

# Data Management Group Assignment

DM Group 5

## Contents

1. Database Design and Implementation
2. Data Generation and Management
3. Data Pipeline Generation
4. Data Analysis

## Database Design and Implementation

### 1.1 E-R Diagram Design:

1. E-R Diagram:

We created an Entity-Relationship diagram for an e-commerce store by first identifying a total of 7 key entities namely suppliers, customers, products, product categories, order details, transactions and promotions. Relationships were created to better define the connection between two entities which included provide, belong to, order and make. Attributes were created to detail the information held within every entity. Order relationship was given multiple attributes to better explain the nature of the relationship.

Assumptions:

- The e-commerce store is based in the UK.
- Multiple suppliers provide multiple products.
- Multiple products belong to a single product category.
- Multiple customers can order multiple products.
- Multiple customers can make multiple transactions.
- Multiple products can be contained in a single order detail.
- Multiple promotions can be applied to multiple products.

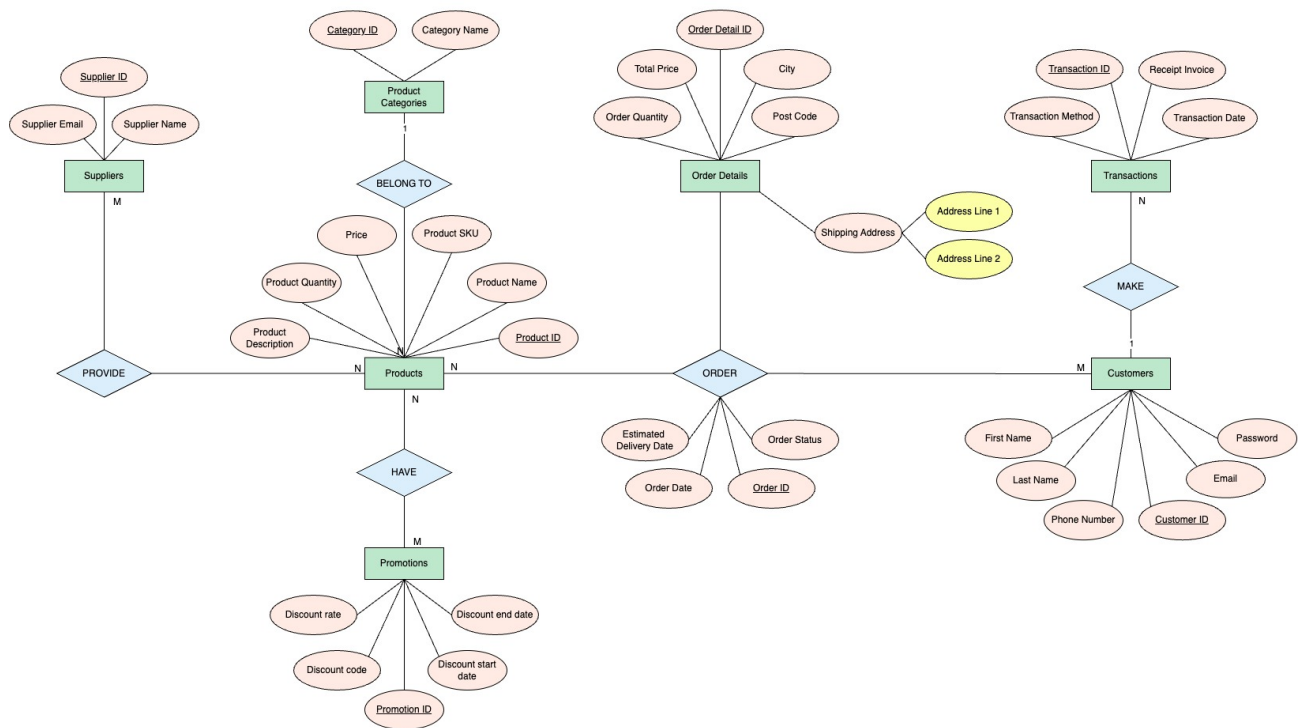


Figure 1: Entity-Relationship Diagram

## 2. Relationship Sets:

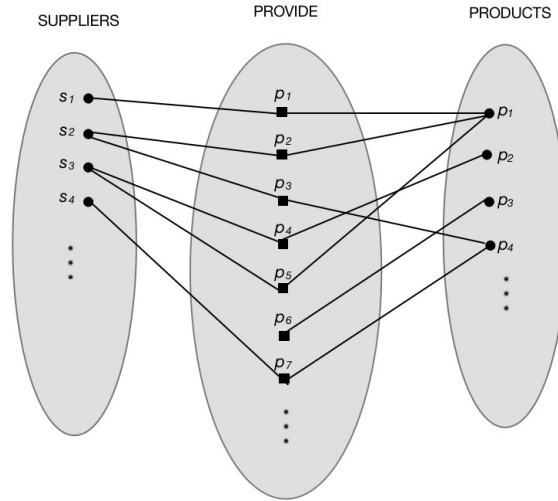


Figure 2: Suppliers PROVIDE products. Many-to-many relationship (M:N).

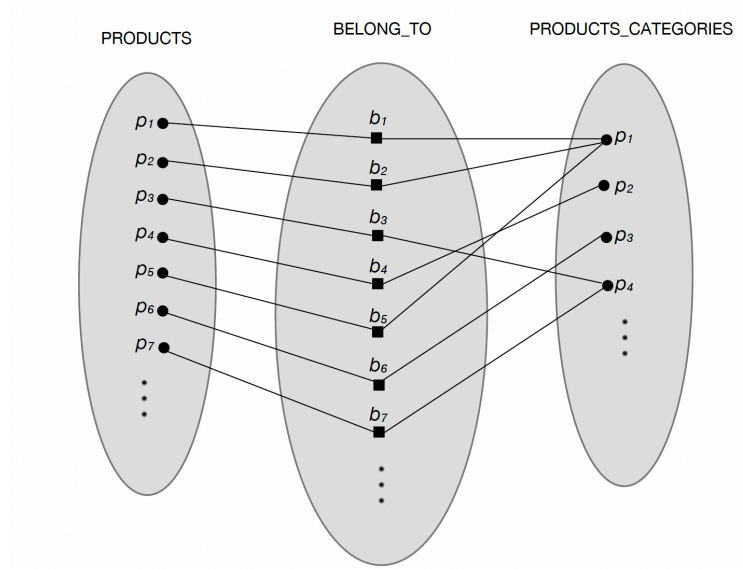


Figure 3: Products BELONG TO Product Categories. Many-to-one relationship (N:1).

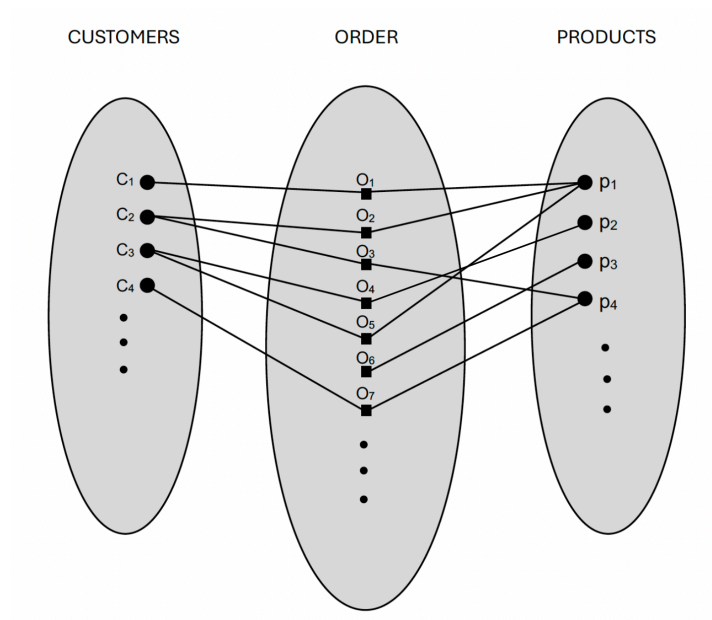


Figure 4: Customers ORDER products. Many-to-many relationship (M:N).

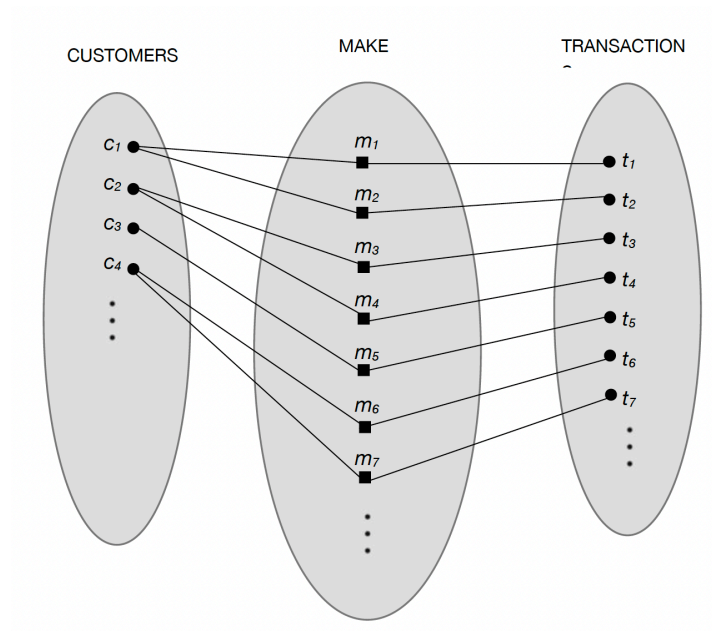


Figure 5: Customers MAKE Transaction. Many-to-many relationship (M:N).

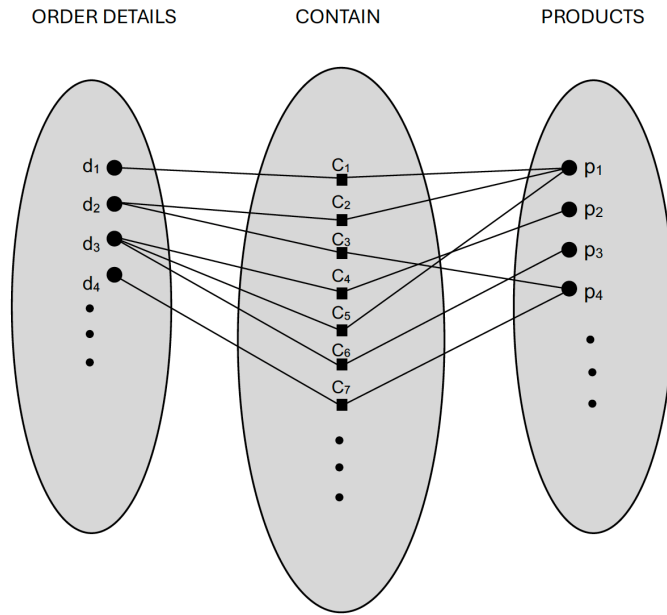


Figure 6: Order Details CONTAIN Products. One-to-many relationship (1:N).

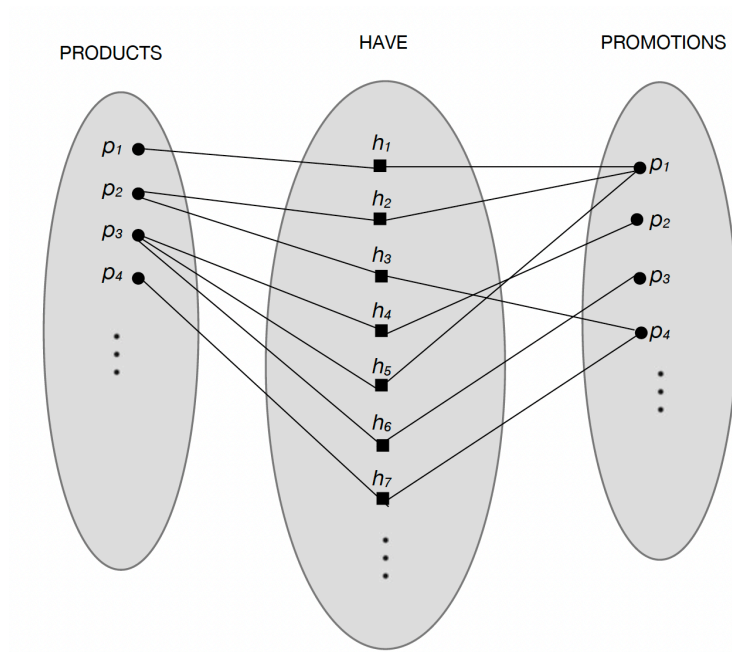


Figure 7: Products HAVE Promotions. Many-to-many relationship (N:M).

