

# Data Management Assignment Group 18

## Table of contents

<b>Introduction</b>	<b>2</b>
<b>Part1 Database Design and Implementation</b>	<b>3</b>
1.1 Entity-Relationship Modelling . . . . .	3
1.2 SQL Database Schema Creation . . . . .	8
<b>Part 2 Data Generation and Management</b>	<b>12</b>
2.1 Synthetic Data Generation . . . . .	12
2.2 Data Import and Quality Assurance . . . . .	20
Reading the files . . . . .	21
Creating a loop to read all files and list number of rows and columns in each file	21
Structure of data . . . . .	22
Number of columns and their column names . . . . .	22
Checking unique id column as the primary key in each table . . . . .	22
Data Validation for Each Table . . . . .	23
Referential Integrity for ensuring data integrity . . . . .	36
<b>Part 3 Data Pipeline Generation</b>	<b>37</b>
3.1 GitHub Repository and Workflow Setup . . . . .	37
3.2 GitHub Actions for Continuous Integration . . . . .	37
<b>Part 4 Data Analysis and Reporting</b>	<b>38</b>
4.1 Advanced Data Analysis in R . . . . .	38
4.1.1 Promotion Discount Trend . . . . .	39
4.1.2 Promotion Count Trend . . . . .	40
Promotion Analysis . . . . .	41
4.1.3 Monthly Revenue Trend . . . . .	41
Monthly Revenue Analysis . . . . .	43
4.1.4 Monthly Best-Selling Products . . . . .	43
Monthly Best-Selling Products Analysis . . . . .	45

4.1.5 Monthly Shipping Efficiency . . . . .	46
4.1.6 Monthly Delivery Efficiency . . . . .	48
Shipping and Delivery Efficiency Analysis . . . . .	50
<b>4.2 Comprehensive Reporting with Quarto . . . . .</b>	<b>51</b>
4.2.1 Demographic Distribution of Customers . . . . .	51
A. The Distribution of Gender across Customers . . . . .	51
B. The Distribution of Age across Customers . . . . .	51
C. The Distribution of Careers across Customers (Top 10) . . . . .	52
D. The Distribution of Geographic Location across Customers (Top 10) . . . . .	53
E. The Current Customer Referral Rate . . . . .	55
Customer Analysis . . . . .	55
4.2.2 Product Portfolio . . . . .	55
A. The Distribution of Product Review Scores (Top 10) . . . . .	55
B. The Number of Products supplied by Different Suppliers . . . . .	56
C. The Product Review Scores of Different Suppliers (Best Top 5) . . . . .	57
D. The Product Review Scores of Different Suppliers (Worst Top 5) . . . . .	58
E. The Top 10 Best Selling Products . . . . .	58
Supplier Analysis . . . . .	60
Product Analysis . . . . .	60
4.2.3 Sales Analysis . . . . .	60
A. Order Refund Rate . . . . .	60
Order Refund Analysis . . . . .	61

## Introduction

E-commerce, which entails the buying and selling of goods and services over the internet, includes wide range of transactions, ranging from the online retail stores to the digital market places. It involves various activities, including online payments, digital marketing, and efficient supply chain management. In this project, the aim is to emphasise the ETL flow, automation and data analysis. This report outlines the framework that has been adopted for managing real-world e-commerce data environment comprehensively, covering end-to-end data management. The framework entails 4 major steps for the creation of the efficient Data Base Management System.

**1) Business Requirement** - This involves thorough understanding of the raw data, including aspects related to business.

**2) Conceptual Database Design** - This includes Entity-Relationship (E-R) modelling, which refers to the creation of the entities, their attributes and the intricate relationships between them. This is the core stage for creating the Database Management System.

**3) Logical Database Design** - This involves translating the E-R model into the relational database, in which the entities are represented in the form of tables.

4) **Physical Database Design** - This step involves creation of the tables using Data Definition Language (DDL) commands to store the data within the database, including the decisions on data types of the attributes.

## Part1 Database Design and Implementation

### 1.1 Entity-Relationship Modelling

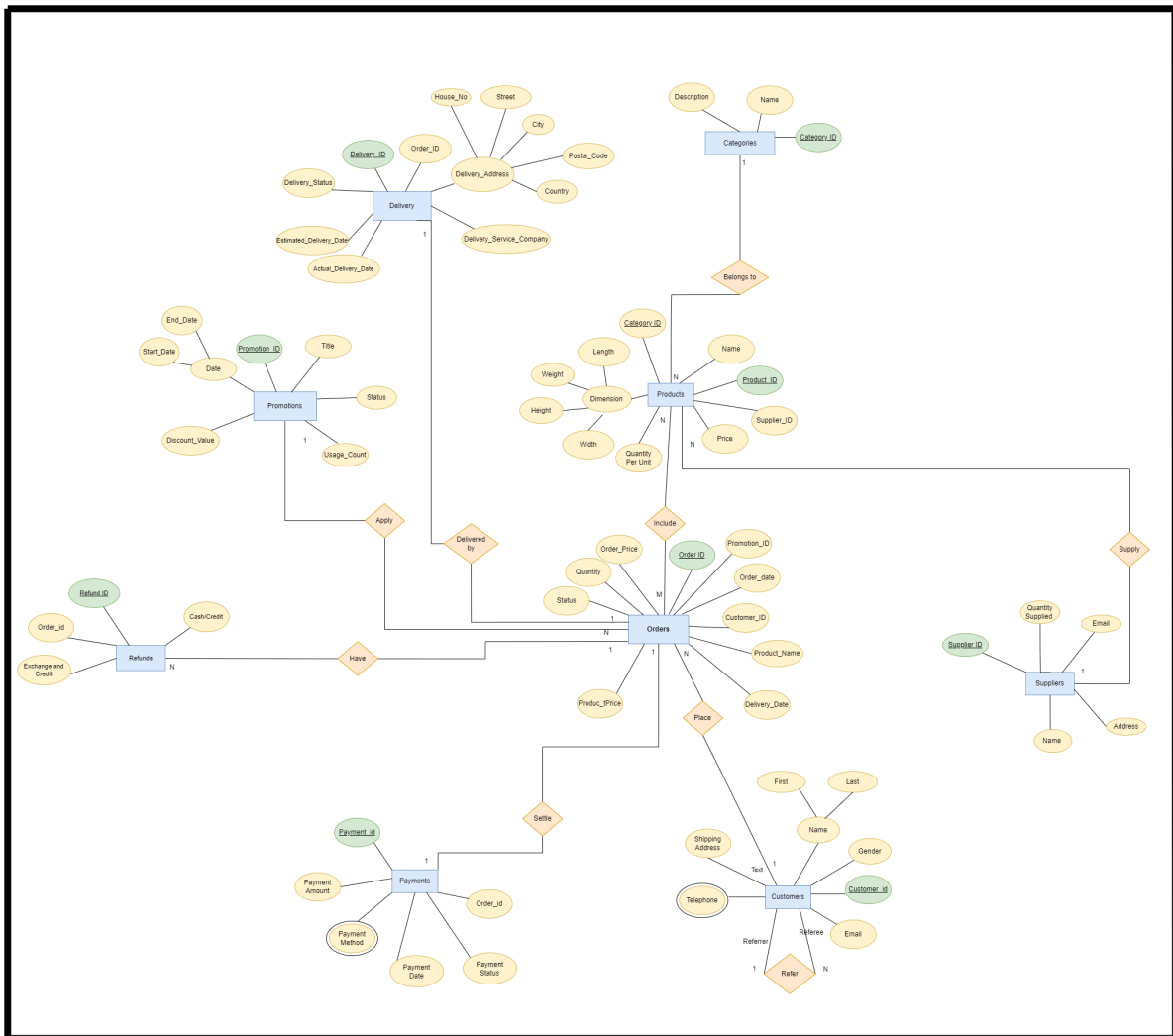


Figure 1: Initial Draft of E-R Diagram

After the initial draft of the E-R diagram, subsequent analysis and data synthesis led to the significant refinements aimed at enhancing the efficiency and clarity of the database structure.

1. The transformation of the “orders” entity into a relationship between “customer” and “product” is justified because of the recognition that an order represents a transaction initiated by a customer for a specific product aligning more closely with real-world e-commerce processes and also this simplified the data model to avoid unnecessary complexity and redundancy. As a result order is an associative entity in the final E-R diagram.
2. Removal of the “payment” entity was justified by the ability to calculate the payment amounts dynamically based on the quantity and price attributes of the order relationship and product entity respectively. It also minimised the risk of discrepancies between the stored payment values and actual order details.
3. The “refund” entity was replaced with the refund status attribute within the order relationship. Recognising that the status of the refund is inherently tied to a specific order, making it more suitable as an attribute within the order relationship rather than a separate entity.
4. The “delivery” entity was replaced with a more detailed “shipment” entity, accommodating attributes such as `shipment_id`, `shipment_date`, and `delivery_date`, to optimise delivery tracking.

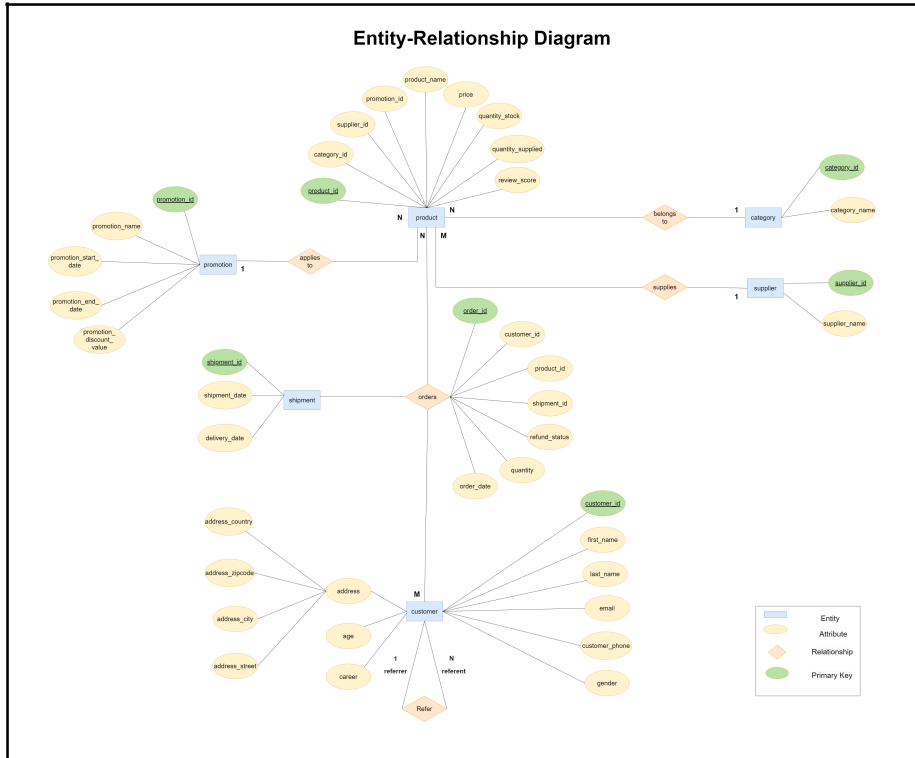


Figure 2: Final ER diagram

After making all the above significant changes the revised E-R diagram has 6 entities that are customer, category, supplier, promotion, shipment, and product. The assumptions that have been taken into consideration for the above E-R modelling are:

- Promotion is only applied to the product.
- There is only one and full final payment for one order that will be calculated using the price, quantity and promotion discount value.
- One order is being shipped and delivered at once, and one order can have many products.
- If one supplier is supplying a particular product, then that product is not going to be supplied by any other supplier. For e.g. if we have product called “ABC Smartphone”. Supplier A exclusively manufactures and supplies the product to our e-commerce platform, and no other supplier can supply the same product.

### Logical Database Schema:

The general form of representing the entities and attributes in logical schema is:

relation\_name {attribute 1, attribute2, attribute3, .....attribute n}

The below represents the logical database schema comprising entities and their attributes derived from the E-R diagram. A single underline represents the Primary Key and a double underline represents the Foreign Key in the table.

1. customer {customer\_id, first\_name, last\_name, email, gender, age, career, customer\_phone, address\_country, address\_zipcode, address\_city, address\_street, referred\_by}
2. category {category\_id, category\_name}
3. supplier {supplier\_id, supplier\_name}
4. promotion {promotion\_id, promotion\_name, promotion\_start\_date, promotion\_end\_date, promotion\_discount\_value}
5. shipment {shipment\_id, shipment\_date, delivery\_date}
6. product {product\_id, category\_id, supplier\_id, promotion\_id, product\_name, price, quantity\_stock, quantity\_supplied, review\_score}
7. orders {order\_id, customer\_id, product\_id, shipment\_id, quantity, refund\_status, order\_date}

Figure 3: Logical Schema

**Relationship sets:**

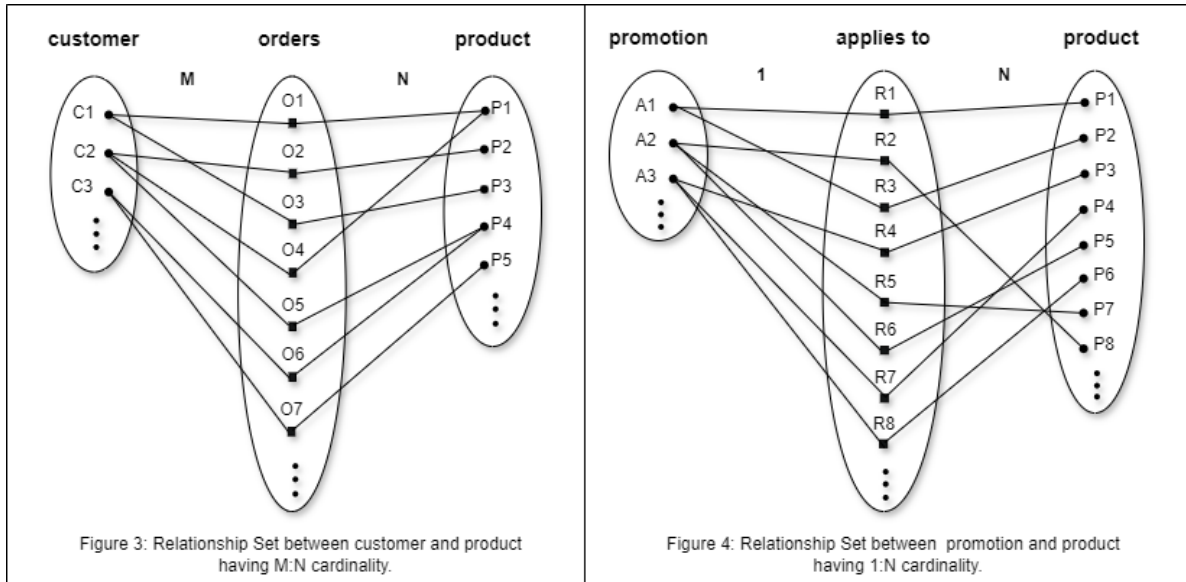


Figure 3 illustrates many-to-many (M:N) relationship between customer and product, indicating that multiple customers can order multiple products, and conversely multiple products can be ordered by various customers.

