WARWICK BUSINESS SCHOOL
THE UNIVERSITY OF WARWICK

Masters Programmes: Group Assignment Cover Sheet

Student Numbers:	**5518804, 2236681, 5577818, 5588817, 5523853**
Module Code:	**IB9HP0**
Module Title:	**Data Management**
Submission Deadline:	**20/03/2024**
Date Submitted:	**20/03/2024**
Word Count:	**2000**
Number of Pages:	**68 (excluding appendix)**
Question Attempted:	**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 Management Assignment Group 18

Table of contents

Introduction 3

Part1 Database Design and Implementation 4
- **1.1 Entity-Relationship Modelling** . . . 4
- **1.2 SQL Database Schema Creation** . . . 9
  - 1.2.1 Strategic Advantage of ETL over ELT . . . 12

Part 2 Data Generation and Management 12
- **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 . . . 42

Part 3 Data Pipeline Generation 42
- **3.1 GitHub Repository and Workflow Setup** . . . 42
- **3.2 GitHub Actions for Continuous Integration** . . . 43

Part 4 Data Analysis and Reporting 46
- **4.1 Advanced Data Analysis in R** . . . 46
  - 4.1.1 Promotion Discount Trend . . . 46
  - 4.1.2 Promotion Count Trend . . . 48
    - Promotion Analysis . . . 49
  - 4.1.3 Monthly Revenue Trend . . . 49
    - Monthly Revenue Analysis . . . 50

4.1.4 Monthly Best-Selling Products 51
Monthly Best-Selling Products Analysis 53
4.1.5 Monthly Shipping Efficiency 53
4.1.6 Monthly Delivery Efficiency 55
Shipping and Delivery Efficiency Analysis 58
4.2 **Comprehensive Reporting with Quarto** 58
4.2.1 Demographic Distribution of Customers 58
A. The Distribution of Gender across Customers 58
B. The Distribution of Age across Customers 58
C. The Distribution of Careers across Customers (Top 10) 59
D. The Distribution of Geographic Location across Customers (Top 10) 60
E. The Current Customer Referral Rate 62
Customer Analysis 63
4.2.2 Product Portfolio 63
A. The Distribution of Product Review Scores (Top 10) 63
B. The Number of Products supplied by Different Suppliers 64
C. The Product Review Scores of Different Suppliers (Best Top 5) . . . 65
D. The Product Review Scores of Different Suppliers (Worst Top 5) . . 65
E. The Top 10 Best Selling Products 66
Supplier Analysis 67
Product Analysis 67
4.2.3 Sales Analysis 67
A. Order Refund Rate 67
Order Refund Analysis 68

Appendices **68**

Introduction

In this project, the aim is to emphasise the ETL flow, automation and data analysis, outlining the framework for managing e-commerce data environment, covering end-to-end data management. The framework entails 4 major steps as follows.

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

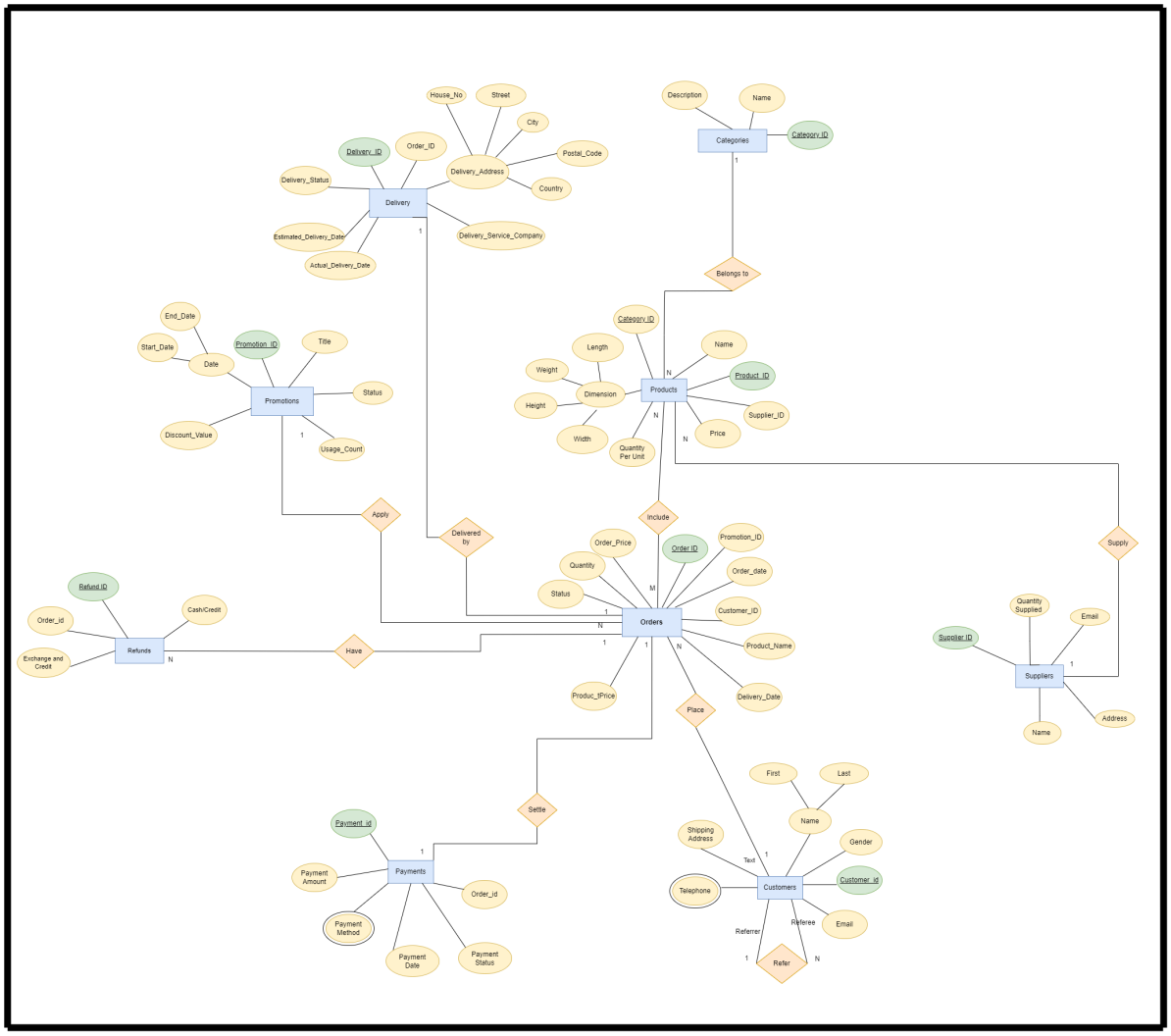


Figure 1: Initial Draft of E-R Diagram

After the initial draft of the E-R diagram, subsequent analysis led to the significant refinements aimed at enhancing the efficieny and clarify of the database structure.

1. 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. Therefore, 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.

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, thus making it 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.

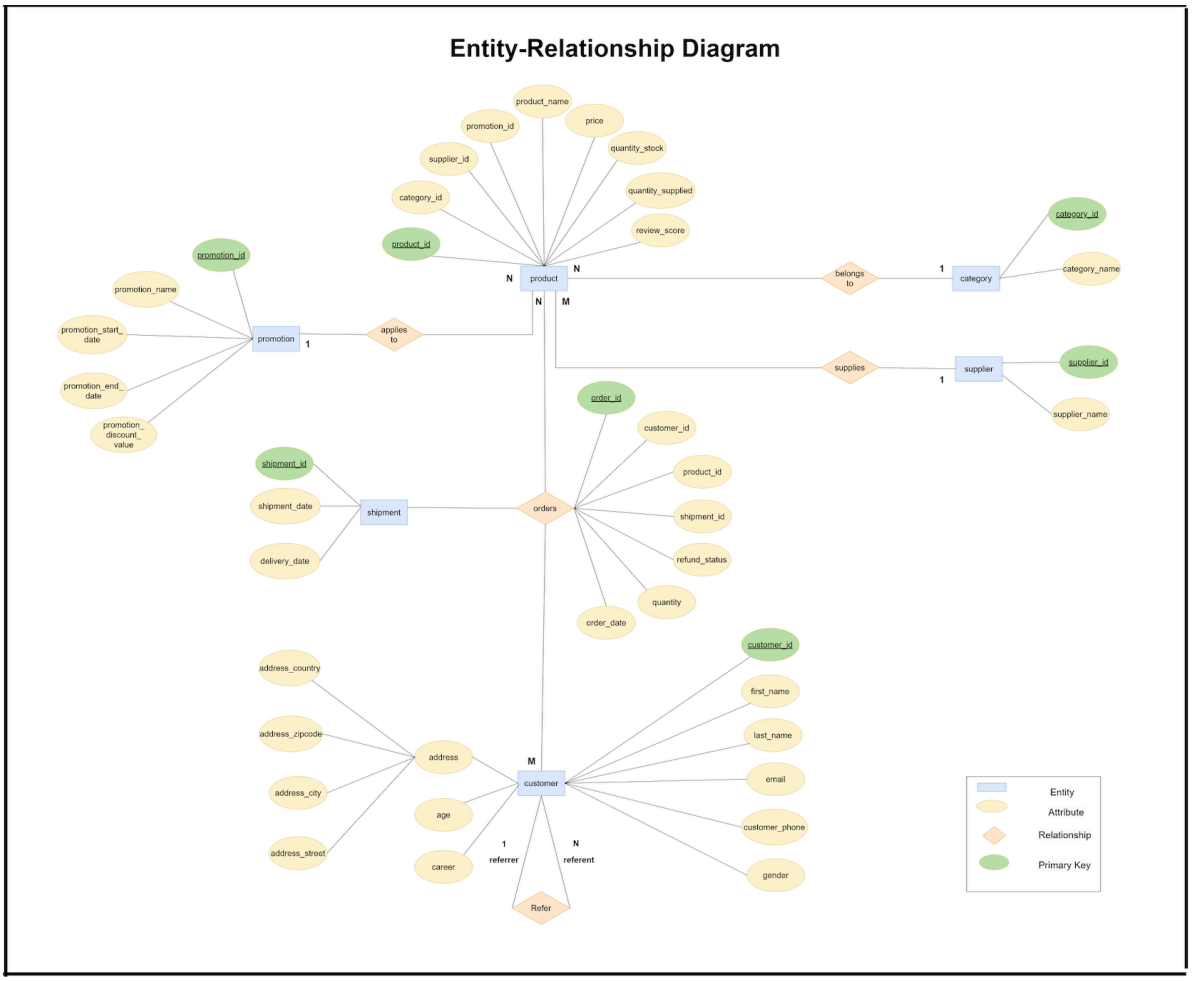


Figure 2: Final ER diagram

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.

Logical Database Schema:

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, quanity_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:

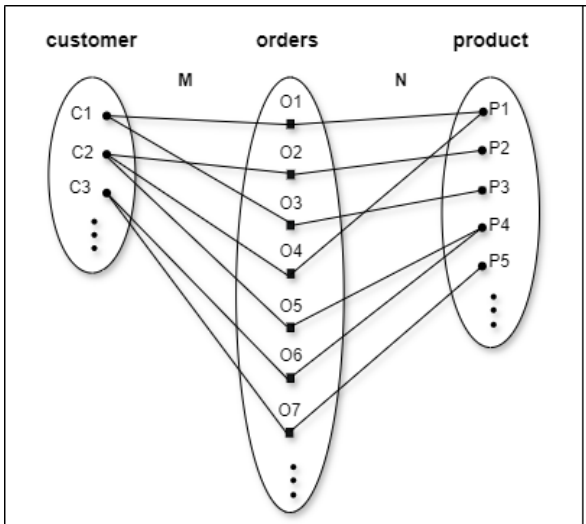


Figure 3: Relationship Set between customer and product having M:N cardinality.

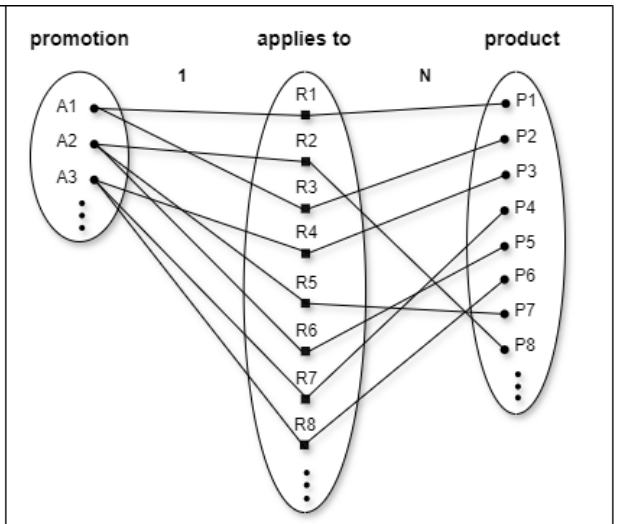


Figure 4: Relationship Set between promotion and product having 1:N cardinality.

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.

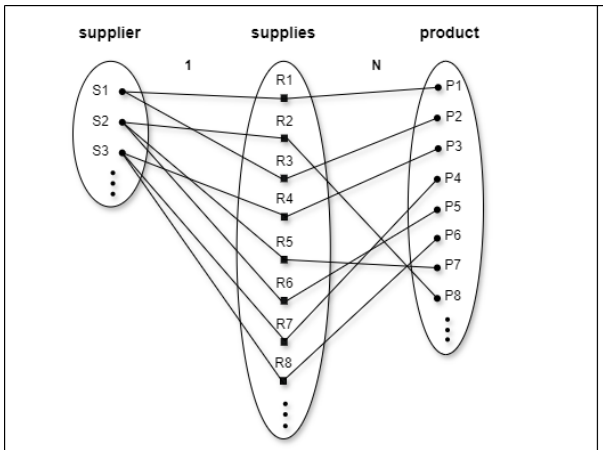


Figure 5: The relationship set between supplier and product having 1:N cardinality.

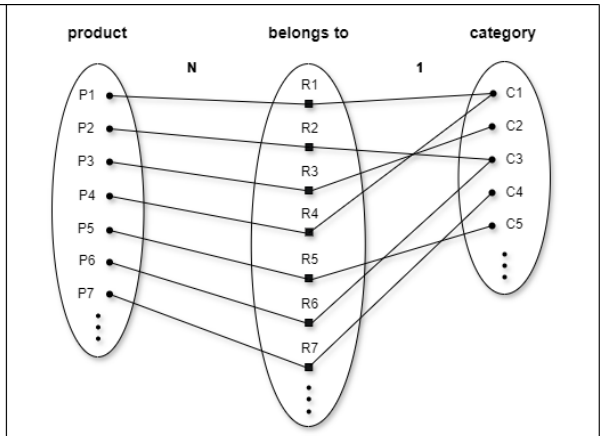


Figure 6: The relationship set between product and category having N:1 cardinality.

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.

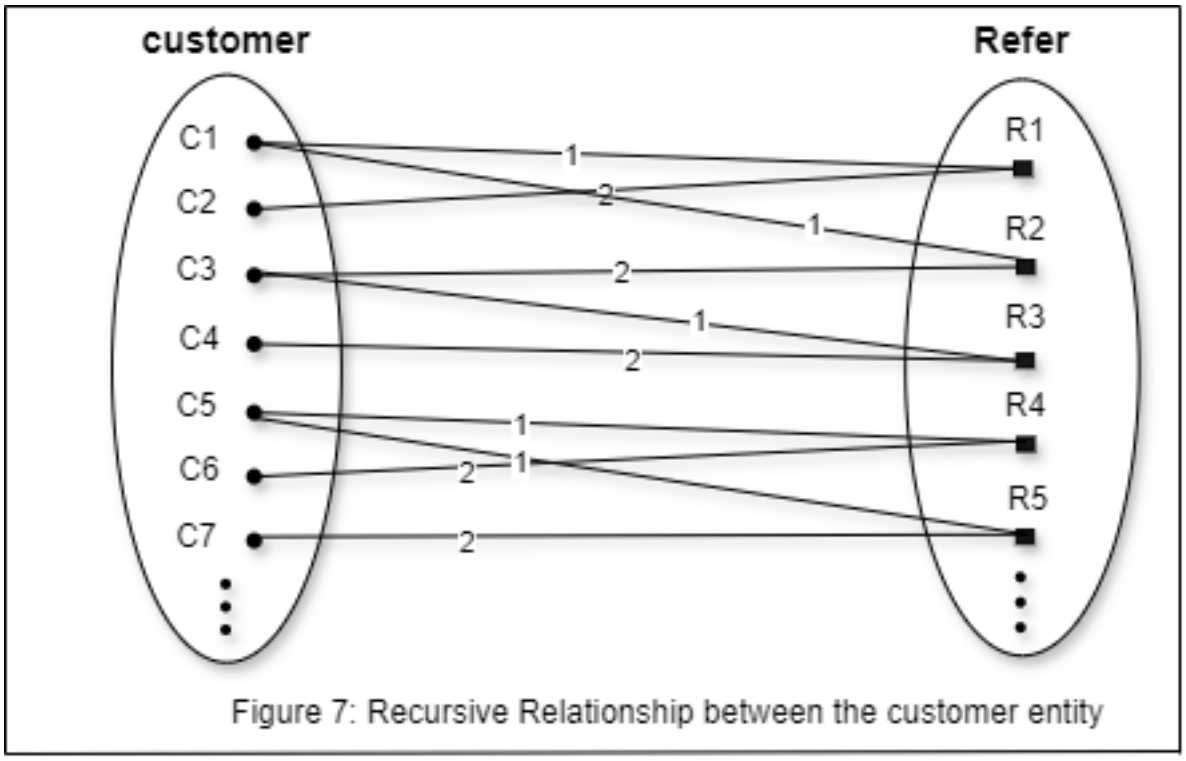


Figure 7: Recursive Relationship between the customer entity

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

```
RSQLite::dbExecute(connection,"

CREATE TABLE IF NOT EXISTS customer (

    customer_id VARCHAR(10) PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(50) NOT NULL,
    gender VARCHAR(20) NOT NULL,
    age INT NOT NULL,
    career VARCHAR(50) NOT NULL,
    customer_phone VARCHAR(20) NOT NULL,
    address_country VARCHAR(50) NOT NULL,
    address_zipcode VARCHAR(20) NOT NULL,
    address_city VARCHAR(20) NOT NULL,
    address_street VARCHAR(50) NOT NULL,
    referred_by VARCHAR(10) NULL

);
")
```

2. Creating category table

```
RSQLite::dbExecute(connection,"

CREATE TABLE IF NOT EXISTS category (

    category_id VARCHAR(10) PRIMARY KEY,
    category_name VARCHAR(50) NOT NULL

);

")
```

3. Creating supplier table

```
RSQLite::dbExecute(connection,"
CREATE TABLE IF NOT EXISTS supplier (

    supplier_id VARCHAR(10) PRIMARY KEY,
    supplier_name VARCHAR(50) NOT NULL

);

")
```

4. Creating promotion table

```
RSQLite::dbExecute(connection,"

CREATE TABLE IF NOT EXISTS promotion (

    promotion_id VARCHAR(10) PRIMARY KEY,
    promotion_name VARCHAR(20) NOT NULL,
    promotion_start_date DATE NOT NULL,
    promotion_end_date DATE NOT NULL,
    promotion_discount_value FLOAT NOT NULL

);

")
```

5. Creating shipment table

```
RSQLite::dbExecute(connection,"

CREATE TABLE IF NOT EXISTS shipment (

    shipment_id VARCHAR(10) PRIMARY KEY,
    shipment_date DATE NOT NULL,
    delivery_date DATE NOT NULL
);

")
```

6. Creating product table

```
RSQLite::dbExecute(connection,"

CREATE TABLE IF NOT EXISTS product (

    product_id VARCHAR(10) PRIMARY KEY,
    category_id VARCHAR(10) NOT NULL,
    supplier_id VARCHAR(10) NOT NULL,
    promotion_id VARCHAR(10) NULL,
    product_name VARCHAR(20) NOT NULL,
    price INT NOT NULL,
    quantity_stock INT NOT NULL,
    quantity_supplied INT NOT NULL,
    review_score FLOAT NOT NULL,
    FOREIGN KEY ('category_id') REFERENCES category('category_id'),
    FOREIGN KEY ('supplier_id') REFERENCES supplier('supplier_id'),
    FOREIGN KEY ('promotion_id') REFERENCES promotion('promotion_id')

);

")
```

7. Creating orders table

```
RSQLite::dbExecute(connection,"

CREATE TABLE IF NOT EXISTS orders (

    order_id VARCHAR(20) NOT NULL,
    customer_id VARCHAR(10) NOT NULL,
    product_id VARCHAR(10) NOT NULL,
    shipment_id VARCHAR(10) NOT NULL,
    quantity INT NOT NULL,
    refund_status VARCHAR(20) NOT NULL,
    order_date DATE NOT NULL,
    FOREIGN KEY ('customer_id') REFERENCES customer('customer_id'),
    FOREIGN KEY ('product_id') REFERENCES product('product_id'),
    FOREIGN KEY ('shipment_id') REFERENCES shipment('shipment_id')
    PRIMARY KEY (order_id,product_id,customer_id,shipment_id)
);

")
```

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

```
RSQLite::dbListTables(connection)
```

1.2.1 Strategic Advantage of ETL over ELT

In the development of the database, the principle of ETL (Extract, Transform, Load) was used rather than ELT (Extract, Load, Transform) as ETL 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 and 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 employed in this project is described below -

Extract – The data 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 – The extracted data was converted into CSV files and was further performed some validation tests. This step was 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.

