "
  CREATE TABLE IF NOT EXISTS 'project_contact_details' (
  'address_id' VARCHAR PRIMARY KEY,
   'buyer_id' VARCHAR,
  'address' VARCHAR,
  'country' TEXT,
  'state' TEXT,
  'city' TEXT,
  'street' VARCHAR,
  'phone_number' VARCHAR(10),
  'address_type' TEXT,
  FOREIGN KEY ('buyer_id') REFERENCES buyer ('buyer_id')
);
")
```

We create the contact detail table and assign address_id as the primary key and buyer_id as the foreign key references to the buyer table. Also, we optimise the storage by specifying the appropriate length of phone_number.

```
## Create Review Table
dbExecute(connection, "
  CREATE TABLE IF NOT EXISTS 'project_review' (
  'review_id' VARCHAR PRIMARY KEY,
    'product_id' VARCHAR,
  'buyer_id' VARCHAR,
  'rating' INT CHECK (rating >= 1 AND rating <= 5),
  'review' TEXT,
  'review_date' DATE,
   FOREIGN KEY ('buyer_id') REFERENCES buyer ('buyer_id'),
   FOREIGN KEY ('product_id') REFERENCES products ('product_id')
```

```
);
")
```

The review table has the primary key review_id and two foreign keys, buyer_id and product_id. The rating field should be an integer, and we set the constraint to accept only ratings within the appropriate range.

```
#Create relationship table project_buyer_orders_products
dbExecute(connection, "
CREATE TABLE IF NOT EXISTS 'project_buyer_orders_products' (
  'buyer_id' VARCHAR(255),
  'product_id' VARCHAR(255),
  'quantity_ordered' INT NOT NULL DEFAULT 1,
  'order_date' DATE,
  'delivery_date' DATE,
  'order_status' TEXT,
  'payment_type' TEXT,
  'address_type' TEXT,
  FOREIGN KEY ('product_id') REFERENCES products ('product_id'),
  FOREIGN KEY ('buyer_id') REFERENCES buyer ('buyer_id')
);
")
```

In the relationship table of buyer orders products, we defined the foreign key constraints referencing the buyer and product table. The quantity_ordered attribute would be an integer and defaulting 1. The order date is DATE type, and the rest are TEXT data.

```
# Create relationship table references
dbExecute(connection, "
CREATE TABLE IF NOT EXISTS 'project_references' (
  'buyer_id' VARCHAR(255),
  'referred_by' VARCHAR(255),
   FOREIGN KEY ('buyer_id') REFERENCES buyer ('buyer_id')
);
")
```

For the reference table, we set up foreign key constraints referencing the buyer table for buyer_id. The referred_by contains the corresponding buyer's ID that referred this buyer to the platform, thus should be VARCHAR.

3. Data Generation and Management

3.1. Synthetic Data Generation

In order to facilitate the Data Manipulation Language (DML) process, synthetic data was generated for each entity, as well as for the M:N relationship in accordance with the physical schema.

```r
# Function to install and load required packages
install_and_load <- function(package) {
  if (!requireNamespace(package, quietly = TRUE)) {
    install.packages(package)
  }
  library(package, character.only = TRUE)
}

# List of necessary packages
packages <- c("dplyr", "lubridate", "stringi", "digest",
              "randomNames", "readr", "readxl", "curl")

# Install and load necessary libraries
sapply(packages, install_and_load)
```

3.1.1. Category Entity

The process commenced by initially identifying a reference dataset in the e-commerce domain. We assume that our e-commerce company is based in the UK. Therefore, once we extracted the relevant categories from the reference dataset, we allocated each one a distinct category id, which was coded to guarantee uniqueness for each id. The process of generating IDs for our entities consistently followed a similar structure, with the aid of ChatGPT.

Prompt in ChatGPT: I want to generate unique identification codes
for my category entity. The code should start with the name of the entity,
followed by 1912000001, with each category adding on to previous id number, and
an additional 3 random digits. The total number of category id's is a parameter
set by me. Provide me the R code.

```r
# Our reference data is this e-commerce dataset
# From kaggle: https://www.kaggle.com/datasets/zusmani/pakistans-largest-ecommerce-dataset


# Define the categories as per the above dataset
# Eexctracted using excel for easier computing as the file is large

categories <- c("Appliances", "Beauty & Grooming", "Books", "Computing",
                "Entertainment", "Health & Sports", "Home & Living",
                "Kids & Baby", "Men's Fashion", "Mobiles & Tablets",
                "School & Education", "Superstore", "Women's Fashion")
# Let's generate primary keys for categories.
# This key style is replicated for all other entities
# Starting with the first letter of the entities
# With a random start id manually set
# And a 3 digit randomly generated number joined at the end

num_categories <- length(categories)  # Number of categories
```

```
start_id_cat <- 1912000001  # Starting ID
set.seed(123)  # Setting seed for reproducibility
random_digits_cat <- sample(100:999, num_categories, replace = TRUE)
# Generate random three-digit numbers

# Create unique category IDs with random digits
category_ids <- paste0("C", start_id_cat:(start_id_cat + num_categories - 1),
                       random_digits_cat)

# Create a data frame with category IDs
categories_df <- data.frame(
  category_id = category_ids,
  category_name = categories
)

categories_df <- categories_df
```

3.1.2. Seller Entity

The next step was to create an entity table for the sellers. The seller names were created by combining adjectives and nouns, as well as phrases for the seller description, using ChatGPT and the prompt sequence shown below.

Prompt in ChatGPT: I want to generate 200 row of data for my seller entity. List 50 nouns, 50 adjectives, and 50 phrases like "seller of, provider of". Then use a combination of nouns and adjectives for names, and use the phrases to describe the categories they sell. The categories sold should be stored as a list for each seller_id. Also provide email, bank name (top 5 UK banks), 8 digit account number, url, password.

To ensure 3NF is not violated, we assume that the seller's name is not unique.

```
# Define the number of seller IDs to generate and starting ID
num_sellers <- 200 # Assuming our company deals with 200 suppliers
start_id_seller <- 1212000001

# Generate random three-digit numbers for seller IDs
set.seed(123)
random_digits_seller <- sample(100:999, num_sellers, replace = TRUE)

# Generate unique seller IDs by combining random digits with starting ID
seller_ids <- paste0("S", start_id_seller:(start_id_seller + num_sellers - 1),
                     random_digits_seller)

# Define the categories for sellers
categories <- c("Appliances", "Beauty & Grooming", "Books", "Computing",
                "Entertainment", "Health & Sports", "Home & Living", "Kids & Baby",
                "Men's Fashion", "Mobiles & Tablets", "School & Education",
```

```
                "Superstore", "Women's Fashion")

# Define adjectives, nouns, and phrases for seller names and descriptions
adjectives <- c("Ultimate", "Supreme", "Global", "Innovative", "Reliable", "Trusted",
                "Eco-friendly", "Dynamic", "Quality", "Premier",
                "Luxury", "Fantastic", "Special", "Premium", "Advanced", "Modern",
                "Smart", "Elegant", "Sleek", "Sustainable",
                "Unique", "Chic", "Stylish", "Vibrant", "Creative", "Dynamic",
                "Innovative", "Elite", "Dynamic",
                "Exquisite", "Fashionable", "Flawless", "Fresh", "Glamorous",
                "Harmonious", "Inspiring", "Majestic", "Opulent",
                "Perfect", "Prime", "Radiant", "Splendid", "Timeless", "Trendy",
                "Upscale", "Vintage", "Wholesome")

nouns <- c("Solutions", "Goods", "Depot", "Distributors", "Merchants", "Traders",
           "Emporium", "Warehouse", "Boutique", "Outfitters",
           "Collections", "Market", "Stores", "Deals", "Corner", "Hub", "Shops",
           "Outlet", "Mart", "Spot",
           "Heaven", "Lab", "Center", "World", "Galaxy", "Palace", "Empire", "Castle",
           "Fortress", "Nest",
           "Grove", "Station", "Base", "Haven", "Hive", "Coast", "Bazaar", "Club",
           "Cave", "Sanctuary",
           "Arena", "Tower", "Mansion", "Forum", "Place", "Square", "Plaza", "Room",
           "Territory")

# List of phrases to generate descriptions
phrases <- c("A seller of a variety of products ranging from: ",
             "A seller that deals with: ",
             "A seller that provides: ",
             "Your destination for finding: ",
             "Specializing in: ",
             "Bringing you the best in: ",
             "Your one-stop shop for: ",
             "Discover the world of: ",
             "Where quality meets: ",
             "Explore our selection of: ",
             "Elevating your experience with: ",
             "Unlocking the potential of: ",
             "Experience the magic of: ",
             "Embrace the beauty of: ",
             "Unleash your passion for: ",
             "Connecting you to: ",
             "Where innovation thrives: ",
             "Fulfilling your needs for: ",
             "Weaving dreams with: ",
             "Crafting excellence with: ",
             "Shaping the future with: ",
```

```
              "Where style meets substance: ",
              "Transforming lives through: ",
              "Empowering you with: ",
              "Inspiring greatness with: ",
              "Building a world of: ",
              "Pioneering new frontiers in: ",
              "Where tradition meets: ",
              "Where dreams become reality: ",
              "Where comfort meets: ",
              "Enabling your journey with: ",
              "Enhancing your lifestyle with: ",
              "Curating the best of: ",
              "Where elegance meets: ",
              "Where passion meets: ",
              "Crafting memories with: ",
              "Where happiness resides in: ",
              "Setting the standard with: ",
              "Elevating everyday life with: ",
              "Where luxury meets: ",
              "Bringing joy through: ",
              "Creating wonders with: ",
              "Nurturing growth with: ",
              "Where beauty blossoms in: ",
              "Forging connections through: ",
              "Unveiling the beauty of: ",
              "Capturing the essence of: ",
              "Crafting dreams with: ",
              "Where adventure awaits in: ",
              "Discovering treasures in: ",
              "Finding serenity in: ",
              "Where elegance reigns in: ",
              "Crafting experiences with: ",
              "Crafting a legacy with: ",
              "Unleashing potential through: ")

# Helper function to generate random string
generate_random_string <- function(length) {
  return(paste0(sample(c(letters, LETTERS, 0:9), length, replace = TRUE),
                collapse = ""))
}

# Helper function to generate readable name without category
generate_readable_name <- function() {
  return(paste(sample(adjectives, 1), sample(nouns, 1)))
}

# Helper function to generate description
```

```r
generate_description <- function(categories) {
  return(paste(sample(phrases, 1), paste(categories, collapse=', '), "."))
}

# Helper function to select random categories
select_categories <- function() {
  return(sample(categories, sample(1:3, 1)))
}

# Define the bank names of suppliers (popular banks in the UK)
bank_names <- c("Natwest", "Barclays", "HSBC", "Santander", "Lloyds")

# Generate random data for 200 sellers
set.seed(123) # for reproducibility

# Generate seller names without category
seller_names <- replicate(200, generate_readable_name(), simplify = TRUE)

# Create the dataframe with seller names
sellers_df <- data.frame(
  seller_id = seller_ids,
  seller_name = seller_names,
  stringsAsFactors = FALSE
)


email_vendors <- c("gmail.com", "yahoo.com", "outlook.com", "hotmail.com", "aol.com")

# Generate random email addresses with popular vendors
emails_sellers <- paste0(seller_names, "@", sample(email_vendors, 200, replace = TRUE))

# Now, generate the rest of the variables and merge them with the existing dataframe
sellers_df$url <- sapply(1:200, function(x) paste0("https://www.",
                                                   gsub(" ", "",
                                                        seller_names[x]
                                                        ), generate_random_string(3),
                                                   ".com"))
sellers_df$categories <- replicate(200, select_categories(), simplify = TRUE)
sellers_df$description <- mapply(generate_description, sellers_df$categories)
sellers_df$email <- emails_sellers
sellers_df$password <- sapply(1:200, function(x) digest(generate_random_string(8),
                                                        algo = "sha256"))
sellers_df$account_number <- sapply(1:200, function(x) paste0(sample(0:9, 8,
                                                                     replace = TRUE),
                                                              collapse = ""))
sellers_df$bank_name <- sample(bank_names, 200, replace = TRUE)
```

```r
# View the first few rows of the sellers dataframe
head(sellers_df)
```

3.1.3. Buyer Entity

The buyer entity was created in a similar manner.

Prompt in ChatGPT: Generate 1000 unique buyer id as per previous code, and add first name, last name, email, password, user type (Basic, Premium, VIP), and expiry date set a year from now.