Figure 4 illustrates one-to-many (1:N) relationship between promotion and product, that a single promotion code may be applied to multiple products, whereas multiple products can be associated with a single promotion code only.

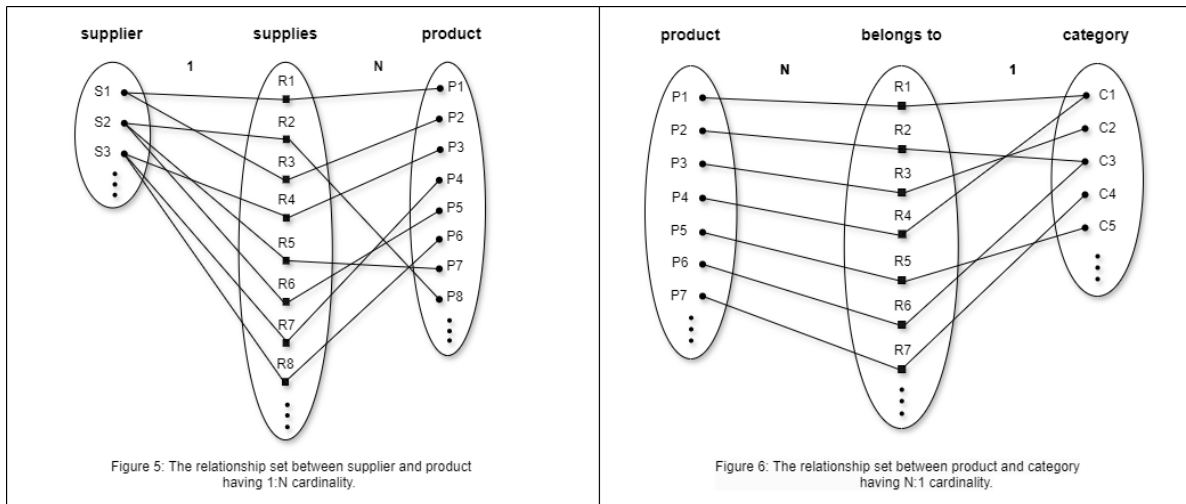


Figure 5 illustrates one-to-many (1:N) relationship between supplier and product. This indicates that a single supplier can supply multiple products, and multiple products can be supplied by a single supplier.

Figure 6 illustrates many-to-one (N:1) relationship between product and category, where multiple products can be categorised under a single category, while each category can have multiple products.

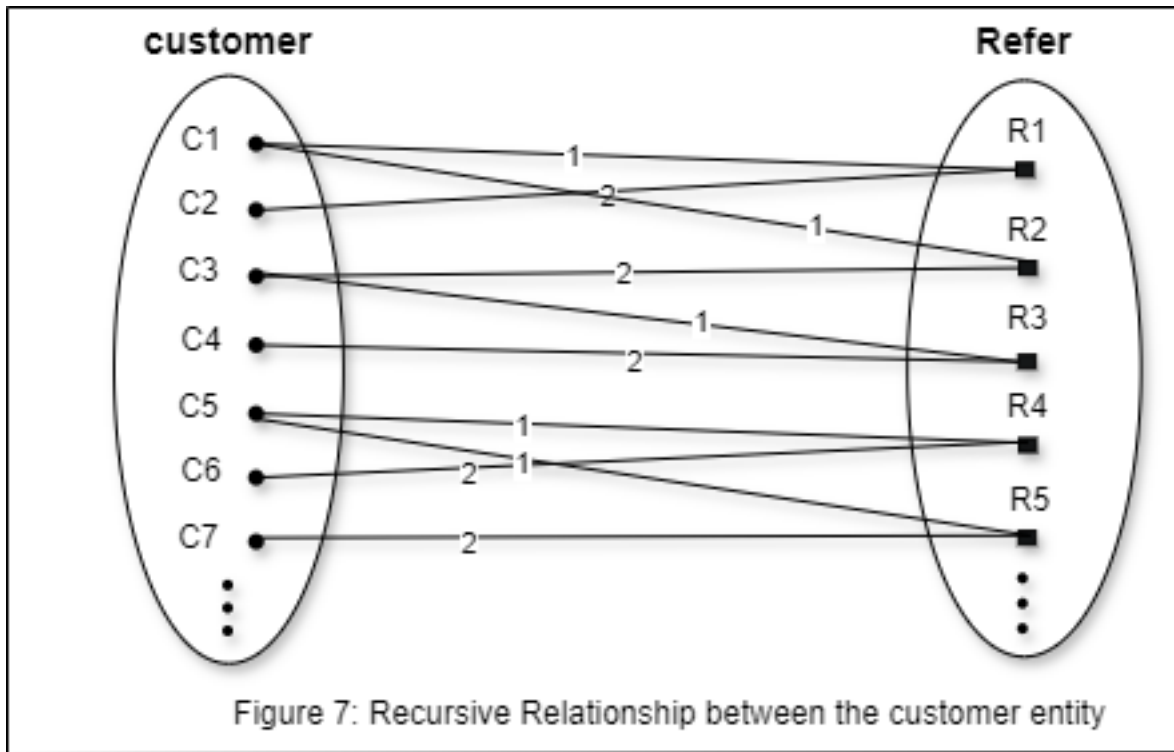


Figure 7 illustrates one-to-many (1:N) recursive relationship involving the customer entity. This indicates that while one customer can refer multiple customers, where each customer can only be referred by one customer. The number “1” above the relationship lines represents the referrer role and “2” represents the referent.

## 1.2 SQL Database Schema Creation

In this part, E-R diagram is translated into a functional SQL database schema.

Creating table schema for all tables

1. Creating customer table

```
1 RSQLite::dbExecute(connection,"
2
3 CREATE TABLE IF NOT EXISTS customer (
4
```



```

5      customer_id VARCHAR(10) PRIMARY KEY,
6      first_name VARCHAR(50) NOT NULL,
7      last_name VARCHAR(50) NOT NULL,
8      email VARCHAR(50) NOT NULL,
9      gender VARCHAR(20) NOT NULL,
10     age INT NOT NULL,
11     career VARCHAR(50) NOT NULL,
12     customer_phone VARCHAR(20) NOT NULL,
13     address_country VARCHAR(50) NOT NULL,
14     address_zipcode VARCHAR(20) NOT NULL,
15     address_city VARCHAR(20) NOT NULL,
16     address_street VARCHAR(50) NOT NULL,
17     referred_by VARCHAR(10) NULL
18
19 );
20 ")

```

## 2. Creating category table

```

1  RSQLite::dbExecute(connection,"
2
3  CREATE TABLE IF NOT EXISTS category (
4
5      category_id VARCHAR(10) PRIMARY KEY,
6      category_name VARCHAR(50) NOT NULL
7
8  );
9
10 ")

```

## 3. Creating supplier table

```

1  RSQLite::dbExecute(connection,"
2  CREATE TABLE IF NOT EXISTS supplier (
3
4      supplier_id VARCHAR(10) PRIMARY KEY,
5      supplier_name VARCHAR(50) NOT NULL
6
7  );
8
9  ")

```

## 4. Creating promotion table

```

1  RSQLite::dbExecute(connection,"
2
3  CREATE TABLE IF NOT EXISTS promotion (
4
5      promotion_id VARCHAR(10) PRIMARY KEY,
6      promotion_name VARCHAR(20) NOT NULL,
7      promotion_start_date DATE NOT NULL,
8      promotion_end_date DATE NOT NULL,
9      promotion_discount_value FLOAT NOT NULL
10
11 );
12
13 ")

```

#### 5. Creating shipment table

```

1  RSQLite::dbExecute(connection,"
2
3  CREATE TABLE IF NOT EXISTS shipment (
4
5      shipment_id VARCHAR(10) PRIMARY KEY,
6      shipment_date DATE NOT NULL,
7      delivery_date DATE NOT NULL
8  );
9
10 ")

```

#### 6. Creating product table

```

1  RSQLite::dbExecute(connection,"
2
3  CREATE TABLE IF NOT EXISTS product (
4
5      product_id VARCHAR(10) PRIMARY KEY,
6      category_id VARCHAR(10) NOT NULL,
7      supplier_id VARCHAR(10) NOT NULL,
8      promotion_id VARCHAR(10) NULL,
9      product_name VARCHAR(20) NOT NULL,
10     price INT NOT NULL,
11     quantity_stock INT NOT NULL,
12     quantity_supplied INT NOT NULL,
13     review_score FLOAT NOT NULL,

```

```

14     FOREIGN KEY ('category_id') REFERENCES category('category_id'),
15     FOREIGN KEY ('supplier_id') REFERENCES supplier('supplier_id'),
16     FOREIGN KEY ('promotion_id') REFERENCES promotion('promotion_id')
17
18 );
19
20 ")
```

## 7. Creating orders table

```

1  RSQLite::dbExecute(connection,"
2
3  CREATE TABLE IF NOT EXISTS orders (
4
5      order_id VARCHAR(20) NOT NULL,
6      customer_id VARCHAR(10) NOT NULL,
7      product_id VARCHAR(10) NOT NULL,
8      shipment_id VARCHAR(10) NOT NULL,
9      quantity INT NOT NULL,
10     refund_status VARCHAR(20) NOT NULL,
11     order_date DATE NOT NULL,
12     FOREIGN KEY ('customer_id') REFERENCES customer('customer_id'),
13     FOREIGN KEY ('product_id') REFERENCES product('product_id'),
14     FOREIGN KEY ('shipment_id') REFERENCES shipment('shipment_id')
15     PRIMARY KEY (order_id,product_id,customer_id,shipment_id)
16
17
18 );
19
20 ")
```

After creating the schema, below lists all of the tables from the database to check what have been created.

```

1  RSQLite::dbListTables(connection)
```

In the development of our database (ecomdata.db), the principle of ETL (Extract, Transform, Load) was used rather than ELT (Extract, Load, Transform) as ETL is more efficient than ELT in various aspects such as in ETL we can clean and transform the data before loading it into the database tables. This indicates that the data warehouse or database has only verified data, resulting in good data quality and consistency. Also, as the transformation happens before loading, any mistakes or inconsistencies may be discovered and handled immediately,

lowering the risk of ingesting the faulty data into the database. The process by which we employed ETL in our project is described below -

**Extract** - The process began with the extraction of data that was generated using both Large Language Models (LLMs) and Mockaroo, which aligns with the extraction phase, where data is collected from various sources and prepared for further processing.

**Transform** - After data extraction, the transformation process involved converting the extracted data into CSV files and further we performed some validation tests. This step was taken to ensure the data quality and consistency, which is the most crucial part of the ETL transformation. Furthermore, normalisation processes were used to standardise the data, for improving its integrity.

**Load** - Finally, the validated and normalised data was loaded into the tables in the database for further analysis.

## Part 2 Data Generation and Management

### 2.1 Synthetic Data Generation

Based on the logical schema, the physical schema involved establishing the details in the database, including tables, columns, relationships, and constraints. Furthermore, the attributes of each entity ensured that the normalisation issues were resolved. Following this, the dependancy between entities prioritised the order when generating the data. Thus, the data within the 1:N entity were created before the M:N entity due to the foreign key for the associative entity.

Table 1: Data Generation Order

Order	Entity / Associative Entity	Number of Observations
1	supplier	20
2	category	20
3	promotion	30
4	customer	200
5	product	200
6	shipment	389
7	orders	389

Based on the nature of each attribute, customer data was created through “Mockaroo” to obtain the initial data.

Field Name	Type	Options
customer_id	Row Number	blank: 0 % $\Sigma$ $\times$
first_name	First Name	blank: 0 % $\Sigma$ $\times$
last_name	Last Name	blank: 0 % $\Sigma$ $\times$
email	Email Address	blank: 0 % $\Sigma$ $\times$
phone_number	Phone	format: ###-###-#### blank: 0 % $\Sigma$ $\times$
address	Street Address	blank: 0 % $\Sigma$ $\times$
city	City	blank: 0 % $\Sigma$ $\times$
state	State	restrict states... All Countries blank: 0 % $\Sigma$ $\times$
country	Country	restrict countries... blank: 0 % $\Sigma$ $\times$
zipcode	Postal Code	blank: 0 % $\Sigma$ $\times$
referred by	Row Number	blank: 0 % $\Sigma$ $\times$

Figure 4: Data generation screenshot on mockaroo website

However, the generated data lacks authentic features, such as age, career, and address, and has no logical meaning for each customer. To make data reflect realistic patterns and distributions, further LLM was used to enhance the practical characteristics of the initial data. The following images show the ChatGPT prompts.



You



MOCK\_DATA\_modified.xlsx

Spreadsheet

Based on the attached file, can you help me improve it to be more realistic

1. Age Distribution - Mean: 35 / Standard deviation: 10

2. Career Distribution:

- 12 to 17 years old: Primarily students, which may include part-time jobs or internships.

- 18 to 22 years old: Students, interns, and young individuals who are just entering the workforce (such as entry-level positions).

- 23 to 30 years old: A wider range of career choices, including positions that require professional degrees or experience.

- 31 years old and above: This includes all professions, assuming that by this age, individuals have stabilized in their career paths, possibly covering mid-level to senior positions.

1. customer phone: begin with "+44" & one space & 10-digit numbers

Figure 5: Data improvement request to ChatGPT for customer table



**You**

[Category Table]

I have a 3NF category table with the following fields:

\*category\_id: It should be a unique identifier of a product category. It should be a 10-digit length character. It is the primary key of the table. It can't be empty.

\*category\_name: It should be the name of the product category. It should be in character format. You should use the real and meaningful category name in an e-commerce company. It can't be empty.

Can you generate 20 records for me and provide me the result in an excel file?