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

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

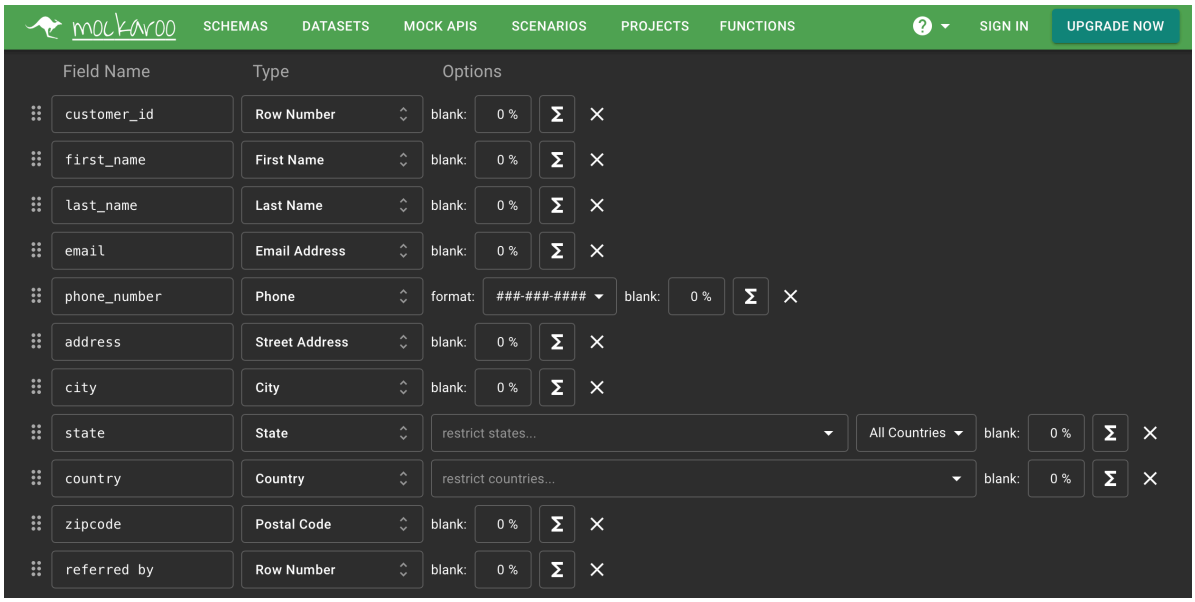


Figure 4: Data generation screenshot on mockaroo website

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.

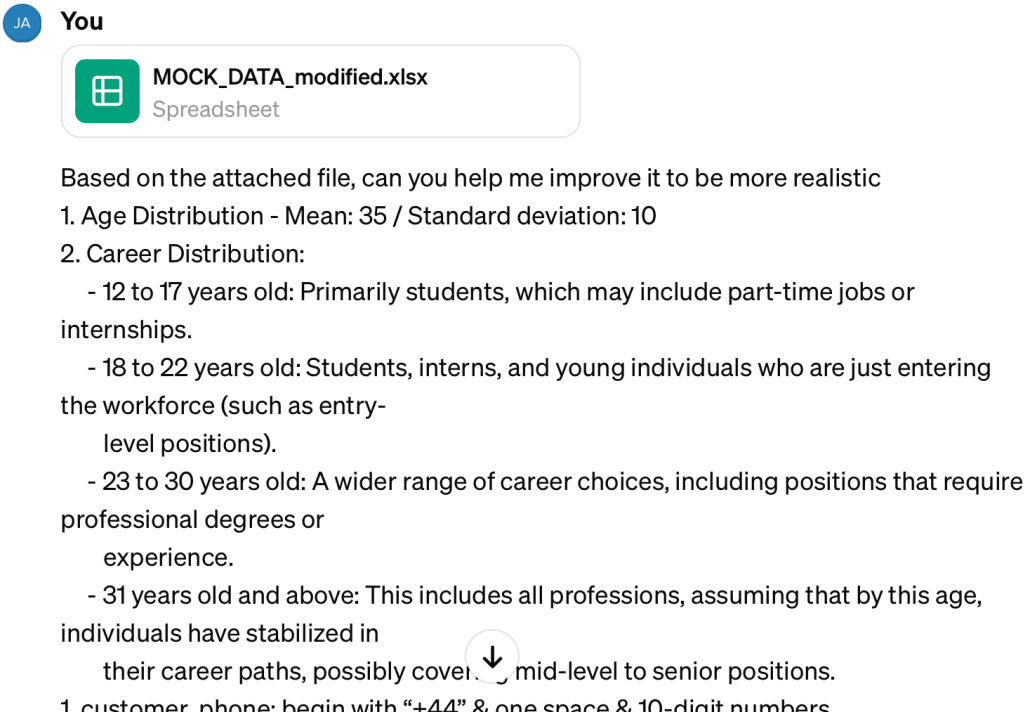


Figure 5: Data improvement request to ChatGPT for customer table

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

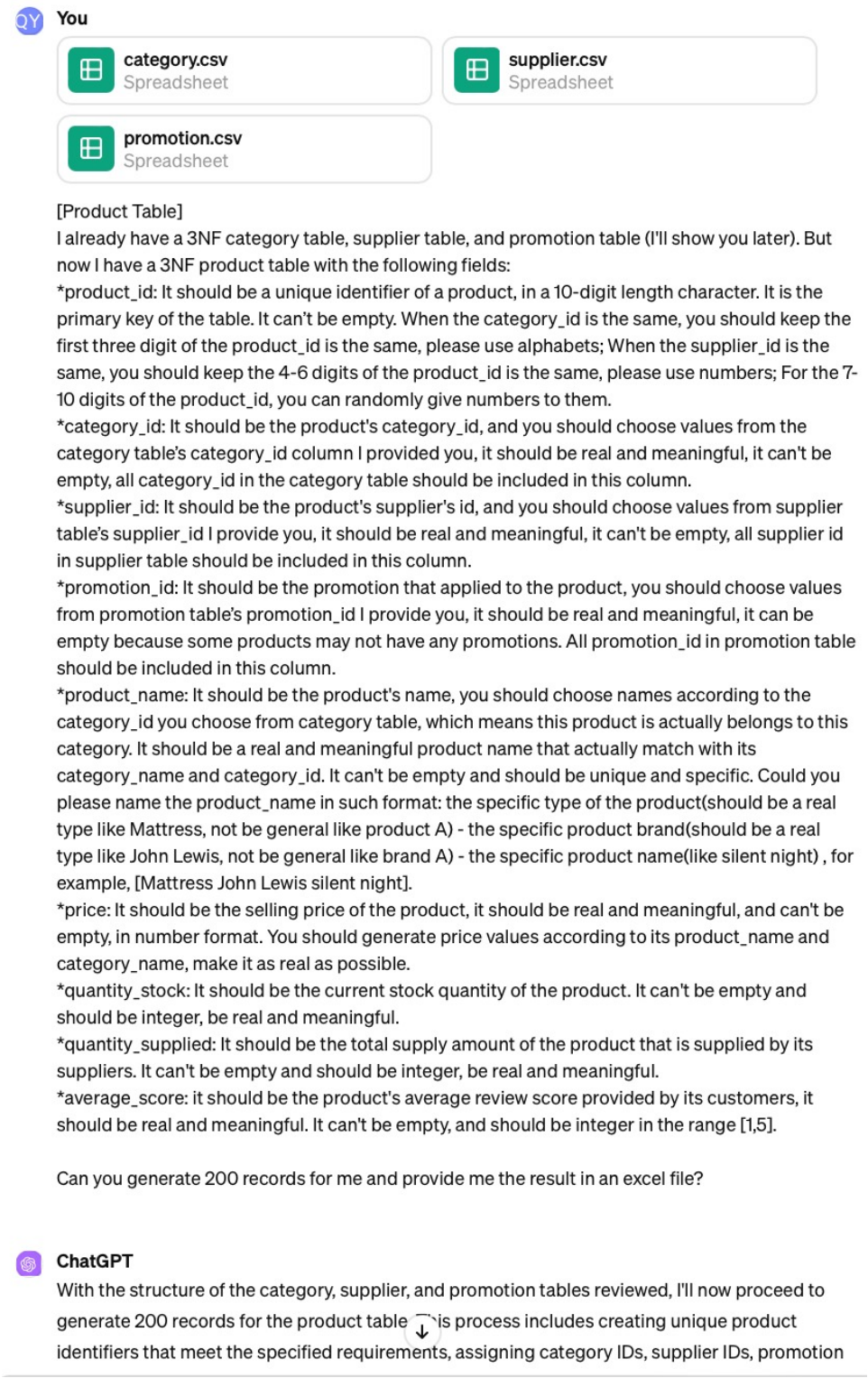


Figure 10: Data improvement request to ChatGPT for product table

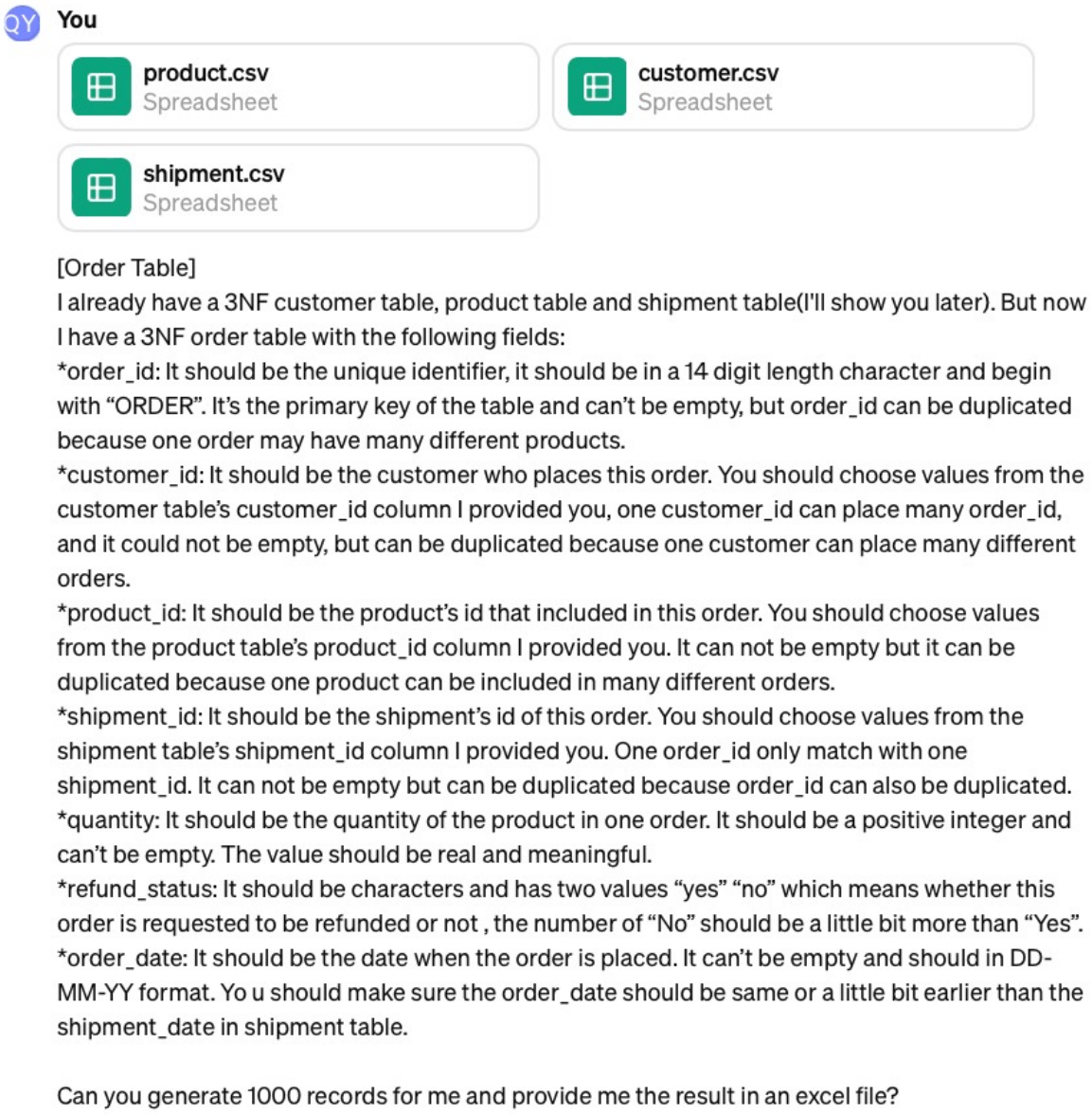


Figure 11: Data improvement request to ChatGPT for order table

2.2 Data Import and Quality Assurance

The ETL approach included extracting data from CSV files, transformed it through validations for data quality, referential integrity, and ensured accurate and normalised data in the database (ecomdata.db). The R scripts validate field correctness, including email formats, age ranges, and proper date formats, etc. Duplicate entries are prevented by checking existing records in

the database and within incoming files. Referential integrity for the 'orders', 'product', and 'customer' tables is maintained by verifying foreign keys before insertion. Additionally, hash checks before and after data loading for 'orders', 'product', and 'shipment' tables safeguard against data alteration.

```
Customer ID does not exist: NA
Product ID does not exist: NA
Shipment ID does not exist: NA
Referential integrity for the row is invalid.
Starting validation for checked new records.
Validation completed for new records.
Starting to insert validated data into the database. Number of records:  2
Data insertion completed successfully.
Data integrity verified: All record hashes match.
Successfully moved processed file to: data_processed/orders6_20240319205817.csv
Finished processing file: data_upload/orders6.csv
```

The output log provides the data importation and integrity verification process for orders table. It excludes the rows with referential integrity issues, confirms the successful insertion of two validated records into the database, and verifies data integrity through matching hash checks. Also, it documents the relocation of processed files to an organised directory.

Reading the files

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

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

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

```
for (variable in all_files) {
  filepath <- paste0("data_upload/",variable)
  file_contents <- readr::read_csv(filepath)

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

  #Printing the number of rows and columns in each file
  print(paste0("The file: ",variable,
               " has: ",
```

```
                format(number_of_rows,big.mark = ","),
                " rows and ",
                number_of_columns," columns"))

  number_of_rows <- nrow(file_contents)

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


  #Printing True if the first column is the primary key column else printing False
  print(paste0(" is ",nrow(unique(file_contents[,1]))==number_of_rows))
}
```

Structure of data

```
for (variable in all_files) {
  filepath <- paste0("data_upload/",variable)
  file_contents <- readr::read_csv(filepath)
  str_data <-str(file_contents)
  print(paste0(str_data,"Sructure of the file ", variable))
}
```

Number of columns and their column names

```
for (variable in all_files) {
  filepath <- paste0("data_upload/",variable)
  file_contents <- readr::read_csv(filepath)
  column_names <-colnames(file_contents)
  print(paste0("File ", variable, " has column as ",column_names))
}
```

Checking unique id column as the primary key in each table

```
for (variable in all_files) {
  filepath <- paste0("data_upload/", variable)
  file_contents <- readr::read_csv(filepath)
```

```
  primary_key <- nrow(unique(file_contents[,1])) == nrow(file_contents)
  print(paste0("Primary key of ", variable, " is ", primary_key))
}
```

Data Validation for Each Table

1. Validation for customer data

```
fetch_existing_customer_ids <- function(connection) {
  query <- "SELECT DISTINCT customer_id FROM customer"
  existing_ids <- dbGetQuery(connection, query)$customer_id
  return(existing_ids)

}

#Customer data validation and Reffered by referencial integrity
validate_and_prepare_customer_data <- function(data, existing_ids) {

  #Validation for customer ID
  customer_id_check <- grepl("^CUST[0-9]{6}$", data$customer_id)
  data <- data[customer_id_check, ]

  #Validation for email
  email_check <- grepl("^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}$", data$email)
  data<- data[email_check, ]

  #Validation for gender
  gender_check <- c("male","female","Female","Male")
  data<- data[data$gender %in% gender_check, ]

  #Validation for age
  age_check <- 1:100
  data <- data[data$age %in% age_check, ]

  #Validation for phone number
  phone_check <- grepl("^\\+44 \\d{10}$", data$customer_phone)
  data <- data[phone_check, ]

  #Validation for zip code
  zipcode_check <- grepl("^\\w{3} \\w{3}$", data$address_zipcode)
  data <- data[zipcode_check, ]
```

```r

  #Reffered by check
  unique_customer_ids <- unique(c(data$customer_id, existing_ids))
  #Validate 'referred by' IDs
  valid_referral_flags <- (data$referred_by == "") | data$referred_by %in% unique_customer_ic
  data <- data[valid_referral_flags, ]

  return(data)
}

#Fetch existing customer IDs from the database
existing_customer_ids <- fetch_existing_customer_ids(connection)

customer_file_paths <- list.files(path = "data_upload", pattern = "customer.*\\.csv$", full.n
#Initialising empty dataframe
customer_possible_data <- data.frame()

customer_primary_key <- "customer_id"

#Read each customer CSV file and check for the existence of the primary key in the database b
for (file_path in customer_file_paths) {
  cat("Starting processing file:", file_path, "\n")
  #Read the current file
  customer_data <- read.csv(file_path)

  #Iterate through each row of the file
  for (i in seq_len(nrow(customer_data))) {
    new_record <- customer_data[i, ]
    primary_key_value <- new_record[[customer_primary_key]]
    conditions <- paste(customer_primary_key, "=", paste0("'", primary_key_value, "'"))

    #Check if a record with the same primary key exists in the database
    record_exists_query <- paste("SELECT COUNT(*) FROM customer WHERE", conditions)
    record_exists_result <- dbGetQuery(connection, record_exists_query)
    record_exists <- record_exists_result[1, 1] > 0

    if(record_exists) {
      cat("Record with primary key", primary_key_value, "already exists in the database.\n")
    }
    if (!record_exists) {
      #Check if the primary key value of the new record is unique in the temporary dataframe
      if (!primary_key_value %in% customer_possible_data[[customer_primary_key]]) {
```

```
        customer_possible_data <- rbind(customer_possible_data, new_record)
      }
    }

    cat("Finished processing file:", file_path, "\n")

  }
  cat("Starting validation for new records.\n")
  customer_possible_data <- validate_and_prepare_customer_data(customer_possible_data, existi
  cat("Validation completed for new records.\n")
}


if (nrow(customer_possible_data) > 0)
{
  cat("Starting to insert validated data into the database. Number of records: ", nrow(custo

  #Ingesting prepared data to our database
  dbWriteTable(connection, name = "customer", value = customer_possible_data, append = TRUE,
  cat("Data insertion completed successfully.\n")
} else
{
  cat("No valid customer data to insert into the database.\n")
}
```