```r
# Define the number of buyer IDs to generate and starting ID
num_buyers <- 800
start_id_buyer <- 1312000001

# Generate random three-digit numbers for buyer IDs
set.seed(123)  # Ensuring reproducibility
random_digits_buyer <- sample(100:999, num_buyers, replace = TRUE)

# Generate unique buyer IDs by combining random digits with starting ID
buyer_ids <- paste0("B", start_id_buyer:(start_id_buyer + num_buyers - 1),
                    random_digits_buyer)

# Load necessary package to generate names
# install.packages("babynames")
library(babynames)

# Generate 1000 unique first names
first_names <- character(0)
while (length(first_names) < 800) {
  first_names <- unique(c(first_names, sample(babynames::babynames$name, 800,
                                              replace = TRUE)))
}
first_names <- head(first_names, 800)

# Generate 1000 unique last names
last_names <- character(0)
while (length(last_names) < 800) {
  last_names <- unique(c(last_names, sample(babynames::babynames$name, 1000,
                                            replace = TRUE)))
}
last_names <- head(last_names, 800)

# Create a dataframe with first and last names
names_df <- data.frame(
  first_name = first_names,
  last_name = last_names
)
```

```r
# Display the first few rows of the dataframe
head(names_df)

# Define the popular email vendors
email_vendors <- c("gmail.com", "yahoo.com", "outlook.com", "hotmail.com", "aol.com")

# Generate random email addresses with popular vendors
emails <- paste0(first_names, ".", last_names, "@", sample(email_vendors, 800,
                                                           replace = TRUE))

# Define a function to hash passwords
hash_password <- function(password) {
  hashed <- openssl::sha256(password)
  return(hashed)
}

# Generate random passwords
passwords <- sapply(1:800, function(x) {
  random_password <- paste0(sample(c(letters, LETTERS, 0:9), 800, replace = TRUE),
                            collapse = "")
  hash_password(random_password)
})

# Generate random user types
user_types <- sample(c("Basic", "Premium", "VIP"), 800, replace = TRUE)

# Generate random expiry dates
# For demonstration purpose, let's assume expiry date is 1 year from current date
expiry_dates <- as.Date(Sys.Date()) + sample(365, 800, replace = TRUE)

# Create a dataframe with all the details
buyer_df <- data.frame(
  buyer_id = buyer_ids,
  first_name = first_names,
  last_name = last_names,
  email = emails,
  password = passwords,
  user_type = user_types,
  expiry_date = expiry_dates
)

# Display the first few rows of the dataframe
head(buyer_df)

buyer_df$buyer_id
```

3.1.4. Contact Detail Entity

Similarly, the dataset containing contact details was created using Mockaroo. However, additional cleaning was necessary to match it with our other datasets. Address id is unique, but address name is not unique.

Prompt in ChatGPT: Generate unique UK style 6 digit PO Box numbers for each city in existing contact dataset.

```
# Buyer has a 1:N relationship with addresses
# Let's add the contact details to the buyer table

# Read the data created with Mockaroo
contact_df <- read_csv("synthetic_data_generation/contact_detail.csv")

# Lets assume the buyer has 1 address each in this case.

# Lets format the address id to our standardized format.

# Define the number of address IDs to generate and starting ID
num_addresses <- 1000
start_id_address <- 1118000001

# Generate random three-digit numbers
set.seed(123)  # Ensuring reproducibility
random_digits_address <- sample(100:999, num_addresses, replace = TRUE)

# Generate unique address IDs with random digits
address_ids <- paste0("A", start_id_address:(start_id_address + num_addresses - 1),
                      random_digits_address)

contact_df$address_id<- address_ids
# The Address column doesnt seem to have appropriate PO box values
# Lets generate 1000 random PO BOX values with r and replace the data in contact_df

# Chat GPT Prompt response:
# Function to generate a PO box number with the specified format

generate_po_box_number <- function(city) {
  first_letter <- substr(city, 1, 1)
  po_box_number <- paste0(first_letter, paste0(sample(LETTERS, 2, replace=TRUE),
                                               collapse=""),
                          sample(0:9, 1, replace=TRUE), paste0(sample(
                            LETTERS, 2, replace=TRUE), collapse=""))
  return(po_box_number)
}

# Generate unique UK-style PO box numbers for each city in contact_df
```

```r
uk_po_box_numbers <- character(nrow(contact_df))

for (i in 1:nrow(contact_df)) {
  city <- contact_df$city[i]
  new_po_box_number <- generate_po_box_number(city)
  uk_po_box_numbers[i] <- new_po_box_number
}

# Replace existing addresses in contact_df$address with just the PO box number
contact_df$address <- uk_po_box_numbers

# Lets clean the city values
# Remove asterisks (*) from city names
contact_df$city <- gsub("\\*", "", contact_df$city)

# CARDINALITY CHECK

# Lets drop the email column as we already have email in buyer_df
contact_df$email <- NULL

# Lets now create a column in contact_df to store buyer IDs
contact_df$buyer_id <- NA

# Shuffle the order of contacts to ensure randomness
contact_df <- contact_df[sample(nrow(contact_df)), ]

# Initialize a counter to keep track of the number of contacts assigned to each buyer
buyer_contact_count <- rep(0, nrow(buyer_df))

# Assign contacts to buyers
for (i in 1:nrow(contact_df)) {
  # Find a buyer who has less than 2 contacts assigned
  eligible_buyers <- which(buyer_contact_count < 2)

  if (length(eligible_buyers) == 0) {
    # If all eligible buyers already have 2 contacts, break the loop
    break
  }

  # Choose a random eligible buyer
  chosen_buyer <- sample(eligible_buyers, 1)

  # Assign the contact to the chosen buyer
  contact_df$buyer_id[i] <- buyer_df$buyer_id[chosen_buyer]

  # Increment the contact count for the chosen buyer
  buyer_contact_count[chosen_buyer] <- buyer_contact_count[chosen_buyer] + 1
```

```
}

# Display the resulting contact_df
head(contact_df)
```

```
# A tibble: 6 x 8
  address_id     address country        state city  street phone_number buyer_id
  <chr>          <chr>   <chr>          <chr> <chr> <chr>         <dbl> <chr>
1 A1118000609267 SXI9ZT  United Kingdom Engl~ Sout~ Stati~   7028695824 B131200~
2 A1118000443430 BYS4XJ  United Kingdom Engl~ Bris~ Stati~   7873418623 B131200~
3 A1118000995732 BWO0IP  United Kingdom Nort~ Bang~ Main ~   7494814099 B131200~
4 A1118000773155 CJS1CW  United Kingdom Engl~ Chic~ High ~   7888285628 B131200~
5 A1118000983704 COS7JW  United Kingdom Engl~ Cove~ High ~   7460512998 B131200~
6 A1118000514811 LQL8PM  United Kingdom Nort~ Lisb~ Main ~   7504007377 B131200~
```

```
# Now let's add the address type to the contact_df
# Assuming the buyer can set the address as work or home

# Add address_type column to contact_df with default value NA
contact_df$address_type <- NA

# Group contact_df by buyer_id and sample one address type for each group
contact_df <- contact_df %>%
  group_by(buyer_id) %>%
  mutate(address_type = sample(c("Home", "Work"), 1)) %>%
  ungroup()  # Ungroup the dataframe

# Display the first few rows of contact_df
head(contact_df)
```

```
# A tibble: 6 x 9
  address_id     address country        state city  street phone_number buyer_id
  <chr>          <chr>   <chr>          <chr> <chr> <chr>         <dbl> <chr>
1 A1118000609267 SXI9ZT  United Kingdom Engl~ Sout~ Stati~   7028695824 B131200~
2 A1118000443430 BYS4XJ  United Kingdom Engl~ Bris~ Stati~   7873418623 B131200~
3 A1118000995732 BWO0IP  United Kingdom Nort~ Bang~ Main ~   7494814099 B131200~
4 A1118000773155 CJS1CW  United Kingdom Engl~ Chic~ High ~   7888285628 B131200~
5 A1118000983704 COS7JW  United Kingdom Engl~ Cove~ High ~   7460512998 B131200~
6 A1118000514811 LQL8PM  United Kingdom Nort~ Lisb~ Main ~   7504007377 B131200~
# i 1 more variable: address_type <chr>
```

3.1.5. Product Entity

The next step was to generate a list of 500 products using the same format as the seller's dataset for product names.

Prompt in ChatGPT: Generate a list of 500 products, with product ID, price, rating, product name, weight, instock (yes, no), available units. Name should be a combination of 50 adjectives and 50 nouns plus its category, assume 90% of products are in stock, of which quantity can go up to 500 units. Category price should fluctuate around the average prices that is stored in category_prices (+- 30%), price should be rounded to nearest .99 to match retail prices.

```
# Define the number of product IDs to generate and starting ID
num_products <- 500
start_id_product <- 1812000001

# Generate random three-digit numbers for product IDs
set.seed(123)  # Ensuring reproducibility
random_digits_product <- sample(100:999, num_products, replace = TRUE)

# Generate unique product IDs by combining random digits with starting ID
product_ids <- paste0("P", start_id_product:(start_id_product + num_products - 1),
                      random_digits_product)

# Set seed for reproducibility
set.seed(123)

# Adjectives and nouns for product names
adjectives <- c("Sleek", "Modern", "Stylish", "Efficient", "Durable", "Compact",
                "Innovative", "Elegant", "Powerful", "Versatile", "Premium",
                "High-Quality", "Portable", "Sophisticated", "Advanced",
                "User-Friendly", "Comfortable", "Reliable",
                "Futuristic", "Eco-Friendly", "Luxurious", "Affordable",
                "Sleek", "Gorgeous", "Practical", "Sustainable", "Chic",
                "Flexible", "Compact", "Robust", "Sleek",
                "Fashionable", "Ergonomic", "Dynamic", "Sleek",
                "Innovative", "High-Performance", "Fashionable", "Trendy",
                "Fashionable", "Sleek", "Stylish", "Fashionable",
                "Elegant", "Dynamic", "Stylish", "Elegant", "Eco-Friendly")

nouns <- c("Refrigerator", "Smartphone", "Television", "Dress", "Laptop",
           "Shirt", "Speaker", "Vacuum Cleaner", "Perfume", "Couch", "Toy",
           "Yoga Mat", "Notebook", "Washing Machine", "Tablet", "Makeup Kit",
           "Curtains", "Baby Stroller", "Running Shoes", "Calculator",
           "Backpack", "Cookware Set", "Luggage", "Sneakers", "Headphones",
           "Jacket", "Printer", "Face Cream", "Bed", "Baby Monitor",
           "Gym Equipment", "Desk", "Gaming Console",
           "Handbag", "Smartwatch", "Air Purifier", "Dining Table",
           "Sunglasses", "Razor", "Blender", "Earphones", "Mattress",
           "Baby Carrier", "Fitness Tracker", "Alarm Clock",
           "Hand Mixer", "Pillow")