**ChatGPT**

I've created an Excel file with 20 records for the category table as requested. You can download the file from the link below:

[Download Category\\_Table.xlsx \[-\]](#)



Figure 6: Data improvement request to ChatGPT for category table



**You**

[Supplier Table]

I have a 3NF supplier table with the following fields:

\*supplier\_id: It should be a unique identifier of a product supplier. It should be a 10-digit length character. It is the primary key of the table. It can't be empty.

\*category\_name: It should be the name of the product supplier. It should be in character format.

You should use the real and meaningful supplier name in an e-commerce company. It can't be empty.

Can you generate 20 records for me and provide me the result in an excel file?



**ChatGPT**

I've created an Excel file with 20 records for the supplier table as requested. You can download the file from the link below:

[Download Supplier\\_Table.xlsx \[-\]](#)



Figure 7: Data improvement request to ChatGPT for supplier table





**You**

[Promotion Table]

I have a 3NF promotion table with the following fields:

\*promotion\_id: It should be a unique identifier of a promotion. It should be a 10-digit length character. It is the primary key of the table. It can't be empty.

\*promotion\_name: It should be the name of a e-commerce promotion. It should be in character format. You should use the real and meaningful promotion name in an e-commerce environment. It can't be empty.

\*promotion\_start\_date: It's the start date of a promotion. It should be in DD-MM-YY format. You should ensure the promotion\_start\_date is real and meaningful, and match with its the promotion\_name. Like the Black Friday promotion should happen in November or December. It can't be empty. The time period should be from 2023 January to 2024 February.

\*promotion\_end\_date: It's the end date of a promotion. It should be in DD-MM-YY format. You should ensure the promotion\_end\_date is real and meaningful, and match with its the promotion\_name. Like the Black Friday promotion should end in November or December. You should also ensure each promotion\_id's promotion\_end\_date is later than its promotion\_start\_date. It can't be empty. The time period should be from 2023 January to 2024 February.

\*promotion\_discount\_value: It's the discount value of one promotion. It should be a float number in range [0,1]. The new price after promotion should be price \* (1-promotion\_discount\_value). It can't be empty. Its value shouldn't be too high and most of them should be smaller than 0.6.

Can you generate 30 records for me and provide me the result in an excel file?



**ChatGPT**

I've generated 30 records for the promotion table with the updated requirements and saved them to an Excel file. You can download the file from the link below:

[Download Promotion\\_Table.xlsx \[?\]](#)



Figure 8: Data improvement request to ChatGPT for promotion table



**You**

[Shipment Table]

I have a 3NF shipment table with the following fields:

\*shipment\_id: It should be a unique identifier of a shipment matching with an order. It should be a 10-digit length character and the first 4 digits should be "SHIP", and the other 6 digits should be numbers. It is the primary key of the table, and it can't be empty.

\*shipment\_date: It should be the date that an order starts to be shipped. It should be in DD-MM-YY format and the time period should be from 2023 January to 2024 March.

\*delivery\_date: It should be the date that an order has been received by the customer. It should be in DD-MM-YY format and the time period should be from 2023 January to 2024 March.

Can you generate 389 records for me and provide me the result in an excel file?



**ChatGPT**

I've generated 389 records for the shipment table as requested and saved them to an Excel file.

You can download the file from the link below:


[Download Shipment\\_Table.xlsx](#) [↗]



Figure 9: Data improvement request to ChatGPT for shipment table

 You

 **category.csv**  
Spreadsheet

 **supplier.csv**  
Spreadsheet

 **promotion.csv**  
Spreadsheet

[Product Table]

I already have a 3NF category table, supplier table, and promotion table (I'll show you later). But now I have a 3NF product table with the following fields:

\*product\_id: It should be a unique identifier of a product, in a 10-digit length character. It is the primary key of the table. It can't be empty. When the category\_id is the same, you should keep the first three digit of the product\_id is the same, please use alphabets; When the supplier\_id is the same, you should keep the 4-6 digits of the product\_id is the same, please use numbers; For the 7-10 digits of the product\_id, you can randomly give numbers to them.

\*category\_id: It should be the product's category\_id, and you should choose values from the category table's category\_id column I provided you, it should be real and meaningful, it can't be empty, all category\_id in the category table should be included in this column.

\*supplier\_id: It should be the product's supplier's id, and you should choose values from supplier table's supplier\_id I provide you, it should be real and meaningful, it can't be empty, all supplier id in supplier table should be included in this column.

\*promotion\_id: It should be the promotion that applied to the product, you should choose values from promotion table's promotion\_id I provide you, it should be real and meaningful, it can be empty because some products may not have any promotions. All promotion\_id in promotion table should be included in this column.

\*product\_name: It should be the product's name, you should choose names according to the category\_id you choose from category table, which means this product is actually belongs to this category. It should be a real and meaningful product name that actually match with its category\_name and category\_id. It can't be empty and should be unique and specific. Could you please name the product\_name in such format: the specific type of the product(should be a real type like Mattress, not be general like product A) - the specific product brand(should be a real type like John Lewis, not be general like brand A) - the specific product name(like silent night) , for example, [Mattress John Lewis silent night].

\*price: It should be the selling price of the product, it should be real and meaningful, and can't be empty, in number format. You should generate price values according to its product\_name and category\_name, make it as real as possible.

\*quantity\_stock: It should be the current stock quantity of the product. It can't be empty and should be integer, be real and meaningful.

\*quantity\_supplied: It should be the total supply amount of the product that is supplied by its suppliers. It can't be empty and should be integer, be real and meaningful.

\*average\_score: it should be the product's average review score provided by its customers, it should be real and meaningful. It can't be empty, and should be integer in the range [1,5].

Can you generate 200 records for me and provide me the result in an excel file?

 ChatGPT

With the structure of the category, supplier, and promotion tables reviewed, I'll now proceed to generate 200 records for the product table. This process includes creating unique product identifiers that meet the specified requirements, assigning category IDs, supplier IDs, promotion

Figure 10: Data improvement request to ChatGPT for product table



You



product.csv  
Spreadsheet



customer.csv  
Spreadsheet



shipment.csv  
Spreadsheet

[Order Table]

I already have a 3NF customer table, product table and shipment table(I'll show you later). But now I have a 3NF order table with the following fields:

\*order\_id: It should be the unique identifier, it should be in a 14 digit length character and begin with "ORDER". It's the primary key of the table and can't be empty, but order\_id can be duplicated because one order may have many different products.

\*customer\_id: It should be the customer who places this order. You should choose values from the customer table's customer\_id column I provided you, one customer\_id can place many order\_id, and it could not be empty, but can be duplicated because one customer can place many different orders.

\*product\_id: It should be the product's id that included in this order. You should choose values from the product table's product\_id column I provided you. It can not be empty but it can be duplicated because one product can be included in many different orders.

\*shipment\_id: It should be the shipment's id of this order. You should choose values from the shipment table's shipment\_id column I provided you. One order\_id only match with one shipment\_id. It can not be empty but can be duplicated because order\_id can also be duplicated.

\*quantity: It should be the quantity of the product in one order. It should be a positive integer and can't be empty. The value should be real and meaningful.

\*refund\_status: It should be characters and has two values "yes" "no" which means whether this order is requested to be refunded or not , the number of "No" should be a little bit more than "Yes".

\*order\_date: It should be the date when the order is placed. It can't be empty and should in DD-MM-YY format. You should make sure the order\_date should be same or a little bit earlier than the shipment\_date in shipment table.

Can you generate 1000 records for me and provide me the result in an excel file?

Figure 11: Data improvement request to ChatGPT for order table

## 2.2 Data Import and Quality Assurance

The report entails the detailed R code that was developed, alongside with the creation of CSV files of the normalised data. Subsequently validation checks were performed on the data to ensure the data quality standards. These validation checks were essential to ensure the accuracy, completeness, and consistency of the data. On the successful validation, the

verified data was loaded into the tables in the database. The database created is named as “ecomdata.db”.

For example, Certain validation checks were performed on the attributes of the customer data in the CSV file to ensure the data accuracy which was essential for the further analysis such as:

- Email - The customer requires to enter a valid email id.
- Gender - It has 4 categories that is “male”, “Male”, “female”, “Female”.
- Age - The age of the customer should be in the range of 1 to 100.
- Zip Code - It should contain 6 characters with a space after 3 characters.
- Referred Check - The new customers that are being referred, their customer id should be presented in the customer\_id column in the customer table.

Similarly, certain validations were performed for other CSV files before loading them in the database.

## Reading the files

Firstly, all the CSV files are listed before doing the validations.

```
1 all_files <- list.files("data_upload/")
2 all_files
```

## Creating a loop to read all files and list number of rows and columns in each file

```
1 for (variable in all_files) {
2   filepath <- paste0("data_upload/",variable)
3   file_contents <- readr::read_csv(filepath)
4
5   number_of_rows <- nrow(file_contents)
6   number_of_columns <- ncol(file_contents)
7
8   #Printing the number of rows and columns in each file
9   print(paste0("The file: ",variable,
10               " has: ",
11               format(number_of_rows,big.mark = ","),
12               " rows and ",
13               number_of_columns," columns"))
}
```

```

14
15   number_of_rows <- nrow(file_contents)
16
17   print(paste0("Checking for file: ",variable))
18
19
20   #Printing True if the first column is the primary key column else printing False
21   print(paste0(" is ",nrow(unique(file_contents[,1]))==number_of_rows))
22 }

```

## Structure of data

```

1 for (variable in all_files) {
2   filepath <- paste0("data_upload/",variable)
3   file_contents <- readr::read_csv(filepath)
4   str_data <-str(file_contents)
5   print(paste0(str_data,"Structure of the file ", variable))
6 }

```

## Number of columns and their column names

```

1 for (variable in all_files) {
2   filepath <- paste0("data_upload/",variable)
3   file_contents <- readr::read_csv(filepath)
4   column_names <-colnames(file_contents)
5   print(paste0("File ", variable, " has column as ",column_names))
6 }

```

## Checking unique id column as the primary key in each table

```

1 for (variable in all_files) {
2   filepath <- paste0("data_upload/", variable)
3   file_contents <- readr::read_csv(filepath)
4   primary_key <- nrow(unique(file_contents[,1])) == nrow(file_contents)
5   print(paste0("Primary key of ", variable, " is ", primary_key))
6 }

```



## Data Validation for Each Table

### 1. Validation for customer data

```
1 fetch_existing_customer_ids <- function(connection) {
2   query <- "SELECT DISTINCT customer_id FROM customer"
3   existing_ids <- dbGetQuery(connection, query)$customer_id
4   return(existing_ids)
5
6 }
7
8 #Customer data validation and Reffered by referencial integrity
9 validate_and_prepare_customer_data <- function(data, existing_ids) {
10
11   #Validation for customer ID
12   customer_id_check <- grepl("^CUST[0-9]{6}$", data$customer_id)
13   data <- data[customer_id_check, ]
14
15   #Validation for email
16   email_check <- grepl("[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\. [a-zA-Z]{2,}$", data$email)
17   data<- data[email_check, ]
18
19   #Validation for gender
20   gender_check <- c("male","female","Female","Male")
21   data<- data[data$gender %in% gender_check, ]
22
23   #Validation for age
24   age_check <- 1:100
25   data <- data[data$age %in% age_check, ]
26
27   #Validation for phone number
28   phone_check <- grepl("^\\+44 \\d{10}$", data$customer_phone)
29   data <- data[phone_check, ]
30
31   #Validation for zip code
32   zipcode_check <- grepl("^\\w{3} \\w{3}$", data$address_zipcode)
33   data <- data[zipcode_check, ]
34
35   #Reffered by check
36   unique_customer_ids <- unique(c(data$customer_id, existing_ids))
37   #Validate 'referred by' IDs
38   valid_referral_flags <- (data$referred_by == "") | data$referred_by %in% unique_customer_ids
```

```

39   data <- data[valid_referral_flags, ]
40
41   return(data)
42 }
43
44 #Fetch existing customer IDs from the database
45 existing_customer_ids <- fetch_existing_customer_ids(connection)
46
47 customer_file_paths <- list.files(path = "data_upload", pattern = "customer.*\\.csv$", full.
48 #Initialising empty dataframe
49 customer_possible_data <- data.frame()
50
51 customer_primary_key <- "customer_id"
52
53 #Read each customer CSV file and check for the existence of the primary key in the database
54 for (file_path in customer_file_paths) {
55   cat("Starting processing file:", file_path, "\n")
56   #Read the current file
57   customer_data <- read.csv(file_path)
58
59   #Iterate through each row of the file
60   for (i in seq_len(nrow(customer_data))) {
61     new_record <- customer_data[i, ]
62     primary_key_value <- new_record[[customer_primary_key]]
63     conditions <- paste(customer_primary_key, "=", paste0("'", primary_key_value, "'"))
64
65     #Check if a record with the same primary key exists in the database
66     record_exists_query <- paste("SELECT COUNT(*) FROM customer WHERE", conditions)
67     record_exists_result <- dbGetQuery(connection, record_exists_query)
68     record_exists <- record_exists_result[1, 1] > 0
69
70     if(record_exists) {
71       cat("Record with primary key", primary_key_value, "already exists in the database.\n")
72     }
73     if (!record_exists) {
74       #Check if the primary key value of the new record is unique in the temporary dataframe
75       if (!primary_key_value %in% customer_possible_data[[customer_primary_key]]) {
76         customer_possible_data <- rbind(customer_possible_data, new_record)
77       }
78     }
79
80     cat("Finished processing file:", file_path, "\n")

```



```

81 }
82 }
83 cat("Starting validation for new records.\n")
84 customer_possible_data <- validate_and_prepare_customer_data(customer_possible_data, exist)
85 cat("Validation completed for new records.\n")
86 }
87
88
89 if (nrow(customer_possible_data) > 0)
90 {
91   cat("Starting to insert validated data into the database. Number of records: ", nrow(customer_possible_data), "\n")
92
93   #Ingesting prepared data to our database
94   dbWriteTable(connection, name = "customer", value = customer_possible_data, append = TRUE, as.data.frame(customer_possible_data))
95   cat("Data insertion completed successfully.\n")
96 } else
97 {
98   cat("No valid customer data to insert into the database.\n")
99 }

```

Recursive relationship in the customer table

In the 'customer' table, 'customer\_id' serves as the primary key, that uniquely identifies each customer and 'referred\_by' is the attribute that defines the customer who refers this customer serving as the link between different customers within the same customer table.