## 1.2 SQL Database Schema Creation:

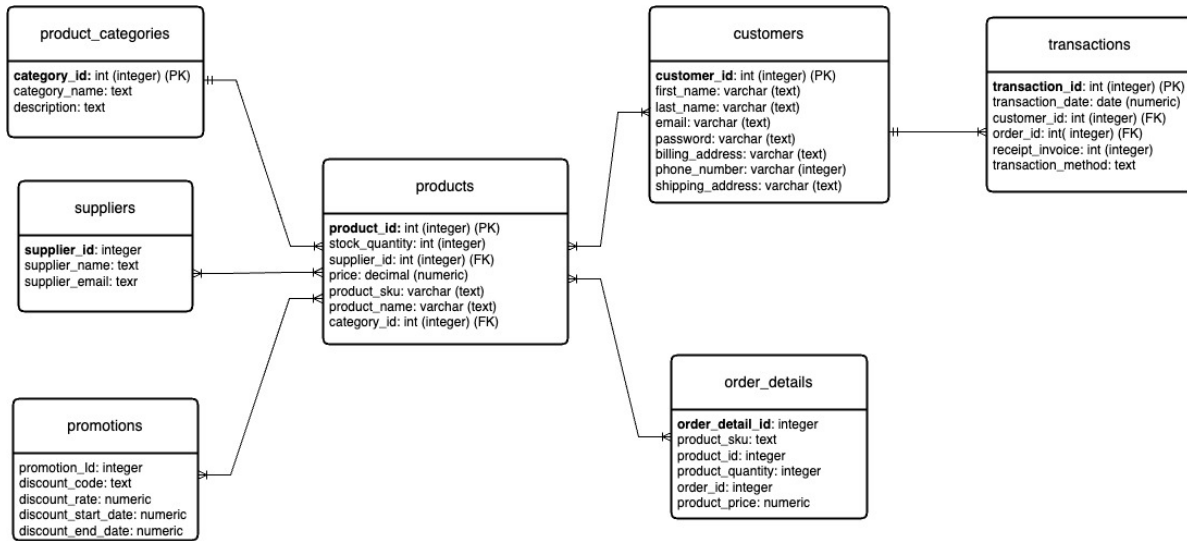


Figure 8: Logical Database Schema

### Logical Schema Table

- Suppliers(supplier\_id, supplier\_name, supplier\_email)
- Products(product\_id, category\_id, supplier\_id, product\_sku, product\_name, price, product\_quantity, product\_description)
- Product Categories(category\_id, category\_name)
- Customers(customer\_id, first\_name, last\_name, email, password, phone\_number)
- Transactions(transaction\_id, customer\_id, order\_id, receipt\_invoice, transaction\_method, transaction\_date)
- Promotions(promotion\_id, discount\_code, discount\_rate, discount\_start\_date, discount\_end\_date)
- Order details(order\_detail\_id, order\_id, product\_id, total\_price, order\_quantity, shipping\_address, city, postcode)
- Order(order\_id, customer\_id, order\_date, estimate\_delivery\_date, order\_status)

Figure 9: Logical Schema Table

Table 1: Entity Attribute Data Dictionary

Entity	Attribute	Description
1. Suppliers	supplier_ID	Primary Key Identifier for each unique supplier
	supplier_name	Name of the supplier
	supplier_email	Email Address of the supplier
2. Products	product_id	Primary Key Identifier for each unique Product
	category_id	Foreign Key Identifier for product categories since product has a many-to-one cardinality (N:1) with product categories
	supplier_id	Foreign Key Identifier for suppliers since product has a many-to-many cardinality (N:M) with suppliers
	product_sku	Unique code assigned to each product for inventory tracking and management
	product_name	Name of the product
	price	Price of the product
	product_quantity	Number of products
	product_description	Short Description of the product
3. Product Categories	category_id	Primary Key Identifier for each unique product category
	category_name	Name of the product category
4. Customers	customer_id	Primary Key Identifier for each unique customer
	first_name	First Name of the customer
	last_name	Last Name of the customer
	email	Email Address of the customer
	password	Account Password of the customer
	phone_number	Phone Number of the customer

Entity	Attribute	Description
5. Transactions	transaction_id	Primary Key Identifier for each unique transaction
	customer_id	Foreign Key Identifier for customers since transactions has a many-to-many cardinality (N:M) with product categories
	order_id	Foreign Key Identifier for Orders since multiple transactions can have multiple orders.
	receipt_invoice	Invoice of the transaction
	transaction_method	Chosen method of transaction
	transaction_date	Date of transaction
6. Promotions	promotion_id	Primary Key Identifier for each unique promotion
	discount_code	Unique code for the discount
	discount_rate	Percentage of discount
	discount_start_date	Start date of the discount
	discount_end_date	End Date of the discount
7. Order details	order_detail_id	Primary Key Identifier for each unique Order detail
	order_id	Foreign Key Identifier for Orders since multiple order details can have multiple orders.
	product_id	Foreign Key Identifier for Products since multiple order details can have multiple order details.
	total_price	Total price of all the orders
	order_quantity	Number of orders
	shipping_address	Composite attribute containing address line 1 and address line 2
	city	Delivery address of the order
	postcode	City location of the order Postal code of the location



Table 2: Relationship Attribute Data Dictionary

Relationship	Attribute	Description
Order	order_id	Primary Key Identifier for each unique Order
	customer_id	Foreign Key Identifier for customers since multiple orders can come from multiple orders.
	order_date	Date of order placement
	estimate_delivery_date	Delivery date of the order
	order_status	Classification of the order depending on its current status i.e., shipped, cancelled or pending

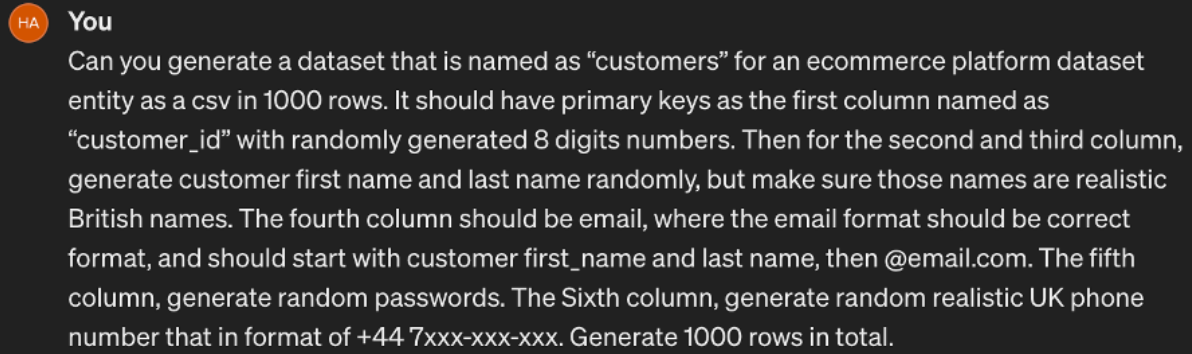
## Data Generation and Management

### 2.1 Synthetic Data Generation:

To generate synthetic data, we utilised ChatGPT. Our approach involved providing ChatGPT with detailed prompts specifying the attributes, relationships, and constraints relevant to each database entity. These prompts ensured that the data generated would reflect realistic e-commerce scenarios while maintaining consistency across entities.

Customers:

We constrained the “Customers” dataset to consist of UK phone numbers, as well as names with matching emails, ensuring a dataset that would closely resemble an actual customer database in the UK.

A screenshot of a chat interface with a dark background. On the left, there is an orange circular icon with the letters 'HA' in white. To its right, the word 'You' is written in white. The main text of the prompt is in a light gray font and describes the requirements for a 'customers' dataset.

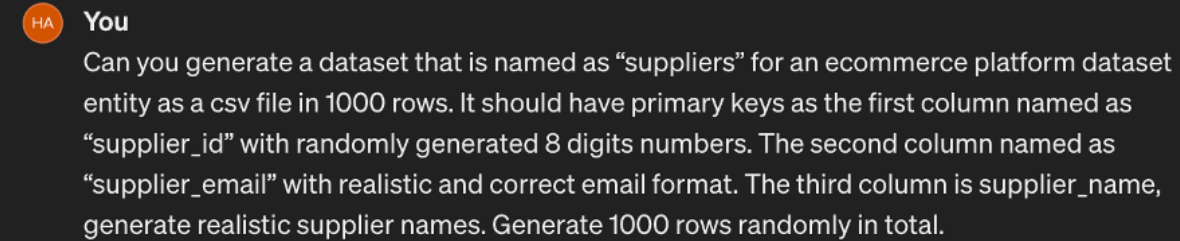
**You**

Can you generate a dataset that is named as “customers” for an ecommerce platform dataset entity as a csv in 1000 rows. It should have primary keys as the first column named as “customer\_id” with randomly generated 8 digits numbers. Then for the second and third column, generate customer first name and last name randomly, but make sure those names are realistic British names. The fourth column should be email, where the email format should be correct format, and should start with customer first\_name and last name, then @email.com. The fifth column, generate random passwords. The Sixth column, generate random realistic UK phone number that in format of +44 7xxx-xxx-xxx. Generate 1000 rows in total.

Figure 10: Customers Prompt.

Suppliers:

In the “Suppliers” data set, we ensured that supplier names matched email formats.

A screenshot of a chat interface with a dark background. On the left, there is an orange circular icon with the letters 'HA' in white. To its right, the word 'You' is written in white. The main text of the prompt is in a light gray font and describes the requirements for a 'suppliers' dataset.

**You**

Can you generate a dataset that is named as “suppliers” for an ecommerce platform dataset entity as a csv file in 1000 rows. It should have primary keys as the first column named as “supplier\_id” with randomly generated 8 digits numbers. The second column named as “supplier\_email” with realistic and correct email format. The third column is supplier\_name, generate realistic supplier names. Generate 1000 rows randomly in total.

Figure 11: Suppliers Prompt.

Product Categories:

The prompt is for the generation of “Product Categories”, with 8 unique category identifiers and corresponding names across 8 entries.

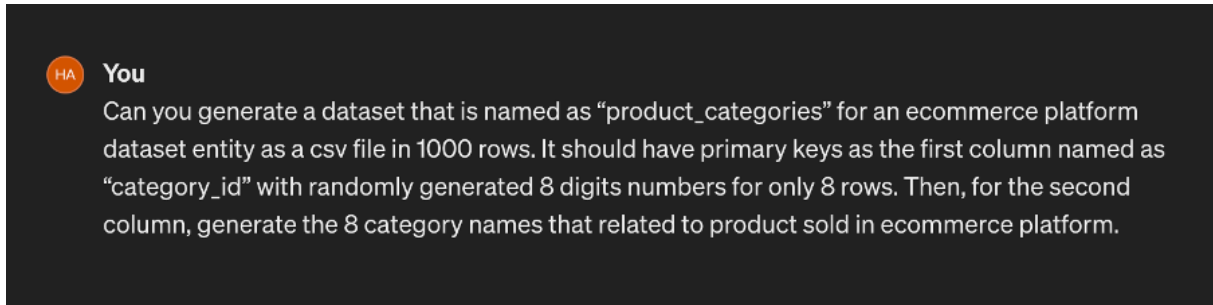


Figure 12: Product Categories Prompt.

Promotions:

For the “Promotions” entity, we focused on generating realistic discount codes and rates, alongside logically constrained start, and end dates for the discounts.

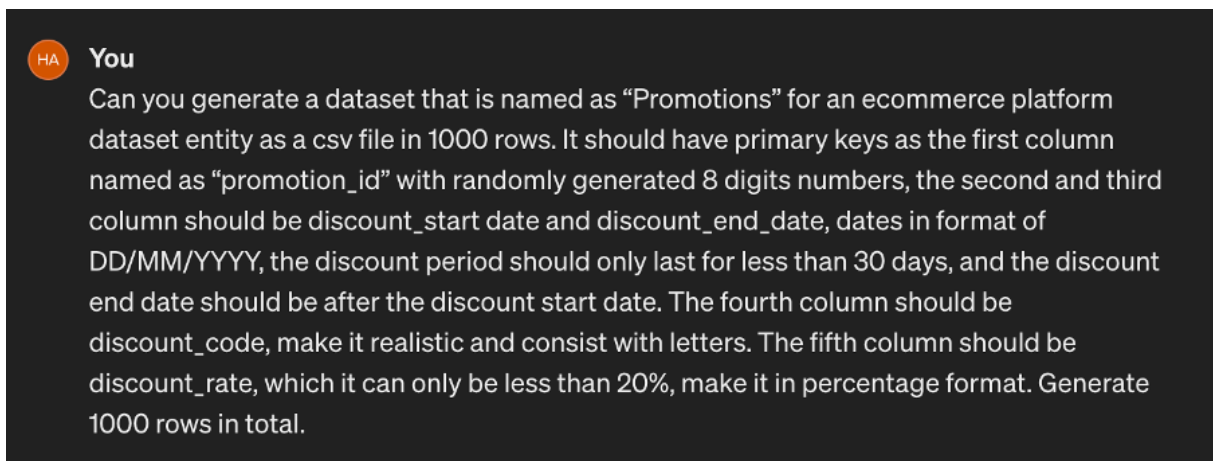


Figure 13: Promotions Prompt.