# Generate product names based on categories
total_products <- 500
```

```r
# Sample data
data <- tibble(
  product_id = product_ids,
  product_name = paste(sample(adjectives, total_products, replace = TRUE),
                       sample(nouns, total_products, replace = TRUE), sep = ' ')
)

# View the first few rows of updated data
head(data)
```

```
# A tibble: 6 x 2
  product_id     product_name
  <chr>          <chr>
1 P1812000001514 Sleek Smartphone
2 P1812000002562 Advanced Couch
3 P1812000003278 Sophisticated Speaker
4 P1812000004625 Stylish Sunglasses
5 P1812000005294 Stylish Fitness Tracker
6 P1812000006917 Fashionable Air Purifier
```

```r
# Generate "in stock" column
data <- data %>%
  mutate(in_stock = sample(c("yes", "no"), total_products, replace = TRUE,
                           prob = c(0.9, 0.1)))

# View the first few rows of updated data
head(data)
```

```
# A tibble: 6 x 3
  product_id     product_name             in_stock
  <chr>          <chr>                    <chr>
1 P1812000001514 Sleek Smartphone         yes
2 P1812000002562 Advanced Couch           no
3 P1812000003278 Sophisticated Speaker    yes
4 P1812000004625 Stylish Sunglasses       no
5 P1812000005294 Stylish Fitness Tracker  yes
6 P1812000006917 Fashionable Air Purifier yes
```

```r
# Generate "available_units" column
data <- data %>%
  mutate(available_units = ifelse(in_stock == "yes",
                                  ifelse(runif(total_products) <= 0.9,
                                         sample(1:500, total_products,
                                                replace = TRUE), 0),
                                  0))
```

```r
# View the first few rows of updated data
head(data)
```

```
# A tibble: 6 x 4
  product_id     product_name             in_stock available_units
  <chr>          <chr>                    <chr>              <dbl>
1 P1812000001514 Sleek Smartphone         yes                  318
2 P1812000002562 Advanced Couch           no                     0
3 P1812000003278 Sophisticated Speaker    yes                  208
4 P1812000004625 Stylish Sunglasses       no                     0
5 P1812000005294 Stylish Fitness Tracker  yes                  279
6 P1812000006917 Fashionable Air Purifier yes                   67
```

```r
# Categories and their nouns
category_nouns <- list(
  "Appliances" = c("Refrigerator", "Vacuum Cleaner", "Blender",
                   "Washing Machine", "Air Purifier", "Hand Mixer"),
  "Mobiles & Tablets" = c("Smartphone", "Tablet", "Smartwatch"),
  "Entertainment" = c("Television", "Speaker", "Gaming Console",
                      "Headphones", "Earphones"),
  "Women's Fashion" = c("Dress", "Shirt", "Handbag", "Sunglasses"),
  "Computing" = c("Laptop", "Notebook", "Printer"),
  "Men's Fashion" = c("Shirt", "Jacket", "Sneakers", "Backpack"),
  "Beauty & Grooming" = c("Perfume", "Makeup Kit", "Face Cream", "Razor"),
  "Home & Living" = c("Curtains", "Couch", "Table", "Bed", "Dining Table",
                      "Luggage", "Mattress", "Cookware Set", "Desk",
                      "Alarm Clock", "Pillow"),
  "Kids & Baby" = c("Toy", "Baby Stroller", "Baby Monitor", "Baby Carrier"),
  "Health & Sports" = c("Running Shoes", "Yoga Mat",
                        "Fitness Tracker", "Gym Equipment"),
  "School & Education" = c("Notebook", "Calculator")
)

# Initialize the category column as an empty character vector
data$category <- ""

# Iterate over each product name to find its category
for (category in names(category_nouns)) {
  for (noun in category_nouns[[category]]) {
    # Check if the product name contains the current noun
    data$category <- ifelse(grepl(noun, data$product_name), category, data$category)
  }
}

# View the first few rows of updated data
head(data)
```

```
# A tibble: 6 x 5
  product_id     product_name             in_stock available_units category
  <chr>          <chr>                    <chr>               <dbl> <chr>
1 P1812000001514 Sleek Smartphone         yes                   318 Mobiles & Ta~
2 P1812000002562 Advanced Couch           no                      0 Home & Living
3 P1812000003278 Sophisticated Speaker    yes                   208 Entertainment
4 P1812000004625 Stylish Sunglasses       no                      0 Women's Fash~
5 P1812000005294 Stylish Fitness Tracker  yes                   279 Health & Spo~
6 P1812000006917 Fashionable Air Purifier yes                    67 Appliances
```

```r
# Let's add the prices using the following information

# Categories and their average prices are extracted
# From the Pakistan e-commerce dataset
# Using excel pivot tables
category_prices <- c(
  "Appliances" = 31.41600882,
  "Beauty & Grooming" = 2.498201959,
  "Books" = 1.373836291,
  "Computing" = 30.92282575,
  "Entertainment" = 54.70450929,
  "Health & Sports" = 2.897034883,
  "Home & Living" = 3.212621698,
  "Kids & Baby" = 1.911669386,
  "Men's Fashion" = 2.578544381,
  "Mobiles & Tablets" = 49.84310222,
  "Others" = 6.320278129,
  "School & Education" = 1.369733365,
  "Superstore" = 1.735174642,
  "Women's Fashion" = 4.9945859
)


# Function to generate random price
# Based on category mean with random percentage increase/decrease
generate_price <- function(category) {
  mean_price <- category_prices[category]
  # Generate random percentage increase/decrease within the range of -30% to 30%
  percentage_change <- runif(1, -0.3, 0.3)
  # Calculate the new price with the percentage change
  price <- mean_price * (1 + percentage_change)
  return(price)
}

# Add price column to data
data <- data %>%
  mutate(price = mapply(generate_price, category))
```

```r
# View the first few rows of updated data
head(data)
```

```
# A tibble: 6 x 6
  product_id     product_name            in_stock available_units category price
  <chr>          <chr>                   <chr>               <dbl> <chr>    <dbl>
1 P1812000001514 Sleek Smartphone        yes                   318 Mobiles~ 59.2
2 P1812000002562 Advanced Couch          no                      0 Home & ~  3.12
3 P1812000003278 Sophisticated Speaker   yes                   208 Enterta~ 50.4
4 P1812000004625 Stylish Sunglasses      no                      0 Women's~  5.96
5 P1812000005294 Stylish Fitness Tracker yes                   279 Health ~  2.92
6 P1812000006917 Fashionable Air Purifi~ yes                    67 Applian~ 36.0
```

```r
# Function to round price to the nearest 0.99
round_to_nearest_99 <- function(price) {
  rounded_price <- round(price - 0.01) + 0.99
  return(rounded_price)
}

# Function to generate random price based on category mean
# With random percentage increase/decrease, rounded to the nearest 0.99
generate_price <- function(category) {
  mean_price <- category_prices[category]
  # Generate random percentage increase/decrease within the range of -30% to 30%
  percentage_change <- runif(1, -0.3, 0.3)
  # Calculate the new price with the percentage change
  price <- mean_price * (1 + percentage_change)
  # Round the price to the nearest 0.99
  rounded_price <- round_to_nearest_99(price)
  return(rounded_price)
}

# Add price column to data
data <- data %>%
  mutate(price = mapply(generate_price, category))

# We need the category_id to represent the 1:N relationship
# Between product and category
merged_data <- merge(data, categories_df, by.x = "category",
                     by.y = "category_name", all.x = TRUE)

# Let's also assign sellers to products
# Function to randomly assign a seller_id to data based on matching categories
assign_seller_id <- function(data, sellers_df) {
  for (i in 1:nrow(data)) {
    matching_sellers <- sellers_df$seller_id[
```

```
      sellers_df$categories %in% data$category[i]]
    if (length(matching_sellers) > 0) {
      data$seller_id[i] <- sample(matching_sellers, 1)
    } else {
      data$seller_id[i] <- NA
    }
  }
  return(data)
}

#CARDINALITY CHECK

# Call the function to assign seller_id to data
data <- assign_seller_id(data, sellers_df)

# Now we can remove categories from the sellers dataset to ensure atomicity.
sellers_df$categories<- NULL

# Merge products_df with categories_df based on matching category names
merged_df <- merge(data, categories_df, by.x = "category",
                   by.y = "category_name", all.x = TRUE)

# Drop the original category column
merged_df$category <- NULL
# View the updated dataframe

products_df<-merged_df
```

3.1.6. Buyer-Orders-Product Relationship

An order dataset was created using 1000 orders from customers in the buyers dataset who ordered the products as previously defined. Additional considerations, such as repeat buyers, order dates, and discount conditions, were defined using ChatGPT.

Buyer id and order date in combination define the composite order id.

Part 1: Creating a list of orders

Prompt in ChatGPT: Generate 1000 orders with order ID, order date (random date and time from September 2022), order status (delivered 60%, returned 35%, cancelled 5%), payment type (cash, card). Assign 30% of buyers from the buyer dataframe as repeat buyers, and the rest randomly sampled for each order. Assign 30% of carriers to represent multiple orders being carried. Assign a list of random products to be selected from the products dataset, with a maximum of 15 products and an average of 3 products per order. Products should be in stock when placing the order.

```
# To create an M:N buyer orders products relationship
# we first need a data that containts the list of orders
# With the products and quantity ordered by each buyer
```

```r
# Define the number of order IDs to generate and starting ID
num_orders <- 1000
start_id_order <- 50001

# Generate random three-digit numbers for unique order IDs
set.seed(123)  # Setting seed for reproducibility
random_digits_order <- sample(100:999, num_orders, replace = TRUE)

# Generate unique order IDs with random digits
order_ids <- paste0("O", start_id_order:(start_id_order + num_orders - 1),
                    random_digits_order)

# Display the first few order IDs
head(order_ids)

# Create sample order data
orders <- data.frame(
  order_id = order_ids,
  order_date = as.POSIXct(sample(seq(as.POSIXct('2022-09-01'),
                                     as.POSIXct('2022-09-30'), by = "day"),
                                 1000, replace = TRUE) + runif(1000) * 86400),
  order_status = sample(c("Delivered", "Cancelled", "Returned"),
                        1000, replace = TRUE, prob = c(0.6, 0.35, 0.05)),
  payment_type = sample(c("Card", "Cash"), 1000, replace = TRUE)
)

# Display the first few rows of the dataset
head(orders)

# Assigning buyer IDs to orders

# Sample 30% of the buyer IDs to represent repeat buyers
repeat_buyer_ids <- sample(
  buyer_df$buyer_id, size = 0.3 * length(buyer_df$buyer_id),
  replace = TRUE)

# Assign buyer IDs to each order randomly, with 30% of orders being from repeat buyers
orders$buyer_id <- sample(c(
  buyer_df$buyer_id, repeat_buyer_ids), size = nrow(orders),
  replace = TRUE)

# Display the first few rows of the orders data frame with buyer IDs assigned
head(orders)


# Assigning products to orders
```

```r
# Define the maximum number of products per order and average number of product IDs
max_products <- 15
average_products <- 3

# Filter products available in stock
# so our system doesn't place orders when inventory is 0
products_df_instock <- products_df %>% filter(available_units > 20)

# Function to generate a list of product IDs for each order
generate_product_list <- function() {
  num_products <- rpois(1, lambda = average_products)
  if(num_products > max_products) num_products <- max_products
  product_list <- sample(products_df_instock$product_id,
                         size = num_products, replace = TRUE)
  return(product_list)
}

# Apply the function to each row of the orders data frame
# To create the products_ordered column
orders$products_ordered <- lapply(1:nrow(orders), function(x) generate_product_list())

# Display the first few rows of the orders data frame
# With products_ordered column
head(orders)

# We also need to make sure that the order
# Contains the address the buyer selected as the delivery location
# So we sample an address id which belongs to a buyer id
# As a buyer can have more than 1 address

# Add a new column to orders dataframe with sampled address type
orders <- orders %>%
  mutate(address_type = sample(c("Home", "Work"), size = nrow(.),
                               replace = TRUE))
```

Part 2: Assigning products to orders

Prompt in ChatGPT: For each product ordered in the orders dataset, it should be a single-row value and must include information from the product dataset.