## 2. Validations for category data

```

1 validate_and_prepare_category_data <- function(data) {
2
3   # Validation for category ID
4   category_id_check <- grepl("^[A-Za-z0-9]{10}$", data$category_id)
5   data <- data[category_id_check,]
6
7   return(data)
8 }
9
10 # Fetch existing category IDs from the database
11
12 category_file_paths <- list.files(path = "data_upload", pattern = "category.*\\.csv$", full.names = TRUE)
13
14 # Define the primary key column for the category table
15 category_primary_key <- "category_id"

```

```

16
17 #Initialising empty data frame
18 category_possible_data <- data.frame()
19
20 # Read each category CSV file and check for the existence of the primary key in the database
21 for (file_path in category_file_paths) {
22
23     cat("Starting processing file:", file_path, "\n")
24
25     # Read the current file
26     category_data <- readr::read_csv(file_path)
27
28     # Iterate through each row of the file
29     for (i in seq_len(nrow(category_data))) {
30         new_record <- category_data[i, ]
31         primary_key_value <- new_record[[category_primary_key]]
32         conditions <- paste(category_primary_key, "=", paste0("'", primary_key_value, "'"))
33
34         # Check if a record with the same primary key exists in the database
35         record_exists_query <- paste("SELECT COUNT(*) FROM category WHERE", conditions)
36         record_exists_result <- dbGetQuery(connection, record_exists_query)
37         record_exists <- record_exists_result[1, 1] > 0
38
39         if(record_exists) {
40             cat("Record with primary key", primary_key_value, "already exists in the database.\n")
41         }
42         if (!record_exists) {
43             # Check if the primary key value of the new record is unique in the temporary dataframe
44             if (!primary_key_value %in% category_possible_data[[category_primary_key]]) {
45                 category_possible_data <- rbind(category_possible_data, new_record)
46             }
47         }
48
49         cat("Finished processing file:", file_path, "\n")
50
51     }
52
53 }
54 cat("Starting validation for new records.\n")
55 category_possible_data <- validate_and_prepare_category_data(category_possible_data)
56 cat("Validation completed for new records.\n")
57

```

```

58 if (nrow(category_possible_data) > 0) {
59   cat("Starting to insert validated data into the database. Number of records: ", nrow(category_possible_data), "\n")
60
61   # Ingesting prepared data to our database
62   dbWriteTable(connection, name = "category", value = category_possible_data, append = TRUE, as.character())
63   cat("Data insertion completed successfully.\n")
64 } else
65 {
66   cat("No valid category data to insert into the database.\n")
67 }

```

### 3. Validations for supplier data

```

1  validate_and_prepare_supplier_data <- function(data) {
2    # Validation for supplier ID
3    supplier_id_check <- grepl("^[A-Za-z0-9]{10}$", data$supplier_id)
4    data <- data[supplier_id_check,]
5
6    return(data)
7  }
8
9  # Fetch existing supplier IDs from the database
10
11  supplier_file_paths <- list.files(path = "data_upload", pattern = "supplier.*\\.csv$", full.names = TRUE)
12
13  # Define the primary key column for the supplier table
14  supplier_primary_key <- "supplier_id"
15
16  #Initialising empty data frame
17  supplier_possible_data <- data.frame()
18
19  # Read each supplier CSV file and check for the existence of the primary key in the database
20  for (file_path in supplier_file_paths) {
21
22    cat("Starting processing file:", file_path, "\n")
23
24    # Read the current file
25    supplier_data <- readr::read_csv(file_path)
26
27    # Iterate through each row of the file
28    for (i in seq_len(nrow(supplier_data))) {
29      new_record <- supplier_data[i, ]

```

```

30 primary_key_value <- new_record[[supplier_primary_key]]
31 conditions <- paste(supplier_primary_key, "=", paste0("'", primary_key_value, "'"))
32
33 # Check if a record with the same primary key exists in the database
34 record_exists_query <- paste("SELECT COUNT(*) FROM supplier WHERE", conditions)
35 record_exists_result <- dbGetQuery(connection, record_exists_query)
36 record_exists <- record_exists_result[1, 1] > 0
37
38 if(record_exists) {
39   cat("Record with primary key", primary_key_value, "already exists in the database.\n")
40 }
41 if (!record_exists) {
42   # Check if the primary key value of the new record is unique in the temporary dataframe
43   if (!primary_key_value %in% supplier_possible_data[[supplier_primary_key]]) {
44     supplier_possible_data <- rbind(supplier_possible_data, new_record)
45   }
46 }
47
48 cat("Finished processing file:", file_path, "\n")
49
50 }
51
52 }
53
54 cat("Starting validation for new records.\n")
55 supplier_possible_data <- validate_and_prepare_supplier_data(supplier_possible_data)
56 cat("Validation completed for new records.\n")
57
58 if (nrow(supplier_possible_data) > 0) {
59   cat("Starting to insert validated data into the database. Number of records: ", nrow(supplier_possible_data), "\n")
60   # Ingesting prepared data to our database
61   dbWriteTable(connection, name = "supplier", value = supplier_possible_data, append = TRUE, as.data.frame(supplier_possible_data))
62   cat("Data insertion completed successfully.\n")
63 } else
64 {
65   cat("No valid supplier data to insert into the database.\n")
66 }

```

#### 4. Validations for promotion data

```

1 validate_and_prepare_promotion_data <- function(data) {
2

```

```

3 # Validation for promotion ID
4 promotion_id_check <- grepl("^[A-Za-z0-9]{10}$", data$promotion_id)
5 data <- data[promotion_id_check, ]
6
7 #Checking for the validation of the promotion_start_date and promotion_end_date in the promotion table
8 date_check <- !is.na(as.Date(data$promotion_start_date, format = "%d-%m-%Y")) &
9 !is.na(as.Date(data$promotion_end_date, format = "%d-%m-%Y")) &
10 as.Date(data$promotion_start_date, format = "%d-%m-%Y") < as.Date(data$promotion_end_date, format = "%d-%m-%Y")
11
12 #Check for the validation of the promotion_start_date and promotion_end_date in the promotion table
13 #promotion_start_date and promotion_end_date should be in correct form for eg 12/11/2023
14 date_format <- "%d-%m-%Y"
15 date_check <- !is.na(as.Date(data$promotion_start_date, format = date_format)) &
16 !is.na(as.Date(data$promotion_end_date, format = date_format)) &
17 as.Date(data$promotion_start_date, format = date_format) < as.Date(data$promotion_end_date, format = date_format)
18 data <- data[date_check,]
19
20
21 #Check for the validation of the column promotion_discount_value in the promotion table.
22 #promotion_discount_value should be <1
23
24 discount_value_check <- !is.na(data$promotion_discount_value) &
25 is.numeric(data$promotion_discount_value) &
26 data$promotion_discount_value < 1
27 data <- data[discount_value_check,]
28 return(data)
29 }
30
31
32 # Fetch existing promotion IDs from the database
33
34 promotion_file_paths <- list.files(path = "data_upload", pattern = "promotion.*\\.csv$", full.names = TRUE)
35
36 # Define the primary key column for the promotion table
37 promotion_primary_key <- "promotion_id"
38
39 #Initialising empty data frame
40 promotion_possible_data <- data.frame()
41
42
43 # Read each promotion CSV file and check for the existence of the primary key in the database
44 for (file_path in promotion_file_paths) {

```

```

45
46 cat("Starting processing file:", file_path, "\n")
47
48 # Read the current file
49 promotion_data <- readr::read_csv(file_path)
50
51 # Iterate through each row of the file
52 for (i in seq_len(nrow(promotion_data))) {
53   new_record <- promotion_data[i, ]
54   primary_key_value <- new_record[[promotion_primary_key]]
55   conditions <- paste(promotion_primary_key, "=", paste0("'", primary_key_value, "'"))
56
57   # Check if a record with the same primary key exists in the database
58   record_exists_query <- paste("SELECT COUNT(*) FROM promotion WHERE", conditions)
59   record_exists_result <- dbGetQuery(connection, record_exists_query)
60   record_exists <- record_exists_result[1, 1] > 0
61
62   if(record_exists) {
63     cat("Record with primary key", primary_key_value, "already exists in the database.\n")
64   }
65   if (!record_exists) {
66     # Check if the primary key value of the new record is unique in the temporary dataframe
67     if (!primary_key_value %in% promotion_possible_data[[promotion_primary_key]]) {
68       promotion_possible_data <- rbind(promotion_possible_data, new_record)
69     }
70   }
71
72   cat("Finished processing file:", file_path, "\n")
73
74 }
75 }
76 cat("Starting validation for new records.\n")
77 promotion_possible_data <- validate_and_prepare_promotion_data(promotion_possible_data)
78 cat("Validation completed for new records.\n")
79
80
81 if (nrow(promotion_possible_data) > 0)
82 {
83   cat("Starting to insert validated data into the database. Number of records: ", nrow(promotion_possible_data), "\n")
84   # Digesting prepared data to our database
85   dbWriteTable(connection, name = "promotion", value = promotion_possible_data, append = TRUE, as.table = FALSE)
86   cat("Data insertion completed successfully.\n")

```

```

87 } else
88 {
89   cat("No valid promotion data to insert into the database.\n")
90 }

```

## 5. Validations for shipment data

```

1  validate_and_prepare_shipment_data <- function(data) {
2    # Validation for shipment ID
3    shipment_id_check <- grepl("^SHIP[0-9]{6}$", data$shipment_id)
4    data <- data[shipment_id_check,]
5
6    # Convert dates from character to Date object
7    data$shipment_date <- as.Date(data$shipment_date, format = "%d-%m-%Y")
8    data$delivery_date <- as.Date(data$delivery_date, format = "%d-%m-%Y")
9
10   # Validation for shipment_date and delivery_date format
11   date_format_check <- !is.na(data$shipment_date) & !is.na(data$delivery_date)
12
13   # Keep only rows with valid date formats
14   data <- data[date_format_check,]
15
16   # Validation for logical order of shipment and delivery dates
17   logical_date_order_check <- data$shipment_date <= data$delivery_date
18
19   # Keep only rows with logical date order
20   data <- data[logical_date_order_check,]
21
22   return(data)
23 }
24
25 # Fetch existing shipment IDs from the database
26
27 shipment_file_paths <- list.files(path = "data_upload", pattern = "shipment.*\\.csv$", full.
28
29 # Define the primary key column for the shipment table
30 shipment_primary_key <- "shipment_id"
31
32 #Initialising empty data frame
33 shipment_possible_data <- data.frame()
34
35 # Read each shipment CSV file and check for the existence of the primary key in the database

```

```

36 for (file_path in shipment_file_paths) {
37
38   cat("Starting processing file:", file_path, "\n")
39
40   # Read the current file
41   shipment_data <- readr::read_csv(file_path)
42
43   # Iterate through each row of the file
44   for (i in seq_len(nrow(shipment_data))) {
45     new_record <- shipment_data[i, ]
46     primary_key_value <- new_record[[shipment_primary_key]]
47     conditions <- paste(shipment_primary_key, "=", paste0("'", primary_key_value, "'"))
48
49     # Check if a record with the same primary key exists in the database
50     record_exists_query <- paste("SELECT COUNT(*) FROM shipment WHERE", conditions)
51     record_exists_result <- dbGetQuery(connection, record_exists_query)
52     record_exists <- record_exists_result[1, 1] > 0
53
54     if(record_exists) {
55       cat("Record with primary key", primary_key_value, "already exists in the database.\n")
56     }
57     if (!record_exists) {
58       # Check if the primary key value of the new record is unique in the temporary dataframe
59       if (!primary_key_value %in% shipment_possible_data[[shipment_primary_key]]) {
60         shipment_possible_data <- rbind(shipment_possible_data, new_record)
61       }
62     }
63
64     cat("Finished processing file:", file_path, "\n")
65
66   }
67 }
68 cat("Starting validation for new records.\n")
69 shipment_possible_data <- validate_and_prepare_shipment_data(shipment_possible_data)
70 cat("Validation completed for new records.\n")
71
72 if (nrow(shipment_possible_data) > 0) {
73   cat("Starting to insert validated data into the database. Number of records: ", nrow(shipment_possible_data), "\n")
74   # Ingesting prepared data to our database
75   dbWriteTable(connection, name = "shipment", value = shipment_possible_data, append = TRUE, as.data.frame = TRUE)
76   cat("Data insertion completed successfully.\n")
77 } else {

```



```

78   cat("No valid shipment data to insert into the database.\n")
79 }

```

## 6. Validations for product data

```

1  validate_and_prepare_product_data <- function(data) {
2
3  # Validation for product ID
4  product_id_check <- grepl("[A-Za-z0-9]{10}$", data$product_id)
5  data <- data[product_id_check, ]
6
7  # Performing validation checks here
8  data <- data[data$review_score >= 1 & data$review_score <= 5, ]
9
10 return(data)
11 }
12
13
14 # Fetch existing product IDs from the database
15
16 product_file_paths <- list.files(path = "data_upload", pattern = "product.*\\.csv$", full.names = TRUE)
17
18 # Define the primary key column for the product table
19 product_primary_key <- "product_id"
20
21 #Initialising empty data frame
22 product_possible_data <- data.frame()
23
24
25 # Read each product CSV file and check for the existence of the primary key in the database
26 for (file_path in product_file_paths) {
27
28   cat("Starting processing file:", file_path, "\n")
29
30   # Read the current file
31   product_data <- readr::read_csv(file_path)
32
33   # Iterate through each row of the file
34   for (i in seq_len(nrow(product_data))) {
35     new_record <- product_data[i, ]
36     primary_key_value <- new_record[[product_primary_key]]
37     conditions <- paste(product_primary_key, "=", paste0("'", primary_key_value, "'"))

```

```

38
39 # Check if a record with the same primary key exists in the database
40 record_exists_query <- paste("SELECT COUNT(*) FROM product WHERE", conditions)
41 record_exists_result <- dbGetQuery(connection, record_exists_query)
42 record_exists <- record_exists_result[1, 1] > 0
43
44 if(record_exists) {
45   cat("Record with primary key", primary_key_value, "already exists in the database.\n")
46 }
47 if (!record_exists) {
48   # Check if the primary key value of the new record is unique in the temporary dataframe
49   if (!primary_key_value %in% product_possible_data[[product_primary_key]]) {
50     product_possible_data <- rbind(product_possible_data, new_record)
51   }
52 }
53
54 cat("Finished processing file:", file_path, "\n")
55
56 }
57 }
58 cat("Starting validation for new records.\n")
59 product_possible_data <- validate_and_prepare_product_data(product_possible_data)
60 cat("Validation completed for new records.\n")
61
62 if (nrow(product_possible_data) > 0)
63 {
64   cat("Starting to insert validated data into the database. Number of records: ", nrow(product.
65   # Digesting prepared data to our database
66   dbWriteTable(connection, name = "product", value = product_possible_data, append = TRUE, row
67   cat("Data insertion completed successfully.\n")
68 } else
69 {
70   cat("No valid product data to insert into the database.\n")
71 }