Products:

The prompt describes generating a “Products” dataset for an e-commerce platform, featuring unique product IDs, associated category and supplier IDs, product names, SKUs, descriptions, prices, and stock quantities, across 1000 rows.



**You**

Can you generate a dataset that is named as “products” for an ecommerce platform dataset entity in csv file in 1000 rows. It should have primary keys as the first column named as “product\_id” with randomly generated 8 digits numbers 1000 rows. In second column, please randomly generate “category\_id” which relates to product\_categories file attached, the third column randomly select and generate “supplier\_id” by using data in the same column name in supplier file. In the fourth column, please generate unique product\_name which is in the category in column 2, also related to product\_categories file. In fifth column, generate random product SKU, sixth column, generate product description of column 4. In seventh column, generate price match to column4 (product name). Last column, provide product quantity (stock quantity) which is realistic.

Figure 14: Products Prompt.

Orders and Order Details:

This prompt outlines the creation of two interconnected datasets, “Orders” and “Order Details,” for an e-commerce platform to maintain consistency. Within the datasets, we focused on imitating a realistic relationship between orders, customers, and products while incorporating accurate order statuses and delivery timelines.



**You**

Please generate 2 files below

Orders

Can you generate a dataset that is named as “orders” for an ecommerce platform dataset entity as csv file in 629 rows. It should have primary keys as the first column named as “order\_id” with randomly generated 8 digits numbers. Second column should be customer\_id, use the customer\_id from the file attached, one order\_id can only related to one customer\_id. The third and fourth column should be estimated\_delivery\_date, and order\_date, where between estimated delivery date and order date, there should be max 6 days differences and order date should be before the estimated delivery date. Dates in format of DD/MM/YYYY. For fifth column order\_status, generate randomly from these three options “shipped”, “pending”, “cancelled” with only 20 values for cancelled. For six column total price should be calculated by the summation of all products price of each order\_id which derived from product\_price multiplied by order\_quantity in order\_details.

Order Details

Generate an “order\_details” dataset for an e-commerce platform database in csv file in 1000 rows. The dataset should have a primary key column named “order\_detail\_id” consisting of randomly generated numbers that are 7 to 8 digits. The “order\_id” column should reference order IDs from the “Orders” data, which can be repeated to reflect multiple items per order. Align each “product\_id” with a specific product from the “Product” dataset, ensuring that the “product\_price” matches the chosen product. Restrict the “order\_quantity” to no more than 4. The dataset should also include realistic UK “shipping\_address”, “city”, and “postcode” fields.



Figure 15: Orders and Order Details Prompt.

Transactions:

In this prompt, we linked each transaction to customer and order details, including distinct invoice numbers and specific payment methods, while aligning transaction dates with their respective order dates.

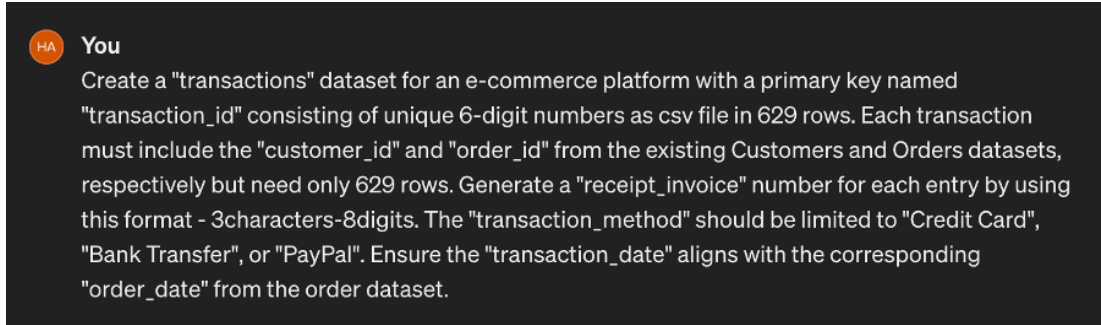


Figure 16: Transactions Prompt.

The prompts enabled us to generate 8 datasets for each entity, setting a strong foundation for robust analysis, given the realistic constraints and patterns put in place.

## 2.2 Data Import and Quality Assurance

### Data Validation:

We employed a series of data validation checks to ensure data quality and assess data integrity. Below are the steps taken as well as their outcomes.

```
library(readr)
library(RSQLite)

#Data Validation
count <- 0
data_files <- list.files("data_uploads/")
```

### Primary Key Validation:

We verified that the first column of each dataset contained unique identifiers, which is essential for primary keys. Our validation checks confirmed that there were no duplicate entries, ensuring the integrity of our data model.

### Duplicate Entries Check:

Our script checked for duplicates in the first column, designed to flag any replication of primary keys. The results confirmed that each dataset had unique identifiers across all entries.

### Missing Values Assessment:

The script aims to detect missing values in any column. The results imply that no missing values were identified across all entries, thus confirming a complete dataset.

```
#Check if the first column of each file is a primary
for (variable in data_files) {
  #to get the filepath of datafiles
  this_filepath <- paste0("data_uploads/", variable)
  #to read the data files
  this_file_contents <- readr::read_csv(this_filepath, col_types = cols())

  number_of_rows <- nrow(this_file_contents)
  print(paste0("Checking for: ", variable))
  #check the number of rows and count of primary keys
  print(paste0(" is ", nrow(unique(this_file_contents[, 1]))==number_of_rows))

  if(nrow(unique(this_file_contents[, 1]))==number_of_rows){
    count <- 1
  }

  #Check duplicates of the first column
  duplicates <- duplicated(this_file_contents[[1]])
  number_of_duplicates <- sum(duplicates)
  print(paste0("Checking for duplicates in '",
               variable, "': ",
               number_of_duplicates, " duplicates found"))

  # the count will be 2 if this validation is passed
  if(number_of_duplicates==0 && count == 1){
    count <- 2
  }
  # Calculate the number of missing values for each column
  missing_values <- colSums(is.na(this_file_contents))

  # Print the results for the current file
  print(paste("Missing values in", basename(this_filepath), ":"))
  print(missing_values)

  if(sum(missing_values == 0) && count == 2){
    count <- 3
  }
}
```

```

[1] "Checking for: customers.csv"
[1] " is TRUE"
[1] "Checking for duplicates in 'customers.csv': 0 duplicates found"
[1] "Missing values in customers.csv :"
```

customer_id	first_name	last_name	email	password	phone
0	0	0	0	0	0

```

[1] "Checking for: customers12.csv"
[1] " is TRUE"
[1] "Checking for duplicates in 'customers12.csv': 0 duplicates found"
[1] "Missing values in customers12.csv :"
```

customer_id	first_name	last_name	email	password	phone
0	0	0	0	0	0

```

[1] "Checking for: order_details.csv"
[1] " is TRUE"
[1] "Checking for duplicates in 'order_details.csv': 0 duplicates found"
[1] "Missing values in order_details.csv :"
```

order_detail_id	order_id	product_id	product_price
0	0	0	0

order_quantity	shipping_Address	city	postcode
0	0	0	0

```

[1] "Checking for: orders.csv"
[1] " is TRUE"
[1] "Checking for duplicates in 'orders.csv': 0 duplicates found"
[1] "Missing values in orders.csv :"
```

order_id	customer_id	estimated_delivery_date
0	0	0

order_date	order_status	total_price
0	0	0

```

[1] "Checking for: product_categories.csv"
[1] " is TRUE"
[1] "Checking for duplicates in 'product_categories.csv': 0 duplicates found"
[1] "Missing values in product_categories.csv :"
```

category_id	category_name
0	0

```

[1] "Checking for: products-new.csv"
[1] " is TRUE"
[1] "Checking for duplicates in 'products-new.csv': 0 duplicates found"
[1] "Missing values in products-new.csv :"
```

product_id	category_id	supplier_id	product_name
0	0	0	0

product_sku	product_description	price	product_quantity
0	0	0	0

```

[1] "Checking for: products.csv"
```

```

[1] " is TRUE"
[1] "Checking for duplicates in 'products.csv': 0 duplicates found"
[1] "Missing values in products.csv :"
      product_id      category_id      supplier_id      product_name
      0              0              0              0
      product_sku product_description      price      product_quantity
      0              0              0              0
[1] "Checking for: promotions.csv"
[1] " is TRUE"
[1] "Checking for duplicates in 'promotions.csv': 0 duplicates found"
[1] "Missing values in promotions.csv :"
      promotion_id discount _start_date      discount_end_date
      0              0              0
      discount_code      discount_rate
      0              0
[1] "Checking for: suppliers.csv"
[1] " is TRUE"
[1] "Checking for duplicates in 'suppliers.csv': 0 duplicates found"
[1] "Missing values in suppliers.csv :"
      supplier_id supplier_email      supplier_name
      0              0              0
[1] "Checking for: transactions.csv"
[1] " is TRUE"
[1] "Checking for duplicates in 'transactions.csv': 0 duplicates found"
[1] "Missing values in transactions.csv :"
      transaction_id      customer_id      order_id      receipt_invoice
      0              0              0              0
transaction_method      transaction_date
      0              0

```

#### Email Format Verification:

We executed a script to ensure that email addresses within our data set follow a certain format of [firstname.lastname@email.com](mailto:firstname.lastname@email.com). The test ensures that all email formats are correct following the specified pattern.

```

#Check email is in format of xxx@xxx.com
# "customers" dataset contains the phone numbers
customer_data_path <- "data_uploads/customers.csv"
customer_data <- read_csv(customer_data_path, col_types = cols())

email_pattern <- "[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$"

```



```
# Assuming 'email_column_name' is the actual name
# of the column containing email addresses
email_column_name <- "email"

# Check if the email format is correct in the specified column
email_format_correct <- grepl(email_pattern,
                              customer_data[[email_column_name]])

# Printing the results
cat("Checking email format for customers dataset:\n")
```

Checking email format for customers dataset:

```
if (all(email_format_correct) && count == 3) {
  cat("All email addresses are in the correct format.\n")
  count <- 4
} else {
  incorrect_emails <- customer_data[[email_column_name]][!email_format_correct]
  cat("The following email addresses are not in the correct format:\n")
  print(incorrect_emails)
}
```

All email addresses are in the correct format.

Phone Number Format Check:

The following code verifies the format of phone number within the “Customers” dataset to ensure that they adhere to the specified UK format. Results suggest the correctness of each phone number, confirming that they align with the format.

```
#Check phone number is in format of +44 7XXX-XXX-XXX
# "customers" dataset contains the phone numbers
customer_data_path <- "data_uploads/customers.csv"
customer_data <- read_csv(customer_data_path, col_types = cols())

# Define the phone number format
phone_pattern <- "^\\+44 7\\d{3}-\\d{3}-\\d{3}$"

# Check if the phone format is correct in the specified column
# Replace 'phone_column_name' with the
# actual name of the column containing phone numbers
```

```
phone_format_correct <- grepl(phone_pattern, customer_data$phone)

# Printing the results
cat("Checking phone format for customers dataset:\n")
```

Checking phone format for customers dataset:

```
if (all(phone_format_correct) && count == 4) {
  cat("All phone numbers are in the correct format.\n")
  count <- 5
} else {
  incorrect_phones <- customer_data$phone_column_name[!phone_format_correct]
  cat("The following phone numbers are not in the correct format:\n")
  print(incorrect_phones)
}
```

All phone numbers are in the correct format.

```
#Printing this message if all validation checks are passed
if (count == 5) {
  cat("The dataset passed all five data validation tests")
}
```

The dataset passed all five data validation tests

Referential integrity:

Established through the careful definition of primary and foreign key constraints across various tables within a relational database. For example, the 'orders' table references the 'customers' table via a foreign key constraint on 'customer\_id', ensuring that each order is linked to an existing customer. Similarly, the 'order\_details' table contains foreign keys that reference both the 'products' table (through 'product\_id') and the 'orders' table (through 'order\_id'), guaranteeing that order details cannot exist without a corresponding product and order. This pattern is consistently applied across other tables. These foreign key constraints ensure that relationships between tables are strictly maintained, preventing orphaned records and preserving the logical integrity of the database schema

### Data Importing:

To import the data sets into the SQLite database after data validation and quality assurance checks, the database connection is established. As the first step, all the tables are dropped from the database using the dbExecute function of RSQLite.