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

```
validate_and_prepare_category_data <- function(data) {

  # Validation for category ID
  category_id_check <- grepl("^[A-Za-z0-9]{10}$", data$category_id)
  data <- data[category_id_check,]

  return(data)
}

```

```r
# Fetch existing category IDs from the database

category_file_paths <- list.files(path = "data_upload", pattern = "category.*\\.csv$", full.r

# Define the primary key column for the category table
category_primary_key <- "category_id"

#Initialising empty data frame
category_possible_data <- data.frame()

# Read each category CSV file and check for the existence of the primary key in the database
for (file_path in category_file_paths) {

  cat("Starting processing file:", file_path, "\n")

  # Read the current file
  category_data <- readr::read_csv(file_path)

  # Iterate through each row of the file
  for (i in seq_len(nrow(category_data))) {
    new_record <- category_data[i, ]
    primary_key_value <- new_record[[category_primary_key]]
    conditions <- paste(category_primary_key, "=", paste0("'", primary_key_value, "'"))

    # Check if a record with the same primary key exists in the database
    record_exists_query <- paste("SELECT COUNT(*) FROM category WHERE", conditions)
    record_exists_result <- dbGetQuery(connection, record_exists_query)
    record_exists <- record_exists_result[1, 1] > 0

    if(record_exists) {
      cat("Record with primary key", primary_key_value, "already exists in the database.\n")
    }
    if (!record_exists) {
      # Check if the primary key value of the new record is unique in the temporary datafram
      if (!primary_key_value %in% category_possible_data[[category_primary_key]]) {
        category_possible_data <- rbind(category_possible_data, new_record)
      }
    }

    cat("Finished processing file:", file_path, "\n")

  }
```

```
}
cat("Starting validation for new records.\n")
category_possible_data <- validate_and_prepare_category_data(category_possible_data)
cat("Validation completed for new records.\n")

if (nrow(category_possible_data) > 0) {
  cat("Starting to insert validated data into the database. Number of records: ", nrow(categ

  # Ingesting prepared data to our database
  dbWriteTable(connection, name = "category", value = category_possible_data, append = TRUE,
  cat("Data insertion completed successfully.\n")
} else
{
  cat("No valid category data to insert into the database.\n")
}
```

3. Validations for supplier data

```
validate_and_prepare_supplier_data <- function(data) {
  # Validation for supplier ID
  supplier_id_check <- grepl("^[A-Za-z0-9]{10}$", data$supplier_id)
  data <- data[supplier_id_check,]

  return(data)
}

# Fetch existing supplier IDs from the database

supplier_file_paths <- list.files(path = "data_upload", pattern = "supplier.*\\.csv$", full.

# Define the primary key column for the supplier table
supplier_primary_key <- "supplier_id"

#Initialising empty data frame
supplier_possible_data <- data.frame()

# Read each supplier CSV file and check for the existence of the primary key in the database
for (file_path in supplier_file_paths) {

  cat("Starting processing file:", file_path, "\n")

```

```r
  # Read the current file
  supplier_data <- readr::read_csv(file_path)

  # Iterate through each row of the file
  for (i in seq_len(nrow(supplier_data))) {
    new_record <- supplier_data[i, ]
    primary_key_value <- new_record[[supplier_primary_key]]
    conditions <- paste(supplier_primary_key, "=", paste0("'", primary_key_value, "'"))

    # Check if a record with the same primary key exists in the database
    record_exists_query <- paste("SELECT COUNT(*) FROM supplier WHERE", conditions)
    record_exists_result <- dbGetQuery(connection, record_exists_query)
    record_exists <- record_exists_result[1, 1] > 0

    if(record_exists) {
      cat("Record with primary key", primary_key_value, "already exists in the database.\n")
    }
    if (!record_exists) {
      # Check if the primary key value of the new record is unique in the temporary datafram
      if (!primary_key_value %in% supplier_possible_data[[supplier_primary_key]]) {
        supplier_possible_data <- rbind(supplier_possible_data, new_record)
      }
    }

    cat("Finished processing file:", file_path, "\n")

  }

}
    cat("Starting validation for new records.\n")
    supplier_possible_data <- validate_and_prepare_supplier_data(supplier_possible_data)
    cat("Validation completed for new records.\n")

if (nrow(supplier_possible_data) > 0) {
  cat("Starting to insert validated data into the database. Number of records: ", nrow(suppli

  # Ingesting  prepared data to our database
  dbWriteTable(connection, name = "supplier", value = supplier_possible_data, append = TRUE,
  cat("Data insertion completed successfully.\n")
} else
{
  cat("No valid supplier data to insert into the database.\n")
```

```
}
```

4. Validations for promotion data

```
validate_and_prepare_promotion_data <- function(data) {

# Validation for promotion ID
promotion_id_check <- grepl("^[A-Za-z0-9]{10}$", data$promotion_id)
data <- data[promotion_id_check, ]

#Checking for the validation of the promotion_start_date and promotion_end_date in the promot
date_check <- !is.na(as.Date(data$promotion_start_date, format = "%d-%m-%Y")) &
!is.na(as.Date(data$promotion_end_date, format = "%d-%m-%Y")) &
as.Date(data$promotion_start_date, format = "%d-%m-%Y") < as.Date(data$promotion_end_date, fo

#Check for the validation of the promotion_start_date and promotion_end_date in the promotion
#promotion_start_date and promotion_end_data should be in correct form for eg 12/11/2023
date_format <- "%d-%m-%Y"
date_check <- !is.na(as.Date(data$promotion_start_date, format = date_format)) &
!is.na(as.Date(data$promotion_end_date, format = date_format)) &
as.Date(data$promotion_start_date, format = date_format) < as.Date(data$promotion_end_date, f
data <- data[date_check,]


#Check for the validation of the column promotion_discount_value in the promotion table.
#promotion_discount_value should be <1

discount_value_check <- !is.na(data$promotion_discount_value) &
is.numeric(data$promotion_discount_value) &
data$promotion_discount_value < 1
data <- data[discount_value_check,]
return(data)
}


# Fetch existing promotion IDs from the database

promotion_file_paths <- list.files(path = "data_upload", pattern = "promotion.*\\.csv$", full

# Define the primary key column for the promotion table
promotion_primary_key <- "promotion_id"

```

```r
#Initialising empty data frame
promotion_possible_data <- data.frame()


# Read each promotion CSV file and check for the existence of the primary key in the database
for (file_path in promotion_file_paths) {

cat("Starting processing file:", file_path, "\n")

# Read the current file
promotion_data <- readr::read_csv(file_path)

# Iterate through each row of the file
for (i in seq_len(nrow(promotion_data))) {
new_record <- promotion_data[i, ]
primary_key_value <- new_record[[promotion_primary_key]]
conditions <- paste(promotion_primary_key, "=", paste0("'", primary_key_value, "'"))

# Check if a record with the same primary key exists in the database
record_exists_query <- paste("SELECT COUNT(*) FROM promotion WHERE", conditions)
record_exists_result <- dbGetQuery(connection, record_exists_query)
record_exists <- record_exists_result[1, 1] > 0

if(record_exists) {
  cat("Record with primary key", primary_key_value, "already exists in the database.\n")
}
if (!record_exists) {
  # Check if the primary key value of the new record is unique in the temporary dataframe
  if (!primary_key_value %in% promotion_possible_data[[promotion_primary_key]]) {
    promotion_possible_data <- rbind(promotion_possible_data, new_record)
  }
}

cat("Finished processing file:", file_path, "\n")

}
}
cat("Starting validation for new records.\n")
promotion_possible_data <- validate_and_prepare_promotion_data(promotion_possible_data)
cat("Validation completed for new records.\n")


```

```r
if (nrow(promotion_possible_data) > 0)
{
cat("Starting to insert validated data into the database. Number of records: ", nrow(promotio
# Digesting prepared data to our database
dbWriteTable(connection, name = "promotion", value = promotion_possible_data, append = TRUE,
cat("Data insertion completed successfully.\n")
} else
{
cat("No valid promotion data to insert into the database.\n")
}
```

5. Validations for shipment data

```r
validate_and_prepare_shipment_data <- function(data) {
    # # Validation for shipment ID
    shipment_id_check <- grepl("^SHIP[0-9]{6}$", data$shipment_id)
    data <- data[shipment_id_check,]

    # Convert dates from character to Date object
    data$shipment_date <- as.Date(data$shipment_date, format = "%d-%m-%Y")
    data$delivery_date <- as.Date(data$delivery_date, format = "%d-%m-%Y")

    # Validation for shipment_date and delivery_date format
    date_format_check <- !is.na(data$shipment_date) & !is.na(data$delivery_date)

    # Keep only rows with valid date formats
    data <- data[date_format_check,]

    # Validation for logical order of shipment and delivery dates
    logical_date_order_check <- data$shipment_date <= data$delivery_date

    # Keep only rows with logical date order
    data <- data[logical_date_order_check,]

    return(data)
  }

  # Fetch existing shipment IDs from the database
  existing_shipment_ids <- dbGetQuery(connection, "SELECT shipment_id FROM shipment")$shipme
  # List all files that have shipment in the title
  shipment_file_paths <- list.files(path = "data_upload", pattern = "shipment.*\\.csv$", ful
  # Destination directory to move after processing the file
```

```r
destination_dir <- "data_processed"
# Ensure the destination directory exists before processing files
if (!dir.exists(destination_dir)) {
  dir.create(destination_dir)
}

# Check if there are new files for shipment table
if (length(shipment_file_paths) > 0) {
  #Initialising empty data frame
  shipment_possible_data <- data.frame(shipment_id = character(), stringsAsFactors = FALSE
  # Read each shipment CSV file and check for the existence of the primary key in the data
  for (file_path in shipment_file_paths) {
    cat("Starting processing file:", file_path, "\n")

    # Read the current file
    shipment_data <- readr::read_csv(file_path, show_col_types = FALSE)

    # Filter out records with existing shipment_id in the database
    unique_shipment_data <- shipment_data[!shipment_data$shipment_id %in% existing_shipmen

    # Combine unique records into the possible data frame
    shipment_possible_data <- rbind(shipment_possible_data, unique_shipment_data)

    # Construct the destination file path
    file_name <- basename(file_path)
    dest_file_path <- file.path(destination_dir, file_name)

    # Construct the destination file path with a timestamp to ensure uniqueness
    file_name <- basename(file_path)
    timestamp <- format(Sys.time(), "%Y%m%d%H%M%S")  # Get current timestamp
    file_ext <- tools::file_ext(file_name)  # Extract file extension
    base_name <- sub(paste0("\\.", file_ext, "$"), "", file_name)  # Remove extension fro
    new_file_name <- paste0(base_name, "_", timestamp, ".", file_ext)  # Create new filena
    dest_file_path <- file.path(destination_dir, new_file_name)  # Construct new destinati

    # Move the processed file to the destination directory
    if (file.rename(file_path, dest_file_path)) {
      cat("Successfully moved processed file to:", dest_file_path, "\n")
    } else {
      cat("Failed to move file:", file_path, "\n")
    }
    cat("Finished processing file:", file_path, "\n")
```

```
72      }
73
74      # Ensure shipment_possible_data contains only unique shipment_ids, no repetition of new
75      shipment_possible_data <- shipment_possible_data %>% distinct(shipment_id, .keep_all = T
76
77
78      if (nrow(shipment_possible_data) > 0) {
79        cat("Starting validation for new records.\n")
80        shipment_possible_data <- validate_and_prepare_shipment_data(shipment_possible_data)
81        cat("Validation completed for new records.\n")
82      }
83
84      # Order data to ensure consistent hashing
85      shipment_possible_data <- shipment_possible_data[order(shipment_possible_data$shipment_i
86
87      if (!is.null(dim(shipment_possible_data)) && dim(shipment_possible_data)[1] > 0) {
88        # Generate pre-load hashes
89        pre_load_hashes <- sapply(1:nrow(shipment_possible_data), function(i) {
90          record <- as.character(unlist(shipment_possible_data[i, ]))
91          digest(paste(record, collapse = "|"), algo = "md5")
92        })
93        cat("Starting to insert validated data into the database. Number of records: ", nrow(s
94        tryCatch({
95          dbWriteTable(connection, name = "shipment", value = shipment_possible_data, append =
96          cat("Data insertion completed successfully.\n")
97        }, error = function(e) {
98          cat("Error inserting data into the database: ", e$message, "\n")
99        })
100       # Fetch the loaded data back for post-load hash comparison
101       loaded_shipment_ids <- sprintf("'%s'", shipment_possible_data$shipment_id)
102       query <- sprintf("SELECT * FROM shipment WHERE shipment_id IN (%s) ORDER BY shipment_i
103
104       retrieved_shipments <- dbGetQuery(connection, query)
105
106       post_load_hashes <- sapply(1:nrow(retrieved_shipments), function(i) {
107         record <- as.character(unlist(retrieved_shipments[i, ]))
108         digest(paste(record, collapse = "|"), algo = "md5")
109       })
110
111       if (nrow(retrieved_shipments) > 0) {
112
113         # Compare hashes
```

```
        identical_hashes <- all(pre_load_hashes == post_load_hashes)
        if (identical_hashes) {
          cat("Data integrity verified: All record hashes match.\n")
        } else {
          cat("Data integrity check failed: Record hashes do not match.\n")
        }
      }
    } else {
      cat("No valid shipment data to insert into the database.\n")
    }
  } else {
    cat("No new files to process.\n")
  }
```

6. Validations for product data

```
# function to validate and check referential integrity
validate_and_prepare_product_data <- function(data, connection) {

 # Validation for product ID
 product_id_check <- grepl("^[A-Za-z0-9]{10}$", data$product_id)
 data <- data[product_id_check, ]
 # Performing validation for review score
 data <- data[data$review_score >= 1 & data$review_score <= 5, ]

 # Fetch existing IDs from reference tables
 valid_category_ids <- dbGetQuery(connection, "SELECT category_id FROM category")$category_id
 valid_supplier_ids <- dbGetQuery(connection, "SELECT supplier_id FROM supplier")$supplier_id
 valid_promotion_ids <- c(dbGetQuery(connection, "SELECT promotion_id FROM promotion")$promot

 # Referential integrity checks
 data <- data[data$category_id %in% valid_category_ids, ]
 data <- data[data$supplier_id %in% valid_supplier_ids, ]
 data <- data[is.na(data$promotion_id) | data$promotion_id %in% valid_promotion_ids, ]

 # Convert quantity_stock and quantity_supplied to numeric to ensure
 # non-numeric values are handled. This introduces NA for any non-numeric values.
 data$quantity_stock <- as.numeric(data$quantity_stock)
 data$quantity_supplied <- as.numeric(data$quantity_supplied)

 # Now proceed with the validations for non-negative, non-NA, and integer for quantity_stock
 data <- data[!is.na(data$quantity_stock) & data$quantity_stock >= 0 & (data$quantity_stock =
```

```
 data <- data[!is.na(data$quantity_supplied) & data$quantity_supplied >= 0 & (data$quantity_s

 # Validation for positive price values
 data <- data[data$price > 0 & !is.na(data$price), ]

 # Validation to ensure product_name is not empty
 data <- data[data$product_name != "" & !is.na(data$product_name), ]

 return(data)
}


# Fetch existing product IDs from the database into a vector
existing_product_ids <- tryCatch({
 dbGetQuery(connection, "SELECT product_id FROM product")$product_id
}, error = function(e) {
 cat("Error fetching existing product IDs: ", e$message, "\n")
 NULL
})

# List files that have 'product' in title
product_file_paths <- list.files(path = "data_upload", pattern = "product.*\\.csv$", full.nam
# Initialising temporary empty data frame for future loading
product_possible_data <- data.frame()

# Read each product CSV file
for (file_path in product_file_paths) {
 cat("Starting processing file:", file_path, "\n")
 # Read the current file
 product_data <- tryCatch({
   readr::read_csv(file_path)
 }, error = function(e) {
   cat("Error reading file", file_path, ": ", e$message, "\n")
   next  # Skip to the next iteration of the loop
 })

 # Iterate through each row of the file and check for uniqueness of primary key in database a
 for (i in seq_len(nrow(product_data))) {
   new_record <- product_data[i, ]
   primary_key_value <- new_record[["product_id"]]

   if (!primary_key_value %in% existing_product_ids) {
```

```
      if (!primary_key_value %in% product_possible_data[["product_id"]]) {
        product_possible_data <- rbind(product_possible_data, new_record)
      }
    } else {
      cat("Record with primary key", primary_key_value, "already exists in the database.\n")
    }
  }
  cat("Finished processing file:", file_path, "\n")
}

# Call for validation and integrity check function
cat("Starting validation for new records.\n")
product_possible_data <- validate_and_prepare_product_data(product_possible_data, connection)
cat("Validation completed for new records.\n")

# Implementing data integrity check for validated and ready to loading data for each row
if ("product_id" %in% names(product_possible_data) && nrow(product_possible_data) > 0) {
  product_possible_data <- product_possible_data[order(product_possible_data$product_id), ]
} else {
  cat("product_id column not found or product_possible_data is empty.\n")
}
pre_load_hashes <- sapply(1:nrow(product_possible_data), function(i) {
  record <- as.character(unlist(product_possible_data[i, ]))
  digest(paste(record, collapse = "|"), algo = "md5")
})

# Ingesting into database command
if (nrow(product_possible_data) > 0) {
  cat("Starting to insert validated data into the database. Number of records: ", nrow(product

  # Ingesting prepared data to our database
  tryCatch({
    dbWriteTable(connection, name = "product", value = product_possible_data, append = TRUE,
    cat("Data insertion completed successfully.\n")
  }, error = function(e) {
    cat("Error inserting data into the database: ", e$message, "\n")

     # Additional error handling logic here
  })

  # Fetch the loaded data back for post-load hash comparison
  loaded_product_ids <- sprintf("'%s'", product_possible_data$product_id)
```

```
 query <- sprintf("SELECT * FROM product WHERE product_id IN (%s) ORDER BY product_id", paste
 retrieved_products <- dbGetQuery(connection, query)

 # Creating hashes for retrieved data from db
 post_load_hashes <- sapply(1:nrow(retrieved_products), function(i) {
   record <- as.character(unlist(retrieved_products[i, ]))
   digest(paste(record, collapse = "|"), algo = "md5")
 })

 # Compare hashes for pre-loaded and retrieved-loaded data
 identical_hashes_product <- all(pre_load_hashes == post_load_hashes)
 if (identical_hashes_product) {
   cat("Data integrity verified: All record hashes match.\n")
 } else {
   cat("Data integrity check failed: Record hashes do not match.\n")
 }
} else {
 cat("No valid product data to insert into the database.\n")
}
```