```r
# Extract relevant columns from the 'products_df' table
normalized_product_data <- products_df[c("product_id", "product_name",
                                         "price", "in_stock", "available_units",
                                         "category_id", "seller_id")]

# Initialize a list to store matched product IDs for each order
matched_products <- list()
```

```
# Iterate over each row of the orders data frame
for (i in 1:nrow(orders)) {
  # Extract product IDs for the current order
  product_ids <- orders$products_ordered[[i]]

  # Check if product IDs are empty
  if (length(product_ids) > 0) {
    # Store the product IDs for the current order
    matched_products[[i]] <- data.frame(order_id = rep(orders$order_id[i],
                                                       length(product_ids)),
                                        product_id = product_ids)
  }
}

# Combine matched product IDs into a single data frame
matched_products_df <- do.call(rbind, matched_products)

# Merge normalized product data with original product data based on product_id
merged_data <- merge(matched_products_df, products_df, by = "product_id")

# View the merged data
head(merged_data)
```

Part 3: Normalised Form for Buyer-Orders-Product (3NF Table Creation)

The order data needed to be normalised so that each ordered product would have its own row in the relationship. We normalised the above dataset by merging it with the constituent buyer, resulting in a M:N buyer orders product relationship.

```
# Merge the merged product data with the orders data based on order_id

buyer_orders_products <- merge(merged_data, orders, by = "order_id")

# Sort by order date
buyer_orders_products <- buyer_orders_products[
  order(buyer_orders_products$order_date), ]

# Initialize a new column for remaining available units
buyer_orders_products$remaining_available_units <- NA

# Initialize the quantity_ordered column in buyer_orders_products dataframe
buyer_orders_products$quantity_ordered <- NA

# Update available_units after each order
for (i in 1:nrow(buyer_orders_products)) {
  product_id <- buyer_orders_products$product_id[i]
  quantity_ordered <- NA  # Initialize quantity_ordered
```

```
  # Determine the quantity_ordered
  if (runif(1) <= 0.9) {
    quantity_ordered <- 1
  } else if (runif(1) <= 0.05) {
    quantity_ordered <- sample(2:5, 1)
  }

  # Ensure quantity_ordered is at least 1
  if (is.na(quantity_ordered) || quantity_ordered < 1) {
    quantity_ordered <- 1
  }

  # Update the quantity_ordered column
  buyer_orders_products$quantity_ordered[i] <- quantity_ordered

  # Update available_units in products_df
  row_index <- which(products_df$product_id == product_id)
  if (length(row_index) > 0) {
    products_df$available_units[row_index] <-
      products_df$available_units[row_index] - quantity_ordered
    products_df$available_units[row_index] <-
      max(products_df$available_units[row_index], 0)
    buyer_orders_products$remaining_available_units[i] <-
      products_df$available_units[row_index]
  } else {
    # Handle the case where product_id is not found in products_df
    # Print a warning message or take appropriate action
    warning(paste("Product ID", product_id, "not found in products_df"))
  }
}

# Drop the available_units column
buyer_orders_products <- buyer_orders_products[
  , !(names(buyer_orders_products) %in% c("available_units"))]

# Display the first few rows of updated data
head(buyer_orders_products)

# Select relevant columns
buyer_orders_products <- buyer_orders_products %>%
  select(
    product_id,
    quantity_ordered,
    order_date,
    order_status,
    payment_type,
    buyer_id,
```

```
    address_type
  )
```

3.1.7. Review Entity

Furthermore, a review entity was created using Mockaroo, and review dates were adjusted to ensure that all reviews were posted after customers received the product.

```
# Dataset of reviews
review_df <- read_csv("synthetic_data_generation/review.csv")

# Let's first make the review id standardized

# Define the number of review IDs to generate and starting ID
num_reviews <- nrow(review_df)
start_id_review <- 1771000001

# Generate random three-digit numbers
set.seed(123)  # Setting seed for reproducibility
random_digits <- sample(100:999, num_reviews, replace = TRUE)

# Generate unique review IDs with random digits
review_ids <- paste0("R", start_id_review:(start_id_review + num_reviews - 1),
                     random_digits)
review_df$review_id<- review_ids

# Since reviews can only be placed on orders that were not cancelled
# i.e, orders that were delivered or returned
# We filter the data as follows:

non_cancelled_orders <- buyer_orders_products %>% filter(order_status != "Cancelled")

# Assign product id and buyer id as foreign keys in the reviews table
# Let's also replace the review dates
sampled_pairs <- non_cancelled_orders[sample(nrow(buyer_orders_products),
                                             nrow(review_df), replace = FALSE),
                                      c("product_id", "buyer_id", "order_status",
                                        "order_date")]

review_df$product_id <- sampled_pairs$product_id
review_df$buyer_id<- sampled_pairs$buyer_id

# Let's replace the review dates, as the same day of delivery
# Assuming people review once they get the product.

# Generate a random number of days between 0 and 7
```

```
random_days <- sample(0:7, nrow(sampled_pairs), replace = TRUE)

# Add the random number of days to the order_date
review_date <- sampled_pairs$order_date + random_days

# Assign the result to the review_date column in review_df
review_df$review_date <- review_date

# Let's drop the NA's

review_df <- na.omit(review_df)

# Push dataframes into csv files
```

3.1.8. Self Referencing on Buyer Entity

This section explores about self-referencing within buyer entities. Self-referencing occurs when a buyer entity engages in transactions with itself or related parties, potentially leading to complications and risks. We'll examine what this means and how it can impact business operations. The following code snippet demonstrates a practical method for identifying and managing self-referencing instances within a buyer entity dataset.

```
# Sample 60% of the buyer_id values
references <- sample(buyer_df$buyer_id, 0.6*nrow(buyer_df), replace = TRUE)

# Randomly assign values to buyer_df$buyer_id for the selected 60%
buyer_df$referred_by <- ifelse(buyer_df$buyer_id %in% references, NA, references)

# Create new table with buyer_df and referred_by

references_df <- buyer_df %>% select(buyer_id, referred_by) %>% na.omit()

buyer_df$referred_by <- NULL
```

3.2. Data Quality Assurance (Validation)

During the data validation process, we carefully examine various attributes across multiple datasets to ensure their accuracy and consistency. First, we ensure that all names are in character format and that phone numbers contain exactly ten digits. Additionally, we ensure that email addresses follow the standard email format. We've added referential integrity checks to ensure that foreign key values exist in the referenced tables and that primary keys like product_id, seller_id, category_id, buyer_id, address_id, and review_id are unique and non-null. For columns marked as unique, such as email addresses in buyer and seller records, we ensure uniqueness and eliminate duplicates.

Data type validation ensures that each column contains the correct type of data, and range constraint checks ensure that numeric values are within acceptable ranges. We validate that specific

attributes, such as UK postcodes, contain precisely 6 digits. Furthermore, we set a maximum length of 25 characters for various names of product, first name, last name, buyer name, seller name, and product name as well as descriptions are limited to 200 words.

Regular expression validation is employed to verify that data conforms to particular patterns, such as email adress format or phone numbers that commence with the digit 7. In order to maintain the accuracy and dependability of the data, any rows that fail to meet the specified validation criteria are promptly eliminated from the dataset.

```
# Create a list to store all entity dataframes
all_entity_data <- list()

# List all entity folders within the "data_upload" directory
entity_folders <- list.dirs(path = "data_upload",
                            full.names = TRUE,
                            recursive = FALSE)

# Loop through each entity folder
for (entity_folder in entity_folders) {
  # Extract the folder name
  entity_name <- basename(entity_folder)

  # List all CSV files in the entity folder
  csv_files <- list.files(path = entity_folder,
                          pattern = "\\.csv$",
                          full.names = TRUE)

  # Initialize an empty dataframe to store data for the entity
  entity_data <- data.frame()

  # Loop through each CSV file in the entity folder
  for (csv_file in csv_files) {
    # Read the CSV file into a dataframe
    csv_data <- read.csv(csv_file)

    # Merge the data from the CSV file into the entity dataframe
    entity_data <- rbind(entity_data, csv_data)
  }

  # Assign the entity dataframe to a variable with a name starting with "project_"
  assign(paste0("project_", entity_name), entity_data)

  # Add the entity dataframe to the list
  all_entity_data[[paste0("project_", entity_name)]] <- entity_data
}

# Check the names of the created entity dataframes
print(names(all_entity_data))
```

```r
# Data Validation
# Function to validate phone numbers
validate_phone_number <- function(phone_number) {
  # Check if the length is 10 and starts with 7
  if (nchar(phone_number) != 10 || substr(phone_number, 1, 1) != "7") {
    return(FALSE)
  }
  # Check if all characters are numeric
  if (!grepl("^[0-9]+$", phone_number)) {
    return(FALSE)
  }
  return(TRUE)
}

# Function to validate first and last names
validate_name <- function(name) {
  # Check if the length is less than 25 characters
  if (nchar(name) > 25) {
    return(FALSE)
  }
  # Check if only alphabets are present
  if (!grepl("^[A-Za-z]+$", name)) {
    return(FALSE)
  }
  return(TRUE)
}

# Function to validate address
validate_address <- function(address) {
  # Check if the length is 6 characters
  if (nchar(address) != 6) {
    return(FALSE)
  }
  # Check if address consists of only alphanumeric characters
  if (!grepl("^[A-Za-z0-9]+$", address)) {
    return(FALSE)
  }
  return(TRUE)
}

# Validate phone numbers
valid_phone <- sapply(project_contact_df$phone_number, validate_phone_number)

# Validate addresses
valid_address <- sapply(project_contact_df$address, validate_address)
```

```r
# Check for invalid phone numbers and addresses
invalid_phone_numbers <- project_contact_df$phone_number[!valid_phone]
invalid_addresses <- project_contact_df$address[!valid_address]

# If there are any invalid entries, remove them
if (length(invalid_phone_numbers) > 0) {
  cat("Invalid phone numbers:", invalid_phone_numbers, "\n")
  project_contact_df <- project_contact_df[valid_phone, ]
}

if (length(invalid_addresses) > 0) {
  cat("Invalid addresses:", invalid_addresses, "\n")
  project_contact_df <- project_contact_df[valid_address, ]
}

# Remove duplicate rows based on email IDs in project_buyer_df
project_buyer_df <- distinct(project_buyer_df, email, .keep_all = TRUE)