ETL (Extract, Transform, Load): The dataset is extracted, then transformed and loaded to the database

```
#db connection
db_connection <- RSQLite::dbConnect(RSQLite::SQLite(),"ecommerce.db")

#Dropping the tables if they exists from the database
invisible(RSQLite::dbExecute(db_connection,
                             "DROP TABLE IF EXISTS customers"))
invisible(RSQLite::dbExecute(db_connection,
                             "DROP TABLE IF EXISTS order_details"))
invisible(RSQLite::dbExecute(db_connection,
                             "DROP TABLE IF EXISTS transactions"))
invisible(RSQLite::dbExecute(db_connection,
                             "DROP TABLE IF EXISTS products"))
invisible(RSQLite::dbExecute(db_connection,
                             "DROP TABLE IF EXISTS product_categories"))
invisible(RSQLite::dbExecute(db_connection,
                             "DROP TABLE IF EXISTS suppliers"))
invisible(RSQLite::dbExecute(db_connection,
                             "DROP TABLE IF EXISTS promotion"))
invisible(RSQLite::dbExecute(db_connection,
                             "DROP TABLE IF EXISTS orders"))
```

After the tables are dropped, the physical database schema is newly created into the SQLite database from the primary data sets in the data uploads folder.

```
# Writing the csv file contents to the database and
# Creating the table with the table_name
RSQLite::dbWriteTable(db_connection,
                      "customers","data_uploads/customers.csv",append=TRUE)
RSQLite::dbWriteTable(db_connection,
                      "order_details",
                      "data_uploads/order_details.csv",append=TRUE)
RSQLite::dbWriteTable(db_connection,
                      "transactions",
                      "data_uploads/transactions.csv",append=TRUE)
RSQLite::dbWriteTable(db_connection,
                      "products",
                      "data_uploads/products.csv",append=TRUE)
RSQLite::dbWriteTable(db_connection,
                      "product_categories",
```

```

        "data_uploads/product_categories.csv",append=TRUE)
RSQLite::dbWriteTable(db_connection,
        "suppliers",
        "data_uploads/suppliers.csv",append=TRUE)
RSQLite::dbWriteTable(db_connection,
        "promotions",
        "data_uploads/promotions.csv",append=TRUE)
RSQLite::dbWriteTable(db_connection,
        "orders",
        "data_uploads/orders.csv",append=TRUE)

#Disconnecting the db connection
RSQLite::dbDisconnect(db_connection)

```

## Data Pipeline Generation

### 3.1 Github Repository and Workflow Setup

To effectively manage and version-control our project, including database files, scripts, and necessary documents, we established a GitHub repository as a centralized location to store and organize them. The GitHub repo link is provided here: [https://github.com/aparna-kiran/dm\\_group\\_5/](https://github.com/aparna-kiran/dm_group_5/)

All project-related scripts, including data validation, database updates, and data analysis, were included in the repository. Any necessary documents, such as project documentation or instructions, were also incorporated. By utilizing version control, we maintained a comprehensive history of changes made to our project, enabling us to track progress and collaborate effectively.

### 3.2 Github Actions for Continuous Integration

To streamline and automate key data management processes, we have used GitHub Actions, including automating data validation, updating databases with new data, and conducting basic data analysis tasks.

We have set up workflows within a file named 'etl.yaml' GitHub that get triggered automatically on the push event to our repositories. This ensures that our data management processes are initiated seamlessly and consistently. We have implemented actions that run data validation scripts to check the integrity and quality of our data, ensuring it meets predefined standards before any further processing.

```
dm_group_5 / .github / workflows / etl.yaml
aparna-kiran Update etl.yaml - file name change
Code Blame 42 lines (48 loc) • 1.24 KB Code 55% faster with GitHub Copilot

1 name: ETL workflow for Group 05
2
3 on:
4   # schedule:
5   #   - cron: '0 */3 * * *' # Run every 3 hours
6   push:
7     branches: [ main ]
8
9 jobs:
10  build:
11    runs-on: ubuntu-latest
12    steps:
13      - name: Checkout code
14        uses: actions/checkout@v2
15      - name: Setup R environment
16        uses: r-lib/actions/setup-r@v2
17        with:
18          r-version: '4.2.0'
19      - name: Cache R packages
20        uses: actions/cache@v2
21        with:
22          path: ${{ env.R_LIBS_USER }}
23          key: ${{ runner.os }}-r-${{ hashFiles('**/lockfile') }}
24          restore-keys: |
25            ${{ runner.os }}-r-
26      - name: Install packages
27        if: steps.cache.outputs.cache-hit != 'true'
28        run: |
29          Rscript -e 'install.packages(c("ggplot2", "dplyr", "readr", "RSQLite", "tidyr", "lubridate", "readxl", "scales"))'
30      - name: Execute R script
31        run: |
32          Rscript R/data_importing.R
33      - name: Execute data automation R script
34        run: |
35          Rscript R/db_automation.R
36      - name: Execute data analysis R script
37        run: |
38          Rscript R/data_transformation.R
39      - name: Add files
40        run: |
41          git config --global user.email "aparnakiran.as@gmail.com"
42          git config --global user.name "aparna-kiran"
```

Figure 17: Github Workflow Code.

```
library(readr)
library(RSQLite)
# Get a list of file names in the directory
directory <- "data_uploads/"

#listing data files
data_files <- list.files(directory, pattern = "\\*.csv$")
data_files
```

```

[1] "customers.csv"          "customers12.csv"          "order_details.csv"
[4] "orders.csv"             "product_categories.csv"   "products-new.csv"
[7] "products.csv"           "promotions.csv"           "suppliers.csv"
[10] "transactions.csv"

```

```

#establishing db connection
db_connection <- RSQLite::dbConnect(RSQLite::SQLite(),"ecommerce.db")

#listing the database table names
table_names <- RSQLite::dbListTables(db_connection)
table_names

```

```

[1] "customers"          "order_details"          "orders"
[4] "product_categories" "products"                "promotions"
[7] "suppliers"          "transactions"

```

Through GitHub Actions, we have automated the integration of new data into our databases, ensuring that our data remains up-to-date and accurate. Then, we automated the execution of basic data analysis scripts. By automating these tasks, we reduced manual effort and the risk of error, leading to increased efficiency in our data management processes.

The code below updates the SQLite database with the new data sets. It checks if any fields in the new data file are already existing in the database and if there are any, then they are not appended to the table.

```

for (tablename in table_names){

  # Filter files containing "customers" in their name
  table_files <- grep(tablename, data_files, ignore.case = TRUE, value = TRUE)

  # Checking if there is more than one dataset for each table name
  if(length(table_files) > 1){
    if(tablename == "customers"){
      for (i in seq_along(table_files)){
        # Importing customers database table
        existing_dataset <- dbReadTable(db_connection, tablename)
        # Defining the file path of new dataset
        filepath <- paste0(directory,table_files[i])
        dataset <- read_csv(filepath, col_types = cols())

        # Making all primary keys of max length
        id_length <- max(nchar(existing_dataset$customer_id))
      }
    }
  }
}

```

```

#adding leading zeroes
existing_dataset$customer_id <- sprintf(
  paste0("%0", id_length, "d"),
  as.numeric(existing_dataset$customer_id))

# Filtering new fields from the new dataset
# that does not exist in the table
filtered_dataset <- dataset[!dataset$customer_id %in%
  existing_dataset$customer_id, ]
RSQLite::dbWriteTable(db_connection,tablename,
  filtered_dataset,
  append=TRUE)

# If there are new fields then append them to the sql table
if(nrow(filtered_dataset) != 0){
  print("Customers are appending")
}
}
}

# Doing the above workflow for products
if(tablename == "products"){
  for (i in seq_along(table_files)){
    existing_dataset <- dbReadTable(db_connection, tablename)
    filepath <- paste0(directory,table_files[i])
    dataset <- read_csv(filepath, col_types = cols())

    id_length <- max(nchar(existing_dataset$product_id))
    existing_dataset$product_id <- sprintf(
      paste0("%0",
                                id_length, "d"),
      as.numeric(
        existing_dataset$product_id
      )
    )

    filtered_dataset <- dataset[!dataset$product_id
      %in%
      existing_dataset$product_id, ]

    RSQLite::dbWriteTable(db_connection,tablename,
      filtered_dataset,
      append=TRUE)
    if(nrow(filtered_dataset) != 0){
      print("Products are appending")
    }
  }
}
}

```

```

    }
  }
}
# Doing the above workflow for order details
if(tablename == "order_details"){
  for (i in seq_along(table_files)){
    existing_dataset <- dbReadTable(db_connection, tablename)
    filepath <- paste0(directory,table_files[i])
    dataset <- read_csv(filepath, col_types = cols())

    id_length <- max(nchar(existing_dataset$order_detail_id))
    existing_dataset$order_detail_id <- sprintf(
      paste0("%0", id_length, "d"),
      as.numeric(existing_dataset$order_detail_id))

    filtered_dataset <- dataset[!dataset$order_detail_id
                                %in%
                                existing_dataset$order_detail_id, ]
    RSQLite::dbWriteTable(db_connection,tablename,
                          new_dataset,
                          append=TRUE)
    if(nrow(filtered_dataset) != 0){
      print("Order Details are appending")
    }
  }
}
# Doing the above workflow for orders
if(tablename == "orders"){
  for (i in seq_along(table_files)){
    existing_dataset <- dbReadTable(db_connection, tablename)
    filepath <- paste0(directory,table_files[i])
    dataset <- read_csv(filepath, col_types = cols())

    id_length <- max(nchar(existing_dataset$order_id))
    existing_dataset$order_id <- sprintf(paste0("%0", id_length, "d"),
                                         as.numeric(
                                           existing_dataset$order_id))

    filtered_dataset <- dataset[!dataset$order_id %in%
                                existing_dataset$order_id, ]
    RSQLite::dbWriteTable(db_connection,tablename,
                          new_dataset,

```



```

                                append=TRUE)
    if(nrow(filtered_dataset) != 0){
      print("Orders are appending")
    }
  }
}
# Doing the above workflow for promotions
if(tablename == "promotions"){
  for (i in seq_along(table_files)){
    existing_dataset <- dbReadTable(db_connection, tablename)
    filepath <- paste0(directory,table_files[i])
    dataset <- read_csv(filepath, col_types = cols())

    id_length <- max(nchar(existing_dataset$promotion_id))
    existing_dataset$promotion_id <- sprintf(
      paste0("%0", id_length, "d"),
      as.numeric(existing_dataset$promotion_id))

    filtered_dataset <- dataset[!dataset$promotion_id %in%
                                existing_dataset$promotion_id, ]
    RSQLite::dbWriteTable(db_connection,tablename,
                          new_dataset,
                          append=TRUE)
    if(nrow(filtered_dataset) != 0){
      print("Promotions are appending")
    }
  }
}
# Doing the above workflow for suppliers
if(tablename == "suppliers"){
  for (i in seq_along(table_files)){
    existing_dataset <- dbReadTable(db_connection, tablename)
    filepath <- paste0(directory,table_files[i])
    dataset <- read_csv(filepath, col_types = cols())

    id_length <- max(nchar(existing_dataset$supplier_id))
    existing_dataset$supplier_id <- sprintf(
      paste0("%0", id_length, "d"),
      as.numeric(existing_dataset$supplier_id))

    filtered_dataset <- dataset[!dataset$supplier_id %in%
                                existing_dataset$supplier_id, ]
  }
}