7. Validation for orders data

```
#Validation for orders data

   # Function to check if given foreign IDs of orders data exist in their respective tables
   check_id_exists <- function(connection, table_name, column_name, ids) {
     query_template <- "SELECT DISTINCT %s FROM %s WHERE %s IN (%s)"
     query <- sprintf(query_template, column_name, table_name, column_name, paste0("'", ids,
     existing_ids <- dbGetQuery(connection, query)[[1]]
     all(ids %in% existing_ids)
   }

   # Function to validate that foreign keys of 'orders' are existing in respective tables
   validate_referential_integrity <- function(new_record) {
     customer_exists <- check_id_exists(connection, "customer", "customer_id", new_record$cus
     product_exists <- check_id_exists(connection, "product", "product_id", new_record$produc
     shipment_exists <- check_id_exists(connection, "shipment", "shipment_id", new_record$shi

     if(product_exists & customer_exists & shipment_exists) {
       return(TRUE)
     } else {
       if(!customer_exists) cat("Customer ID does not exist:", new_record$customer_id, "\n")
```

```
      if(!product_exists) cat("Product ID does not exist:", new_record$product_id, "\n")
      if(!shipment_exists) cat("Shipment ID does not exist:", new_record$shipment_id, "\n")
      return(FALSE)
    }
  }


  # Referential integrity of orders table
  # Function to check if given foreign IDs of orders data exist in their respective tables
  check_id_exists <- function(connection, table_name, column_name, ids) {
    query_template <- "SELECT DISTINCT %s FROM %s WHERE %s IN (%s)"
    query <- sprintf(query_template, column_name, table_name, column_name, paste0("'", ids,
    existing_ids <- dbGetQuery(connection, query)[[1]]
    all(ids %in% existing_ids)
  }

  # Function to validate that foreign keys of 'orders' are existing in respective tables
  validate_referential_integrity <- function(new_record) {
    customer_exists <- check_id_exists(connection, "customer", "customer_id", new_record$cus
    product_exists <- check_id_exists(connection, "product", "product_id", new_record$produc
    shipment_exists <- check_id_exists(connection, "shipment", "shipment_id", new_record$shi

    if(product_exists & customer_exists & shipment_exists) {
      return(TRUE)
    } else {
      if(!customer_exists) cat("Customer ID does not exist:", new_record$customer_id, "\n")
      if(!product_exists) cat("Product ID does not exist:", new_record$product_id, "\n")
      if(!shipment_exists) cat("Shipment ID does not exist:", new_record$shipment_id, "\n")
      return(FALSE)
    }
  }



  # Validation function of orders table

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

    customer_id_check <- grepl("^CUST[0-9]{6}$", data$customer_id)
```

```r
    data <- data[customer_id_check, ]

    product_id_check <- grepl("^[A-Za-z0-9]{10}$", data$product_id)
    data <- data[product_id_check, ]

    shipment_id_check <- grepl("^SHIP[0-9]{6}$", data$shipment_id)
    data <- data[shipment_id_check,]

    # Convert order_date strings to Date objects
    converted_dates <- as.Date(data$order_date, format = "%d-%m-%Y")

    # Efficient validation for order_date format and range
    valid_dates <- !is.na(converted_dates) & converted_dates >= as.Date("01-01-2023")

    # Filter data based on the valid_dates check
    data <- data[valid_dates, ]

    # Validation for order_date format
    order_date_format_check <- !is.na(as.Date(data$order_date, format = "%d-%m-%Y"))
    data <- data[order_date_format_check,]


    return(data)
  }

  # List all files that have 'orders' in name
  orders_file_paths <- list.files(path = "data_upload", pattern = "orders.*\\.csv$", full.na
  # Create a temporary df for data to possibly upload
  orders_possible_data <- data.frame()
  # Set a data_processed directory for transfering the file
  destination_dir <- "data_processed"

  # Ensure the destination directory exists
  if (!dir.exists(destination_dir)) {
    dir.create(destination_dir)
  }

  # Check if there are new files for the orders table
  if (length(orders_file_paths) > 0) {
    # Read each orders CSV file and check for the existence of the composite primary key in
    for (file_path in orders_file_paths) {
      orders_data <- readr::read_csv(file_path, show_col_types = FALSE)
```

```
105         cat("Starting processing file:", file_path, "\n")
106         # Iterate through each row of the file
107         for (i in seq_len(nrow(orders_data))) {
108           new_record <- orders_data[i, ]
109 
110           # Check for the Referential integrity
111           if(validate_referential_integrity(new_record)) {
112             # Construct the condition to check the composite primary key (order_id, product_id
113             conditions <- sprintf("order_id = '%s' AND product_id = '%s' AND customer_id = '%s
114                                   new_record$order_id, new_record$product_id, new_record$custo
115 
116             # Check if a record with the same composite primary key exists in the database
117             record_exists_query <- paste("SELECT COUNT(*) FROM orders WHERE", conditions)
118             record_exists_result <- dbGetQuery(connection, record_exists_query)
119             record_exists <- record_exists_result[1, 1] > 0
120 
121 
122             if(!record_exists) {
123               # Construct a unique identifier for the composite primary key for potential new
124               composite_key <- paste(new_record$order_id, new_record$product_id, new_record$cu
125 
126               # Function to check if the composite primary key is unique in the temporary data
127               existing_keys <- sapply(1:nrow(orders_possible_data), function(i) {
128                 paste(orders_possible_data[i, "order_id"], orders_possible_data[i, "product_id
129                       orders_possible_data[i, "shipment_id"], sep = "-")
130               })
131 
132               if (!composite_key %in% existing_keys) {
133                 # If new potential row has unique primary key then move it to temprorary df
134                 orders_possible_data <- rbind(orders_possible_data, new_record)
135               } else {
136                 cat("Record with composite primary key already exists in temporary data.\n")
137               }
138             } else {
139               cat("Record with composite primary key already exists in the database.\n")
140             }
141           } else {
142             cat("Referential integrity for the row is invalid.\n")
143           }
144         }
145         cat("Starting validation for checked new records.\n")
146         orders_possible_data <- validate_and_prepare_orders_data(orders_possible_data)
```

```
147         cat("Validation completed for new records.\n")
148
149         # Sorting the data to maintain ordering
150         if (nrow(orders_possible_data) > 0) {
151           orders_possible_data <- orders_possible_data[order(orders_possible_data$order_id, or
152                                                              orders_possible_data$shipment_id
153
154           # Hashing the ready to upload data for data integrity check
155           pre_load_hashes <- sapply(1:nrow(orders_possible_data), function(i) {
156             record <- as.character(unlist(orders_possible_data[i, ]))
157             digest(paste(record, collapse = "|"), algo = "md5")
158           })
159         }
160
161         # Orders ingestion
162         if (nrow(orders_possible_data) > 0) {
163           cat("Starting to insert validated data into the database. Number of records: ", nrow
164           # Attempt to ingest prepared data to our database
165           tryCatch({
166             dbWriteTable(connection, name = "orders", value = orders_possible_data, append = T
167             cat("Data insertion completed successfully.\n")
168           }, error = function(e) {
169             cat("Error inserting data into the database: ", e$message, "\n")
170           })
171           # Fetch the loaded data back for post-load hash comparison
172           loaded_order_ids <- sprintf("'%s'", orders_possible_data$order_id)
173           query <- sprintf("SELECT * FROM orders WHERE order_id IN (%s) ORDER BY order_id, pro
174           retrieved_orders <- dbGetQuery(connection, query)
175           # Creating hashes for retrieved data from db
176           post_load_hashes <- sapply(1:nrow(retrieved_orders), function(i) {
177             record <- as.character(unlist(retrieved_orders[i, ]))
178             digest(paste(record, collapse = "|"), algo = "md5")
179           })
180
181           # Compare hashes for pre-loaded and retrieved-loaded data
182           identical_hashes <- all(pre_load_hashes == post_load_hashes)
183           if (identical_hashes) {
184             cat("Data integrity verified: All record hashes match.\n")
185           } else {
186             cat("Data integrity check failed: Record hashes do not match.\n")
187           }
188         } else {
```

```
        cat("No valid orders data to insert into the database.\n")
      }
      # Move the processed file to 'data_processed' directory with a timestamp
      file_name <- basename(file_path)
      timestamp <- format(Sys.time(), "%Y%m%d%H%M%S")
      file_ext <- tools::file_ext(file_name)
      base_name <- sub(paste0("\\.", file_ext, "$"), "", file_name)
      new_file_name <- paste0(base_name, "_", timestamp, ".", file_ext)
      dest_file_path <- file.path(destination_dir, new_file_name)

      if (file.rename(file_path, dest_file_path)) {
        cat("Successfully moved processed file to:", dest_file_path, "\n")
      } else {
        cat("Failed to move file:", file_path, "\n")
      }
      cat("Finished processing file:", file_path, "\n")
    }
  } else {
    # If there are no new files, skip processing and notify the user
    cat("No new files to process.\n") }
```

Referential Integrity for ensuring data integrity

Foreign key is the crucial component of a database that enforces referential integrity, ensuring that a value appearing in one entity should also exists in another entity as a foreign key. Basically, it 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. It also acted as the central hub of the 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

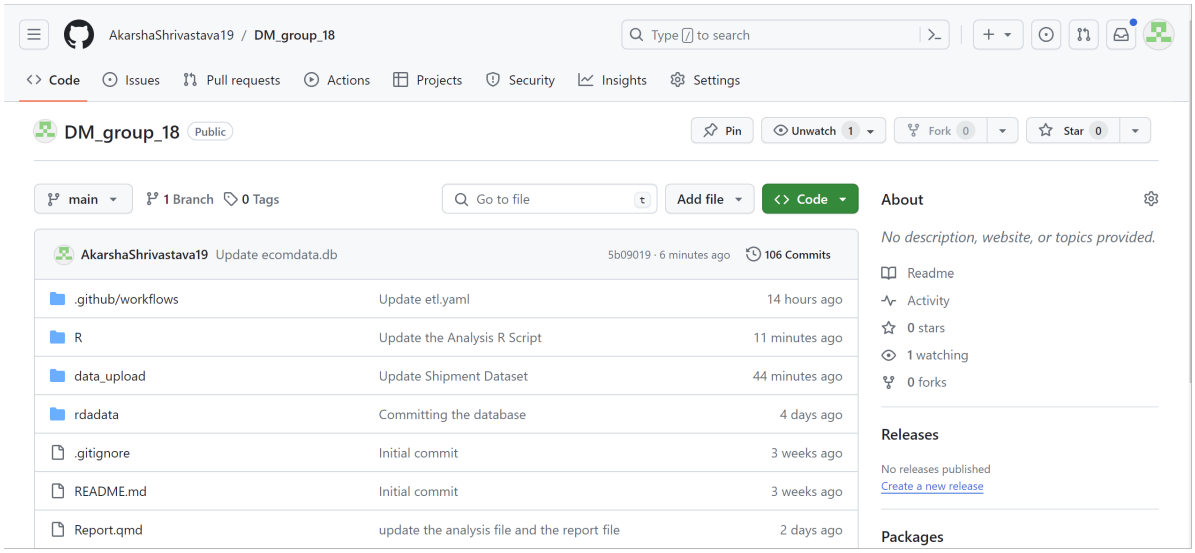
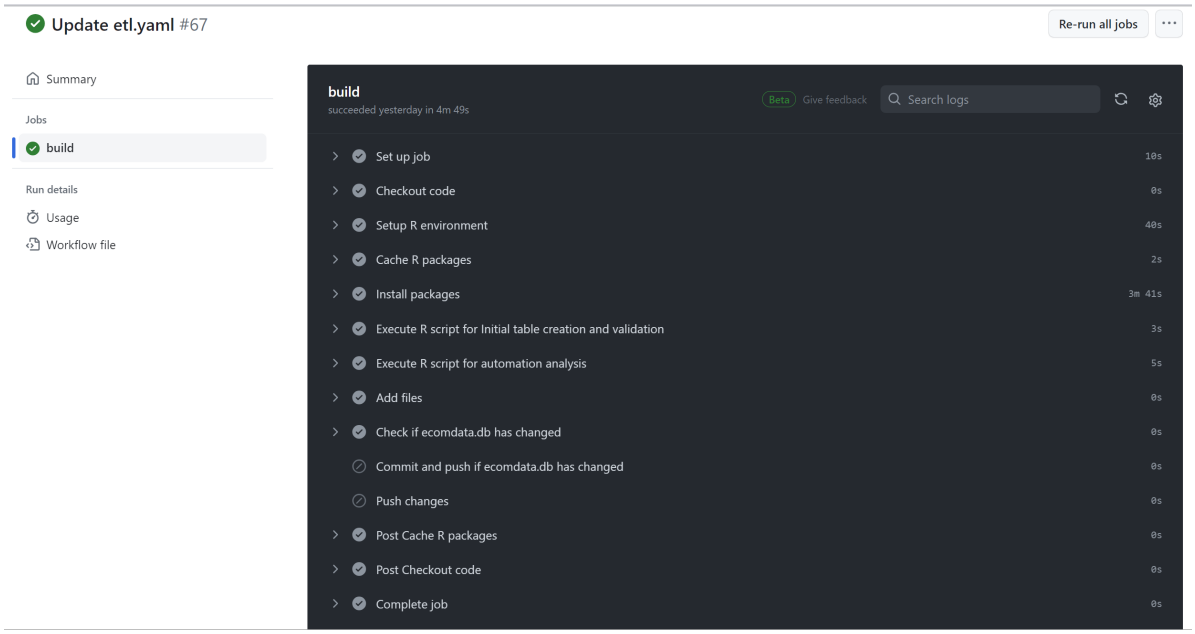


Figure 12: Created GitHub Repository

3.2 GitHub Actions for Continuous Integration

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

It runs multiple jobs seamlessly on the latest Ubuntu Environment as shown below.



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 rapidly detecting the issues.

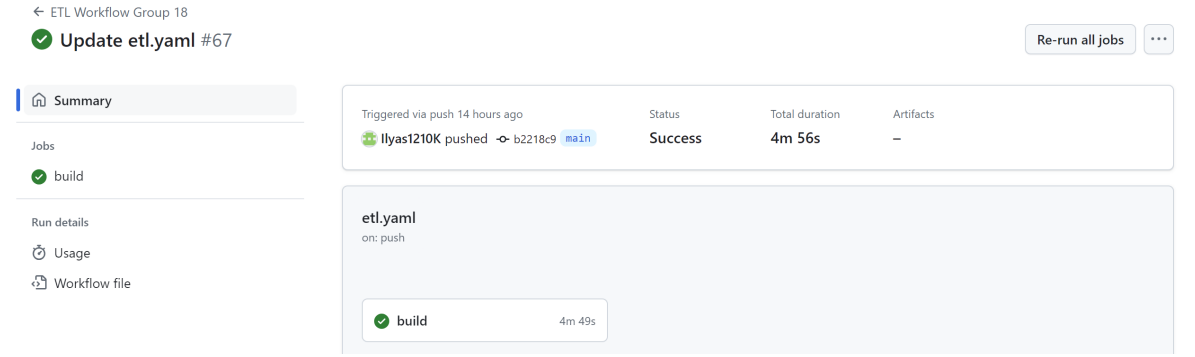


Figure 13: Successfully Build Workflow

The yaml code is added below:

```
name: ETL Workflow Group 18

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 packages
    if: steps.cache.outputs.cache-hit != 'true'
    run: |
      Rscript -e
'install.packages(c("ggplot2","dplyr","readr","RSQLite","DBI","lubridate","tidyr",
"knitr", "digest"))'

  - name: Execute R script for Initial table creation and validation
    run: |
      Rscript R/data_workflow.R

  - name: Execute R script for automation analysis
    run: |
      Rscript R/Analysis.R

  - name: Add files
    run: |
      git config --global user.email "akarsha.shrivastava@warwick.ac.uk"
      git config --global user.name "AkarshaShrivastava19"

  - name: Check if ecomdata.db has changed
    id: check-db
    run: |
      git diff --exit-code --name-only | grep 'ecomdata.db' && echo "::set-output
name=changed::true" || echo "::set-output name=changed::false"

  - name: Commit and push if ecomdata.db has changed
    if: steps.check-db.outputs.changed == 'true'
    run: |
      git add ecomdata.db
      git commit -m "Update ecomdata.db"
      git push

  - name: Push changes
    if: steps.check-db.outputs.changed == 'true'
    uses: ad-m/github-push-action@v0.6.0
    with:
        github_token: ${{ secrets.GITHUB_TOKEN }}
        branch: main
```

Part 4 Data Analysis and Reporting

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

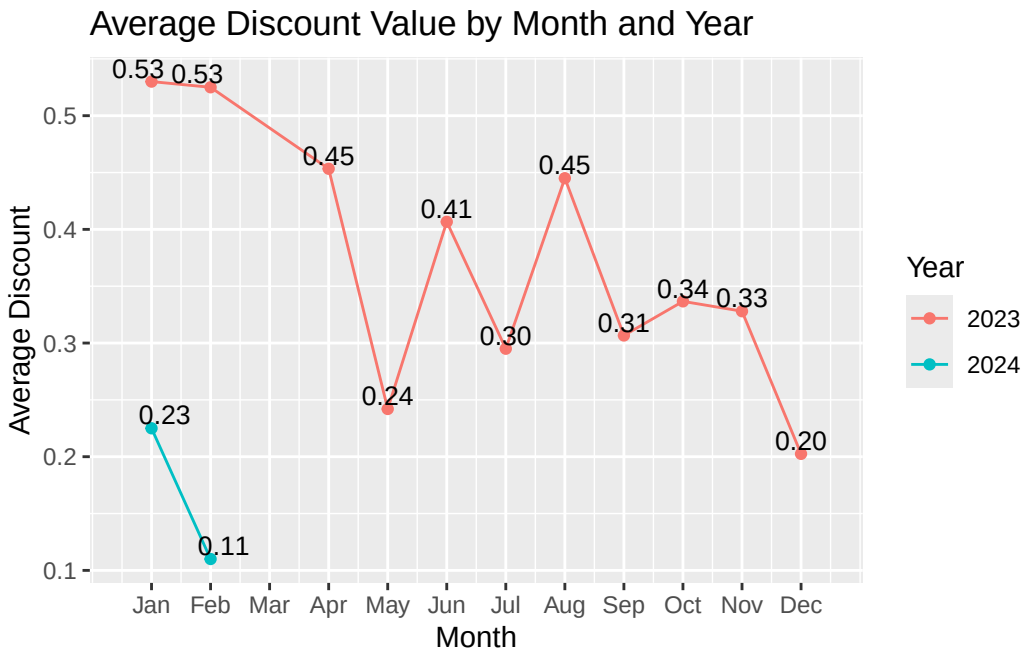
4.1 Advanced Data Analysis in R

```r
# Retrieve data from the database
customer <- dbGetQuery(connection, "SELECT * FROM customer")
product <- dbGetQuery(connection, "SELECT * FROM product")
supplier <- dbGetQuery(connection, "SELECT * FROM supplier")
category <- dbGetQuery(connection, "SELECT * FROM category")
shipment <- dbGetQuery(connection, "SELECT * FROM shipment")
promotion <- dbGetQuery(connection, "SELECT * FROM promotion")
orders <- dbGetQuery(connection, "SELECT * FROM orders")
```

4.1.1 Promotion Discount Trend

```r
# Convert Date Format
promotion <- promotion %>%
  mutate(promotion_start_date = dmy(promotion_start_date),
         promotion_end_date = dmy(promotion_end_date))

# Generate records for each month that each promotion spans
data_expanded <- promotion %>%
  rowwise() %>%
  mutate(months = list(seq(from = promotion_start_date,
                           to = promotion_end_date,
                           by = "month"))) %>%
  unnest(months) %>%
  mutate(year = year(months), month = month(months)) %>%
  group_by(promotion_id, year, month) %>%
  summarise(promotion_discount_value = mean(promotion_discount_value), .groups = 'drop')

# Calculate the average discount value for each month
average_discounts <- data_expanded %>%
  group_by(year, month) %>%
  summarise(average_discount = mean(promotion_discount_value))

# Specify the dimensions of the plot
```