# Remove duplicate rows based on email IDs in project_sellers_df
project_sellers_df <- distinct(project_sellers_df, email, .keep_all = TRUE)



# Ensure referential integrity between tables

# Ensure category_id in project_products_df exists in project_categories_df
if (any(!project_products_df$category_id %in% project_categories_df$category_id)) {
  stop(
    "Foreign key violation: category_id in project_products_df does not exist
    in project_categories_df")
}

# Ensure buyer_id in project_contact_df exists in project_buyer_df
if (any(!project_contact_df$buyer_id %in% project_buyer_df$buyer_id)) {
  stop(
    "Foreign key violation: buyer_id in project_contact_df does not exist
    in project_buyer_df")
}

# Ensure buyer_id in project_review_df exists in project_buyer_df
if (any(!project_review_df$buyer_id %in% project_buyer_df$buyer_id)) {
  stop(
    "Foreign key violation: buyer_id in project_review_df does not exist
    in project_buyer_df")
}

# Ensure product_id in project_review_df exists in project_products_df
```

```r
if (any(!project_review_df$product_id %in% project_products_df$product_id)) {
  stop(
    "Foreign key violation: product_id in project_review_df does not exist
    in project_products_df")
}

# Ensure buyer_id and product_id in project_buyer_orders_products exist
# in project_buyer_df and project_products_df respectively
if (any(!project_buyer_orders_products$buyer_id %in% project_buyer_df$buyer_id) ||
    any(!project_buyer_orders_products$product_id %in% project_products_df$product_id)) {
  stop("
       Foreign key violation: buyer_id or product_id in project_buyer_orders_products
       do not exist in project_buyer_df or project_products_df")
}

# Ensure buyer_id and referred_by in project_references_df exist in project_buyer_df
if (any(!project_references_df$buyer_id %in% project_buyer_df$buyer_id) ||
    any(!project_references_df$referred_by %in% project_buyer_df$buyer_id)) {
  stop("
       Foreign key violation: buyer_id or referred_by in project_references_df
       do not exist in project_buyer_df")
}

# Check for new data if available in the datasets.
# Function to check if data exists in a table
data_exists <- function(connection, table_name) {
  query <- paste0("SELECT COUNT(*) FROM ", table_name)
  result <- dbGetQuery(connection, query)
  return(result[[1]] > 0)
}

# Function to insert data into a table from a data frame
insert_data_from_df <- function(connection, table_name, data_frame) {
  # Check if data already exists in the table
  if (data_exists(connection, table_name)) {
    cat("Data already exists in", table_name, "\n")
    return()
  }

  # If data doesn't exist, insert it into the table
  dbWriteTable(connection, table_name, data_frame, row.names = FALSE, append = TRUE)
  cat("Data inserted into", table_name, "\n")
}


# Verify the table was created by listing all tables in the database
# RSQLite::dbListTables(connection)
```

3.3. Data Import (Insert)

After validation we updated our data frames and created tables using SQL using "CREATE FUNCTION". Then we added our dataset to each table using "INSERT INTO function". In case of a new csv file is identified, we check the dataset for validation, and after validation we create a table for that file in the name of "project_filename".

```
# Load necessary libraries
library(readr)
library(RSQLite)

# Create a database connection
connection <- RSQLite::dbConnect(RSQLite::SQLite(),"database/group32.db")


## Products Table
dbExecute(connection, "
  CREATE TABLE IF NOT EXISTS 'project_products' (
  'product_id' VARCHAR(255) PRIMARY KEY,
    'seller_id' VARCHAR(10) NOT NULL,
  'category_id' VARCHAR(10) NOT NULL,
  'product_name' TEXT NOT NULL,
  'in_stock' BIT NOT NULL DEFAULT 0,
  'available_units' INT NOT NULL DEFAULT 0,
  'price' MONEY NOT NULL CHECK (price > 0),
  FOREIGN KEY ('seller_id') REFERENCES project_seller ('seller_id'),
  FOREIGN KEY ('category_id') REFERENCES project_category('category_id')
);
")


## Create Seller Table
dbExecute(connection, "
CREATE TABLE IF NOT EXISTS 'project_seller' (
'seller_id' VARCHAR(10) PRIMARY KEY,
'seller_name' TEXT NOT NULL,
'url' VARCHAR(255),
'description' TEXT,
'email' VARCHAR(255) UNIQUE,
'password' VARCHAR(255),
'account_number' VARCHAR(255),
'bank_name' TEXT
);
")


## Create category Table
dbExecute(connection, "
```

```
CREATE TABLE IF NOT EXISTS 'project_categories' (
'category_id' VARCHAR(255) PRIMARY KEY,
'category_name' TEXT NOT NULL
);
")

## Create buyer Table
dbExecute(connection, "
  CREATE TABLE IF NOT EXISTS 'project_buyer' (
  'buyer_id' VARCHAR PRIMARY KEY,
  'first_name' TEXT NOT NULL,
  'last_name' TEXT,
  'email' VARCHAR NOT NULL UNIQUE,
  'password' VARCHAR NOT NULL,
  'user_type' TEXT,
  'expiry_date' DATE
);
")


## Create contact details Table
dbExecute(connection, "
  CREATE TABLE IF NOT EXISTS 'project_contact_details' (
  'address_id' VARCHAR PRIMARY KEY,
   'buyer_id' VARCHAR,
  'address' VARCHAR,
  'country' TEXT,
  'state' TEXT,
  'city' TEXT,
  'street' VARCHAR,
  'phone_number' VARCHAR(10),
  'address_type' TEXT,
  FOREIGN KEY ('buyer_id') REFERENCES project_buyer ('buyer_id')
);
")

## Create Review Table
dbExecute(connection, "
  CREATE TABLE IF NOT EXISTS 'project_review' (
  'review_id' VARCHAR PRIMARY KEY,
    'product_id' VARCHAR,
  'buyer_id' VARCHAR,
  'rating' INT CHECK (rating >= 1 AND rating <= 5),
  'review' TEXT,
  'review_date' DATE,
   FOREIGN KEY ('buyer_id') REFERENCES project_buyer ('buyer_id'),
   FOREIGN KEY ('product_id') REFERENCES project_products ('product_id')
```

```
);
")

# Create relationship table project_buyer_orders_products
dbExecute(connection, "
CREATE TABLE IF NOT EXISTS 'project_buyer_orders_products' (
  'buyer_id' VARCHAR(255),
  'product_id' VARCHAR(255),
  'quantity_ordered' INT NOT NULL DEFAULT 1,
  'order_date' DATE,
  'delivery_date' DATE,
  'order_status' TEXT,
  'payment_type' TEXT,
  'address_type' TEXT,
  FOREIGN KEY ('product_id') REFERENCES project_products ('product_id'),
  FOREIGN KEY ('buyer_id') REFERENCES project_buyer ('buyer_id')
);
")

# Create relationship table references
dbExecute(connection, "
CREATE TABLE IF NOT EXISTS 'project_references' (
  'buyer_id' VARCHAR(255),
  'referred_by' VARCHAR(255),
   FOREIGN KEY ('buyer_id') REFERENCES project_buyer ('buyer_id')
);
")

# Verify the table was created by listing all tables in the database
RSQLite::dbListTables(connection)

# Create a list to store all entity dataframes
all_entity_data <- list()

# List all entity folders within the "data_upload" directory
entity_folders <- list.dirs(path = "data_upload", full.names = TRUE, recursive = FALSE)

# Loop through each entity folder
for (entity_folder in entity_folders) {
  # Extract the folder name
  entity_name <- basename(entity_folder)

  # List all CSV files in the entity folder
  csv_files <- list.files(path = entity_folder, pattern = "\\.csv$", full.names = TRUE)

  # Initialize an empty dataframe to store data for the entity
  entity_data <- data.frame()
```

```
  # Loop through each CSV file in the entity folder
  for (csv_file in csv_files) {
    # Read the CSV file into a dataframe
    csv_data <- read.csv(csv_file)

    # Merge the data from the CSV file into the entity dataframe
    entity_data <- rbind(entity_data, csv_data)
  }

  # Assign the entity dataframe to a variable with a name starting with "project_"
  assign(paste0("project_", entity_name), entity_data)

  # Add the entity dataframe to the list
  all_entity_data[[paste0("project_", entity_name)]] <- entity_data
}

# Check the names of the created entity dataframes
print(names(all_entity_data))


# Check for new data if available in the datasets.
# Function to check if data exists in a table
data_exists <- function(connection, table_name, data_frame) {
  # Construct the query to check for existence of data
  query <- paste0("SELECT COUNT(*) FROM ", table_name)
  result <- dbGetQuery(connection, query)
  return(result[[1]] > 0)
}

# Function to insert data into a table if it doesn't exist
insert_data_if_not_exists <- function(connection, table_name, data_frame) {
  # Check if data already exists in the table
  if (data_exists(connection, table_name)) {
    cat("Data already exists in", table_name, "\n")
    return()
  }

  # Extract column names
  columns <- names(data_frame)

  # Construct the INSERT INTO SQL query
  insert_query <- paste0("INSERT INTO '", table_name, "' (",
                         paste0("'", columns, "'",
                                collapse = ", "), ") VALUES ")

  # Loop through each row of the data frame and insert values
  for (i in 1:nrow(data_frame)) {
```

```
    values <- paste0("(", paste0("'",
                                 gsub("'", "''",
                                      unlist(data_frame[i,]))
                                 , "'", collapse = ","), ")")
    dbExecute(connection, paste0(insert_query, values))
  }

  cat("Data inserted into", table_name, "\n")
}

# Insert data from data frames into respective tables
insert_data_if_not_exists(connection, "project_products", project_products_df)
insert_data_if_not_exists(connection, "project_seller", project_sellers_df)
insert_data_if_not_exists(connection, "project_categories", project_categories_df)
insert_data_if_not_exists(connection, "project_buyer", project_buyer_df)
insert_data_if_not_exists(connection, "project_contact_details", project_contact_df)
insert_data_if_not_exists(connection, "project_review", project_review_df)
insert_data_if_not_exists(connection, "project_buyer_orders_products",
                          project_buyer_orders_products)
insert_data_if_not_exists(connection, "project_references", project_references_df)

# Verify the table was created by listing all tables in the database
RSQLite::dbListTables(connection)

# Optionally, verify the data was inserted
products_table <- dbReadTable(connection, "project_products")
print(products_table)

buyer_table <- dbReadTable(connection, "project_buyer")
print(buyer_table)
```

4. Data Pipeline Generation

4.1 GitHub Repository and Workflow Setup

A GitHub repository workflow was created in order to improve productivity, manage code quality, and ensure efficient overall project management. The repo was created by following the steps below:

1. **Create GitHub repository**: Creating a collaborative repository on Github by setting visibility to public and other user restrictions.
2. **Add GitHub Collaborators**: All the team members are added to the repository.
3. **Create RStudio Project**: The URL for the Github repository is then inputed during the creation of a new RStudio project from a Git repository.

4. **Push update on R**: For each change or update, the new edits of the code are saved as a new branch. Then commit and push changes made in R.
5. **Pull Changes on GitHub**: Once the changes are committed, they are pulled on GitHub under the main repository branch.

GitHub repository for this project can be accessed thorugh this link: https://github.com/paraggattani/Data-Management

4.2 GitHub Action for Continuous Integration

For the purpose of continuous integration, a GitHub Action was established to streamline the workflow. The creation process involved the following steps.

1. Using the YAML template accessible under the GitHub Actions tab, a new workflow titled 'etl.yaml' was generated.
2. The configuration of the action was done using an existing template. Within the code, the token was edited and the jobs were tailored to execute specific tasks based on our project as follows:

–Run in GitHub–

