Data Management Group Assignment

DM Group 5

Contents

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. After analysis of the ER diagram, to efficiently define the data held within shipping address attribute, it was given further attributes and used as a composite attribute for order details.

   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.

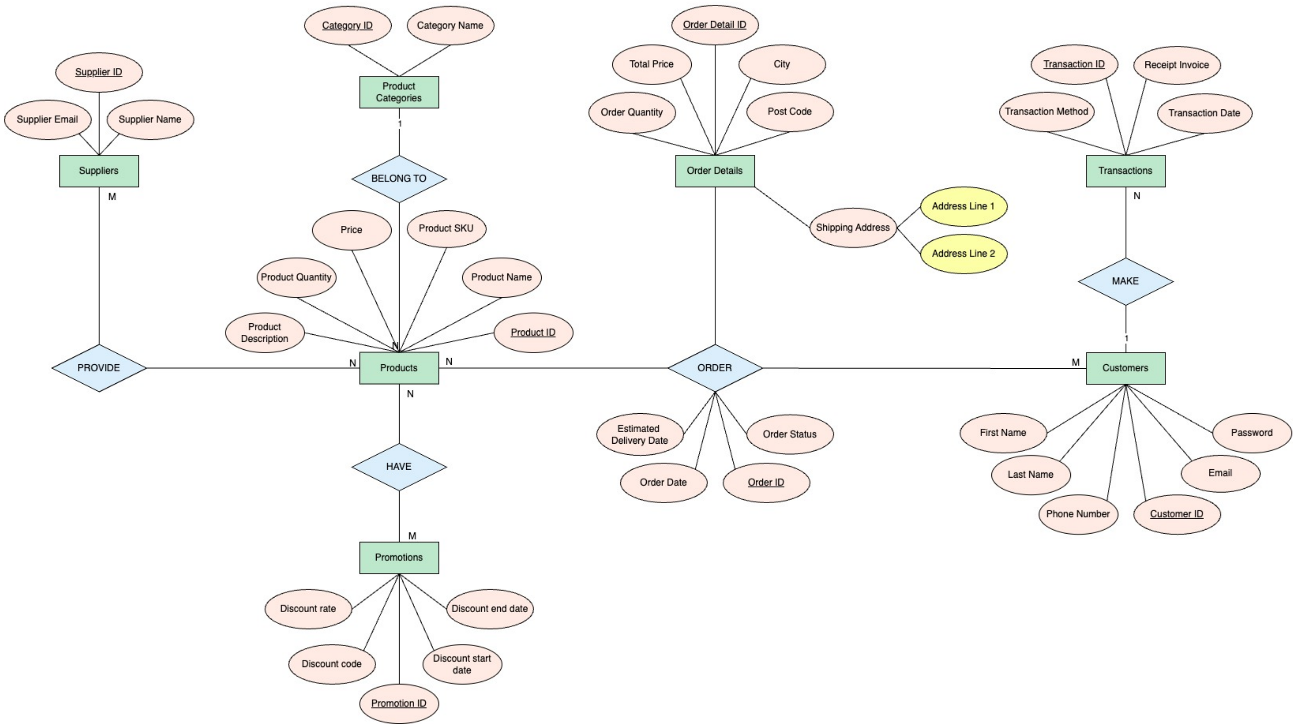


Figure 1: Entity-Relationship Diagram

2. Relationship Sets:

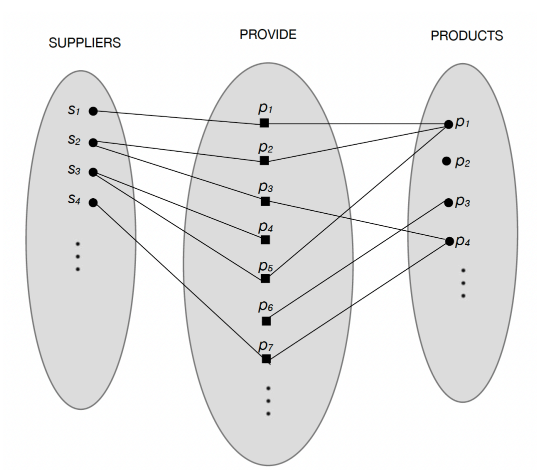


Figure 2: Suppliers PROVIDE products. Many-to-many relationship (M:N).