```

```

    RSQLite::dbWriteTable(db_connection,tablename,
                          new_dataset,
                          append=TRUE)
    if(nrow(filtered_dataset) != 0){
      print("Suppliers are appending")
    }
  }
}

# Doing the above workflow for transactions
if(tablename == "transactions"){
  for (i in seq_along(table_files)){
    existing_dataset <- dbReadTable(db_connection, tablename)
    filepath <- paste0(directory,table_files[i])
    dataset <- read_csv(filepath, col_types = cols())

    id_length <- max(nchar(existing_dataset$transaction_id))
    existing_dataset$transaction_id <- sprintf(
      paste0("%0", id_length, "d"),
      as.numeric(existing_dataset$transaction_id))

    filtered_dataset <- dataset[!dataset$transaction_id
                                %in%
                                existing_dataset$transaction_id, ]
    RSQLite::dbWriteTable(db_connection,tablename,
                          new_dataset,
                          append=TRUE)
    if(nrow(filtered_dataset) != 0){
      print("Transactions are appending")
    }
  }
}

# Doing the above workflow for product categories
if(tablename == "product_categories"){
  for (i in seq_along(table_files)){
    existing_dataset <- dbReadTable(db_connection, tablename)
    filepath <- paste0(directory,table_files[i])
    dataset <- read_csv(filepath, col_types = cols())

    id_length <- max(nchar(existing_dataset$category_id))
    existing_dataset$category_id <- sprintf(
      paste0("%0", id_length, "d"),
      as.numeric(existing_dataset$category_id))
  }
}

```

```

        filtered_dataset <- dataset[!dataset$category_id %in%
                                     existing_dataset$category_id, ]
        RSQLite::dbWriteTable(db_connection,tablename,
                               new_dataset,
                               append=TRUE)
        if(nrow(filtered_dataset) != 0){
            print("Product Categories are appending")
        }
    }
}
}
}
}

```

```
[1] "Products are appending"
```

```
RSQLite::dbDisconnect(db_connection)
```

## Data Analysis

### 4.1, 4.2 Advanced Data Analysis in R and Reporting with Quarto

In our e-commerce platform, we extract key characteristics such as total orders, total revenue, and best-selling products. We analyze these results across different time series, including seasonally, quarterly, and annually to uncover patterns and trends.

```

library(dplyr)
library(ggplot2)
library(tidyr)
library(lubridate)
library(readxl)
library(scales)

```

```

# Reading required database
db_connection <- RSQLite::dbConnect(RSQLite::SQLite(),"ecommerce.db")

customers <- RSQLite::dbReadTable(db_connection, "customers")
order_details <- RSQLite::dbReadTable(db_connection, "order_details")
orders <- RSQLite::dbReadTable(db_connection, "orders")
product_categories <- RSQLite::dbReadTable(db_connection, "product_categories")
products <- RSQLite::dbReadTable(db_connection, "products")

```

```

promotions <- RSQLite::dbReadTable(db_connection, "promotions")
suppliers <- RSQLite::dbReadTable(db_connection, "suppliers")
transactions <- RSQLite::dbReadTable(db_connection, "transactions")

```

```

#Standardise date format
orders$order_date <- as.Date(orders$order_date, format= "%d/%m/%Y")
orders$estimated_delivery_date <- as.Date(orders$estimated_delivery_date,
                                          format = "%d/%m/%Y")

```

```

# Exclude "cancelled" orders and calculate total revenue per year
orders <- orders %>%
  mutate(
    order_id = as.numeric(order_id),
  )
yearly_revenue <- orders %>%
  filter(order_status != "cancelled") %>%
  inner_join(order_details, by = "order_id") %>%
  mutate(year = year(order_date)) %>%
  group_by(year) %>%
  summarize(total_revenue = sum(product_price * order_quantity),
            .groups = 'drop')
print(yearly_revenue)

```

```

# A tibble: 4 x 2
  year total_revenue
<dbl>         <dbl>
1  2020         90492.
2  2021         86787.
3  2022         79055.
4  2023         74111.

```

```

# Adjusting the total_revenue to be in thousands for visualization
yearly_revenue$total_revenue <- yearly_revenue$total_revenue / 1000

# Plotting line chart for the yearly revenue with y-axis values in thousands

yearly_revenue_plot <- ggplot(yearly_revenue, aes(x = as.factor(year),
                                                    y = total_revenue,
                                                    group = 1)) +

  geom_line(color = "blue") +
  geom_point(color = "red", size = 3) +

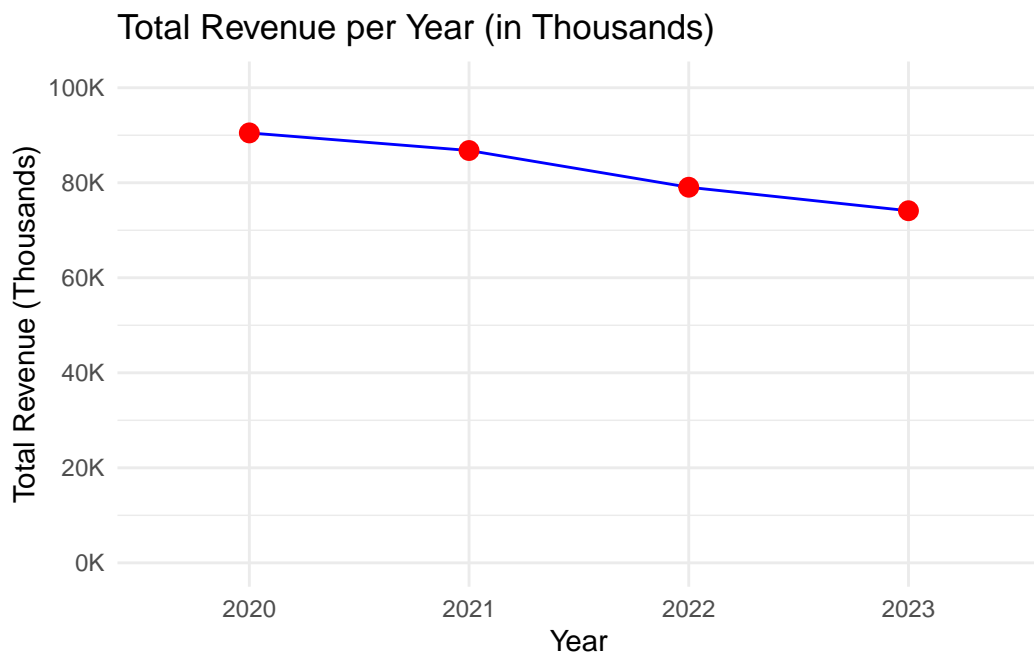
```

```

labs(title = "Total Revenue per Year (in Thousands)",
     x = "Year", y = "Total Revenue (Thousands)") +
theme_minimal() +
scale_y_continuous(labels = label_number(suffix = "K"),
                  limits = c(0, max(yearly_revenue$total_revenue,
                                    na.rm = TRUE) + 10),
                  breaks = seq(0, max(yearly_revenue$total_revenue,
                                    na.rm = TRUE) + 10, by = 20))

ggsave(plot = yearly_revenue_plot,
       filename = "images/yearly_revenue_plot.png",
       width = 10, height = 6, dpi = 300)
print(yearly_revenue_plot)

```



A continuously decreasing pattern is observed in total revenues per year between 2020 and 2023. Starting at a peak in 2020 with a total revenue of more than £90,000, there is a noticeable year-over-year decrease through to 2023, where the revenue has its lowest value at around £74,000.

```

# Calculate total revenue per quarter across all years
quarterly_revenue <- orders %>%
  filter(order_status != "cancelled") %>% # Exclude cancelled orders
  inner_join(order_details, by = "order_id") %>%
  mutate(quarter = paste("Q", quarter(order_date), sep="")) %>%

```

```

group_by(quarter) %>%
  summarize(total_revenue = sum(product_price * order_quantity),
            .groups = 'drop') %>%
  mutate(quarter = factor(quarter, levels = c("Q1", "Q2", "Q3", "Q4")))
print(quarterly_revenue)

```

```

# A tibble: 4 x 2
  quarter total_revenue
  <fct>      <dbl>
1 Q1         85081.
2 Q2         87830.
3 Q3         82276.
4 Q4         75258.

```

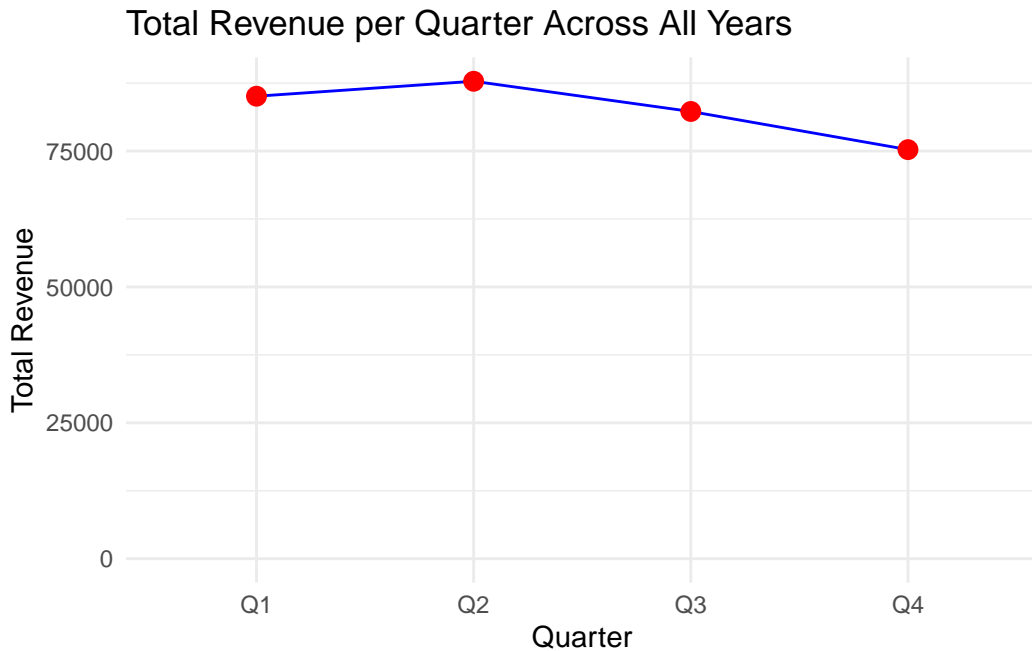
```

quarterly_revenue_all_years_plot <- ggplot(quarterly_revenue,
      aes(x = quarter,
          y = total_revenue,
          group = 1)) +

  geom_line(color = "blue") +
  geom_point(color = "red", size = 3) +
  labs(title = "Total Revenue per Quarter Across All Years",
       x = "Quarter", y = "Total Revenue") +
  theme_minimal() +
  scale_y_continuous(limits = c(0, NA)) # Start y-axis at 0

ggsave(plot = quarterly_revenue_all_years_plot,
      filename = "images/quarterly_revenue_all_years_plot.png",
      width = 10, height = 6, dpi = 300)
print(quarterly_revenue_all_years_plot)

```



The figure displaying the total revenue per quarter across all years (2020-2023) shows that the sum of the total revenue in all years varies across the quarters, with the highest total revenue recorded in Q2 and the lowest in Q4. It is observed that Q1 has a relatively high total revenue overall, continued with an increasing trend in Q2. However, after Q2, a consistent year-over-year decline in the sum of total revenue values is seen between Q2 and Q4.

```
# Calculate the quarterly revenue for each year
detailed_quarterly_revenue <- orders %>%
  filter(order_status != "cancelled") %>% # Exclude cancelled orders
  inner_join(order_details, by = "order_id") %>%
  mutate(year_quarter = paste(year(order_date), "Q",
                              quarter(order_date), sep="")) %>%
  group_by(year_quarter) %>%
  summarize(total_revenue = sum(product_price * order_quantity),
            .groups = 'drop') %>%
  arrange(year_quarter)
print(detailed_quarterly_revenue)
```

```
# A tibble: 16 x 2
  year_quarter total_revenue
  <chr>         <dbl>
1 2020Q1         22906.
2 2020Q2         24685.
3 2020Q3         24345.
```