```

## 7. Validation for orders data

```

1 validate_and_prepare_orders_data <- function(data){
2   # Checking format of order id
3   order_id_check <- grepl("^ORDER[0-9]{9}$", data$order_id)
4   data <- data[order_id_check, ]
5

```

```

6 # Checking format of customer id
7 customer_id_check <- grepl("^CUST[0-9]{6}$", data$customer_id)
8 data <- data[customer_id_check, ]
9
10 # Checking format of product id
11 product_id_check <- grepl("[A-Za-z0-9]{10}$", data$product_id)
12 data <- data[product_id_check, ]
13
14 # Checking format of shipment id
15 shipment_id_check <- grepl("^SHIP[0-9]{6}$", data$shipment_id)
16 data <- data[shipment_id_check,]
17
18 # Validation for order_date format
19 order_date_format_check <- !is.na(as.Date(data$order_date, format = "%d-%m-%Y"))
20 data <- data[order_date_format_check,]
21
22
23 return(data)
24 }
25
26
27 orders_file_paths <- list.files(path = "data_upload", pattern = "orders.*\\.csv$", full.names = TRUE)
28 orders_possible_data <- data.frame()
29
30
31 # Read each orders CSV file and check for the existence of the composite primary key in the database
32 for (file_path in orders_file_paths) {
33   orders_data <- readr::read_csv(file_path)
34
35   # Iterate through each row of the file
36   for (i in seq_len(nrow(orders_data))) {
37     new_record <- orders_data[i, ]
38     # primary_key_value <- new_record[[orders_primary_key]]
39     # Construct the condition to check the composite primary key (order_id, product_id, customer_id, shipment_id)
40     conditions <- sprintf("order_id = '%s' AND product_id = '%s' AND customer_id = '%s' AND shipment_id = '%s'",
41                           new_record$order_id, new_record$product_id, new_record$customer_id, new_record$shipment_id)
42
43     # Check if a record with the same composite primary key exists in the database
44     record_exists_query <- paste("SELECT COUNT(*) FROM orders WHERE", conditions)
45     record_exists_result <- dbGetQuery(connection, record_exists_query)
46     record_exists <- record_exists_result[1, 1] > 0
47

```

```

48
49 if(!record_exists) {
50   # Construct a unique identifier for the composite primary key
51   composite_key <- paste(new_record$order_id, new_record$product_id, new_record$customer_id, n
52
53   # Check if the composite primary key is unique in the temporary dataframe
54   existing_keys <- sapply(1:nrow(orders_possible_data), function(i) {
55     paste(orders_possible_data[i, "order_id"], orders_possible_data[i, "product_id"], orders_p
56   })
57
58   if (!composite_key %in% existing_keys) {
59     orders_possible_data <- rbind(orders_possible_data, new_record)
60   } else {
61     cat("Record with composite primary key already exists in temporary data.\n")
62   }
63   } else {
64     cat("Record with composite primary key already exists in the database.\n")
65   }
66 }
67 }
68 orders_possible_data <- validate_and_prepare_orders_data(orders_possible_data)
69
70 if (nrow(orders_possible_data) > 0) {
71   cat("Starting to insert validated data into the database. Number of records: ", nrow(orders
72
73   # Ingesting prepared data to our database
74
75   dbWriteTable(connection, name = "orders", value = orders_possible_data, append = TRUE, row
76   cat("Data insertion completed successfully.\n")
77 } else {
78   cat("No valid orders data to insert into the database.\n")
79 }

```

## Referential Integrity for ensuring data integrity

The foreign key is the crucial component of a database that enforces referential integrity, ensuring that a value appearing in one relation for a specified set of attributes also exists in another relation for a corresponding set of the attributes. In all, referential integrity ensures that a value referenced in one table exists in another table, maintaining the integrity and consistency of the data.

In the 'category' table, 'category\_id' serves as the primary key, that uniquely identifies each

category of the product. On the other hand, in products table ‘category\_id’ serves as the foreign key. It means if one of the category ids is removed from the category table then the it should be removed from the product table as well.

## Part 3 Data Pipeline Generation

### 3.1 GitHub Repository and Workflow Setup

In this part, a GitHub repository named “DM\_Group\_18” was created and connected it to the Posit cloud to manage and version control our project. It also acted as the central hub of our project which helped the team members to collaborate effectively. In the repository the CSV files were uploaded in the folder named “data\_upload”. Also, the R scripts for database schema creation, validation and analysis could be found in the folder named “R”.

The URL of the repository - [https://github.com/AkarshaShrivastava19/DM\\_group\\_18](https://github.com/AkarshaShrivastava19/DM_group_18)

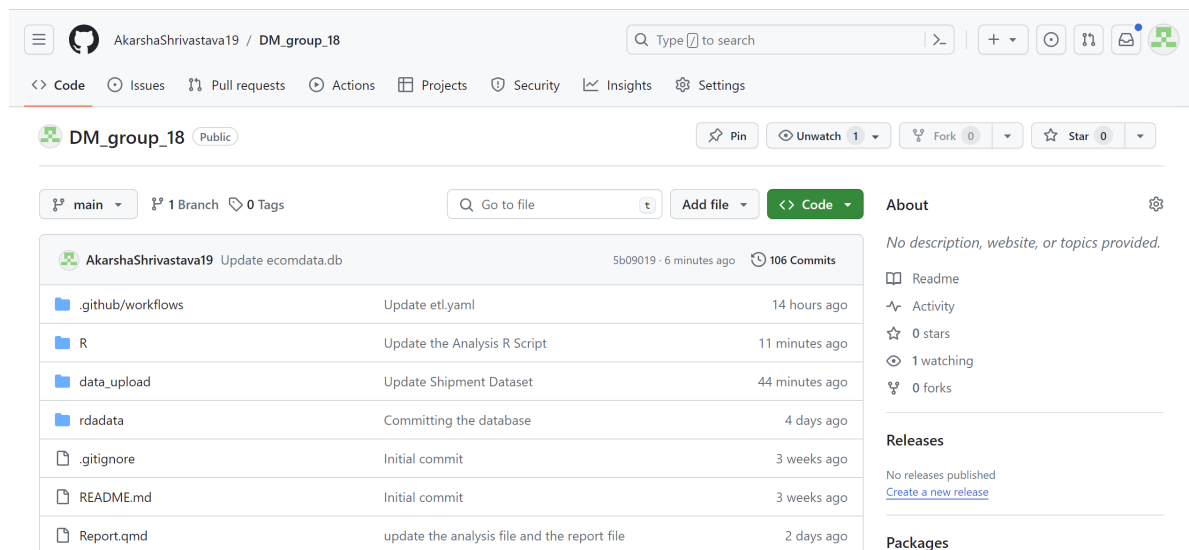


Figure 12: Created GitHub Repository

### 3.2 GitHub Actions for Continuous Integration

Here, we implemented GitHub actions to automate different stages of our data pipeline such as database updates, data validation and data analysis. The workflow was configured to trigger each time we push new changes to the main branch from Posit cloud ensuring that the automated tasks were executed in response to the relevant changes.

This workflow runs seamlessly on the latest Ubuntu Environment and has multiple jobs such as setting up the R environment, installing packages, running the R scripts and updating the database with the latest data. So, after the detection of any changes the workflow activates in response to any “pull” and “push” request. We configured to automatically update the database, run data validations and analysis.

By implementing this, we significantly reduced manual intervention required for this process thus increasing the efficiency and reducing human errors. It also helped us in rapid detection and resolution of the issues.

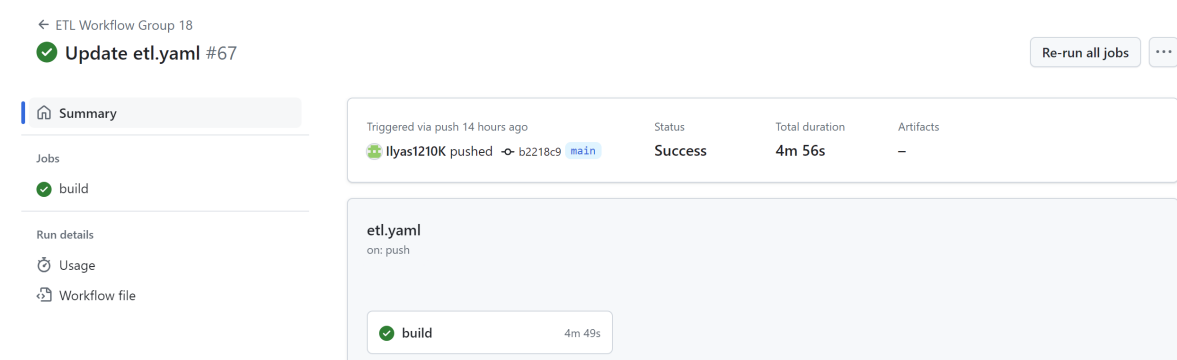


Figure 13: Successfully Build Workflow

## Part 4 Data Analysis and Reporting

The following data analysis was performed on the generated e-commerce data.

### 4.1 Advanced Data Analysis in R

```

1 # Retrieve data from the database
2 customer <- dbGetQuery(connection, "SELECT * FROM customer")
3 product <- dbGetQuery(connection, "SELECT * FROM product")
4 supplier <- dbGetQuery(connection, "SELECT * FROM supplier")
5 category <- dbGetQuery(connection, "SELECT * FROM category")
6 shipment <- dbGetQuery(connection, "SELECT * FROM shipment")
7 promotion <- dbGetQuery(connection, "SELECT * FROM promotion")
8 orders <- dbGetQuery(connection, "SELECT * FROM orders")

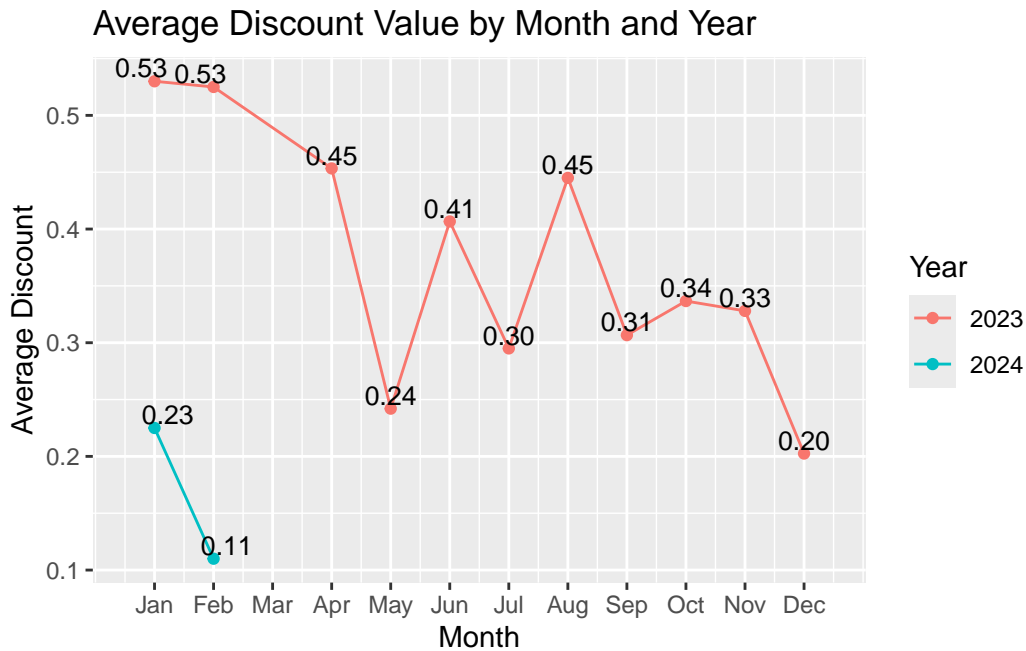
```

### 4.1.1 Promotion Discount Trend

```
1 # Convert Date Format
2 promotion <- promotion %>%
3   mutate(promotion_start_date = dmy(promotion_start_date),
4           promotion_end_date = dmy(promotion_end_date))
5
6 # Generate records for each month that each promotion spans
7 data_expanded <- promotion %>%
8   rowwise() %>%
9   mutate(months = list(seq(from = promotion_start_date,
10                           to = promotion_end_date,
11                           by = "month")))) %>%
12   unnest(months) %>%
13   mutate(year = year(months), month = month(months)) %>%
14   group_by(promotion_id, year, month) %>%
15   summarise(promotion_discount_value = mean(promotion_discount_value), .groups = 'drop')
16
17 # Calculate the average discount value for each month
18 average_discounts <- data_expanded %>%
19   group_by(year, month) %>%
20   summarise(average_discount = mean(promotion_discount_value))
```

`summarise()` has grouped output by 'year'. You can override using the  
`.groups` argument.

```
1 # Specify the dimensions of the plot
2 width <- 12
3 height <- 8
4
5 # Visualise the average discount value for different months and years
6 g_promotionvalue <- ggplot(average_discounts, aes(x = month, y = average_discount, group = year)) +
7   geom_line() +
8   geom_point() +
9   scale_x_continuous(breaks = 1:12, labels = month.abb) +
10   geom_text(aes(label = sprintf("%.2f", average_discount)), position = position_dodge(width = 1)) +
11   labs(title = "Average Discount Value by Month and Year",
12        x = "Month",
13        y = "Average Discount",
14        color = "Year")
15 print(g_promotionvalue)
```



```

1 # Dynamically generate filename with current date and time
2 filename <- paste0("promotion_discount_trend_", format(Sys.time(), "%Y%m%d_%H%M%S"), ".png")
3
4 # Save the plot with the dynamic filename
5 ggsave(filename, plot = g_promotionvalue, width = width, height = height)