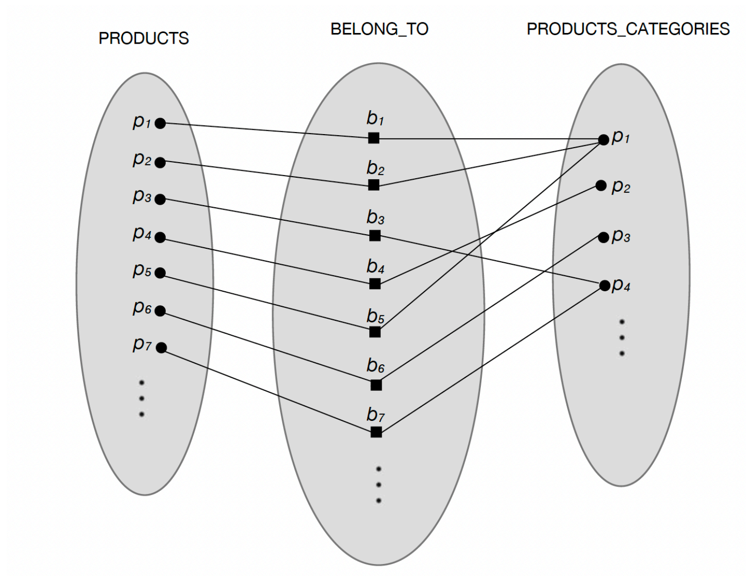


Figure 3: Products BELONG TO Product Categories. Many-to-one relationship (N:1).

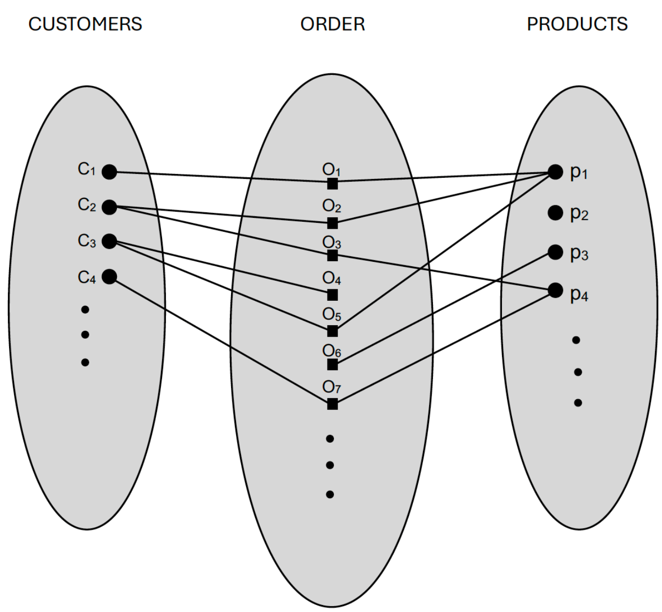


Figure 4: Customers ORDER products. Many-to-many relationship (M:N).

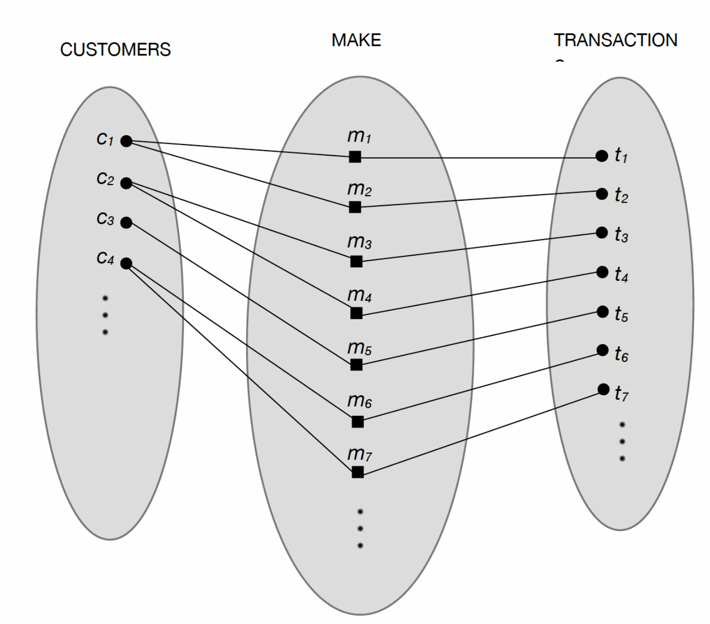


Figure 5: Customers MAKE Transaction. Many-to-many relationship (M:N).