4	2020Q4	18556.
5	2021Q1	19545.
6	2021Q2	24216.
7	2021Q3	22439.
8	2021Q4	20588.
9	2022Q1	17318.
10	2022Q2	18523.
11	2022Q3	25417.
12	2022Q4	17797.
13	2023Q1	25312.
14	2023Q2	20406.
15	2023Q3	10076.
16	2023Q4	18317.

```
detailed_quarterly_revenue <- detailed_quarterly_revenue %>%
  mutate(year = substr(year_quarter, 1, 4),
         quarter = paste("Q", substr(year_quarter, 6, 7), sep=""))
# Add "Q" prefix to quarter

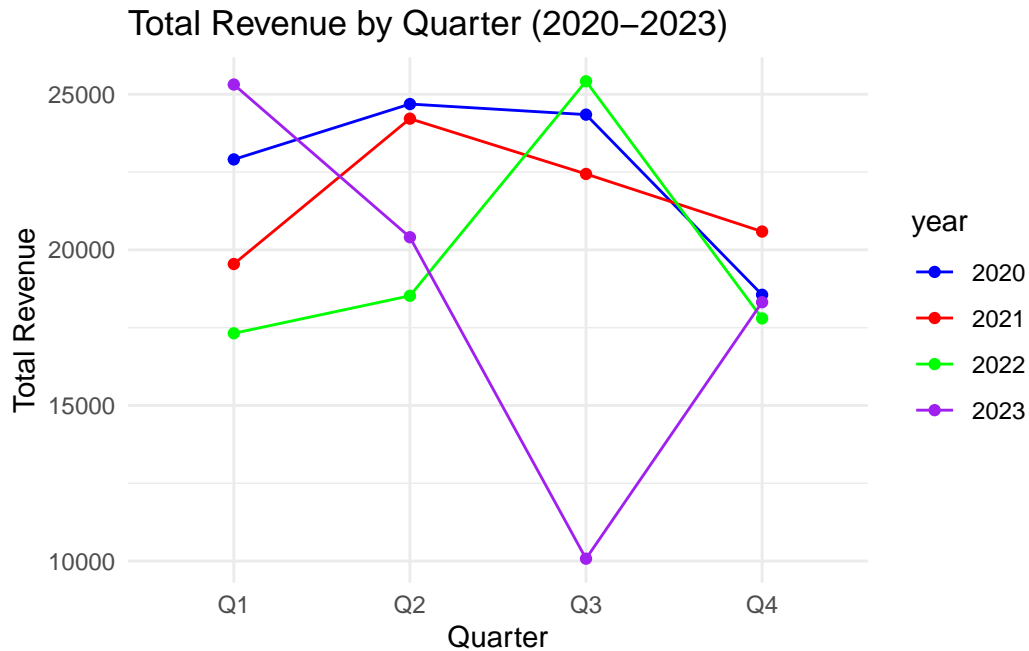
# Plot the line chart for quarterly revenue for each year
quarterly_revenue_plot <- ggplot(detailed_quarterly_revenue,
                                aes(x = quarter, y = total_revenue,
                                    group = year, color = year)) +

  geom_line() +
  geom_point() +
  labs(title = "Total Revenue by Quarter (2020-2023)",
       x = "Quarter", y = "Total Revenue") +
  scale_color_manual(values = c("2020" = "blue",
                                "2021" = "red", "2022" = "green",
                                "2023" = "purple")) +

  theme_minimal()

ggsave(plot = quarterly_revenue_plot,
       filename = "images/quarterly_revenue_plot.png",
       width = 10, height = 6, dpi = 300)
print(quarterly_revenue_plot )
```





Analysing the total revenue trends from 2020 to 2023, a recurrent increase is observed from Q1 to Q2 each year except in 2023, where a drop is observed from around £25,000 to £20,000. The transition from Q2 to Q3 in 2022 is marked by a significant rise from around £18,000 to £25,000, contrasted by 2023's sharp decline from around £20,000 to £10,000. While Q3 generally records the peak revenue, especially in 2022; Q4 often sees a decrease, except for 2023, where a rebound to around £18,000 defies the usual downward trend.

```
# Filter out cancelled orders
non_cancelled_orders <- orders %>%
  filter(order_status != "cancelled")

# Join non_cancelled_orders with order_details
total_revenue_by_city <- non_cancelled_orders %>%
  inner_join(order_details, by = "order_id") %>%
  group_by(city) %>%
  summarize(total_revenue = sum(product_price * order_quantity),
            .groups = 'drop')
print(total_revenue_by_city)
```

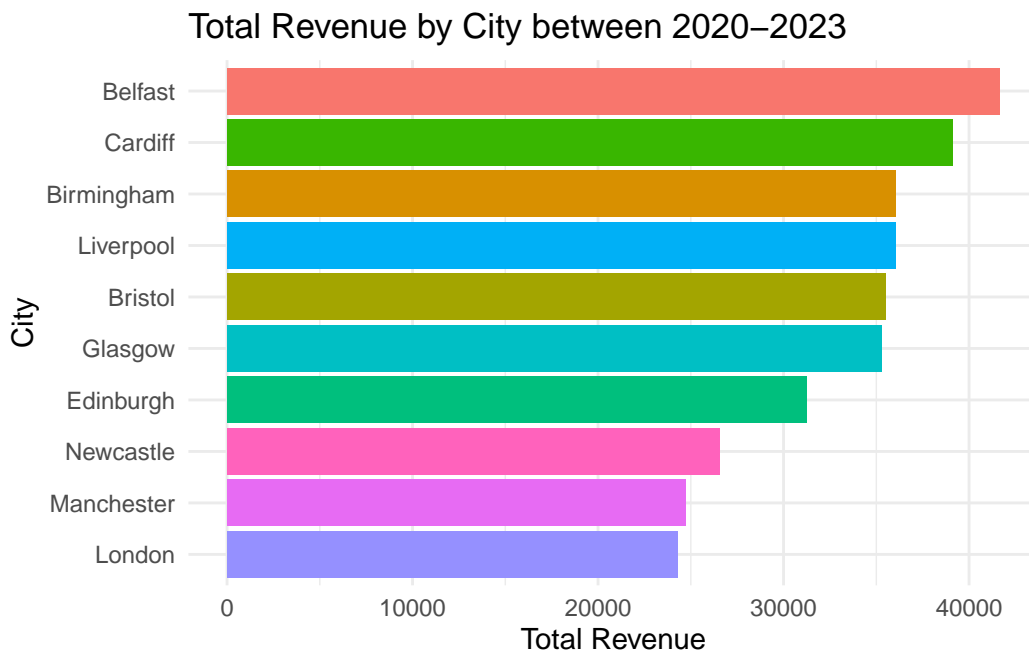
```
# A tibble: 10 x 2
  city      total_revenue
  <chr>      <dbl>
1 Belfast    41666.
2 Birmingham 36052.
```

3	Bristol	35525.
4	Cardiff	39112.
5	Edinburgh	31221.
6	Glasgow	35281.
7	Liverpool	36032.
8	London	24305.
9	Manchester	24712.
10	Newcastle	26539

```
# Plot the bar chart for total revenue by city
revenue_by_city_plot <- ggplot(total_revenue_by_city,
                               aes(x = reorder(city, total_revenue),
                                   y = total_revenue, fill = city)) +

  geom_col() +
  labs(title = "Total Revenue by City between 2020-2023",
       x = "City", y = "Total Revenue") +
  theme_minimal() +
  coord_flip() +
  guides(fill = "none") # Removes the legend

ggsave(plot = revenue_by_city_plot,
       filename = "images/revenue_by_city_plot.png",
       width = 10, height = 6, dpi = 300)
print(revenue_by_city_plot)
```



Analysing the total revenue across different cities, Belfast emerges on the top with approximately £41,000. London and Manchester, notable for their economic significance, are at the lower end among the first 10 cities listed, each with around £24,000 to £25,000. Geographically, this indicates a diverse economic landscape, with the highest revenues not necessarily in the traditionally dominant economic centers of the UK.

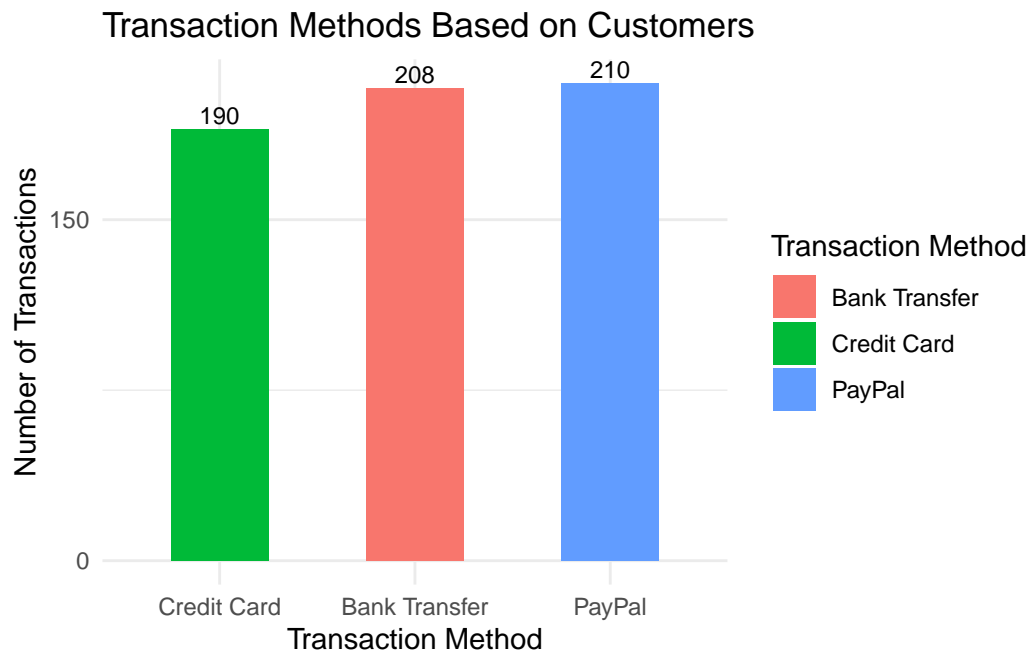
```
# Exclude the cancelled orders
# Calculate the numbers for each transaction method used
transactions <- transactions %>%
  mutate(order_id = as.numeric(order_id))
non_cancelled_orders <- orders %>%
  filter(order_status != "cancelled")
transactions_summary <- non_cancelled_orders %>%
  inner_join(transactions, by = "order_id") %>%
  group_by(transaction_method) %>%
  summarize(number_of_transactions = n(), .groups = 'drop')

# Find the maximum number of transactions
max_transactions <- max(transactions_summary$number_of_transactions)

# Bar chart for usage of transaction methods
transaction_methods_plot <- ggplot(transactions_summary,
                                   aes(x = reorder(transaction_method,
                                                    number_of_transactions),
                                       y = number_of_transactions,
                                       fill=transaction_method)) +

  geom_col(width = 0.5) +
  geom_text(aes(label = number_of_transactions,
                vjust = -0.3, color = "black", size = 3.3) +
  labs(title = "Transaction Methods Based on Customers",
        x = "Transaction Method",
        y = "Number of Transactions",
        fill = "Transaction Method") +
  theme_minimal() +
  scale_y_continuous(breaks = seq(0, max_transactions, by = 150))

ggsave(plot = transaction_methods_plot,
        filename = "images/transaction_methods_plot.png",
        width = 10, height = 6, dpi = 300)
print(transaction_methods_plot)
```



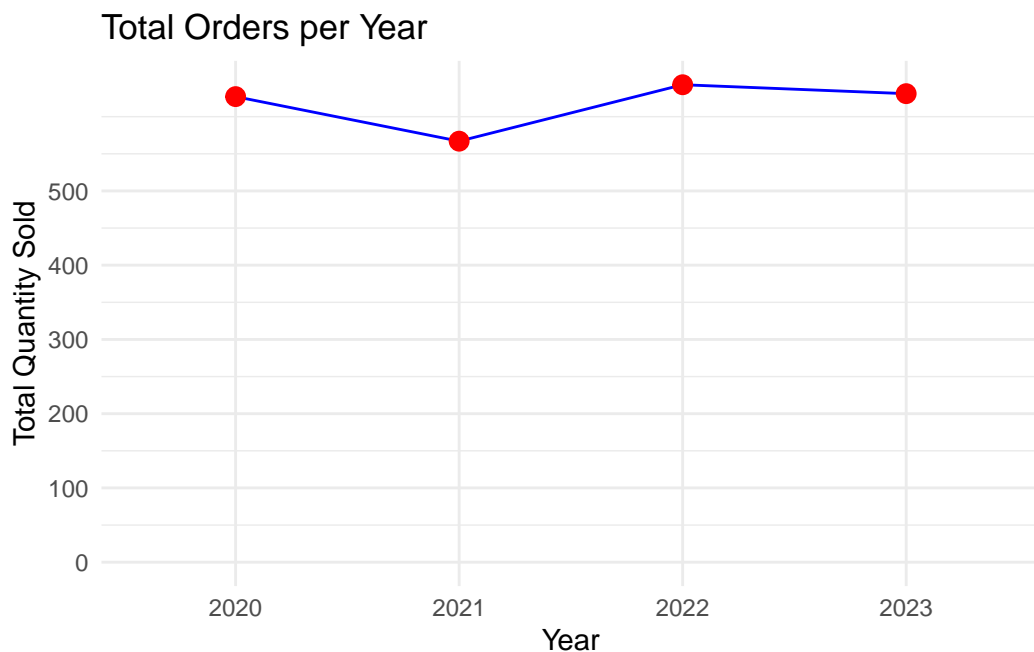
Transaction data from our e-commerce platform reveals a balanced preference for payment methods. PayPal leads marginally with 210 transactions, followed by bank transfers at 208 and credit cards at 190.

```
# Exclude "cancelled" orders and calculate the total orders per year
yearly_orders_quantity <- orders %>%
  filter(order_status != "cancelled") %>%
  inner_join(order_details, by = "order_id") %>%
  mutate(year = year(order_date)) %>%
  group_by(year) %>%
  summarize(total_quantity = sum(order_quantity), .groups = 'drop')
# Summarize by total quantity
print(yearly_orders_quantity)
```

```
# A tibble: 4 x 2
  year total_quantity
  <dbl>         <int>
1  2020             627
2  2021             567
3  2022             643
4  2023             631
```

```
# Plotting the total orders per year
yearly_orders_plot <- ggplot(yearly_orders_quantity,
                             aes(x = as.factor(year),
                                 y = total_quantity, group = 1)) +
  geom_line(color = "blue") +
  geom_point(color = "red", size = 3) +
  labs(title = "Total Orders per Year", x = "Year",
       y = "Total Quantity Sold") +
  theme_minimal() +
  scale_y_continuous(limits = c(0, NA), breaks = seq(0, 500, by = 100))

ggsave(plot = yearly_orders_plot,
       filename = "images/yearly_orders_plot.png",
       width = 10, height = 6, dpi = 300)
print(yearly_orders_plot)
```