```yaml
name: Group ETL Workflows for group 32

on:
  schedule:
    - cron: '0 */3 * * *' # Run every 3 hours
  push:
    branches: [ main ]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v2
      - name: Setup R environment
        uses: r-lib/actions/setup-r@v2
        with:
          r-version: '4.2.0'
      - name: Cache R packages
        uses: actions/cache@v2
        with:
          path: ${{ env.R_LIBS_USER }}
          key: ${{ runner.os }}-r-${{ hashFiles('**/lockfile') }}
          restore-keys: |
            ${{ runner.os }}-r-

      - name: Install system dependencies for sf package
        run: |
          sudo apt-get install libgdal-dev libgeos-dev libproj-dev
      - name: Install system dependencies for rnaturalearth package
        run: |
          sudo apt-get install libudunits2-dev
```

Figure 3: GitHub Code Part 1

```yaml
- name: Install tidyr package
  if: steps.cache.outputs.cache-hit != 'true'
  run: |
    Rscript -e 'install.packages("tidyr")'

- name: Install packages
  if: steps.cache.outputs.cache-hit != 'true'
  run: |
    Rscript -e 'install.packages(c("ggplot2", "dplyr", "readr", "RSQLite", "sf", "rnaturalearth", "rnaturalearthdata"))'

- name: Execute database update script
  run: |
    Rscript R/update_database.R

- name: Add files
  run: |
    git config --global user.email "gattaniparag2812@gmail.com"
    git config --global user.name "paraggattani"
    git add --all figures/
- name: Commit files
  run: |
    git commit -m "Updates"
- name: Push changes
  uses: ad-m/github-push-action@v0.6.0
  with:
      github_token: ${{ secrets.GITHUB_TOKEN }}
      branch: main
```

Figure 4: GitHub Code Part 2

5. Data Analysis

In this part, we look more closely at how the data in the database can be analyzed. Data analysis is an important part of database management because it gives information that helps stakeholders make decisions and improves the way of business works. We want to find trends, patterns, and useful information that can help with strategic decisions and operational changes by carefully looking at and figuring out the data.

```r
# 1
# Total sales
# Calculating the revenue for a month


sales<- RSQLite::dbGetQuery(connection,"
          SELECT a.price, datetime(b.order_date, 'unixepoch') AS date_time
          FROM project_buyer_orders_products b
          INNER JOIN project_products a ON b.product_id = a.product_id " )


sales_by_date <- sales %>%
  separate(date_time, into = c("date", "time"), sep = " ")

average_revenue <-sales_by_date %>%
  group_by(date) %>%
  summarise(total_revenue = sum(price)) %>%
```

```
  summarise(average_revenue = sum(total_revenue) / n_distinct(date))

total_sales<-RSQLite::dbGetQuery(connection, "
          SELECT SUM(a.price) AS total_sales
          FROM project_buyer_orders_products b
          INNER JOIN project_products a ON b.product_id = a.product_id " )
total_sales
```

```
  total_sales
1    51442.38
```

The very first analysis that we did is to find the total sales in a month. The result of the code shows that the e-commerce gains £51,442.38 of revenue for 1 motnh.

```
# 2
# Best performing products by revenue

products <- RSQLite::dbGetQuery(connection,
  "SELECT SUM(a.price) AS revenue,
  b.product_id,
  a.product_name
  FROM project_buyer_orders_products b
  INNER JOIN project_products a ON b.product_id = a.product_id
  GROUP BY a.product_id
  ORDER BY revenue DESC" )

top_10_products <- RSQLite::dbGetQuery(connection,
                                       "SELECT SUM(a.price) AS revenue,
                                       b.product_id,
                                       a.product_name
                                       FROM project_buyer_orders_products b
                                       INNER JOIN project_products a
                                       ON b.product_id = a.product_id
                                       GROUP BY a.product_id
                                       ORDER BY revenue DESC
                                       LIMIT 10" )

top_10 <- ggplot(top_10_products, aes(x = reorder(product_name, -revenue),
                                      y = revenue, fill = product_name)) +
  geom_bar(stat = "identity",show.legend = FALSE) +
  labs(title = "Top 10 Products by Revenue", x = "Product Name", y = "Revenue") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

top_10
```

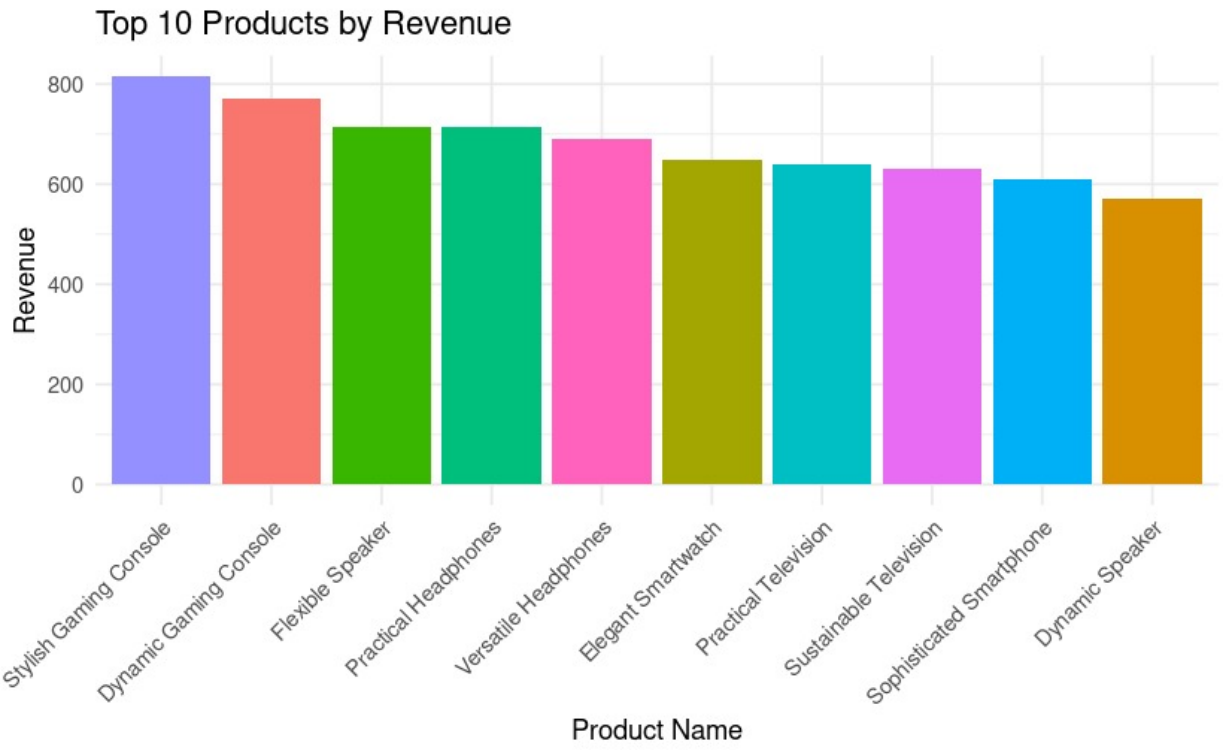

Figure 5: Top 10 Products by Revenue

We can see top 10 products that generates more revenue than the other products in chart above. All 10 products category are electronics and gadgets.

```
# 3
# Revenue by categories
# We calculate the total revenue of each category

revenue_by_categories <- RSQLite::dbGetQuery(connection,
                                             "SELECT c.category_name,
                                             SUM(b.price) AS revenue
                                        FROM project_buyer_orders_products a
                                        INNER JOIN project_products b
                                        ON a.product_id = b.product_id
                                        INNER JOIN project_categories c
                                        ON b.category_id = c.category_id
                                        GROUP BY c.category_name
                                        ORDER BY revenue DESC
                                        " )

# Finding the revenue for each category over time
revenue_by_categories_per_date <- RSQLite::dbGetQuery(connection,
                                                      "SELECT
                                                      c.category_name,
```

```
                                                    b.price,
                                                    datetime(a.order_date,
                                                    'unixepoch')
                                                    AS date_time
                                        FROM project_buyer_orders_products a
                                        INNER JOIN project_products b
                                        ON a.product_id = b.product_id
                                        INNER JOIN project_categories c
                                        ON b.category_id = c.category_id

                                        " )
revenue_by_categories_per_date<- revenue_by_categories_per_date %>%
  separate(date_time, into = c("date", "time"), sep = " ")


revenue_by_categories_per_date <- revenue_by_categories_per_date  %>%
  group_by(category_name, date) %>%
  summarise(sum_price = sum(price)) %>%
  ungroup()
# Revenue by category over time
revenue_by_categories_plot <- ggplot(revenue_by_categories_per_date, aes(
  x = date, y = sum_price, group = category_name, color = category_name)) +
  geom_line() +
  facet_wrap(~ category_name, scales = "free_y") +
  labs(title = "Revenue by Category Over Time", x = "Date", y = "Revenue") +
  theme_minimal() +
  theme(
    axis.text.x = element_blank(),
    legend.position = "none")

revenue_by_categories_plot
```

```
# Ggplot for revenue by category bar_plot
revenue_by_categories <- revenue_by_categories[order(-revenue_by_categories$revenue),]

revenue_by_categories_bar <- ggplot(
  revenue_by_categories, aes(reorder(category_name, -revenue),
                             y = revenue, fill = category_name)) +
  geom_bar(stat = "identity") +
  labs(title = "Revenue by Category", fill = "Category") +
  theme_minimal() +
  theme(legend.position = "none")