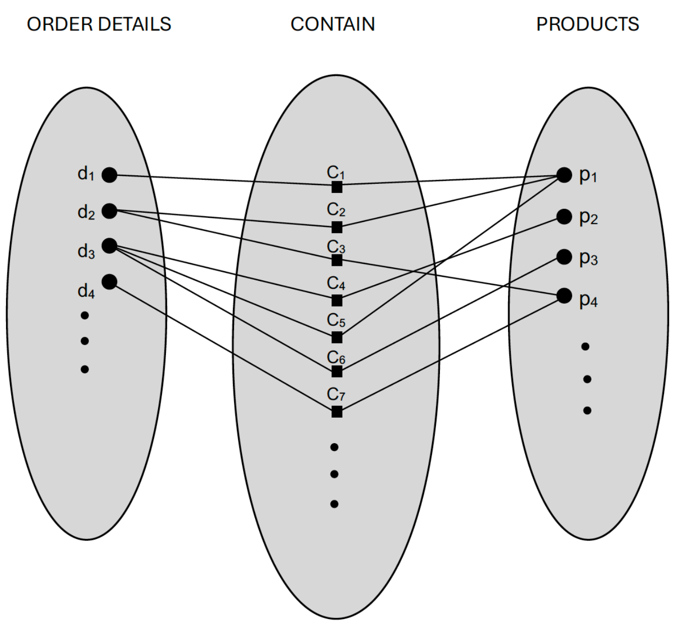


Figure 6: Order Details CONTAIN Products. One-to-many relationship (1:N).

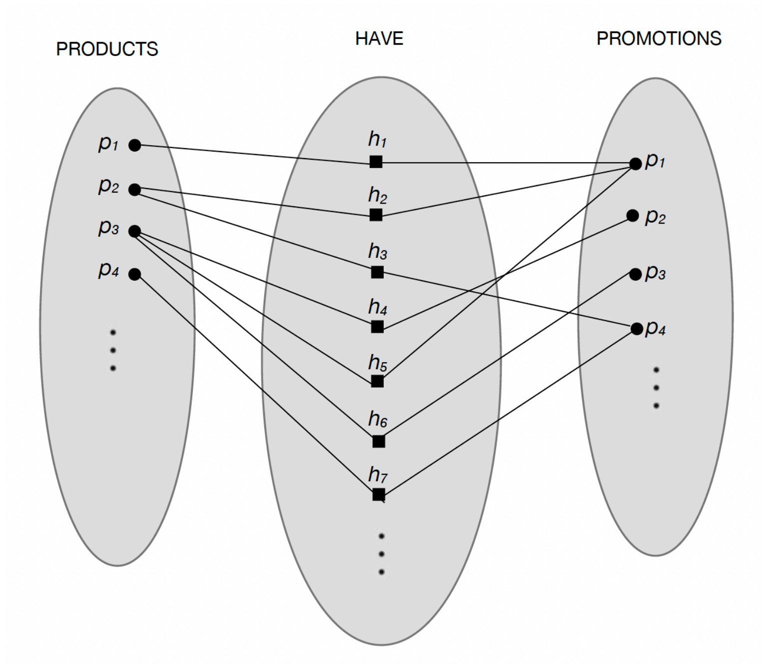


Figure 7: Products HAVE Promotions. Many-to-many relationship (N:M).

1.2 SQL Database Schema Creation:

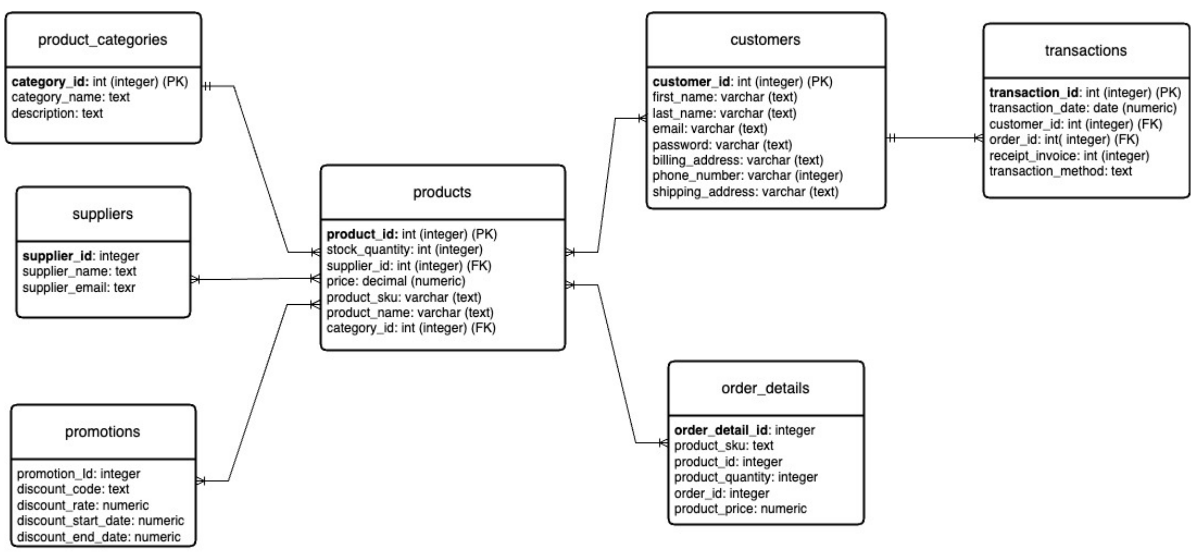


Figure 8: Logical Database Schema

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

Entity	Attribute	Description
	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
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