Total order numbers has the lowest value in 2021 and peak value in 2022, starting with a slight decline from 627 orders in 2020 to 567 in 2021, followed by a significant increase to 643 orders in 2022, and a subsequent decrease to 631 orders in 2023.

```
# Calculate total orders per season for 4 different years
# Excluding cancelled orders
detailed_seasonal_orders_quantity <- orders %>%
  filter(order_status != "cancelled") %>%
```

```

inner_join(order_details, by = "order_id") %>%
mutate(year = year(order_date),
       month = month(order_date),
       season = case_when(
         month %in% c(12, 1, 2) ~ "Winter",
         month %in% c(3, 4, 5) ~ "Spring",
         month %in% c(6, 7, 8) ~ "Summer",
         month %in% c(9, 10, 11) ~ "Fall"
       )) %>%
mutate(season = factor(season,
                      levels = c("Fall", "Winter",
                                "Spring", "Summer"))) %>%

group_by(year, season) %>%
summarize(total_quantity = sum(order_quantity),
          .groups = 'drop') # Summarize by total quantity
print(detailed_seasonal_orders_quantity)

```

```

# A tibble: 16 x 3
  year season total_quantity
  <dbl> <fct>         <int>
1  2020 Fall           161
2  2020 Winter         141
3  2020 Spring         198
4  2020 Summer         127
5  2021 Fall           119
6  2021 Winter         125
7  2021 Spring         181
8  2021 Summer         142
9  2022 Fall           188
10 2022 Winter         160
11 2022 Spring         137
12 2022 Summer         158
13 2023 Fall           169
14 2023 Winter         153
15 2023 Spring         165
16 2023 Summer         144

```

```

# Plotting the seasonal orders for each year
seasonal_orders_plot <- ggplot(detailed_seasonal_orders_quantity,
                              aes(x = season, y = total_quantity,
                                  color = as.factor(year), group = year)) +

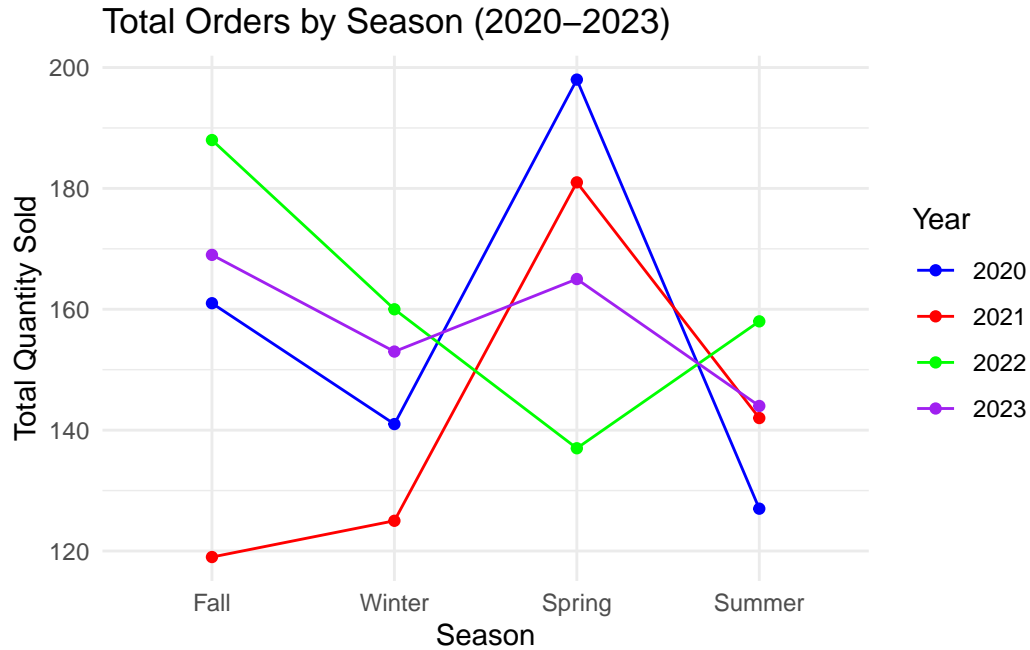
```

```

geom_line() +
geom_point() +
scale_color_manual(values = c("2020" = "blue", "2021" = "red",
                              "2022" = "green", "2023" = "purple"),
                  name = "Year") +
labs(title = "Total Orders by Season (2020-2023)",
     x = "Season",
     y = "Total Quantity Sold",
     color = "Year") +
theme_minimal()

ggsave(plot = seasonal_orders_plot,
       filename = "images/seasonal_orders_plot.png",
       width = 10, height = 6, dpi = 300)
print(seasonal_orders_plot)

```



The seasonal trend for total orders from 2020 to 2023 shows considerable variability, with no consistent pattern emerging across the years. The spring of 2020 marked the highest number of orders at 198. Notably, orders in the fall have shown growth, particularly from 2021 to 2022 where they rose from 119 to 188. The winter of 2022 also experienced a significant rise to 160 orders, exceeding the counts of both preceding years.

```

# Exclude "cancelled" orders
non_cancelled_orders <- orders %>%
  filter(order_status != "cancelled")

# Calculate total quantity sold for each product
# Find the top 5 best-selling products
top_selling_products <- non_cancelled_orders %>%
  inner_join(order_details, by = "order_id") %>%
  group_by(product_id) %>%
  summarise(total_quantity_sold = sum(order_quantity, na.rm = TRUE)) %>%
  ungroup() %>%
  arrange(desc(total_quantity_sold)) %>%
  slice_max(order_by = total_quantity_sold, n = 5) %>%
  distinct(product_id, .keep_all = TRUE)

# Join with the products data frame to get the product names
top_selling_products_with_names <- top_selling_products %>%
  left_join(products, by = "product_id") %>%
  select(product_name, total_quantity_sold) %>%
  arrange(desc(total_quantity_sold)) %>%
  slice_head(n = 10)
print(top_selling_products_with_names)

```

```

# A tibble: 6 x 2
  product_name      total_quantity_sold
  <chr>              <int>
1 Whole Grain Bread 4zv          38
2 Yoga Mat WZQ                36
3 The Great Adventure E10        35
4 Yoga Mat 6ne                 33
5 Organic Honey H4w             32
6 Eau de Toilette MZw           32

```

```

best_selling_products_plot <- ggplot(top_selling_products_with_names,
  aes(x = reorder(product_name,
    total_quantity_sold),
    y = total_quantity_sold,
    fill = product_name)) +
  geom_bar(stat = "identity") +
  coord_flip() + # Flip the axes to make it a horizontal bar chart
  geom_text(aes(label = scales::comma(total_quantity_sold)),

```



```

    position = position_dodge(width = 0.9),
    hjust = -0.2, size = 3) +
  labs(title = "Top 5 Best-Selling Products", x = "Product Name",
    y = "Total Quantity Sold") +
  scale_y_continuous(labels = scales::comma) +
  # Format the Y axis labels with commas
  scale_fill_brewer(palette = "Set3") +
  theme_minimal() +
  theme(axis.text.y = element_text(angle = 0), legend.position = "none")

ggsave(plot = best_selling_products_plot,
  filename = "images/best_selling_products_plot.png",
  width = 10,
  height = 6, dpi = 300)
print(best_selling_products_plot)

```



```

# Ensure order_date is Date type and calculate season
orders <- orders %>%
  mutate(
    order_date = as.Date(order_date, format = "%Y-%m-%d"),
    # Convert order_date to Date type
    season = case_when(
      month(order_date) %in% c(12, 1, 2) ~ "Winter",

```

```

    month(order_date) %in% c(3, 4, 5) ~ "Spring",
    month(order_date) %in% c(6, 7, 8) ~ "Summer",
    month(order_date) %in% c(9, 10, 11) ~ "Fall"
  )
)

# Filter out cancelled orders from the orders dataframe
non_cancelled_orders <- orders %>%
  filter(order_status != "cancelled")

# Join non_cancelled_orders with order_details
# Join non_cancelled_orders with products to get product names
sales_data <- non_cancelled_orders %>%
  inner_join(order_details, by = "order_id") %>%
  inner_join(products, by = "product_id")

# Group by season and product_name
# To summarize total quantity sold across all years
seasonal_sales <- sales_data %>%
  group_by(season, product_name) %>%
  summarise(total_quantity_sold = sum(order_quantity, na.rm = TRUE),
            .groups = "drop")

# For each season find top 10 products with the highest total quantity sold
top_5_seasonal_sales <- seasonal_sales %>%
  arrange(season, desc(total_quantity_sold)) %>%
  group_by(season) %>%
  slice_max(order_by = total_quantity_sold, n = 5, with_ties = FALSE) %>%
  ungroup()
print(top_5_seasonal_sales)

```

```

# A tibble: 20 x 3
  season product_name      total_quantity_sold
  <chr>   <chr>                <int>
1 Fall   Whole Grain Bread 4zv      16
2 Fall   Organic Honey 17d        13
3 Fall   Summer Dress Dzb         13
4 Fall   Yoga Mat 6ne             12
5 Fall   Organic Honey NNp         11
6 Spring Camping Tent gTa      17
7 Spring Premium Blender vXj     12
8 Spring The Great Adventure E10 12

```

9	Spring Eau de Toilette dis	11
10	Spring Organic Honey WKG	11
11	Summer Smartphone X12 Jmx	15
12	Summer Leather Wallet 509	14
13	Summer Whole Grain Bread 4zv	13
14	Summer Cooking 101 nxw	12
15	Summer Laptop Pro 15 mgo	12
16	Winter Eau de Toilette MZw	14
17	Winter Organic Honey H4w	13
18	Winter Yoga Mat WZQ	13
19	Winter Smartphone X12 eDX	11
20	Winter Smartphone X12 x1B	11

Throughout 2020 to 2023, ‘Whole Grain Bread 4zv’ is a consistent favorite, leading in fall with 16 units and remaining strong in summer with 13, while ‘Camping Tent gTa’ tops spring sales at 17 units, aligning with outdoor activity in warmer weather. Tech products peak in summer and winter, with ‘Smartphone X12 Jmx’ selling 15 units in summer and ‘Eau de Toilette MZw’ becoming the winter favorite at 14 units.

```
# Filter out cancelled orders
non_cancelled_orders <- orders %>%
  filter(order_status != "cancelled")

# Calculate total quantity sold for each product
product_sales <- non_cancelled_orders %>%
  inner_join(order_details, by = "order_id") %>%
  group_by(product_id) %>%
  summarise(total_quantity_sold = sum(order_quantity, na.rm = TRUE)) %>%
  ungroup()

# join with the 'products' data frame to get the product names
product_sales_with_names <- product_sales %>%
  left_join(products, by = "product_id") %>%
  select(product_name, product_id, total_quantity_sold)

# Arrange the data by total quantity sold to find the least selling products
least_selling_products <- product_sales_with_names %>%
  arrange(total_quantity_sold) %>%
  slice_head(n = 3)
print(least_selling_products)
```

```
# A tibble: 3 x 3
```

	product_name	product_id	total_quantity_sold
	<chr>	<int>	<int>
1	The Great Adventure Lfr	4958170	1
2	Smartphone X12 qbk	13142164	1
3	The Great Adventure zve	14323341	1