```

#### 4.1.2 Promotion Count Trend

```

1 # Calculate the number of times a promotion appears in each month
2 promotion_counts <- data_expanded %>%
3   count(year, month)
4
5 # Visualise the number of times promotions appear in different years and months
6 g_promotioncount <- ggplot(promotion_counts, aes(x = month, y = n, fill = as.factor(year)))
7   geom_bar(stat = "identity", position = "dodge") +
8   scale_x_continuous(breaks = 1:12, labels = month.abb) +
9   geom_text(aes(label = n), position = position_dodge(width = 0.9), vjust = -0.2, size = 3.5)
10  theme(axis.title = element_text(size = 12)) +
11  labs(title = "Number of Promotions by Month and Year",
12       x = "Month",
13       y = "Number of Promotions",

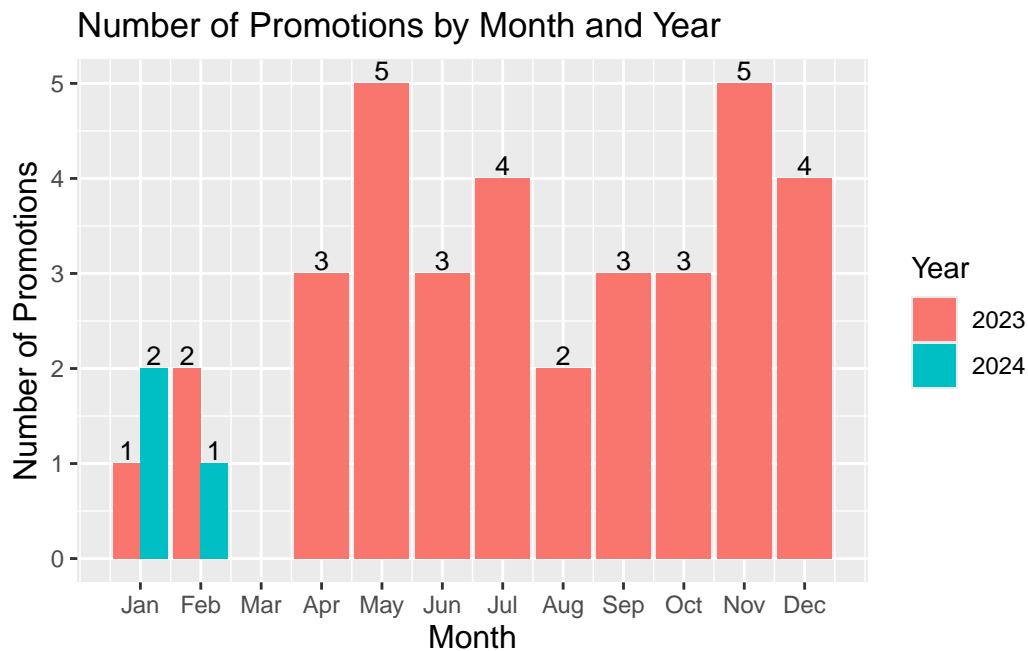
```



```

14     fill = "Year")
15 print(g_promotioncount)

```



```

1 # Dynamically generate filename with current date and time
2 filename <- paste0("promotion_number_trend_", format(Sys.time(), "%Y%m%d_%H%M%S"), ".png")
3
4 # Save the plot with the dynamic filename
5 ggsave(filename, plot = g_promotioncount, width = width, height = height)

```

## Promotion Analysis

The above graphs illustrate the highest discounts that were offered during January and February, i.e. 53% off on the product price. However, more promotional events were offered in other months to attract more customers to shop through our website.

### 4.1.3 Monthly Revenue Trend

```

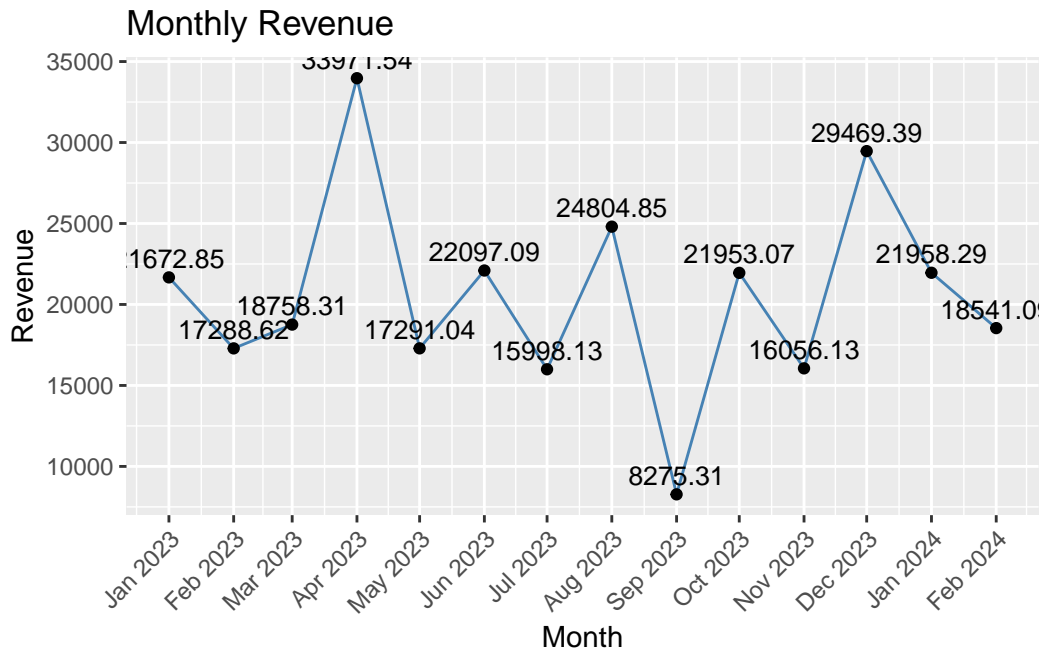
1 # Preprocessed date formats
2 orders <- orders %>% mutate(order_date = dmy(order_date))
3 promotion <- promotion %>%
4   mutate(start_date = promotion_start_date,

```

```

5     end_date = promotion_end_date,
6     promotion_discount_value = if_else(is.na(promotion_discount_value), 0, promotion_di
7
8 # Merge orders with products for pricing information
9 order_products <- orders %>%
10   left_join(product, by = "product_id")
11
12 # Make sure there are no missing prices or quantities
13 order_products <- order_products %>%
14   mutate(price = if_else(is.na(price), 0, price),
15          quantity = if_else(is.na(quantity), 0, quantity))
16
17 # Combine orders, products and promotions to take into account discounts during promotions
18 order_products_promotions <- order_products %>%
19   left_join(promotion, by = "promotion_id") %>%
20   mutate(is_promotion = if_else(order_date >= start_date & order_date <= end_date, TRUE, FALSE),
21          revenue = price * quantity * if_else(is_promotion, 1 - promotion_discount_value, 1))
22
23 # Remove any missing income values generated in the calculation
24 order_products_promotions <- order_products_promotions %>%
25   filter(!is.na(revenue))
26
27 # Calculation of gross monthly income
28 monthly_revenue <- order_products_promotions %>%
29   mutate(month = floor_date(order_date, "month")) %>%
30   group_by(month) %>%
31   summarize(total_revenue = sum(revenue, na.rm = TRUE))
32
33 # Visualisation of monthly income
34 (g_monthlyrevenue <- ggplot(monthly_revenue, aes(x = month, y = total_revenue)) +
35   geom_line(color = "steelblue") +
36   geom_point() +
37   scale_x_date(date_labels = "%b %Y", date_breaks = "1 month") +
38   geom_text(aes(label = sprintf("%.2f", total_revenue)), position = position_dodge(width = 0
39   theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
40   labs(title = "Monthly Revenue", x = "Month", y = "Revenue"))

```



```

1 # Dynamically generate filename with current date and time
2 filename <- paste0("monthly_revenue_",
3                   format(Sys.time(), "%Y%m%d_%H%M%S"), ".png")
4
5 # Save the plot with the dynamic filename
6 ggsave(filename, plot = g_monthlyrevenue, width = width, height = height)

```

### Monthly Revenue Analysis

By analysing the revenue in 2023 and 2024, it is observed that the month of April in 2023 records the highest revenue, credited to the promotion event with a high discount rate, followed by revenue in December 2023. Although no attractive discount offers were applied in the month of December 2023, still an outstanding revenue was observed because of the festive season in the United Kingdom.

#### 4.1.4 Monthly Best-Selling Products

```

1 # Calculate total monthly revenue per product
2 monthly_product_revenue <- order_products_promotions %>%
3   mutate(month = floor_date(order_date, "month")) %>%
4   group_by(month, product_id) %>%
5   summarize(total_revenue_product = sum(revenue, na.rm = TRUE))

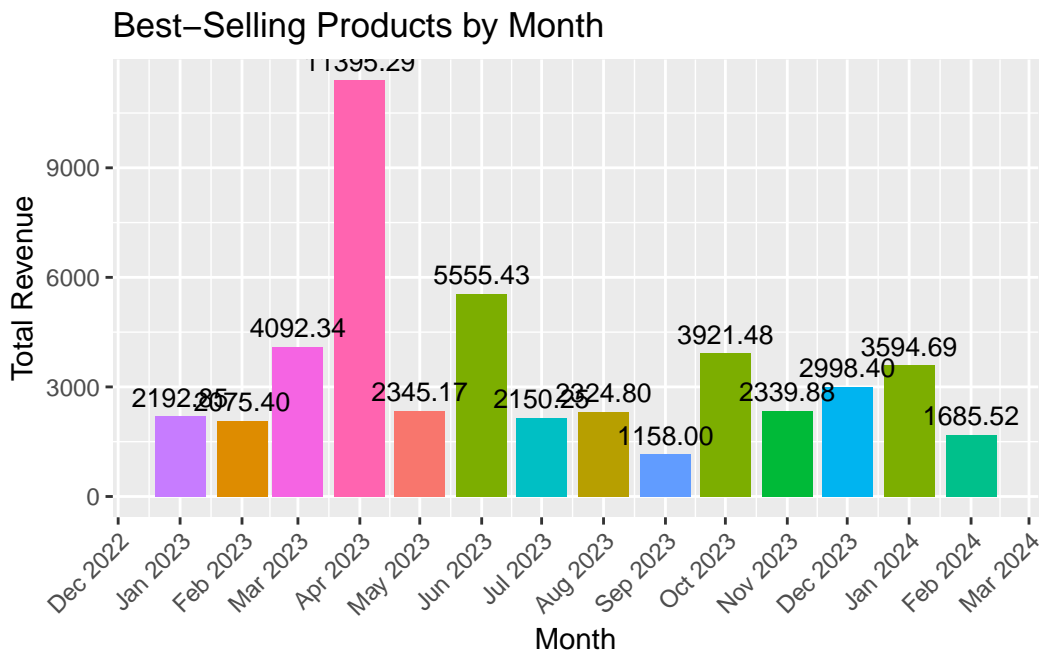
```

`summarise()` has grouped output by 'month'. You can override using the `.groups` argument.

```

1 # Select top earning products per month
2 best_selling_products_each_month <- monthly_product_revenue %>%
3   group_by(month) %>%
4   slice_max(total_revenue_product, n = 1) %>%
5   ungroup() %>%
6   select(month, product_id, total_revenue_product)
7
8 best_selling_products_each_month <- merge(best_selling_products_each_month, product[, c("product_id", "product_name")])
9
10 # Visualise the top earning products and their revenues per month
11 g_bestseller_product_monthly <- ggplot(best_selling_products_each_month, aes(x = month, y = total_revenue_product)) +
12   geom_col(show.legend = FALSE) +
13   geom_text(aes(label = sprintf("%.2f", total_revenue_product)), position = position_dodge(width = 0.5)) +
14   theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
15   scale_x_date(date_labels = "%b %Y", date_breaks = "1 month") +
16   theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
17   labs(title = "Best-Selling Products by Month", x = "Month", y = "Total Revenue")
18
19 print(g_bestseller_product_monthly)

```



```

1 # Dynamically generate filename with current date and time
2 filename <- paste0("monthly_bestseller_product_",
3                   format(Sys.time(), "%Y%m%d_%H%M%S"), ".png")
4
5 # Save the plot with the dynamic filename
6 ggsave(filename, plot = g_bestseller_product_monthly, width = width, height = height)

1 best_selling_products_each_month$month <- as.Date(best_selling_products_each_month$month, "%Y-%m-%d")
2
3 best_selling_products_each_month$YearMonth <- format(best_selling_products_each_month$month, "%Y-%m")
4
5 best_selling_products_each_month <- best_selling_products_each_month %>%
6   arrange(YearMonth)
7
8 table_to_display <- best_selling_products_each_month %>%
9   select(YearMonth, product_name, total_revenue_product) %>%
10  rename('Total Revenue' = total_revenue_product)
11
12 # Display the table with kable
13 kable(table_to_display, caption = "Monthly Best-Selling Products", col.names = c("Time", "Product Name", "Total Revenue"))

```

Table 2: Monthly Best-Selling Products

Time	Product Name	Total Revenue
2023-01	Wall Paint Behr Premium Plus Ultra	2192.85
2023-02	Garden Tool Corona Bypass Pruner	2075.40
2023-03	Wall Paint Kilz Complete Coat	4092.34
2023-04	Wall Paint Olympic One	11395.29
2023-05	Facial Cleanser Cetaphil Gentle Skin Cleanser	2345.17
2023-06	Laptop Acer Swift 5	5555.43
2023-07	Pasta Barilla Linguine	2150.25
2023-08	Guitar Epiphone SG Standard	2324.80
2023-09	Stroller Baby Jogger City Mini GT2	1158.00
2023-10	Laptop Acer Swift 5	3921.48
2023-11	Office Chair Boss Office Products Executive	2339.88
2023-12	Refrigerator Samsung RS27T5561SR	2998.40
2024-01	Laptop Acer Swift 5	3594.69
2024-02	Office Chair Serta Big and Tall	1685.52

## Monthly Best-Selling Products Analysis

It is observed that Wall Paint Olympic One has become the top best-selling product in 2023 April which creates an extremely high revenue for the e-commerce company.