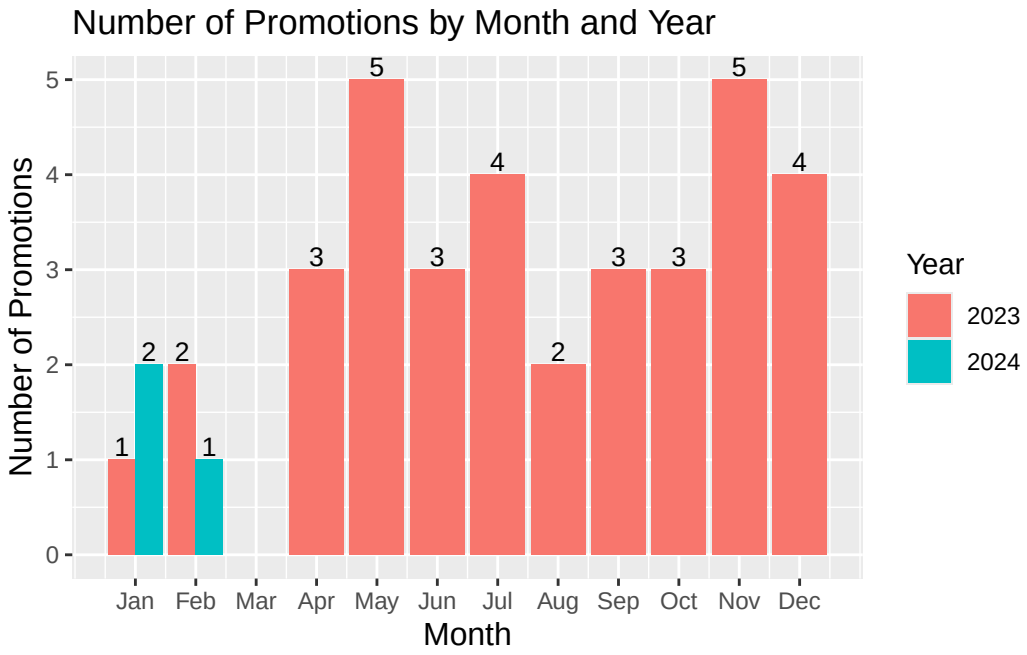
```r
width <- 12
height <- 8

# Visualise the average discount value for different months and years
g_promotionvalue <- ggplot(average_discounts, aes(x = month, y = average_discount, group = ye
  geom_line() +
  geom_point() +
  scale_x_continuous(breaks = 1:12, labels = month.abb) +
  geom_text(aes(label = sprintf("%.2f", average_discount)), position = position_dodge(width =
  labs(title = "Average Discount Value by Month and Year",
       x = "Month",
       y = "Average Discount",
       color = "Year")
print(g_promotionvalue)
```




```r
# Dynamically generate filename with current date and time
filename <- paste0("promotion_discount_trend_", format(Sys.time(), "%Y%m%d_%H%M%S"), ".png")

# Save the plot with the dynamic filename
ggsave(filename, plot = g_promotionvalue, width = width, height = height)
```

4.1.2 Promotion Count Trend

```r
# Calculate the number of times a promotion appears in each month
promotion_counts <- data_expanded %>%
  count(year, month)

# Visualise the number of times promotions appear in different years and months
g_promotioncount <- ggplot(promotion_counts, aes(x = month, y = n, fill = as.factor(year))) +
  geom_bar(stat = "identity", position = "dodge") +
  scale_x_continuous(breaks = 1:12, labels = month.abb) +
  geom_text(aes(label = n), position = position_dodge(width = 0.9), vjust = -0.2, size = 3.5)
  theme(axis.title = element_text(size = 12)) +
  labs(title = "Number of Promotions by Month and Year",
       x = "Month",
       y = "Number of Promotions",
       fill = "Year")
print(g_promotioncount)
```




```r
# Dynamically generate filename with current date and time
filename <- paste0("promotion_number_trend_", format(Sys.time(), "%Y%m%d_%H%M%S"), ".png")

# Save the plot with the dynamic filename
ggsave(filename, plot = g_promotioncount, width = width, height = height)
```

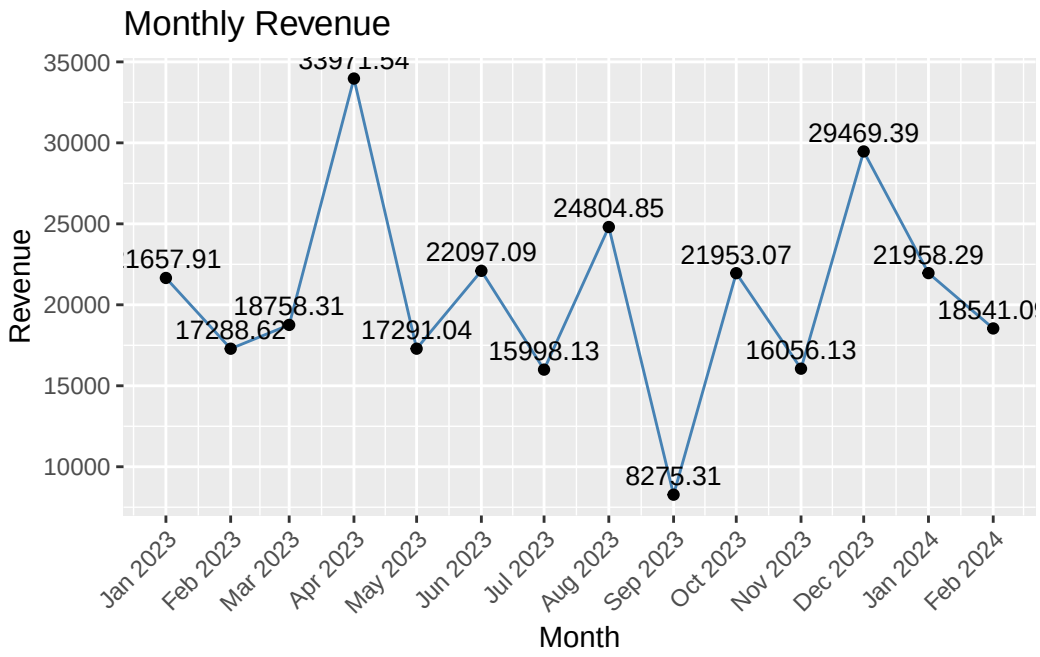
Promotion Analysis

The above graphs illustrate the highest discounts that were offered during January and February, i.e. 53% off on the product price.

4.1.3 Monthly Revenue Trend

```r
# Preprocessed date formats
orders <- orders %>% mutate(order_date = dmy(order_date))
promotion <- promotion %>%
  mutate(start_date = promotion_start_date,
         end_date = promotion_end_date,
         promotion_discount_value = if_else(is.na(promotion_discount_value), 0, promotion_dis

# Merge orders with products for pricing information
order_products <- orders %>%
  left_join(product, by = "product_id")

# Make sure there are no missing prices or quantities
order_products <- order_products %>%
  mutate(price = if_else(is.na(price), 0, price),
         quantity = if_else(is.na(quantity), 0, quantity))

# Combine orders, products and promotions to take into account discounts during promotions
order_products_promotions <- order_products %>%
  left_join(promotion, by = "promotion_id") %>%
  mutate(is_promotion = if_else(order_date >= start_date & order_date <= end_date, TRUE, FALS
         revenue = price * quantity * if_else(is_promotion, 1 - promotion_discount_value, 1))

# Remove any missing income values generated in the calculation
order_products_promotions <- order_products_promotions %>%
  filter(!is.na(revenue))

# Calculation of gross monthly income
monthly_revenue <- order_products_promotions %>%
  mutate(month = floor_date(order_date, "month")) %>%
  group_by(month) %>%
  summarize(total_revenue = sum(revenue, na.rm = TRUE))

# Visualisation of monthly income
(g_monthlyrevenue <- ggplot(monthly_revenue, aes(x = month, y = total_revenue)) +
```

```
  geom_line(color = "steelblue") +
  geom_point() +
  scale_x_date(date_labels = "%b %Y", date_breaks = "1 month") +
  geom_text(aes(label = sprintf("%.2f", total_revenue)), position = position_dodge(width = 0
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  labs(title = "Monthly Revenue", x = "Month", y = "Revenue"))
```




```
# Dynamically generate filename with current date and time
filename <- paste0("monthly_revenue_",
                   format(Sys.time(), "%Y%m%d_%H%M%S"), ".png")

# Save the plot with the dynamic filename
ggsave(filename, plot = g_monthlyrevenue, width = width, height = height)
```

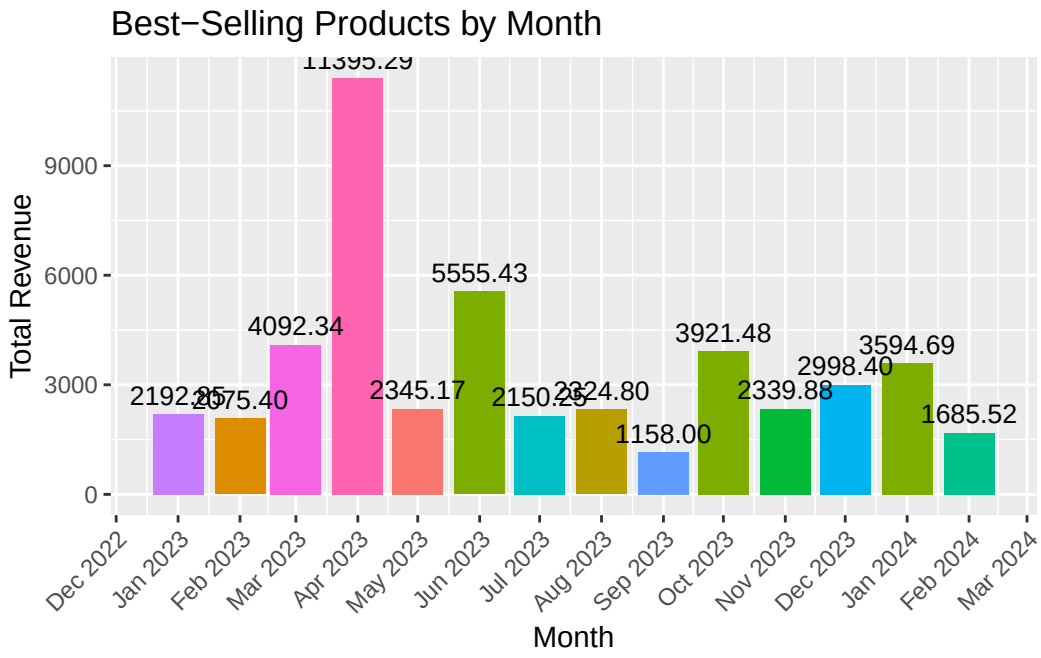
Monthly Revenue Analysis

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

Graph for Monthly Best-Selling Products

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

# Select top earning products per month
best_selling_products_each_month <- monthly_product_revenue %>%
  group_by(month) %>%
  slice_max(total_revenue_product, n = 1) %>%
  ungroup() %>%
  select(month, product_id, total_revenue_product)

best_selling_products_each_month <- merge(best_selling_products_each_month, product[, c("proc

# Visualise the top earning products and their revenues per month
g_bestseller_product_monthly <- ggplot(best_selling_products_each_month, aes(x = month, y = t
  geom_col(show.legend = FALSE) +
  geom_text(aes(label = sprintf("%.2f", total_revenue_product)), position = position_dodge(wi
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  scale_x_date(date_labels = "%b %Y", date_breaks = "1 month") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  labs(title = "Best-Selling Products by Month", x = "Month", y = "Total Revenue")

print(g_bestseller_product_monthly)
```




```
# Dynamically generate filename with current date and time
filename <- paste0("monthly_bestseller_product_",
                   format(Sys.time(), "%Y%m%d_%H%M%S"), ".png")

# Save the plot with the dynamic filename
ggsave(filename, plot = g_bestseller_product_monthly, width = width, height = height)
```

Table for Monthly Best-Selling Products

```
best_selling_products_each_month$month <- as.Date(best_selling_products_each_month$month, "%

best_selling_products_each_month$YearMonth <- format(best_selling_products_each_month$month,

best_selling_products_each_month <- best_selling_products_each_month %>%
  arrange(YearMonth)

table_to_display <- best_selling_products_each_month %>%
  select(YearMonth, product_name, total_revenue_product) %>%
  rename('Total Revenue' = total_revenue_product)

# Display the table with kable
kable(table_to_display, caption = "Monthly Best-Selling Products", col.names = c("Time", "Pr
```

Table 1: 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.

4.1.5 Monthly Shipping Efficiency

```
## Monthly Shipping Efficiency
# Convert dates to Date objects
shipment$shipment_date <- as.Date(shipment$shipment_date, origin = "1970-01-01")

shipment_unique <- shipment %>% distinct(shipment_id, .keep_all = TRUE)

# Merge orders and shipment data on order_id
combined_data <- merge(orders, shipment, by = "shipment_id")

# Calculate shipping duration in days
combined_data$shipping_duration <- as.numeric(difftime(combined_data$shipment_date, combined

# Calculate monthly statistics
monthly_stats <- combined_data %>%
  mutate(month = floor_date(order_date, "month")) %>%
```

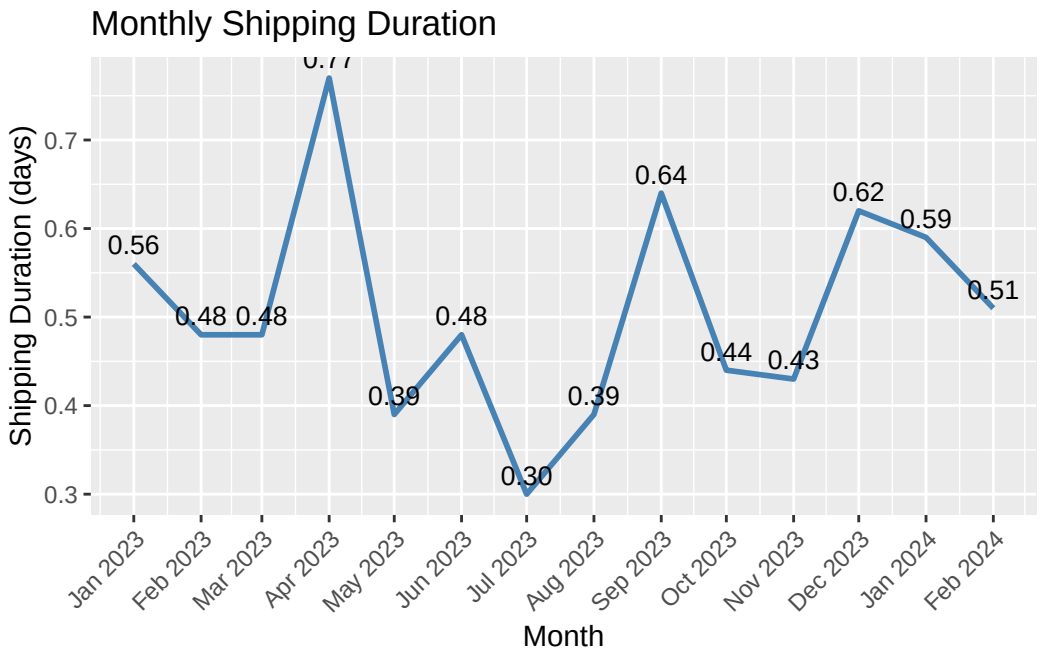
```r
  group_by(month) %>%
  summarise(Average_Shipping_Duration = round(mean(shipping_duration),2),
            Min_Shipping_Duration = min(shipping_duration),
            Max_Shipping_Duration = max(shipping_duration))

table_to_display <- monthly_stats %>%
  mutate(Time = format(month, "%Y-%m")) %>%
  select(Time, Average_Shipping_Duration, Min_Shipping_Duration, Max_Shipping_Duration)

# Display the table with kable
kable(table_to_display, caption = "Monthly Shipping Duration", col.names = c("Time", "Averag
```

Table 2: 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
2023-06	0.48	0	1
2023-07	0.30	0	1
2023-08	0.39	0	1
2023-09	0.64	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

```r
# Visualize the statistics
g_shiping_efficiency <- ggplot(monthly_stats, aes(x = month)) +
  geom_line(aes(y = Average_Shipping_Duration), color = "steelblue", size = 1) +
  geom_text(aes(y = Average_Shipping_Duration,
                label = sprintf("%.2f", Average_Shipping_Duration)),
            position = position_dodge(width = 0.9), vjust = -0.5, size = 3.5) +
  scale_x_date(date_labels = "%b %Y",
               date_breaks = "1 month",
               limits = c(min(monthly_stats$month), max(monthly_stats$month))) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
```

```
  labs(title = "Monthly Shipping Duration",
       x = "Month", y = "Shipping Duration (days)")

print(g_shiping_efficiency)
```




```
# Dynamically generate filename with current date and time
filename <- paste0("monthly_shiping_efficiency_",
                   format(Sys.time(), "%Y%m%d_%H%M%S"), ".png")

# Save the plot with the dynamic filename
ggsave(filename, plot = g_shiping_efficiency, width = width, height = height)
```

4.1.6 Monthly Delivery Efficiency

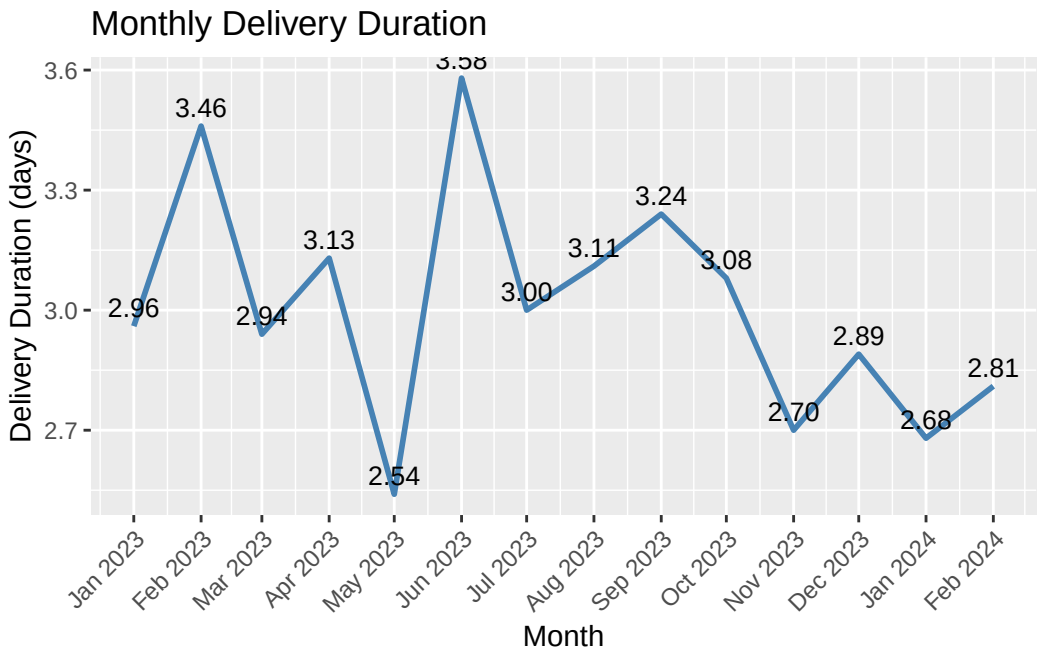
```
## Monthly Delivery Efficiency
# Convert dates to Date objects
shipment$delivery_date <- as.Date(shipment$delivery_date, origin = "1970-01-01")

shipment_unique <- shipment %>% distinct(shipment_id, .keep_all = TRUE)

# Merge orders and shipment data on order_id
```

```r
combined_data <- merge(orders, shipment, by = "shipment_id")