In our e-commerce platform, the products ‘The Great Adventure Lfr’, ‘Smartphone X12 qbk’, ‘The Great Adventure zve’, ‘Premium Blender OaQ’ and ‘Camping Tent UeT’ have the lowest sales, with only one unit sold for each.

```
# Filter out cancelled orders
non_cancelled_orders <- orders %>%
  filter(order_status != "cancelled")

# Join non_cancelled_orders with order_details, then join with products
sales_data <- non_cancelled_orders %>%
  inner_join(order_details, by = "order_id") %>%
  inner_join(products, by = "product_id")

# Join sales data with product categories to get category names
sales_data_with_categories <- sales_data %>%
  inner_join(product_categories, by = "category_id")

# Aggregate total sales per category
# Considering the quantity sold and the product price
category_sales_totals <- sales_data_with_categories %>%
  group_by(category_name) %>%
  summarise(total_sales = sum(order_quantity, na.rm = TRUE)) %>%
  ungroup()

# Sort to find the best-selling categories
best_selling_category <- category_sales_totals %>%
  arrange(desc(total_sales)) %>%
  slice_head(n = 5)

# Print the top 5 best-selling product categories
print(best_selling_category)
```

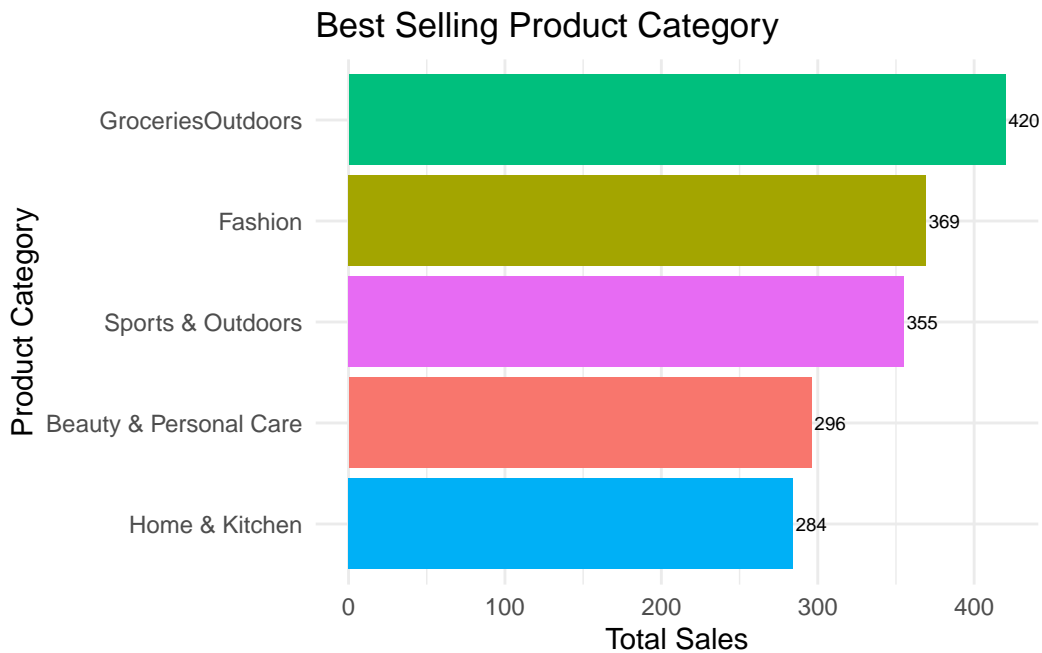
```
# A tibble: 5 x 2
  category_name      total_sales
  <chr>            <int>
1 "GroceriesOutdoors\r"      420
```

2 "Fashion\r"	369
3 "Sports & Outdoors\r"	355
4 "Beauty & Personal Care\r"	296
5 "Home & Kitchen\r"	284

```
# Plot the bar chart
best_selling_categories_plot <- ggplot(best_selling_category,
                                     aes(x = total_sales,
                                         y = reorder(category_name,
                                                       total_sales),
                                         fill = category_name)) +

  geom_bar(stat = "identity") +
  geom_text(aes(label = total_sales, y = reorder(category_name,
                                                  total_sales)),
            position = position_dodge(width = 0.5),
            hjust = -0.1,
            size = 2.5,
            color = "black") +
  labs(title = "Best Selling Product Category",
       x = "Total Sales",
       y = "Product Category") +
  theme_minimal() +
  theme(legend.position = "none")

ggsave(plot = best_selling_categories_plot,
       filename = "images/best_selling_categories_plot.png",
       width = 10, height = 6, dpi = 300)
print(best_selling_categories_plot)
```



```
# Ensure order_date is Date type and define seasons
orders <- orders %>%
  mutate(
    order_date = as.Date(order_date, format = "%Y-%m-%d"),
    season = case_when(
      month(order_date) %in% c(12, 1, 2) ~ "Winter",
      month(order_date) %in% c(3, 4, 5) ~ "Spring",
      month(order_date) %in% c(6, 7, 8) ~ "Summer",
      month(order_date) %in% c(9, 10, 11) ~ "Fall"
    )
  )

# Exclude "cancelled" orders
non_cancelled_orders <- orders %>%
  filter(order_status != "cancelled")

# Join the order details with non_cancelled_orders
# Join the order details with products
# To get category_id for each product sold
sales_data <- order_details %>%
  inner_join(non_cancelled_orders, by = "order_id") %>%
  inner_join(products, by = "product_id")

# Join the sales data with product categories to get category names
```

```
category_sales_data <- sales_data %>%
  inner_join(product_categories, by = "category_id")

# Aggregate the total sales per category per season
category_season_sales_totals <- category_sales_data %>%
  group_by(season, category_name) %>%
  summarise(total_sales = sum(order_quantity, na.rm = TRUE)) %>%
  ungroup()
```

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

```
# Find the top 5 categories with the highest total sales for each season
best_selling_categories_per_season <- category_season_sales_totals %>%
  arrange(desc(total_sales)) %>%
  group_by(season) %>%
  slice_max(order_by = total_sales, n = 5) %>%
  ungroup()

# Arrange the result in order
best_selling_categories_per_season <- best_selling_categories_per_season %>%
  arrange(season, desc(total_sales))
print(best_selling_categories_per_season)
```

```
# A tibble: 20 x 3
  season category_name      total_sales
  <chr>   <chr>             <int>
1 Fall   "GroceriesOutdoors\r"    120
2 Fall   "Fashion\r"              103
3 Fall   "Sports & Outdoors\r"    83
4 Fall   "Beauty & Personal Care\r" 80
5 Fall   "Home & Kitchen\r"       74
6 Spring "GroceriesOutdoors\r"    113
7 Spring "Sports & Outdoors\r"    103
8 Spring "Fashion\r"              99
9 Spring "Toys & Games\r"        91
10 Spring "Books\r"              90
11 Summer "GroceriesOutdoors\r"    102
12 Summer "Fashion\r"          91
13 Summer "Sports & Outdoors\r" 80
14 Summer "Electronics\r"    66
```

15	Summer	"Home & Kitchen\r"	63
16	Winter	"Beauty & Personal Care\r"	92
17	Winter	"Sports & Outdoors\r"	89
18	Winter	"GroceriesOutdoors\r"	85
19	Winter	"Home & Kitchen\r"	84
20	Winter	"Fashion\r"	76

In the fall, Groceries lead the best-selling product category with 120 units, followed by Fashion and Sports & Outdoors. In the spring, Groceries again topped the list with 113 units, while Sports & Outdoors rose to a close second with 103 units, overtaking Fashion which was at 99 units. Toys & Games and Books show strong sales in spring, at 91 and 90 units, indicating a surge in leisure and educational activities. This pattern suggests that essentials like groceries have a steady demand, while other categories like Sports & Outdoors and Books gain seasonal traction.

```
orders <- orders %>%
  mutate(
    order_date = as.Date(order_date, format = "%Y-%m-%d"),
    year = year(order_date)
  )
# Exclude "cancelled" orders
non_cancelled_orders <- orders %>%
  filter(order_status != "cancelled")

# Calculate the total sales per supplier per year
supplier_sales <- order_details %>%
  inner_join(non_cancelled_orders, by = "order_id") %>%
  inner_join(products, by = "product_id") %>%
  group_by(year, supplier_id) %>%
  summarise(total_sales = sum(order_quantity, na.rm = TRUE)) %>%
  ungroup()
```

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

```
# Convert supplier_id to character in both supplier_sales and
# suppliers before joining
supplier_sales <- supplier_sales %>%
  mutate(supplier_id = as.numeric(supplier_id))

suppliers <- suppliers %>%
```



```

mutate(supplier_id = as.numeric(supplier_id))

# Join with suppliers to get supplier names and select relevant columns
supplier_sales <- supplier_sales %>%
  inner_join(suppliers, by = "supplier_id") %>%
  select(year, supplier_name, total_sales) %>%
  arrange(year, desc(total_sales))

# For each year, identify the best-selling suppliers
best_selling_suppliers_per_year <- supplier_sales %>%
  group_by(year) %>%
  slice_max(order_by = total_sales, n = 1) %>%
  ungroup()
print(best_selling_suppliers_per_year)

```

```

# A tibble: 4 x 3
  year supplier_name      total_sales
  <dbl> <chr>             <int>
1  2020 "Martin-Barnett\r"      15
2  2021 "Archer LLC\r"       11
3  2022 "Harvey-Hines\r"     17
4  2023 "Perkins Mathews and Ray\r" 15

```

The top supplier sales per year show ‘Martin-Barnett’ leading in 2020 with 15 sales, a dip to 11 sales with ‘Archer LLC’ in 2021, ‘Harvey-Hines’ at 17 sales among suppliers in 2022, and a return to 15 sales with ‘Perkins Mathews and Ray’ in 2023.

```

# Ensure order_date is Date type and define seasons
orders <- orders %>%
  mutate(
    order_date = as.Date(order_date, format = "%d/%m/%Y"),
    # Adjust format as necessary
    season = case_when(
      month(order_date) %in% c(12, 1, 2) ~ "Winter",
      month(order_date) %in% c(3, 4, 5) ~ "Spring",
      month(order_date) %in% c(6, 7, 8) ~ "Summer",
      month(order_date) %in% c(9, 10, 11) ~ "Fall"
    )
  )

# Exclude "cancelled" orders

```

```

non_cancelled_orders <- orders %>%
  filter(order_status != "cancelled")

# Calculate the total sales per supplier per season
seasonal_supplier_sales <- order_details %>%
  inner_join(non_cancelled_orders, by = "order_id") %>%
  inner_join(products, by = "product_id") %>%
  group_by(season, supplier_id) %>%
  summarise(total_sales = sum(product_quantity, na.rm = TRUE),
            .groups = "drop") %>%
  ungroup()

# Join with the suppliers dataframe to get the supplier names
seasonal_supplier_sales <- seasonal_supplier_sales %>%
  inner_join(suppliers, by = "supplier_id") %>%
  select(season, supplier_name, total_sales) %>%
  arrange(season, desc(total_sales))

# For each season, find the best-selling suppliers
best_selling_suppliers_per_season <- seasonal_supplier_sales %>%
  group_by(season) %>%
  slice_max(order_by = total_sales, n = 1) %>%
  ungroup()

print(best_selling_suppliers_per_season)

```

```

# A tibble: 4 x 3
  season supplier_name      total_sales
  <chr>   <chr>              <int>
1 Fall   "King  Watkins and Bowersa\r"      839
2 Spring "King  Watkins and Bowersa\r"    1678
3 Summer "Moon  Gillespie and Vargas\r"   402
4 Winter "King  Watkins and Bowersa\r"    1678

```

## Conclusion and Future Improvements:

Our analysis of the WBS e-commerce platform from 2020 to 2023 revealed a slight preference for PayPal and identified Belfast as an unexpected frontrunner in sales, surpassing major economic centers like London and Manchester. To counter the trend of declining revenue a multifaceted approach is needed. This includes product discounts, product diversification,

and the enhancement of payment options to spur future growth. The analysis of quarterly revenue and seasonal sales underscores the necessity of adaptive strategies to navigate the cyclical nature of sales effectively, optimizing revenue generation year-round. Moreover, exploring the unique revenue trends and market dynamics of each city is essential for creating tailored strategies that meet regional demands and consumer preferences. Overall, our detailed report highlights the critical improvement on market segmentation and data-informed decision-making to foster long-term expansion in the market.