#### 4.1.5 Monthly Shipping Efficiency

```

1  ## Monthly Shipping Efficiency
2  # Convert dates to Date objects
3  shipment$shipment_date <- as.Date(shipment$shipment_date, origin = "1970-01-01")
4
5  shipment_unique <- shipment %>% distinct(shipment_id, .keep_all = TRUE)
6
7  # Merge orders and shipment data on order_id
8  combined_data <- merge(orders, shipment, by = "shipment_id")
9
10 # Calculate shipping duration in days
11 combined_data$shipping_duration <- as.numeric(difftime(combined_data$shipment_date, combined.
12
13 # Calculate monthly statistics
14 monthly_stats <- combined_data %>%
15   mutate(month = floor_date(order_date, "month")) %>%
16   group_by(month) %>%
17   summarise(Average_Shipping_Duration = round(mean(shipping_duration),2),
18             Min_Shipping_Duration = min(shipping_duration),
19             Max_Shipping_Duration = max(shipping_duration))
20
21 table_to_display <- monthly_stats %>%
22   mutate(Time = format(month, "%Y-%m")) %>%
23   select(Time, Average_Shipping_Duration, Min_Shipping_Duration, Max_Shipping_Duration)
24
25 # Display the table with kable
26 kable(table_to_display, caption = "Monthly Shipping Duration", col.names = c("Time", "Average

```

Table 3: Monthly Shipping Duration

Time	Average Duration	Min Duration	Max Duration
2023-01	0.56	0	1
2023-02	0.48	0	1
2023-03	0.48	0	1
2023-04	0.77	0	1
2023-05	0.39	0	1

Time	Average Duration	Min Duration	Max Duration
2023-06	0.48	0	1
2023-07	0.30	0	1
2023-08	0.39	0	1
2023-09	0.65	0	1
2023-10	0.44	0	1
2023-11	0.43	0	1
2023-12	0.62	0	1
2024-01	0.59	0	1
2024-02	0.51	0	1

```

1 # Visualize the statistics
2 g_shipping_efficiency <- ggplot(monthly_stats, aes(x = month)) +
3   geom_line(aes(y = Average_Shipping_Duration), color = "steelblue", size = 1) +
4   geom_text(aes(y = Average_Shipping_Duration,
5                 label = sprintf("%.2f", Average_Shipping_Duration)),
6             position = position_dodge(width = 0.9), vjust = -0.5, size = 3.5) +
7   scale_x_date(date_labels = "%b %Y",
8               date_breaks = "1 month",
9               limits = c(min(monthly_stats$month), max(monthly_stats$month))) +
10  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
11  labs(title = "Monthly Shipping Duration",
12       x = "Month", y = "Shipping Duration (days)")

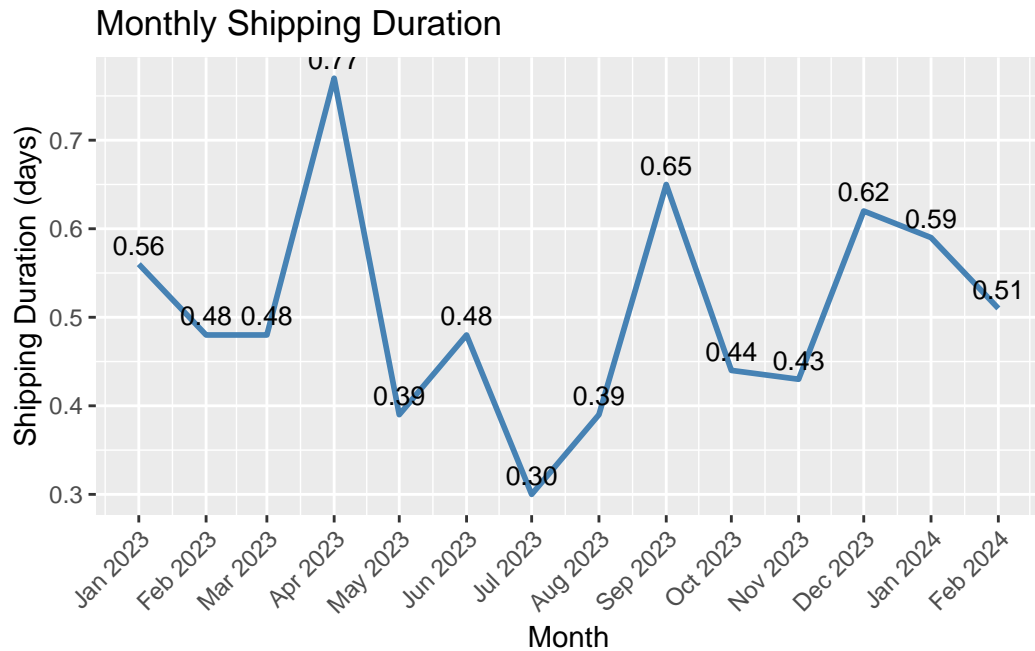
```

Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.  
i Please use `linewidth` instead.

```

1 print(g_shipping_efficiency)

```



```

1 # Dynamically generate filename with current date and time
2 filename <- paste0("monthly_shipping_efficiency_",
3                   format(Sys.time(), "%Y%m%d_%H%M%S"), ".png")
4
5 # Save the plot with the dynamic filename
6 ggsave(filename, plot = g_shipping_efficiency, width = width, height = height)

```

#### 4.1.6 Monthly Delivery Efficiency

```

1 ## Monthly Delivery Efficiency
2 # Convert dates to Date objects
3 shipment$delivery_date <- as.Date(shipment$delivery_date, origin = "1970-01-01")
4
5 shipment_unique <- shipment %>% distinct(shipment_id, .keep_all = TRUE)
6
7 # Merge orders and shipment data on order_id
8 combined_data <- merge(orders, shipment, by = "shipment_id")
9
10 # Calculate delivery duration in days
11 combined_data$delivery_duration <- as.numeric(difftime(combined_data$delivery_date, combined.
12
13 # Calculate monthly statistics

```



```

14 monthly_stats <- combined_data %>%
15   mutate(month = floor_date(order_date, "month")) %>%
16   group_by(month) %>%
17   summarise(Average_Delivery_Duration = round(mean(delivery_duration),2),
18             Min_Delivery_Duration = min(delivery_duration),
19             Max_Delivery_Duration = max(delivery_duration))
20
21 table_to_display <- monthly_stats %>%
22   mutate(Time = format(month, "%Y-%m")) %>%
23   select(Time, Average_Delivery_Duration, Min_Delivery_Duration, Max_Delivery_Duration)
24
25 # Calculate the overall average delivery duration
26 overall_avg_delivery <- mean(combined_data$delivery_duration, na.rm = TRUE)
27
28 # Display the table with kable
29 kable(table_to_display, caption = "Monthly Delivery Duration", col.names = c("Time", "Average

```

Table 4: Monthly Delivery Duration

Time	Average Duration	Min Duration	Max Duration
2023-01	2.93	1	5
2023-02	3.46	1	5
2023-03	2.94	1	5
2023-04	3.13	1	5
2023-05	2.54	1	5
2023-06	3.58	1	5
2023-07	3.00	1	5
2023-08	3.11	1	5
2023-09	3.15	1	5
2023-10	3.08	1	5
2023-11	2.70	1	5
2023-12	2.89	1	5
2024-01	2.68	1	5
2024-02	2.81	1	5

```

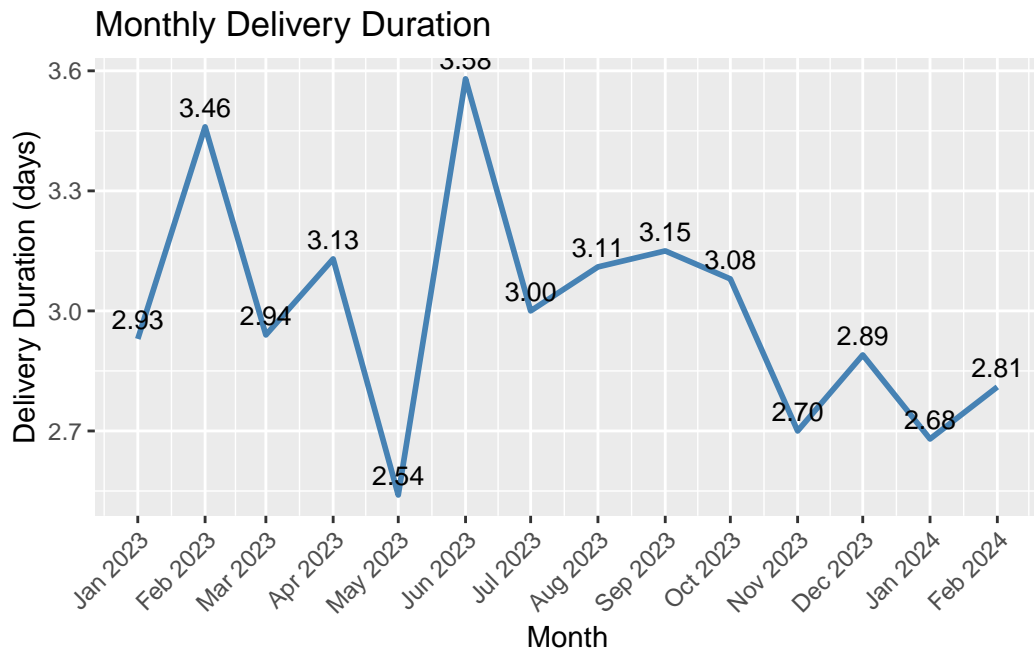
1 # Visualize the statistics
2 g_delivery_efficiency <- ggplot(monthly_stats, aes(x = month)) +
3   geom_line(aes(y = Average_Delivery_Duration), color = "steelblue", size = 1) +
4   geom_text(aes(y = Average_Delivery_Duration,
5                 label = sprintf("%.2f", Average_Delivery_Duration)),

```

```

6         position = position_dodge(width = 0.9), vjust = -0.5, size = 3.5) +
7     scale_x_date(date_labels = "%b %Y",
8                 date_breaks = "1 month",
9                 limits = c(min(monthly_stats$month), max(monthly_stats$month))) +
10    theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
11    labs(title = "Monthly Delivery Duration",
12         x = "Month", y = "Delivery Duration (days)")
13 print(g_delivery_efficiency)

```



```

1 # Dynamically generate filename with current date and time
2 filename <- paste0("monthly_delivery_efficiency_",
3                   format(Sys.time(), "%Y%m%d_%H%M%S"), ".png")
4
5 # Save the plot with the dynamic filename
6 ggsave(filename, plot = g_delivery_efficiency, width = width, height = height)

```

## Shipping and Delivery Efficiency Analysis

From the above analysis it is observed that the average duration ranges between 0.30 and 0.77 days for shipment efficiency and ranges between 2.54 and 3.58 days for delivery efficiency. Both indicates the efficient capability of processing parcels for the customers.

## 4.2 Comprehensive Reporting with Quarto

### 4.2.1 Demographic Distribution of Customers

#### A. The Distribution of Gender across Customers

```
1 SELECT
2     gender,
3     COUNT(*) AS GenderCount,
4     CONCAT(ROUND((COUNT(*) * 100.0) / (SELECT COUNT(*) FROM customer), 2), '%') AS Percentage
5 FROM
6     customer
7 GROUP BY
8     gender
9 ORDER BY
10    Percentage DESC;
```

Table 5: 2 records

gender	GenderCount	Percentage
Male	110	55.28%
Female	89	44.72%

#### B. The Distribution of Age across Customers

```
1 SELECT
2     AgeGroup,
3     COUNT(*) AS Count,
4     CONCAT(ROUND((COUNT(*) * 100.0) / (SELECT COUNT(*) FROM customer), 2), '%') AS Percentage
5 FROM (
6     SELECT
7         customer_id,
8         CASE
9             WHEN age >= 0 AND age < 18 THEN '0-18'
10            WHEN age >= 18 AND age < 30 THEN '19-30'
11            WHEN age >= 30 AND age < 40 THEN '31-40'
12            WHEN age >= 40 AND age < 50 THEN '41-50'
13            WHEN age >= 50 AND age < 60 THEN '51-60'
14            WHEN age >= 60 AND age < 70 THEN '61-70'
15            WHEN age >= 70 THEN '71+'

```

```

16         ELSE 'Unknown'
17     END AS AgeGroup
18 FROM customer
19 ) AS AgeCategories
20 GROUP BY AgeGroup
21 ORDER BY Count DESC;

```

Table 6: 6 records

AgeGroup	Count	Percentage
31-40	68	34.17%
41-50	58	29.15%
19-30	54	27.14%
51-60	16	8.04%
0-18	2	1.01%
61-70	1	0.5%

### C. The Distribution of Careers across Customers (Top 10)

```

1 SELECT
2     career,
3     COUNT(*) AS CareerCount,
4     CONCAT(ROUND((COUNT(*) * 100.0) / (SELECT COUNT(*) FROM customer), 2), '%') AS Percentage
5 FROM
6     customer
7 GROUP BY
8     career
9 ORDER BY
10     CareerCount DESC
11 LIMIT 10;

```

Table 7: Displaying records 1 - 10

career	CareerCount	Percentage
Senior Nurse	17	8.54%
Accountant	17	8.54%
Sales Director	16	8.04%
Senior Data Analyst	15	7.54%
Product Manager	13	6.53%
Graphic Designer	12	6.03%

career	CareerCount	Percentage
Senior Teacher	11	5.53%
Software Engineer	10	5.03%
HR Manager	10	5.03%
Sales Manager	9	4.52%

#### D. The Distribution of Geographic Location across Customers (Top 10)

```

1 SELECT
2     address_city,
3     COUNT(*) AS CityCount,
4     CONCAT(ROUND((COUNT(*) * 100.0) / (SELECT COUNT(*) FROM customer), 2), '%') AS Percentage
5 FROM
6     customer
7 GROUP BY
8     address_city
9 ORDER BY
10    CityCount DESC
11 LIMIT 10;

```

Table 8: Displaying records 1 - 10

address_city	CityCount	Percentage
Oxford	14	7.04%
Newcastle	14	7.04%
Birmingham	14	7.04%
Sheffield	13	6.53%
Liverpool	13	6.53%
Nottingham	12	6.03%
Leeds	12	6.03%
Derby	11	5.53%
London	10	5.03%
Brighton	10	5.03%

```

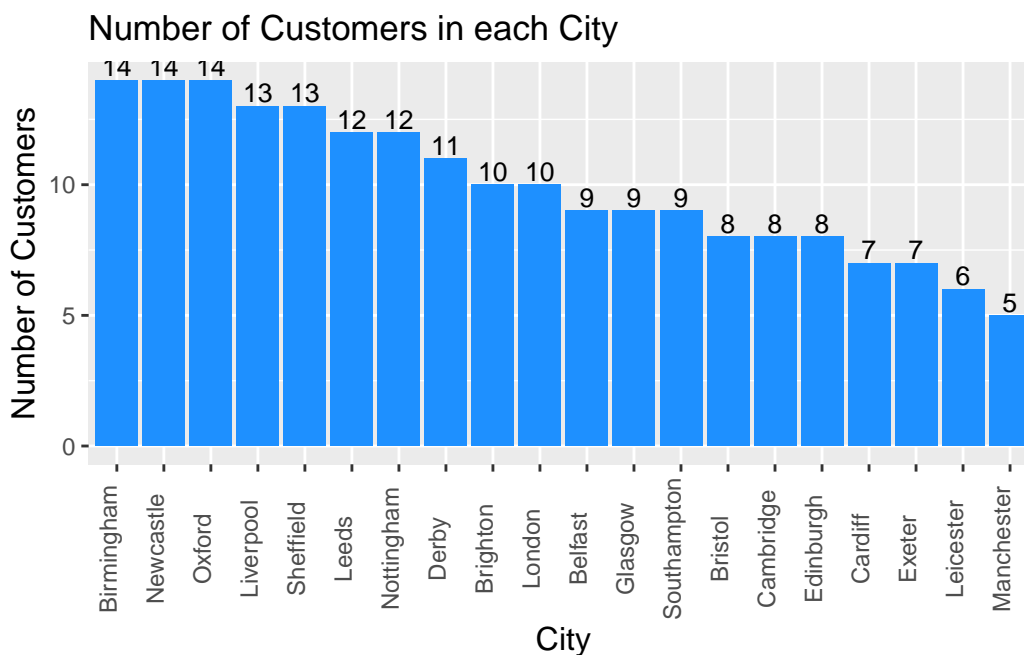
1 # Group by city and count the number of customer in each city
2 customer_city_count <- customer %>%
3   group_by(address_city) %>%
4   summarise(number_of_customers = n()) %>%
5   arrange(desc(number_of_customers))