# Calculate delivery duration in days
combined_data$delivery_duration <- as.numeric(difftime(combined_data$delivery_date, combined_

# Calculate monthly statistics
monthly_stats <- combined_data %>%
  mutate(month = floor_date(order_date, "month")) %>%
  group_by(month) %>%
  summarise(Average_Delivery_Duration = round(mean(delivery_duration),2),
            Min_Delivery_Duration = min(delivery_duration),
            Max_Delivery_Duration = max(delivery_duration))

table_to_display <- monthly_stats %>%
  mutate(Time = format(month, "%Y-%m")) %>%
  select(Time, Average_Delivery_Duration, Min_Delivery_Duration, Max_Delivery_Duration)

# Calculate the overall average delivery duration
overall_avg_delivery <- mean(combined_data$delivery_duration, na.rm = TRUE)

# Display the table with kable
kable(table_to_display, caption = "Monthly Delivery Duration", col.names = c("Time", "Average
```

Table 3: Monthly Delivery Duration

Time	Average Duration	Min Duration	Max Duration
2023-01	2.96	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	3.24	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

```r
# Visualize the statistics
g_delivery_efficiency <- ggplot(monthly_stats, aes(x = month)) +
    geom_line(aes(y = Average_Delivery_Duration), color = "steelblue", size = 1) +
    geom_text(aes(y = Average_Delivery_Duration,
                  label = sprintf("%.2f", Average_Delivery_Duration)),
              position = position_dodge(width = 0.9), vjust = -0.5, size = 3.5) +
    scale_x_date(date_labels = "%b %Y",
                 date_breaks = "1 month",
                 limits = c(min(monthly_stats$month), max(monthly_stats$month))) +
    theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
    labs(title = "Monthly Delivery Duration",
         x = "Month", y = "Delivery Duration (days)")
print(g_delivery_efficiency)
```




```r
# Dynamically generate filename with current date and time
filename <- paste0("monthly_delivery_efficiency_",
                   format(Sys.time(), "%Y%m%d_%H%M%S"), ".png")

# Save the plot with the dynamic filename
ggsave(filename, plot = g_delivery_efficiency, width = width, height = height)
```

Shipping and Delivery Efficiency 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, indicating 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

```
SELECT
    gender,
    COUNT(*) AS GenderCount,
    CONCAT(ROUND((COUNT(*) * 100.0) / (SELECT COUNT(*) FROM customer), 2), '%') AS Percentage
FROM
    customer
GROUP BY
    gender
ORDER BY
    Percentage DESC;
```

Table 4: 2 records

gender	GenderCount	Percentage
Male	110	55.28%
Female	89	44.72%

B. The Distribution of Age across Customers

```
SELECT
    AgeGroup,
    COUNT(*) AS Count,
    CONCAT(ROUND((COUNT(*) * 100.0) / (SELECT COUNT(*) FROM customer), 2), '%') AS Percentage
FROM (
    SELECT
        customer_id,
        CASE
            WHEN age >= 0 AND age < 18 THEN '0-18'
```

```
            WHEN age >= 18 AND age < 30 THEN '19-30'
            WHEN age >= 30 AND age < 40 THEN '31-40'
            WHEN age >= 40 AND age < 50 THEN '41-50'
            WHEN age >= 50 AND age < 60 THEN '51-60'
            WHEN age >= 60 AND age < 70 THEN '61-70'
            WHEN age >= 70 THEN '71+'
            ELSE 'Unknown'
        END AS AgeGroup
    FROM customer
) AS AgeCategories
GROUP BY AgeGroup
ORDER BY Count DESC;
```

Table 5: 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)

```
SELECT
    career,
    COUNT(*) AS CareerCount,
    CONCAT(ROUND((COUNT(*) * 100.0) / (SELECT COUNT(*) FROM customer), 2), '%') AS Percentage
FROM
    customer
GROUP BY
    career
ORDER BY
    CareerCount DESC
LIMIT 10;
```

Table 6: 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%
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)

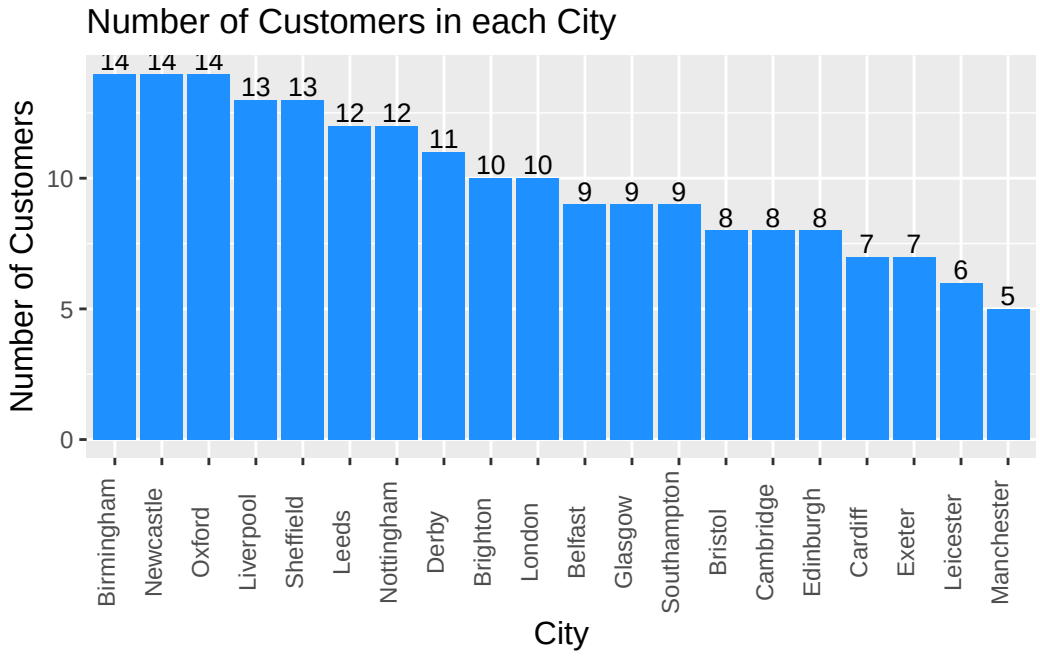
```sql
SELECT
    address_city,
    COUNT(*) AS CityCount,
    CONCAT(ROUND((COUNT(*) * 100.0) / (SELECT COUNT(*) FROM customer), 2), '%') AS Percentage
FROM
    customer
GROUP BY
    address_city
ORDER BY
    CityCount DESC
LIMIT 10;
```

Table 7: 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%

address_city	CityCount	Percentage

```r
# Group by city and count the number of customer in each city
customer_city_count <- customer %>%
  group_by(address_city) %>%
  summarise(number_of_customers = n()) %>%
  arrange(desc(number_of_customers))

# Specify the dimensions of the plot
width <- 12
height <- 8

# Use ggplot to create a bar chart showing the number of customers in each city
g_customer <- ggplot(customer_city_count,
                     aes(x = reorder(address_city, -number_of_customers),
                         y = number_of_customers)) +
  geom_col(fill = "dodgerblue") +
  geom_text(aes(label = number_of_customers),
            position = position_dodge(width = 0.9), vjust = -0.2,
            size = 3.5) +
  theme(axis.text.x = element_text(angle = 90, hjust = 0.5, vjust = 0),
        axis.title = element_text(size = 12)) +
  labs(title = "Number of Customers in each City",
       x = "City",
       y = "Number of Customers")
print(g_customer)
```




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

# Save the plot with the dynamic filename
ggsave(filename, plot = g_customer, width = width, height = height)
```

E. The Current Customer Referral Rate

```
SELECT
    COUNT(CASE WHEN referred_by != '' AND referred_by IS NOT NULL THEN 1 END) AS Customer_wi
    COUNT(*) AS total_customer,
    CONCAT(ROUND((COUNT(CASE WHEN referred_by != '' AND referred_by IS NOT NULL THEN 1 END) *
FROM
    customer;
```

Table 8: 1 records

Customer_with_Referral	total_customer	referral_rate
74	199	37.19%

Customer Analysis

The gender of the customers is evenly distributed, with 55% males and 45% females. 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.

The career distribution states that customers are employeed in diverse industries and have different job positions.

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

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)

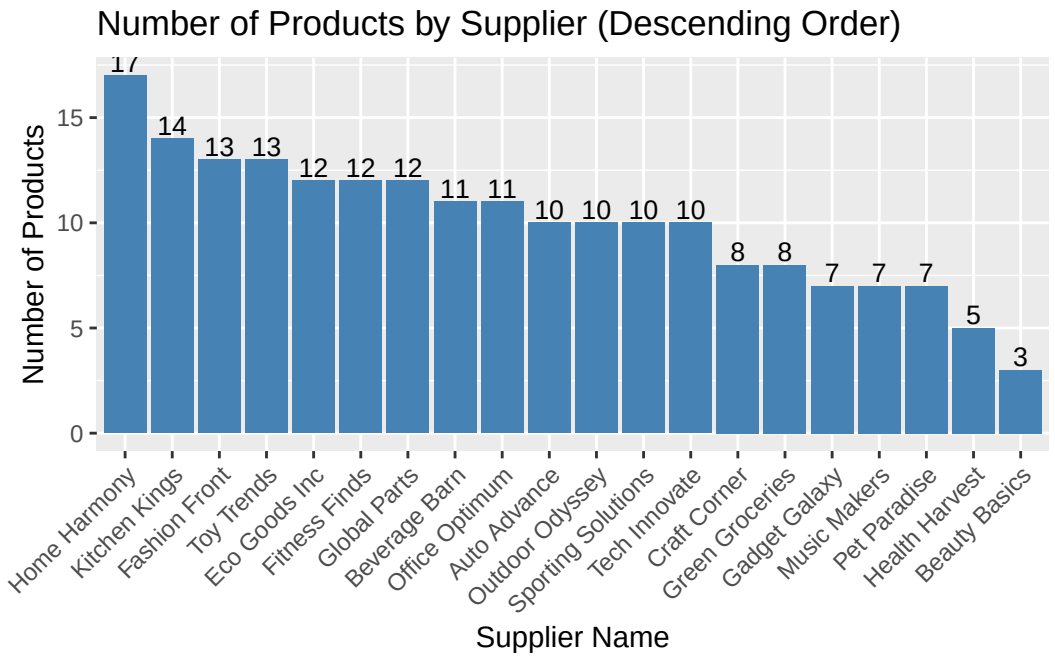
```sql
SELECT
    product_name, review_score
FROM
    product
ORDER BY
    review_score DESC
LIMIT 10;
```

Table 9: 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
Book Penguin Classics Pride and Prejudice	4.89
Laptop HP Spectre x360	4.89
Lipstick NARS Orgasm	4.87
Drill Milwaukee Brushless Cordless Drill	4.84
Lipstick Revlon Super Lustrous Lipstick	4.82
Sofa IKEA Kivik	4.82

B. The Number of Products supplied by Different Suppliers

```r
# Perform an inner join to combine 'product' with 'supplier' on 'supplier_id'
joint_supplier_product <- inner_join(product, supplier, by = "supplier_id")

# Group by supplier_name and count the number of products for each supplier
product_count_by_supplier <- joint_supplier_product %>%
  group_by(supplier_name) %>%
  summarise(number_of_products = n())

# Specify the dimensions of the plot
width <- 12
height <- 8

# Use ggplot to create a bar chart showing the number of products for each supplier
g_supplier <- ggplot(product_count_by_supplier, aes(x = reorder(supplier_name, -number_of_pro
  geom_col(fill = "steelblue") +
  geom_text(aes(label = number_of_products), position = position_dodge(width = 0.9), vjust =
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  labs(title = "Number of Products by Supplier (Descending Order)",
       x = "Supplier Name",
       y = "Number of Products")
print(g_supplier)
```



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

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

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

Table 10: 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)

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

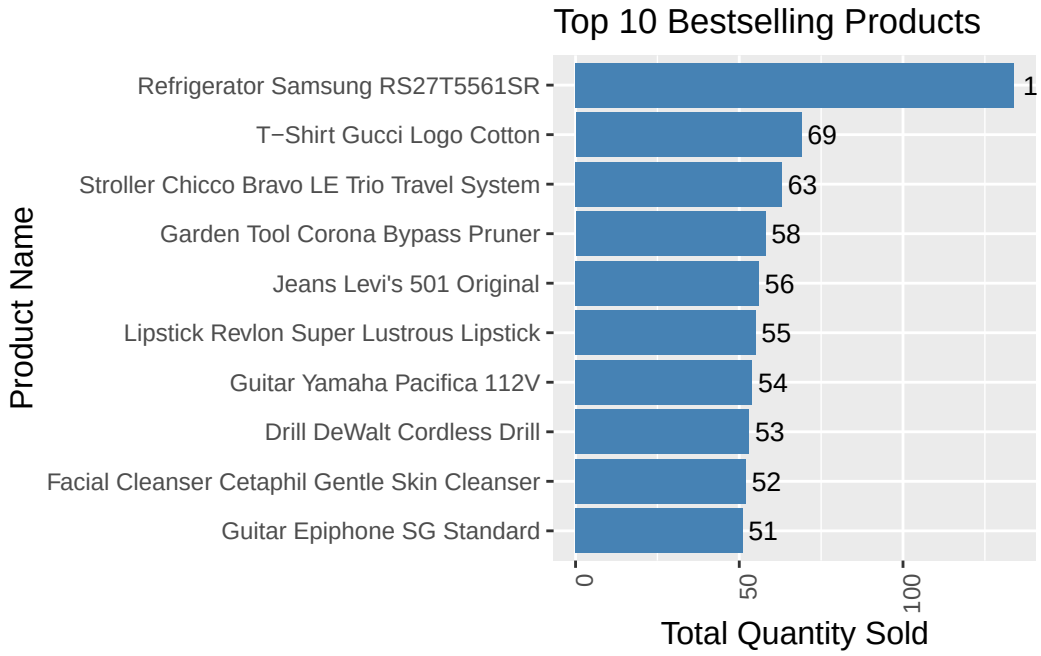
Table 11: 5 records

supplier_name	average_review_score
Fitness Finds	2.96
Auto Advance	2.99

supplier_name	average_review_score
Craft Corner	3.02
Kitchen Kings	3.04
Health Harvest	3.05

E. The Top 10 Best Selling Products

```r
# Perform an inner join to combine 'orders' with 'product' on 'product_id'
joint_order_product <- inner_join(orders, product, by = "product_id")

# Calculate the total quantity sold for each product
product_sales_volume <- joint_order_product %>%
  group_by(product_name) %>%
  summarise(total_quantity_sold = sum(quantity)) # Assuming 'quantity' exists in your orders
# Processing the text of product name
product_sales_volume$product_name <- iconv(product_sales_volume$product_name, "UTF-8", "ASCI

# Choose only the top 10 products based on total quantity sold
top_product_sales_volume <- product_sales_volume %>%
  arrange(desc(total_quantity_sold)) %>%
  slice_head(n = 10)

# Specify the dimensions of the plot
width <- 12
height <- 8

# Use ggplot to create a bar chart showing the total quantity sold for each product
g_topproduct <- ggplot(top_product_sales_volume, aes(x = reorder(product_name, total_quantity
  geom_col(fill = "steelblue") +
  geom_text(aes(label = total_quantity_sold), position = position_dodge(width = 0.9), hjust =
  coord_flip() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1),
        axis.title = element_text(size = 12)) +
  labs(title = "Top 10 Bestselling Products",
       x = "Product Name",
       y = "Total Quantity Sold")
print(g_topproduct)
```




```r
# Dynamically generate filename with current date and time
filename <- paste0("top10_products_by_quantity_", format(Sys.time(), "%Y%m%d_%H%M%S"), ".png"

# Save the plot with the dynamic filename
ggsave(filename, plot = g_topproduct, width = width, height = height)
```

Supplier Analysis

It shows that the products are supplied by various suppliers, ranging from household items, fashion stores, and toy wholesalers to fitness equipment suppliers. 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.

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

```sql
SELECT
    COUNT(CASE WHEN refund_status = 'yes' THEN 1 END) AS refund_orders,
    COUNT(*) AS total_orders,
    CONCAT(ROUND((COUNT(CASE WHEN refund_status = 'yes' THEN 1 END) * 1.0 / COUNT(*)) * 100,
FROM (
    SELECT DISTINCT order_id, refund_status
    FROM orders
) AS unique_orders;
```

Table 12: 1 records

refund_orders	total_orders	refund_rate_percentage
151	387	39.02%

Order Refund Analysis

The order refund rate is 39%, higher than the IMRG's 2020 (benchmarking data); the average e-commerce business saw a 15% return rate. It is a signal for customer reduction since when customers see high refund rates they may be less likely to purchase the products.

Appendices

We asked AI how we can use cat() function in our code. We used cat() in R script to check how the validation and insertion process is going.

You

do you think that I can implement in my code somewhere the same cat() function to track the progress of validation of new customers data?

ChatGPT

Absolutely, using `cat()` function to log progress and key information during the validation and data processing steps is an excellent way to track progress and debug your script. It can provide you with immediate feedback in the R console about what your script is doing, which can be particularly helpful for long-running scripts or when processing large datasets.

Here are some places you might consider adding `cat()` statements in your script to log progress and important milestones:

1. Start and End of Processing Each File:

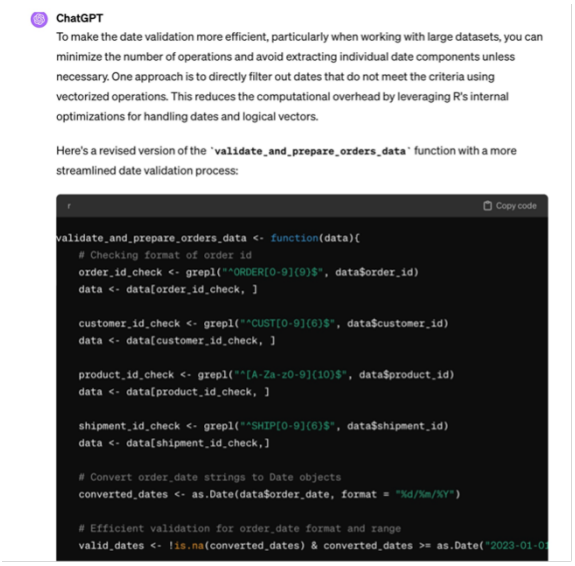
r Copy code

```r
for (file_path in customer_file_paths) {
  cat("Starting processing file:", file_path, "\n")
  # Read and process the file...

  cat("Finished processing file:", file_path, "\n")
}
```

2. Before and After Validation:

Inside the loop where you're iterating through each row of the customer data:

We asked AI how to make our validation code for tables more efficient, before it was checking separately which was inefficient. After that prompt, we updated our validation parts and created functions for each of table validations.