revenue_by_categories_bar
```

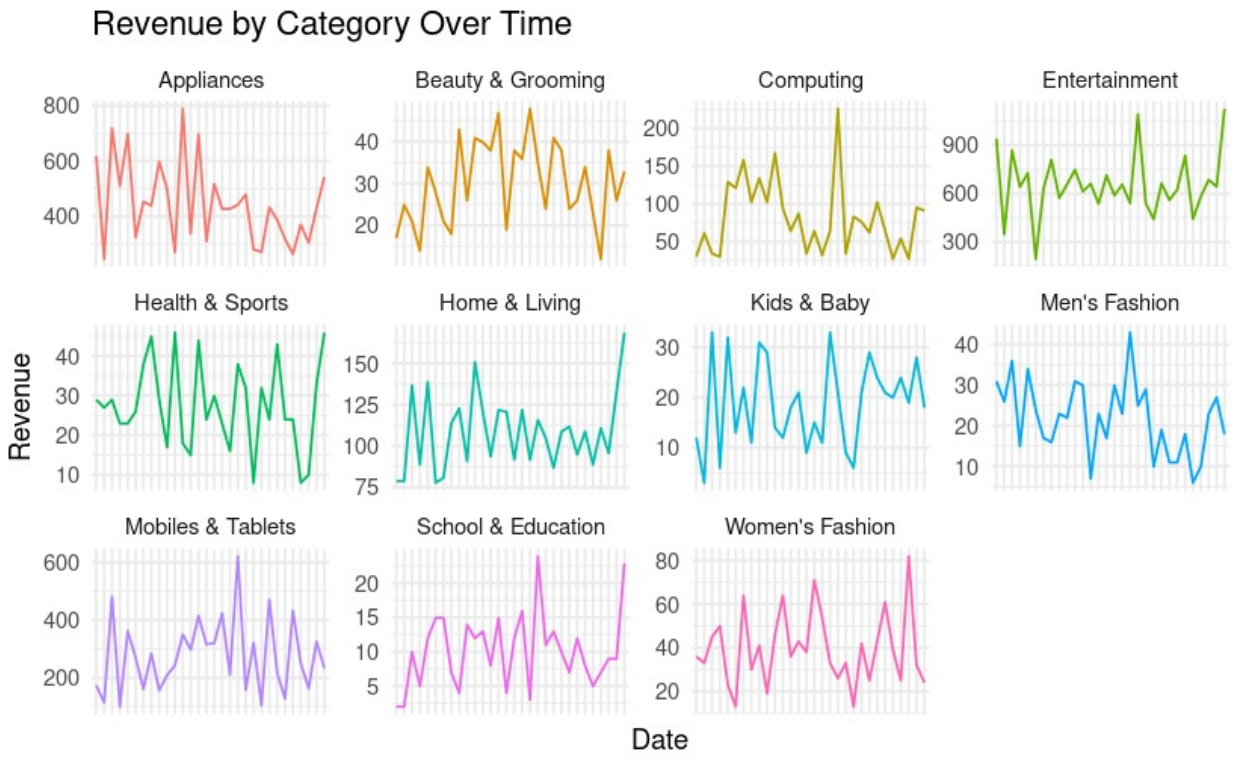

Figure 6: Revenue by Category Over Time

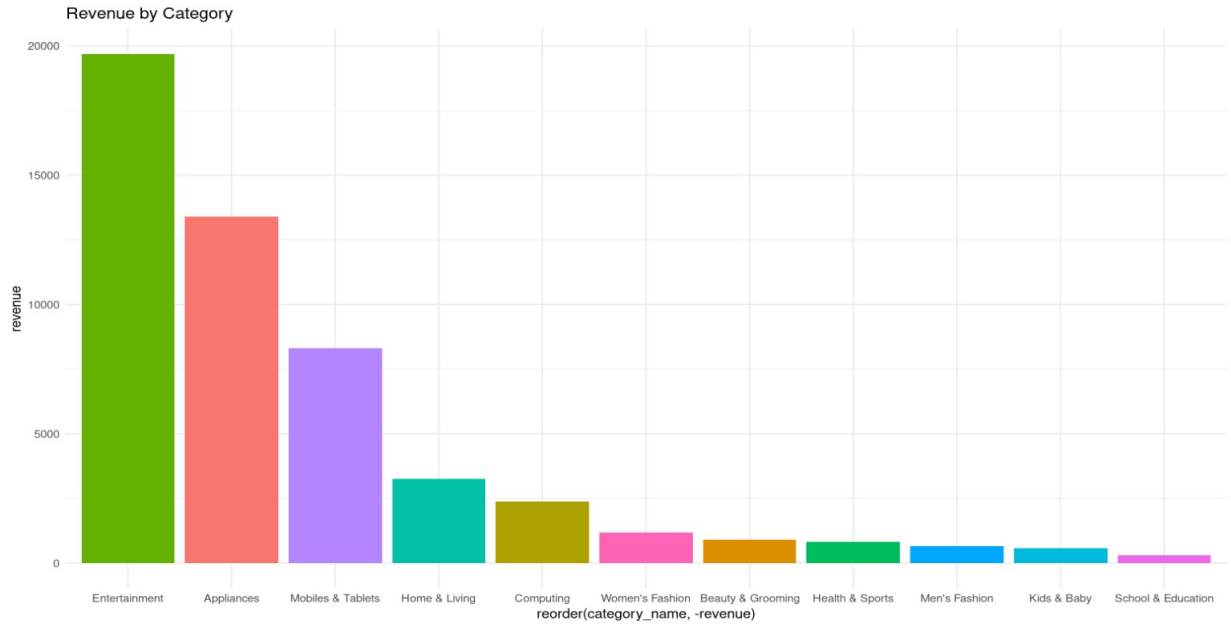

Figure 7: Revenue by Category

The bar chart above shows us the revenue brought in by each category. The highest revenue comes from entertainment, which is almost £20,000. School and education is category with the lowest

revenue, with revenue < £400.

```
# 4
# Top sellers
# We check which sellers are making the most revenue

top_sellers <- RSQLite::dbGetQuery(connection,
                                   "SELECT SUM(b.price) AS revenue, c.seller_name
                                          FROM project_buyer_orders_products a
                                          INNER JOIN project_products b
                                          ON a.product_id = b.product_id
                                          INNER JOIN project_seller c
                                          ON b.seller_id = c.seller_id
                                          GROUP BY c.seller_name
                                          ORDER BY revenue DESC
                                           " )

top_10_sellers <- RSQLite::dbGetQuery(connection,
                                      "SELECT SUM(b.price) AS revenue, c.seller_name
                                          FROM project_buyer_orders_products a
                                          INNER JOIN project_products b
                                          ON a.product_id = b.product_id
                                          INNER JOIN project_seller c
                                          ON b.seller_id = c.seller_id
                                          GROUP BY c.seller_name
                                          ORDER BY revenue DESC
                                          LIMIT 10" )
top_10_sellers_plot <- ggplot(top_10_sellers, aes(x = reorder(
  seller_name, -revenue), y = revenue, fill = seller_name)) +
  geom_bar(stat = "identity",show.legend = FALSE) +
  labs(title = "Top 10 Sellers by Revenue", x = "Seller Name", y = "Revenue") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

top_10_sellers_plot
```

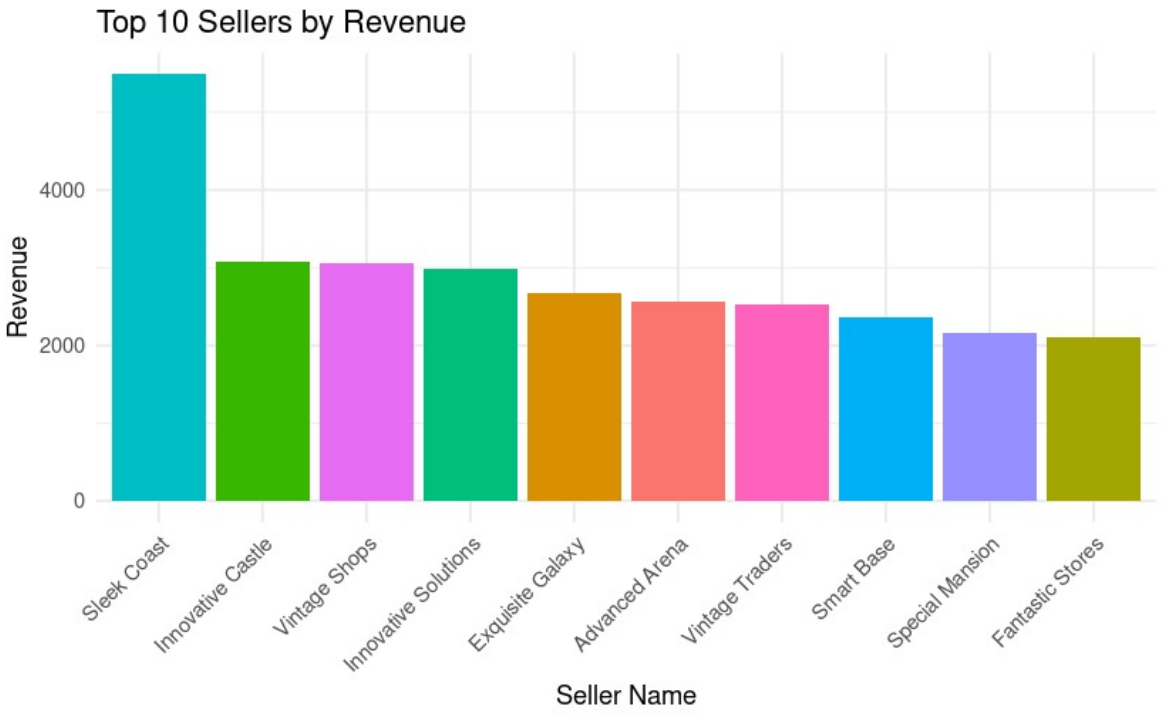

Figure 8: Top 10 Sellers by Revenue

The bar plot above shows top 10 sellers that generate most revenue. The highest revenue comes from seller "Sleek Cost" for £5,489.10.

```
# 5
# Calculating sales by state
sales_by_state <- RSQLite::dbGetQuery(connection,
                                      "SELECT SUM(b.price) AS revenue, c.state
                                      FROM project_buyer_orders_products a
                                      INNER JOIN project_products b
                                      ON a.product_id = b.product_id
                                      INNER JOIN project_contact_details c
                                      ON a.buyer_id = c.buyer_id
                                      GROUP BY c.state
                                      ORDER BY revenue DESC" )
sales_by_state
```

```
   revenue            state
1 44112.12          England
2  7400.70         Scotland
3  7204.05            Wales
4  5673.60 Northern Ireland
```

The result of the codes above shows that England has the highest sales overall, followed by Scotland, Wales and Northern Ireland.

```r
# 6
# Sales_by_city
sales_by_city <- RSQLite::dbGetQuery(connection,
                                     "SELECT SUM(b.price) AS revenue, c.city
                                      FROM project_buyer_orders_products a
                                      INNER JOIN project_products b
                                      ON a.product_id = b.product_id
                                      INNER JOIN project_contact_details c
                                      ON a.buyer_id = c.buyer_id
                                      GROUP BY c.city
                                      ORDER BY revenue DESC" )


sales_by_top10_city <- RSQLite::dbGetQuery(connection,
                                           "SELECT SUM(b.price) AS revenue,
                                           c.city, c.state
                                      FROM project_buyer_orders_products a
                                      INNER JOIN project_products b
                                      ON a.product_id = b.product_id
                                      INNER JOIN project_contact_details c
                                      ON a.buyer_id = c.buyer_id
                                      GROUP BY c.city
                                      ORDER BY revenue DESC
                                      LIMIT 10" )


cities_uk <- data.frame(
  city = c("Edinburgh", "Bangor", "Derby", "St Davids", "Ripon",
           "Doncaster", "Canterbury", "Coventry", "Lisburn", "St Asaph"),
  lon = c(-3.1883, -4.1267, -1.4721, -5.2717, -1.5211,
          -1.1307, 1.0756, -1.5118, -6.0332, -3.4436),
  lat = c(55.9533, 53.2268, 52.9228, 51.8815, 54.1361,
          53.5228, 51.2808, 52.4068, 54.5097, 53.2577)
)

merged_data <- merge(cities_uk, sales_by_top10_city, by = "city", all.x = TRUE)

world <- ne_countries(scale = "medium", returnclass = "sf")
uk <- world[world$admin == 'United Kingdom', ]
df_sf <- st_as_sf(merged_data, coords = c("lon", "lat"), crs = 4326)

# city_colors <- c("London" = "red", "Manchester" = "blue",
#                  "Birmingham" = "green", "Leeds" = "yellow",
#                  "Glasgow" = "purple", "Liverpool" = "orange",
#                  "Newcastle" = "pink", "Sheffield" = "brown",
#                  "Bristol" = "cyan", "Edinburgh" = "magenta")
```