```

```

6
7 # Specify the dimensions of the plot
8 width <- 12
9 height <- 8
10
11 # Use ggplot to create a bar chart showing the number of customers in each city
12 g_customer <- ggplot(customer_city_count,
13                       aes(x = reorder(address_city, -number_of_customers),
14                           y = number_of_customers)) +
15   geom_col(fill = "dodgerblue") +
16   geom_text(aes(label = number_of_customers),
17             position = position_dodge(width = 0.9), vjust = -0.2,
18             size = 3.5) +
19   theme(axis.text.x = element_text(angle = 90, hjust = 0.5, vjust = 0),
20         axis.title = element_text(size = 12)) +
21   labs(title = "Number of Customers in each City",
22        x = "City",
23        y = "Number of Customers")
24 print(g_customer)

```



```

1 # Dynamically generate filename with current date and time
2 filename <- paste0("geographical distribution of customers_",
3                   format(Sys.time(), "%Y%m%d_%H%M%S"), ".png")

```

```

4
5 # Save the plot with the dynamic filename
6 ggsave(filename, plot = g_customer, width = width, height = height)

```

## E. The Current Customer Referral Rate

```

1 SELECT
2     COUNT(CASE WHEN referred_by != '' AND referred_by IS NOT NULL THEN 1 END) AS Customer_wi
3     COUNT(*) AS total_customer,
4     CONCAT(ROUND((COUNT(CASE WHEN referred_by != '' AND referred_by IS NOT NULL THEN 1 END) :
5 FROM
6     customer;

```

Table 9: 1 records

Customer_with_Referral	total_customer	referral_rate
74	199	37.19%

## Customer Analysis

The gender of the customers is quite evenly distributed, with 55% males and 45% females and their age is mainly located in the senior group, who are in the age group of 31-40 years and 41-50 years, accounting for 34% and 29%, of males and females respectively. Young adults aged between 19-30 also comprise approximately 24% of the customers.

The career distribution, states that customers are employed in diverse industries and have different job positions, but most of them have high number of working experience according to their job titles.

The customers currently live in big cities around the United Kingdom, represents people living in big cities who shop online more frequently.

Reviewing the customer referral rate, it is observed that 37% of customers are referred by existing customers, showing a moderate satisfaction level from our customers.

### 4.2.2 Product Portfolio

#### A. The Distribution of Product Review Scores (Top 10)

```

1 SELECT
2     product_name, review_score
3 FROM
4     product
5 ORDER BY
6     review_score DESC
7 LIMIT 10;

```

Table 10: Displaying records 1 - 10

product_name	review_score
Office Chair Herman Miller Aeron	4.98
Craft Kit Faber-Castell Young Artist Essentials Set	4.96
Bed Frame Wayfair Zinus Upholstered Platform Bed	4.94
Smartphone Samsung Galaxy S21 Ultra	4.90
Laptop HP Spectre x360	4.89
Book Penguin Classics Pride and Prejudice	4.89
Lipstick NARS Orgasm	4.87
Drill Milwaukee Brushless Cordless Drill	4.84
Sofa IKEA Kivik	4.82
Lipstick Revlon Super Lustrous Lipstick	4.82

## B. The Number of Products supplied by Different Suppliers

```

1 # Perform an inner join to combine 'product' with 'supplier' on 'supplier_id'
2 joint_supplier_product <- inner_join(product, supplier, by = "supplier_id")
3
4 # Group by supplier_name and count the number of products for each supplier
5 product_count_by_supplier <- joint_supplier_product %>%
6     group_by(supplier_name) %>%
7     summarise(number_of_products = n())
8
9 # Specify the dimensions of the plot
10 width <- 12
11 height <- 8
12
13 # Use ggplot to create a bar chart showing the number of products for each supplier
14 g_supplier <- ggplot(product_count_by_supplier, aes(x = reorder(supplier_name, -number_of_products)))
15     geom_col(fill = "steelblue") +
16     geom_text(aes(label = number_of_products), position = position_dodge(width = 0.9), vjust =

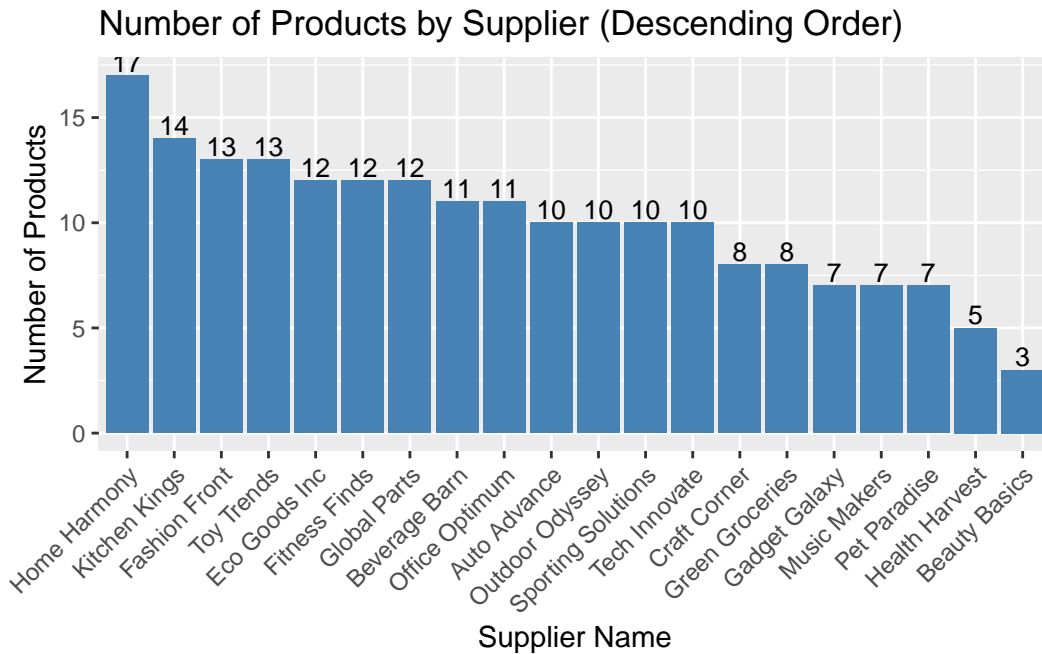
```



```

17 theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
18 labs(title = "Number of Products by Supplier (Descending Order)",
19       x = "Supplier Name",
20       y = "Number of Products")
21 print(g_supplier)

```



```

1 # Dynamically generate filename with current date and time
2 filename <- paste0("the product number supplied by supplier_", format(Sys.time(), "%Y%m%d_%H%M%S"))
3
4 # Save the plot with the dynamic filename
5 ggsave(filename, plot = g_supplier, width = width, height = height)

```

### C. The Product Review Scores of Different Suppliers (Best Top 5)

```

1 SELECT s.supplier_name, ROUND(AVG(p.review_score), 2) AS average_review_score
2 FROM product p
3 JOIN supplier s ON p.supplier_id = s.supplier_id
4 GROUP BY s.supplier_name
5 ORDER BY average_review_score DESC
6 LIMIT 5;

```

Table 11: 5 records

supplier_name	average_review_score
Global Parts	3.90
Pet Paradise	3.81
Office Optimum	3.76
Fashion Front	3.73
Music Makers	3.62

#### D. The Product Review Scores of Different Suppliers (Worst Top 5)

```

1 SELECT s.supplier_name, ROUND(AVG(p.review_score), 2) AS average_review_score
2 FROM product p
3 JOIN supplier s ON p.supplier_id = s.supplier_id
4 GROUP BY s.supplier_name
5 ORDER BY average_review_score ASC
6 LIMIT 5;
```

Table 12: 5 records

supplier_name	average_review_score
Fitness Finds	2.96
Auto Advance	2.99
Craft Corner	3.02
Kitchen Kings	3.04
Health Harvest	3.05

#### E. The Top 10 Best Selling Products

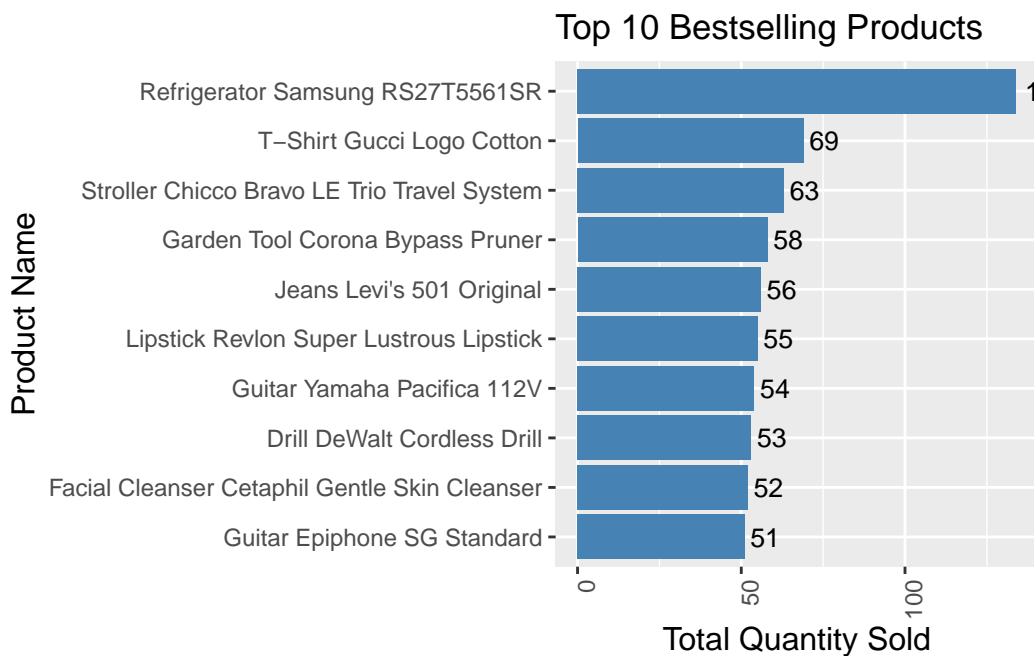
```

1 # Perform an inner join to combine 'orders' with 'product' on 'product_id'
2 joint_order_product <- inner_join(orders, product, by = "product_id")
3
4 # Calculate the total quantity sold for each product
5 product_sales_volume <- joint_order_product %>%
6   group_by(product_name) %>%
7   summarise(total_quantity_sold = sum(quantity)) # Assuming 'quantity' exists in your orders
8 # Processing the text of product name
9 product_sales_volume$product_name <- iconv(product_sales_volume$product_name, "UTF-8", "ASCII")
10
```

```

11 # Choose only the top 10 products based on total quantity sold
12 top_product_sales_volume <- product_sales_volume %>%
13   arrange(desc(total_quantity_sold)) %>%
14   slice_head(n = 10)
15
16 # Specify the dimensions of the plot
17 width <- 12
18 height <- 8
19
20 # Use ggplot to create a bar chart showing the total quantity sold for each product
21 g_topproduct <- ggplot(top_product_sales_volume, aes(x = reorder(product_name, total_quantity_sold))) +
22   geom_col(fill = "steelblue") +
23   geom_text(aes(label = total_quantity_sold), position = position_dodge(width = 0.9), hjust = 1) +
24   coord_flip() +
25   theme(axis.text.x = element_text(angle = 90, hjust = 1),
26         axis.title = element_text(size = 12)) +
27   labs(title = "Top 10 Bestselling Products",
28        x = "Product Name",
29        y = "Total Quantity Sold")
30 print(g_topproduct)

```



```

1 # Dynamically generate filename with current date and time
2 filename <- paste0("top10_products_by_quantity_", format(Sys.time(), "%Y%m%d_%H%M%S"), ".png")
3
4 # Save the plot with the dynamic filename
5 ggsave(filename, plot = g_topproduct, width = width, height = height)

```

## Supplier Analysis

The analysis shows that the products are supplied by various suppliers, ranging from household items, fashion stores, and toy wholesalers to fitness equipment suppliers. The e-commerce platform has a diversity of products to attract more potential customers. Breaking down into the suppliers' satisfaction, the top five have more than 3.5 out of 5 scores, and the worst five have approximately 3 scores, with no significant difference among the suppliers currently. Considering the product quality, suppliers with much higher product quality could be contacted further.

## Product Analysis

The reviewed scores shows that the top 10 products are mainly from the product category electronic devices and household items such as refrigerators, T-shirt,etc

### 4.2.3 Sales Analysis

#### A. Order Refund Rate

```

1 SELECT
2     COUNT(CASE WHEN refund_status = 'yes' THEN 1 END) AS refund_orders,
3     COUNT(*) AS total_orders,
4     CONCAT(ROUND((COUNT(CASE WHEN refund_status = 'yes' THEN 1 END) * 1.0 / COUNT(*)) * 100,
5 FROM (
6     SELECT DISTINCT order_id, refund_status
7     FROM orders
8 ) AS unique_orders;

```

Table 13: 1 records

refund_orders	total_orders	refund_rate_percentage
152	389	39.07%

### **Order Refund Analysis**

The order refund rate is 39.07%, higher than the IMRG's 2020 ([benchmarking data](#)); the average e-commerce business saw a 15% return rate. Refunding rates can damage the reputation of the business since when customers see high refund rates; they may be less likely to purchase the products in the future.