ChatGPT

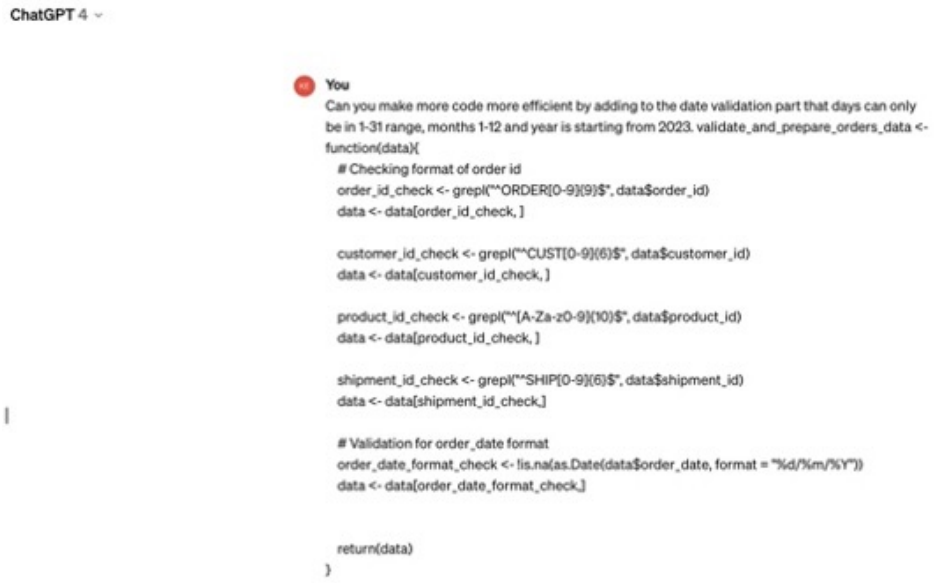
To make the date validation more efficient, particularly when working with large datasets, you can minimize the number of operations and avoid extracting individual date components unless necessary. One approach is to directly filter out dates that do not meet the criteria using vectorized operations. This reduces the computational overhead by leveraging R's internal optimizations for handling dates and logical vectors.

Here's a revised version of the `validate_and_prepare_orders_data` function with a more streamlined date validation process:

Copy code

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

  customer_id_check <- grepl("^CUST[0-9]{6}$", data$customer_id)
  data <- data[customer_id_check, ]

  product_id_check <- grepl("^[A-Za-z0-9]{10}$", data$product_id)
  data <- data[product_id_check, ]

  shipment_id_check <- grepl("^SHIP[0-9]{6}$", data$shipment_id)
  data <- data[shipment_id_check,]

  # Convert order_date strings to Date objects
  converted_dates <- as.Date(data$order_date, format = "%d/%m/%Y")

  # Efficient validation for order_date format and range
  valid_dates <- !is.na(converted_dates) & converted_dates >= as.Date("2023-01-01
```

We asked AI to improve my code for orders table validation function, so I can add one more validation check that days are in range of 1-31, months in 1-12 and years starts from 2023 (established of the e-commerce shop). I used the provided code in orders table validation part/



ChatGPT 4

You

Can you make more code more efficient by adding to the date validation part that days can only be in 1-31 range, months 1-12 and year is starting from 2023. validate_and_prepare_orders_data <- function(data){

```
  # Checking format of order id
  order_id_check <- grepl("^ORDER[0-9]{9}$", data$order_id)
  data <- data[order_id_check, ]

  customer_id_check <- grepl("^CUST[0-9]{6}$", data$customer_id)
  data <- data[customer_id_check, ]

  product_id_check <- grepl("^[A-Za-z0-9]{10}$", data$product_id)
  data <- data[product_id_check, ]

  shipment_id_check <- grepl("^SHIP[0-9]{6}$", data$shipment_id)
  data <- data[shipment_id_check,]

  # Validation for order_date format
  order_date_format_check <- !is.na(as.Date(data$order_date, format = "%d/%m/%Y"))
  data <- data[order_date_format_check,]


  return(data)
}
```

This is the reply from AI, we provided him my code for products table from R script and asked him how he can increase efficiency of the code. He suggested to use batch checking method to improve it. Based on that we constructed more efficient code for creating a batch of existing 'product_id' in database and then compare new upcoming primary keys with this vector.

Optimizing Record Existence Checks

Your loop through each product record to check its existence in the database before insertion is effective but could be optimized, especially for large datasets:

1. **Batch Checking**: Instead of individual queries for each `product_id`, consider fetching existing `product_id`s once into a vector and then checking existence in R. This reduces database queries and can significantly speed up the process.
2. **Temporary Dataframe Uniqueness**: The check for uniqueness in `product_possible_data` is done in an efficient manner. However, for very large dataframes, consider alternatives like `dplyr::distinct()` or maintaining a set of seen keys to avoid the performance cost of subsetting on each iteration.

We provided AI my chunk for products table and asked him to help me integrate referential integrity. We used and adjusted provided code in Product table chunk, the code verifies that in new potential data foreign keys are valid and are presented in associated tables.

ChatGPT

To implement referential integrity checks in your script for the `product` table, you'll need to verify that each `product` record's `category_id`, `supplier_id`, and `promotion_id` exist in their respective reference tables (`category`, `supplier`, `promotion`) before attempting to insert the record into the `product` table. This process ensures that all foreign keys in the `product` table point to existing records, maintaining the integrity of your database relationships.

Here's a way to integrate referential integrity checks into your `validate_and_prepare_product_data` function:

Enhanced Validation Function with Referential Integrity Checks

r Copy code

```r
validate_and_prepare_product_data <- function(data, connection) {
  # Validation for product ID
  product_id_check <- grepl("^[A-Za-z0-9]{10}$", data$product_id)
  data <- data[product_id_check, ]

  # Validation for review_score range
  data <- data[data$review_score >= 1 & data$review_score <= 5, ]

  # Fetch existing IDs from reference tables
  valid_category_ids <- dbGetQuery(connection, "SELECT category_id FROM category")$ca
  valid_supplier_ids <- dbGetQuery(connection, "SELECT supplier_id FROM supplier")$su
  valid_promotion_ids <- c(dbGetQuery(connection, "SELECT promotion_id FROM promotion

  # Referential integrity checks
  data <- data[data$category_id %in% valid_category_ids, ]
  data <- data[data$supplier_id %in% valid_supplier_ids, ]
  data <- data[is.na(data$promotion_id) | data$promotion_id %in% valid_promotion_ids,

  return(data)
```

We asked to improve validation function of product table to incorporate more validation constrains on quantity stock, quantity supplied and price. The code was used in validation function of product table.

You

Help me to produce a validation code for 'quantity_stock', "quantity_supplied" that they are non-negative and integer number (fractional numbers are not allowed). also, 'price' should be only positive values. And 'product_name' cannot be empty

< 2 / 2 >

ChatGPT

To extend the `validate_and_prepare_product_data` function with additional validations for `quantity_stock`, `quantity_supplied`, `price`, and ensuring `product_name` is not empty, you can incorporate these checks using logical indexing in R. Here's how you might update your function to include these validations:

Updated Validation Function

r — Copy code

```r
validate_and_prepare_product_data <- function(data, connection) {
  # Initial validation for product ID and review_score range
  product_id_check <- grepl("^[A-Za-z0-9]{10}$", data$product_id)
  data <- data[product_id_check, ]
  data <- data[data$review_score >= 1 & data$review_score <= 5, ]

  # Fetch existing IDs from reference tables for referential integrity checks
  valid_category_ids <- dbGetQuery(connection, "SELECT category_id FROM category")$ca
  valid_supplier_ids <- dbGetQuery(connection, "SELECT supplier_id FROM supplier")$su
  valid_promotion_ids <- c(dbGetQuery(connection, "SELECT promotion_id FROM promotion

  # Referential integrity checks
  data <- data[data$category_id %in% valid_category_ids, ]
  data <- data[data$supplier_id %in% valid_supplier_ids, ]
  data <- data[is.na(data$promotion_id) | data$promotion_id %in% valid_promotion_ids,

  # Validation for non-negative and integer quantity_stock and quantity_supplied
```

We asked AI about TryCatch function and how it can be used in our code. We used it R sciprts to handle errors and warnings.

You

do I need to incorporate tryCatch in my code above?

ChatGPT

Incorporating `tryCatch` into your script can significantly enhance its robustness, especially when dealing with external resources like databases or files where unexpected issues might arise. `tryCatch` in R is a mechanism for handling errors and warnings gracefully, allowing your script to continue running or to perform specific actions when encountering issues, rather than halting execution unexpectedly.

Here are a few scenarios in your script where `tryCatch` could be particularly beneficial:

1. Database Queries

When fetching existing `product_id`s or performing any database operation, there's always a chance of encountering database connection issues, query syntax errors, or timeouts. Wrapping these operations in `tryCatch` blocks can help handle such errors more gracefully.

r Copy code

```r
existing_product_ids <- tryCatch({
    dbGetQuery(connection, "SELECT product_id FROM product")$product_id
}, error = function(e) {
    cat("Error fetching existing product IDs: ", e$message, "\n")
    NULL  # Return NULL or an appropriate value to indicate failure
})
```

2. Reading Files

Reading files with `readr::read_csv` could fail due to issues like file access permissions, incorrect file paths, or malformed CSV content. Using `t↓atch` here allows you to log the error and skip the problematic file without stopping the script.

We asked Chatgpt to use data integrity checks using hashing. Originally we wanted to verify the data integrity before and after loading for the loading data frame. However, due to data type changes and other possible issues they were not validated even if data frames' values were similar.

ChatGPT

To incorporate data integrity checks using hashing before loading and after loading the validated data into the database, you can use the `digest` package in R. Here's how you can modify your script to include these steps:

Step 1: Install and Load the `digest` Package

If you haven't already, install the `digest` package and then load it in your script. This package provides functions for creating hash digests of arbitrary R objects.

r Copy code

```r
if (!requireNamespace("digest", quietly = TRUE)) {
  install.packages("digest")
}
library(digest)
```

Step 2: Compute Hash Before Validation and Loading

After the validation function but before the data loading into the database, compute a hash of `product_possible_data` to capture its state post-validation but pre-loading:

r Copy code

```r
# Ensure product_possible_data is sorted by a consistent key to ensure identical orde
product_possible_data <- product_possible_data[order(product_possible_data$product_ic

# Compute hash of product_possible_data after validation but before loading
pre_load_hash <- digest(product_possible_data, algo = "md5")
cat("Pre-load hash of product_possible_data: ", pre_load_hash, "\n")
```

This prompt helped us to construct the code that is used to hash each row of validated data before loading and after loading. It is done to ensure that loading data and loaded data are the same and was not corrupted during the loading process. The similar codes was used in shipment, orders and product tables data integrity checks.

You

it is not working anyway. Where else in my code I can use hashing to ensure data integrity?

ChatGPT

If directly comparing pre-load and post-load data using hashes isn't yielding the expected results due to subtle discrepancies that are hard to control or identify (like minor floating-point differences, data type conversions, or class attribute issues), you might consider alternative approaches to ensure data integrity. Here are some strategies focusing on hashing that might work better in your context:

Hash Individual Records Before Loading and After Retrieval

Instead of hashing the entire dataset at once, consider hashing individual records (rows) before loading them into the database and then hashing the retrieved records again to compare. This approach can help pinpoint exactly where discrepancies occur.

1. **Hash Each Record Before Loading**: Iterate over each row in `product_possible_data`, convert it to a character string or a consistent representation, hash it, and store the hash in a new vector or column.
2. **Load Data into the Database**: Proceed with loading your data into the database as you normally would.
3. **Retrieve and Hash Each Record After Loading**: After loading, retrieve the data (ensuring the same order or using identifiers to match records), hash each retrieved record in the same manner as before, and compare individual hashes.

Implementing Individual Record Hashing

r Copy code

```r
# Example for hashing individual records before loading
pre_load_hashes <- sapply(1:nrow(product_possible_data), function(i) {
  record <- as.character(unlist(product_possible_data[i, ]))
  digest(paste(record, collapse = "|"), algo = "md5")
```

We prompted Chatgpt to improve code for orders table using batching processing. We provided our initial code and then based on AI suggestions and iterative prompts process, we improved code to be more efficient and used it in orders table chunks.

You

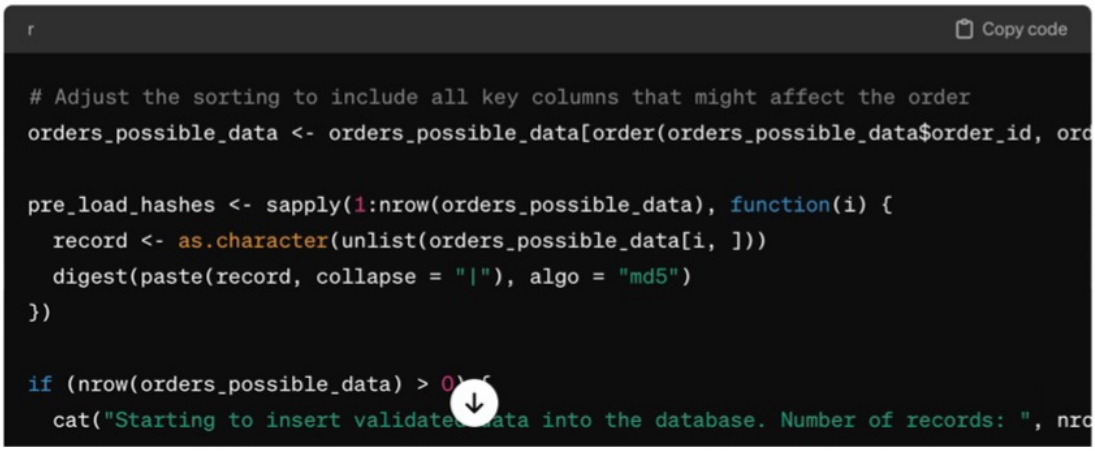
it seems that it is sorted by order id but the values with the same order then have different orders in possible and retrieved data.

ChatGPT

If the sorted `orders_possible_data` and the retrieved data from the database do not maintain the same order for rows with the same `order_id` due to other varying fields, you need a more stable sorting mechanism. One way to address this issue is to sort the data by multiple columns that collectively ensure a unique ordering for every row. This approach would keep the order of rows consistent between the pre-load and post-load datasets.

Given the structure of your `orders` data, it's likely that sorting by `order_id` alone isn't sufficient because each `order_id` could be associated with multiple product, customer, and shipment IDs. Sorting by all these fields should ensure that the order of the data remains consistent between the pre-load phase and the post-load phase.

Here's how you can adjust the sorting mechanism to sort by multiple columns, ensuring consistent order in both `orders_possible_data` and the retrieved data:



We prompted AI to help us implement the code for data integrity check for orders table code. After iterative process of prompting and adjusting the code, we implemented data integrity check. So, the code now creates a hash number for validated rows that are ready to be included to data base. After that it retrieves the same loaded data and creates a hash for them. After that it is being compared.

ChatGPT

Your current script processes each order record individually, which is straightforward but might not be the most efficient way, especially if there are many records to process. Batch processing can significantly improve performance by reducing the number of database queries and transactions. Here's how you can modify your code to use batch processing:

Step 1: Collect All IDs First and Perform Bulk Checks

Instead of checking the existence of each ID in the database one by one, collect all unique IDs first and then check their existence in bulk. This can drastically reduce the number of database queries.

```r
# Example for batch checking product IDs
unique_product_ids <- unique(orders_data$product_id)
existing_product_ids <- check_id_exists(connection, "product", "product_id", unique_p

# Similarly for customer_ids and shipment_ids
```

Step 2: Validate Referential Integrity in Batches

After getting the existing IDs in bulk, you can check each record against these lists to validate referential integrity without querying the database each time.

```r
# Example modification of validate_referential_integrity function to work with pre-fe
validate_referential_integrity_batch <- function(new_record, existing_product_ids, ex
  product_exists <- new_record$product_id %in% existing_product_ids
  customer_exists <- new_record$customer_id %in% existing_customer_ids
  shipment_exists <- new_record$shipment_id %in% existing_shipment_ids
```

In this prompt we asked AI to help improve our shipment table code. We wanted to improve checking for existing primary keys in table, so it will query the database once and store it as a vector. It helped to create more efficient code that queries the database once instead of iterative one by one checking.

You

Improve my code for shipment table. So instead of iterative process of querying the existing database to check for uniqueness of primary id. So, fetch existing in current database primary keys to increase performance.

```
validate_and_prepare_shipment_data <- function(data) {
    # Validation for shipment ID
    shipment_id_check <- grepl("^SHIP[0-9]{6}$", data$shipment_id)
    data <- data[shipment_id_check,]

    # Validation for shipment_date and delivery_date format
    date_format_check <- !is.na(as.Date(data$shipment_date, format = "%d/%m/%Y")) &
     !is.na(as.Date(data$delivery_date, format = "%d/%m/%Y"))

    data <- data[date_format_check,]

    #Validation for loogical order of shipment and delivery dates
    logical_date_order_check <- as.Date(data$shipment_date) < as.Date(data$delivery_date)
    data <- data[logical_date_order_check,]
    return(data)
  }

  # Fetch existing shipment IDs from the database

  shipment_file_paths <- list.files(path = "data_upload", pattern = "shipment.*\\.csv$", full.names =
TRUE)

  # Define the primary key column for the shipment table
  shipment_primary_key <- "shipment_id"

  #Initialising empty data frame
  shipment_possible_data <- data.frame()

  # Read each shipment CSV file and chec. . the existence of the primary key in the database
before appending
```

ChatGPT

Improving your code for handling the `shipment` table by fetching existing primary keys from the database in one go, rather than checking for uniqueness iteratively for each row, is a great idea for enhancing performance. This approach reduces the number of database queries, thereby minimizing I/O overhead and speeding up the process. Here's how you can modify your code to incorporate this improvement:

1. **Fetch Existing Primary Keys**: Before processing the CSV files, query the database to fetch all existing `shipment_id`s. Store these IDs in a vector or similar data structure for quick lookup.
2. **Check for Uniqueness Against the Fetched List**: As you process each new record, check if its `shipment_id` is in the list of fetched IDs to determine its uniqueness, instead of querying the database for each record.

Here's the revised code incorporating these improvements:

r Copy code

```r
library(readr)
library(RSQLite)
library(dplyr)

validate_and_prepare_shipment_data <- function(data) {
  # Validation for shipment ID
  shipment_id_check <- grepl("^SHIP[0-9]{6}$", data$shipment_id)
  data <- data[shipment_id_check, ]

  # Validation for shipment_date and delivery_date format
  date_format_check <- !is.na(as.Date(data$shipment_date, format = "%d/%m/%Y")) &
    !is.na(as.Date(data$delivery_date, format = "%d/%m/%Y"))
  data <- data[date_format_check, ]
```

```
  data <- data[logical_date_order_check, ]

  return(data)
}

# Fetch existing shipment IDs from the database
existing_shipment_ids <- dbGetQuery(connection, "SELECT shipment_id FROM shipment")$s

shipment_file_paths <- list.files(path = "data_upload", pattern = "shipment.*\\.csv$"

# Define the primary key column for the shipment table
shipment_primary_key <- "shipment_id"

# Initialising empty data frame
shipment_possible_data <- data.frame()

for (file_path in shipment_file_paths) {
  cat("Starting processing file:", file_path, "\n")

  # Read the current file
  shipment_data <- readr::read_csv(file_path)

  # Filter out records with existing shipment_id in the database
  unique_shipment_data <- shipment_data[!shipment_data[[shipment_primary_key]] %in% e

  # Combine unique records into the possible data frame
  if(nrow(unique_shipment_data) > 0){
    shipment_possible_data <- rbind(shipment_possible_data, unique_shipment_data)
  }

  cat("Finished processing file:", file_path, "\n")
}
```

We also asked AI to improve the code to ensure data integrity before and after loading the validated data to the database. The AI did not provided the usable code, so it took many iterations of prompting and also applying the existing programming language knowledge.

You

Incorporate in my shipment table code the data integrity check before and after loading the table as I did for product table. The example code is provided: Implementing data integrity check for validated and ready to loading data for each row

```
product_possible_data <- product_possible_data[order(product_possible_data$product_id), ]
pre_load_hashes <- sapply(1:nrow(product_possible_data), function(i) {
 record <- as.character(unlist(product_possible_data[i, ]))
 digest(paste(record, collapse = "|"), algo = "md5")
})

# Ingesting into database command
if (nrow(product_possible_data) > 0) {
  cat("Starting to insert validated data into the database. Number of records: ",
nrow(product_possible_data), "\n")