```
highest_rev_plot <- ggplot() +
  geom_sf(data = uk, fill = "white", color = "black") +
  geom_sf(data = df_sf, aes(color = city)) +
  geom_text(data = merged_data , aes(x = lon, y = lat, label = revenue),
            size = 3, hjust = 0, nudge_x = 0.05) +
  scale_color_discrete(name = "City") +
  theme_void() +
  theme(legend.position = "bottom")+
  labs(title = "top 10 cities with highest revenue")

highest_rev_plot
```

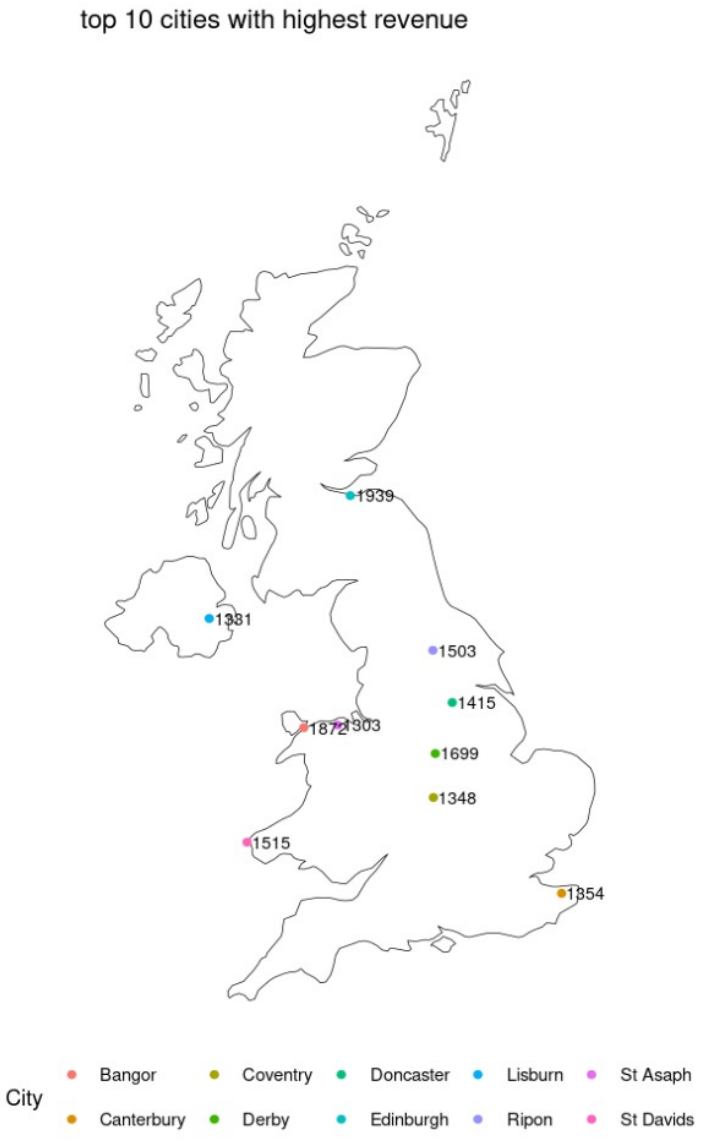

Figure 9: Top 10 Cities with Highest Revenue

The city with the highest sales is Edinburgh in Scotland, but most of the cities with the highest sales belong to England, confirming our previous finding where England has the highest sales overall.

```
# 7
# Using sql to create a dataframe to find the sale per product
Sales_user_type <- RSQLite::dbGetQuery(connection,
                                       "SELECT a.price, b.buyer_id,
                                       c.user_type,datetime(b.order_date,
                                       'unixepoch')
                                       AS date_time
                                      FROM project_products a
                                      INNER JOIN project_buyer_orders_products b
                                      ON a.product_id = b.product_id
                                      INNER JOIN project_buyer c
                                      ON b.buyer_id = c.buyer_id ")

# Using R to sum sales per order
revenue_per_order <- Sales_user_type  %>%
  group_by(date_time, buyer_id, user_type) %>%
  summarise(sum_price = sum(price)) %>%
  ungroup()

# Counting orders per buyer
number_of_orders_per_buyer <- revenue_per_order %>%
  count(buyer_id)

Count_of_buyers <- RSQLite::dbGetQuery(connection,"
                                       SELECT DISTINCT(buyer_id)
                                       FROM project_buyer ")

total_orders_per_buyer <- left_join(Count_of_buyers,
                                    number_of_orders_per_buyer,
                                    by = "buyer_id") %>%
  mutate(order_quantity = ifelse(is.na(n), 0, n))

total_orders_per_buyer<- total_orders_per_buyer %>%
  select(-n)

orders_plot_data <-total_orders_per_buyer %>%
  group_by(order_quantity) %>%
  summarise(num_buyers = n_distinct(buyer_id))

# Plot bar graph
buyers_by_orders_plot <- ggplot(orders_plot_data, aes(x = order_quantity,
                                                      y = num_buyers)) +
  geom_bar(stat = "identity", fill = "skyblue") +
  labs(title = "Number of Buyers by Number of Orders",
       x = "Number of Orders", y = "Number of Buyers") +
```

```
  theme_minimal()

buyers_by_orders_plot
```

Figure 10: Number of Buyers by Number of Orders

The bar plot above indicates most of the buyers have only ordered once or have never ordered before. Some marketing strategies can be developed to increase consumption. For example, we can advertise a free trial for new customers, and send promotional codes or vouchers to existing customers for repeat purchases.

```
# 8
# Dividing the date and time column to just get date

revenue_per_order <- revenue_per_order %>% separate(date_time,
                                                    into = c("date", "time"),
```

```
                                                    sep = " ")

# Adding discounts and shipping
final_balance <- revenue_per_order %>% mutate(
  discount = ifelse(user_type == "VIP", 0.1, 0),
  price_after_discount = sum_price * (1 - discount),
  shipping = case_when(
    sum_price < 100 & user_type != "VIP" & user_type != "Premium" ~ 10,
    user_type %in% c("Premium", "VIP") ~ 0,
    TRUE ~ NA_real_
    ),
  total = price_after_discount + shipping )

average_revenue_after_shipping_discounts <- final_balance %>%
  summarise(total_revenue = sum(total))

# Finding total revenue after adding the discount and shipping cost
total_revenue <- sum(final_balance$total, na.rm = TRUE)
total_revenue
```

```
[1] 40828.59
```

The total revenue decreases by 6.4% after taking into account discounts and shipping costs.

```
# 9
# Finding total sales by the user type

Sales_by_user_type <- RSQLite::dbGetQuery(connection,
                                          "SELECT SUM(b.price)
                                          AS revenue, c.user_type
                                          FROM project_buyer_orders_products a
                                          INNER JOIN project_products b
                                          ON a.product_id = b.product_id
                                          INNER JOIN project_buyer c
                                          ON a.buyer_id = c.buyer_id
                                          GROUP BY c.user_type
                                          ORDER BY revenue DESC" )
Sales_by_user_type
```

```
   revenue user_type
1 19893.32   Premium
2 15889.43       VIP
3 15659.63     Basic
```

As per codes above, premium customers bring more revenue to sellers than other types of buyers. Although the amount spent by basic buyers is similar to that of VIP buyers, it includes the shipping costs without discount.

```r
# 10
# Finding average order revenue

average_revenue_by_order <- mean(revenue_per_order$sum_price)

# The average revenue we make from an order is 93.70 pounds
average_revenue_by_order
```

```
[1] 93.70197
```

Using chunk part above, we found that the average revenue for 1 order is around £93.70.

```r
# 11
# Finding the average number of products in an order
# Writing an SQL query to get the product_id , buyer_id, order_date
# we use unique combinations of buyer_id and order_date as one order_id
Orders <- RSQLite::dbGetQuery(connection,
                              "SELECT a.product_id, b.buyer_id, b.order_date
                              FROM project_buyer_orders_products b
                              INNER JOIN project_products a
                              ON b.product_id = a.product_id" )


# Using dplyr to count how many products have the same buyer_id and date_time
# To see the average quantity of products in an order
order_quantity <- Orders %>%
  group_by( buyer_id, order_date) %>%
  summarise(count = n_distinct(product_id))

average_order_quantity <- round(mean(order_quantity$count))

# From this we can see that on average we have 3 products in an order
average_order_quantity
```

```
[1] 3
```

In average, there are 3 products in 1 order of a customer.

```r
# Count the occurrences of each product_id
top_products <- project_buyer_orders_products %>%
  count(product_id, sort = TRUE) %>%
  top_n(10)

# Display the top 10 products sold
print(top_products)
```

```r
# Plotting the top 10 products sold
plot <- ggplot(top_products, aes(x = reorder(product_id, n), y = n)) +
  geom_bar(stat = "identity", fill = "skyblue") +
  labs(title = "Top 10 Products Sold",
       x = "Product ID",
       y = "Count") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
  # Rotate x-axis labels for better readability

# Define the directory to save the figures
figure_directory <- "figures/"

# Create the directory if it doesn't exist
if (!dir.exists(figure_directory)) {
  dir.create(figure_directory)
}

# Save each plot as an image
this_filename_date <- as.character(Sys.Date())
this_filename_time <- as.character(format(Sys.time(), format = "%H_%M"))

# Save top_10 plot
ggsave(filename = paste0(figure_directory, "top_10_", this_filename_date,
                         "_", this_filename_time, ".png"),
       width = 6, height = 6, plot = top_10)

# Save revenue_by_categories_plot
ggsave(filename = paste0(figure_directory, "revenue_by_categories_plot_",
                         this_filename_date, "_",
                         this_filename_time, ".png"),
       plot = revenue_by_categories_plot)

# Save revenue_by_categories_bar plot
ggsave(filename = paste0(figure_directory,
                         "revenue_by_categories_bar_",
                         this_filename_date, "_",
                         this_filename_time, ".png"),
       plot = revenue_by_categories_bar)

# Save top_10_sellers_plot
ggsave(filename = paste0(figure_directory, "top_10_sellers_plot_",
                         this_filename_date, "_", this_filename_time, ".png"),
       plot = top_10_sellers_plot)

# Save highest_rev_plot
ggsave(filename = paste0(figure_directory, "highest_rev_plot_",
                         this_filename_date, "_", this_filename_time, ".png"),
```

```
       plot = highest_rev_plot)

# Save buyers_by_orders_plot
ggsave(filename = paste0(figure_directory, "buyers_by_orders_plot_",
                         this_filename_date, "_", this_filename_time, ".png"),
       plot = buyers_by_orders_plot)
```

We include the codes above to make sure the plots for every analysis updated smoothly on database with integration in GitHub.

6. Conclusions

The comprehensive project described above takes a holistic approach to simulating and managing an e-commerce environment. The project ensures a thorough understanding of e-commerce dynamics by approaching database design, data generation, pipeline development, and analysis in a systematic manner. The creation of a strong Entity-Relationship (E-R) diagram, which is then translated into logical and physical schemas, provides a solid foundation for database implementation. Synthetic data generation makes accurate simulations possible, while GitHub Actions simplify workflow management and version control.

Extensive data analysis with R yields useful insights into various aspects of e-commerce operations, such as sales trends, product performance, and buyer behavior. These insights make it possible to make more informed decisions and optimize strategies.

Moving forward, it is recommended to improve data validation processes to ensure even greater accuracy and reliability. Furthermore, improving data visualization techniques can help to communicate insights to stakeholders more effectively. Furthermore, advanced analytics and machine learning algorithms can reveal deeper patterns and trends in data, allowing for further optimization of e-commerce operations and strategies.

In conclusion, the project demonstrates a comprehensive approach to e-commerce management, laying the groundwork for future research and development in the ever-changing e-commerce landscape.