 # Ingesting prepared data to our database
   tryCatch({
   dbWriteTable(connection, name = "product", value = product_possible_data, append = TRUE,
row.names = FALSE)
   cat("Data insertion completed successfully.\n")
   }, error = function(e) {
  cat("Error inserting data into the database: ", e$message, "\n")
  # Additional error handling logic here
     })

  # Fetch the loaded data back for post-load hash comparison
   loaded_product_ids <- sprintf("'%s'", product_possible_data$product_id)
   query <- sprintf("SELECT * FROM product WHERE product_id IN (%s) ORDER BY product_id",
paste(loaded_product_ids, collapse = ", "))
   retrieved_products <- dbGetQuery(connection, query)
  # Creating hashes for retrieved data from db
   post_load_hashes <- sapply(1:nrow(retrieved_products), function(i) {
    record <- as.character(unlist(retrieved_ ↓ ducts[i, ]))
    digest(paste(record, collapse = "|"), algo = "md5")
```

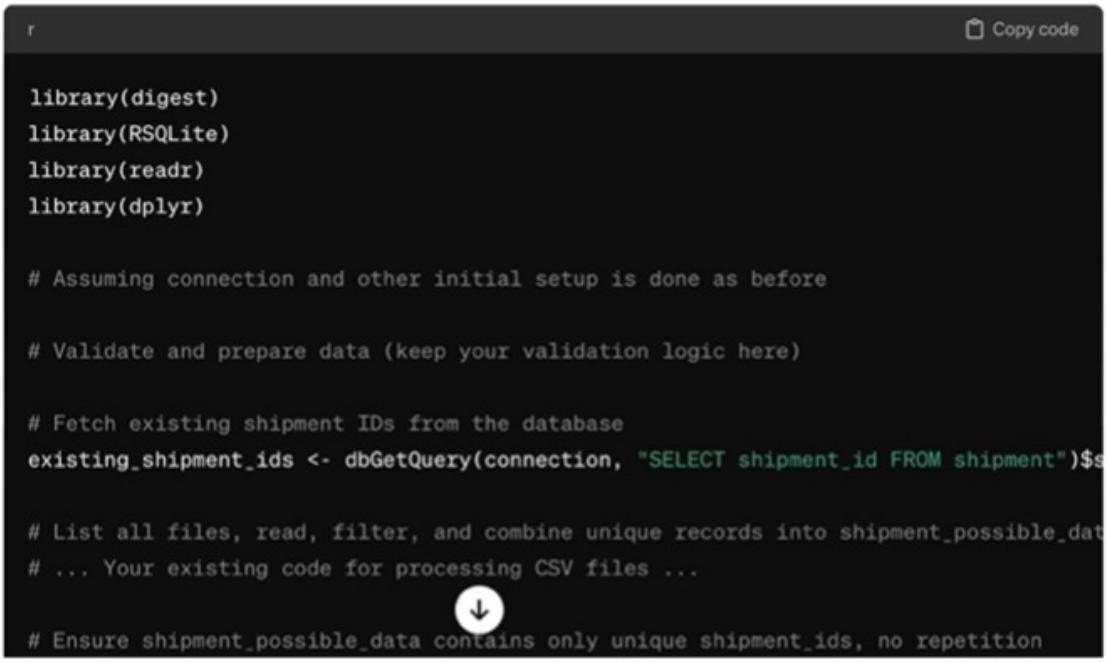
```
pre_load_hashes <- sapply(1:nrow(shipment_possible_data), function(i) {
  record <- as.character(unlist(shipment_possible_data[i, ]))
  digest(paste(record, collapse = "|"), algo = "md5")
})

if (nrow(shipment_possible_data) > 0) {
  cat("Starting to insert validated data into the database. Number of records: ", nro

  # Attempt to insert data into the database
  tryCatch({
    dbWriteTable(connection, name = "shipment", value = shipment_possible_data, apper
    cat("Data insertion completed successfully.\n")
  }, error = function(e) {
    cat("Error inserting data into the database: ", e$message, "\n")
  })

  # Fetch the loaded data back for post-load hash comparison
  loaded_shipment_ids <- sprintf("'%s'", shipment_possible_data$shipment_id)
  query <- sprintf("SELECT * FROM shipment WHERE shipment_id IN (%s) ORDER BY shipmer
  retrieved_shipments <- dbGetQuery(connection, query)

  # Generate post-load hashes
  post_load_hashes <- sapply(1:nrow(retrieved_shipments), function(i) {
    record <- as.character(unlist(retrieved_shipments[i, ]))
    digest(paste(record, collapse = "|"), algo = "md5")
  })

  # Compare hashes
  identical_hashes <- all(pre_load_hashes == post_load_hashes)
  if (identical_hashes) {
    cat("Data integrity verified: All record hashes match.\n")
  } else {
    cat("Data integrity check fail      Record hashes do not match.\n")
  }
```

ChatGPT

To incorporate a data integrity check for the `shipment` table similar to the approach you've taken with the `product` table, we'll follow the same steps: generating pre-load hashes for each row of validated data, inserting the data into the database, retrieving the inserted data, and finally comparing post-load hashes to the pre-load hashes to verify data integrity. Here's how you can adapt your shipment table code to include these steps:

1. **Generate Pre-Load Hashes** for the validated `shipment_possible_data`.
2. **Insert the Data** into the `shipment` table.
3. **Retrieve the Inserted Data** based on the primary keys (`shipment_id`).
4. **Generate Post-Load Hashes** for the retrieved data.
5. **Compare Pre-Load and Post-Load Hashes** to verify data integrity.

Here's the adapted code:



We prompted AI to improve the code cause we noticed that our code was not checking for duplicates of rows inside of the new upcoming files. AI guided us to construct a code that also validates for uniqueness inside of new potential files. The similar code was used in all other tables validation part.

You

okay, but does the code checks if there are multiple new data but they have the same primary key?

ChatGPT

Your current implementation checks for the uniqueness of `shipment_id` (the primary key) against existing IDs in the database but does not account for the possibility of encountering multiple new records with the same `shipment_id` within the batch of data being processed. This scenario could lead to potential duplicates within `shipment_possible_data` before it's inserted into the database, especially if these duplicates span across different CSV files or are present within a single file.

To ensure each `shipment_id` in `shipment_possible_data` is unique before attempting to insert the data into the database, you should add an additional step to filter out duplicates from `shipment_possible_data` after it's been populated from the CSV files but before the validation step. A straightforward way to do this is to use the `distinct()` function from the `dplyr` package, which allows you to retain only unique rows based on specified columns (in this case, `shipment_id`).

Here's how you can modify your code to include this step:

r Copy code

```r
library(dplyr)

# After combining all unique records into shipment_possible_data
shipment_possible_data <- rbind(shipment_possible_data, unique_shipment_data)

# Ensure shipment_possible_data contains only unique shipment_ids
# before proceeding with validation and insertion.
shipment_possible_data <- shipment_possible_data %>% distinct(shipment_id, .keep_all

# Now proceed with validation and ther processing
shipment_possible_data <- validate_and_prepare_shipment_data(shipment_possible_data)
```

We also decided to implement a feature that is going to move the source files after the data is processed. We asked AI if it is possible to do and also he provided a code that could be used to implement this adjustment.

You

I have one more question to you. So, right now all new data files (csv) are being uploaded to 'data_upload' folder. And after loading these files to database, they are still saved in this folder. Is it possible to make a code that after ingestion of data, it transfers the used csv file to another folder?

ChatGPT

Yes, it's definitely possible and a good practice to move processed files to another directory after their data has been successfully ingested into the database. This helps in keeping your working directory organized and avoids re-processing the same files inadvertently. In R, you can use the `file.rename()` function to move files from one directory to another. Here's how you can integrate this into your existing code:

1. **Define Your Destination Directory**: Before the loop that processes each CSV file, define the path to the directory where you want to move the processed files. If this directory does not exist, you should create it.
2. **Move Processed Files**: After a file has been successfully processed and its data has been ingested into the database, use `file.rename()` to move it to the defined directory.

Here's an adapted version of your script with these additions:

r Copy code

```r
# Assuming existing setup and libraries are already loaded

# Define the source and destination directories
source_dir <- "data_upload"
destination_dir <- "data_processed"

# Create the destination directory if it does not exist
if (!dir.exists(destination_dir)) {
```

ChatGPT

If you want every processed file to be uniquely named by appending a timestamp and then moved to the 'data_processed' folder, you can slightly adjust your script to always generate a new filename with a timestamp, regardless of whether the file already exists in the destination directory. This approach ensures that each file is uniquely identified and preserved without overwriting any existing files.

Here's how you can modify your code to achieve this:

r Copy code

```r
for (file_path in shipment_file_paths) {
    cat("Starting processing file:", file_path, "\n")

    # Read the current file
    shipment_data <- readr::read_csv(file_path, show_col_types = FALSE)

    # [Your processing logic here]

    # Construct the destination file path with a timestamp to ensure uniqueness
    file_name <- basename(file_path)
    timestamp <- format(Sys.time(), "%Y%m%d%H%M%S")  # Get current timestamp
    file_ext <- tools::file_ext(file_name)  # Extract file extension
    base_name <- sub(paste0("\\.", file_ext, "$"), "", file_name)  # Remove extension
    new_file_name <- paste0(base_name, "_", timestamp, ".", file_ext)  # Create new f
    dest_file_path <- file.path(destination_dir, new_file_name)  # Construct new dest

    # Move the processed file to the destination directory
    if (file.rename(file_path, dest_file_path)) {
        cat("Successfully moved processed file to:", dest_file_path, "\n")
    } else {
        cat("Failed to move file:", file_path, "\n")
    }
    cat("Finished processing file: ile_path, "\n")
```

You

I want that every file after processing is added with time stamp and moved to 'data processed' folder dest_file_path <- file.path(destination_dir, file_name)

```
    # Check if the processed file already exists in the destination directory
 if (file.exists(dest_file_path)) {
  # Generate a new file name to avoid overwriting
  # For simplicity, append a timestamp or a unique identifier
  timestamp <- format(Sys.time(), "%Y%m%d%H%M%S")
  file_ext <- tools::file_ext(file_name)
  base_name <- sub(paste0("\\.", file_ext, "$"), "", file_name)
  new_file_name <- paste0(base_name, "_", timestamp, ".", file_ext)
  dest_file_path <- file.path(destination_dir, new_file_name)
 }
  # Move the processed file to the destination directory
  if (file.rename(file_path, dest_file_path)) {
   cat("Successfully moved processed file to:", dest_file_path, "\n")
  } else {
   cat("Failed to move file:", file_path, "\n")
   }
  cat("Finished processing file:", file_path, "\n")
 }
```

ChatGPT

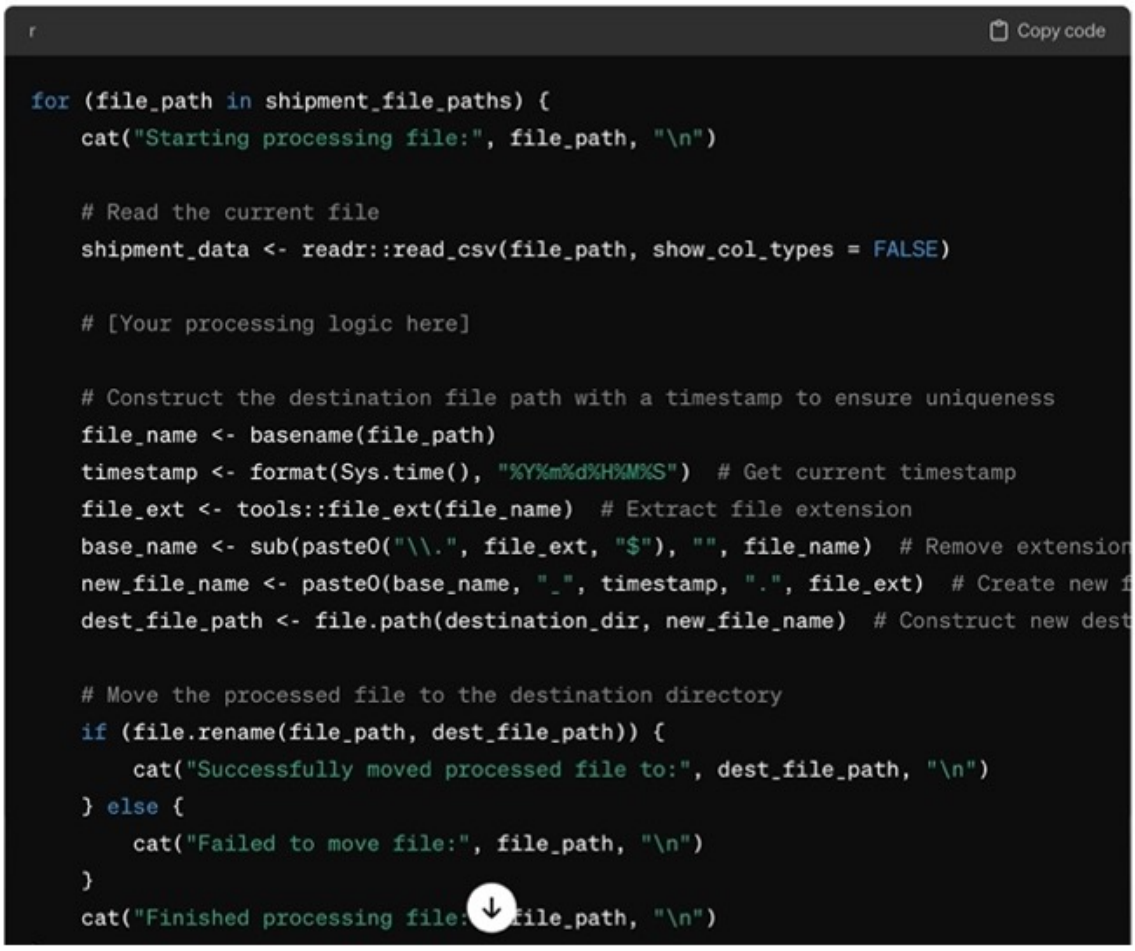
If you want every processed file to be uniquely named by appending a timestamp and then moved to the 'data_processed' folder, you can slightly adjust your script to always generate a new filename with a timestamp, regardless of whether the file already exists in the destination directory. This approach ensures that each file is uniquely identified and preserved without overwriting any existing files.

Here's how you can modify your code to achieve this:

ChatGPT

If you want every processed file to be uniquely named by appending a timestamp and then moved to the 'data_processed' folder, you can slightly adjust your script to always generate a new filename with a timestamp, regardless of whether the file already exists in the destination directory. This approach ensures that each file is uniquely identified and preserved without overwriting any existing files.

Here's how you can modify your code to achieve this:



AI also helped us to solve the issue which we encountered when our workflow failed to run. The problem was in applying mathematical function on non-numeric values. Also, another problem was in product table that if there was no data existing in temporary data frame, it gave the error when tried to order the frame by primary key. Based on the suggestions provided by AI, changes were implemented to restore the run process.

You

look at this error and wait for my code: I want you after that to solve this issue Record with primar key SOH1230566 already exists in the database.
Finished processing file: data_upload/product.csv
Starting validation for new records.
Error in floor(data$quantity_stock) :
non-numeric argument to mathematical function
Calls: validate_and_prepare_product_data -> [-> [.data.frame
Execution halted
Error: Process completed with exit code 1.

ChatGPT

The error message you've shared indicates two primary issues in your code or process:

1. **Duplicate Primary Key Error**: "Record with primary key SOH1230566 already exists in the database." This suggests that your process attempted to insert a record into the database with a primary key (`SOH1230566`) that already exists in the target table. Primary keys must be unique in a database table, so you cannot have two records with the same primary key value.
2. **Non-numeric Argument Error**: "non-numeric argument to mathematical function." This error occurs in the `floor(data$quantity_stock)` function call, indicating that `data$quantity_stock` contains non-numeric values, whereas `floor()` requires numeric input to round down to the nearest whole number.

To address these issues, consider the following solutions:

For the Duplicate Primary Key Error:

- **Check for Existing Records**: Before inserting data into the database, check if a record with the same primary key already exists. If it does, decide whether to skip the insertion, update the existing record, or handle it differently bas on your requirements.
- **Ensure Unique Primary Keys**: If you're generating primary keys, ensure the generation logic

Revised Numeric Validation Check

Before performing the `floor()` operation, convert `quantity_stock` and `quantity_supplied` to numeric values, ensuring any non-numeric entries are handled gracefully. This can be done using `as.numeric()`, which will convert non-numeric values to `NA`. After this conversion, you can proceed with your existing checks, but it's also wise to add a step to handle or filter out `NA` values resulting from non-numeric to numeric conversion.

r Copy code

```r
# Before the validation checks for quantity_stock and quantity_supplied
# Convert to numeric, handling non-numeric values gracefully
data$quantity_stock <- as.numeric(data$quantity_stock)
data$quantity_supplied <- as.numeric(data$quantity_supplied)

# Any conversion issues will result in NA values. You might want to handle these NAs
# For example, you can decide to filter out these NAs or replace them with a default
# Here we're just ensuring the remaining validation checks only apply to non-NA, nume

data <- data[!is.na(data$quantity_stock) & data$quantity_stock >= 0 & (data$quantity_
data <- data[!is.na(data$quantity_supplied) & data$quantity_supplied >= 0 & (data$qua
```

Handling Conversion Errors

When converting to numeric, `as.numeric()` will turn any non-numeric strings to `NA`. This means if `quantity_stock` or `quantity_supplied` contains such values, they will become `NA` after conversion. You should decide how to handle these cases—whether to filter them out or replace them with a default value (e.g., 0) depends on your specific requirements and data integrity policies.

You

> # Implementing data integrity check for validated and ready to loading data for each row
> product_possible_data <- product_possible_data[order(product_possible_data$product_id),]
Error in order(product_possible_data$product_id) :
 argument 1 is not a vector

ChatGPT

The error message you're encountering, `argument 1 is not a vector`, suggests that the attempt to order your `product_possible_data` dataframe by `product_id` is failing because `order(product_possible_data$product_id)` is not receiving a vector as expected. This issue can arise if `product_possible_data` is not properly initialized as a dataframe or if `product_id` does not exist as a column in your dataframe at the point where the ordering is attempted.

There are a couple of ways this issue might be resolved, depending on the state of `product_possible_data` at the time the error occurs:

Ensure `product_possible_data` is Properly Initialized as a Dataframe

Make sure that `product_possible_data` is initialized as a dataframe with the expected columns before you attempt to add rows to it or perform operations like ordering. For example, if `product_possible_data` is expected to have columns matching those in your CSV files, you could initialize it like this:

r Copy code

```r
# Assuming 'product_data_columns' contains the names of the columns expected in 'proc
product_possible_data <- data.frame(matrix(ncol = length(product_data_columns), nrow
colnames(product_possible_data) <- product_data_columns
```

This ensures `product_possible_data` is ataframe with the correct structure.

Check for the Existence of `product_id` Before Ordering

Before attempting to order `product_possible_data` by `product_id`, ensure that `product_id` exists as a column. This can be particularly important if `product_possible_data` could be empty or if its structure might vary:

```r
# Implementing data integrity check for validated and ready to loading data for each
if ("product_id" %in% names(product_possible_data) && nrow(product_possible_data) > 0
  product_possible_data <- product_possible_data[order(product_possible_data$product_
} else {
  cat("product_id column not found or product_possible_data is empty.\n")
}
```

This approach checks that `product_id` is a column within `product_possible_data` and that there are rows to order. If these conditions are not met, it prints a message indicating the issue rather than attempting the operation that would lead to an error.

Summary

- **Initialization**: Ensure `product_possible_data` is correctly initialized as a dataframe with the expected structure from the start.
- **Conditional Ordering**: Perform ordering operations conditionally, based on the presence of the column you're ordering by and the existence of rows in your dataframe.

These steps should help you avoid the error and ensure that your data manipulation operations proceed smoothly.