Entity	Attribute	Description
	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 Delivery address of the order
	city	City location of the order
	postcode	Postal code of the location

««< HEAD +—————+————————+———————————-+ | Relationship | Attribute | Description | +:=============+:=======================+:========
| Order | order_id | Primary Key | | | | | | | | Identifier for each unqiue 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 |
+—————+————————+———————————-+ ======= +—————
+————————+———————————-+ | 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 | +————-+——————–+——————————-+ »»»> 6eaaadec5a613392b7972d187dd98ade8344a16f

: Relationship Attribute Data Dictionary

Data Generation and Management

2.1 Synthetic Data Generation:

To generate synthetic data for our e-commerce database, 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. See the following prompt:

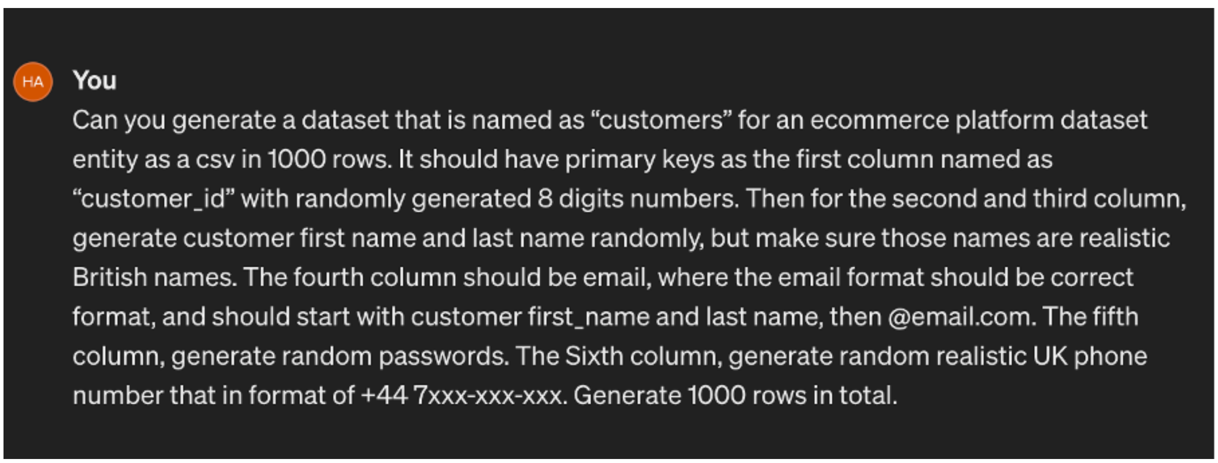


Figure 10: Customers Prompt.

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

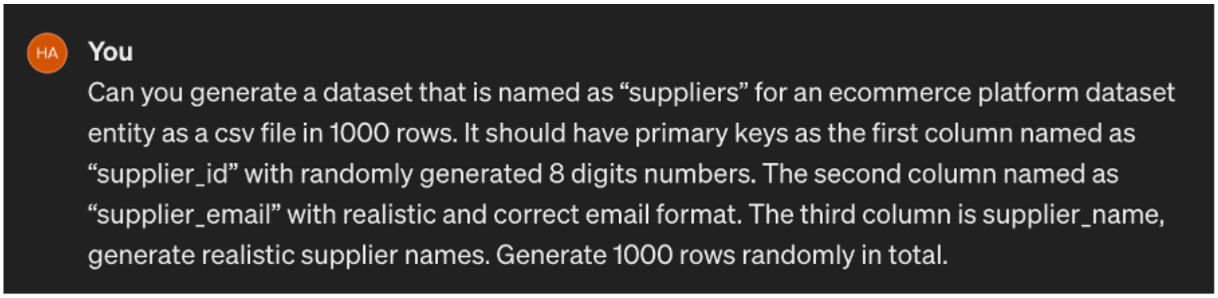


Figure 11: Suppliers Prompt.

Product Categories:
The prompt is for the generation of "Product Categories", with 8 unique category identifiers and corresponding names across 8 entries. See the following prompt:

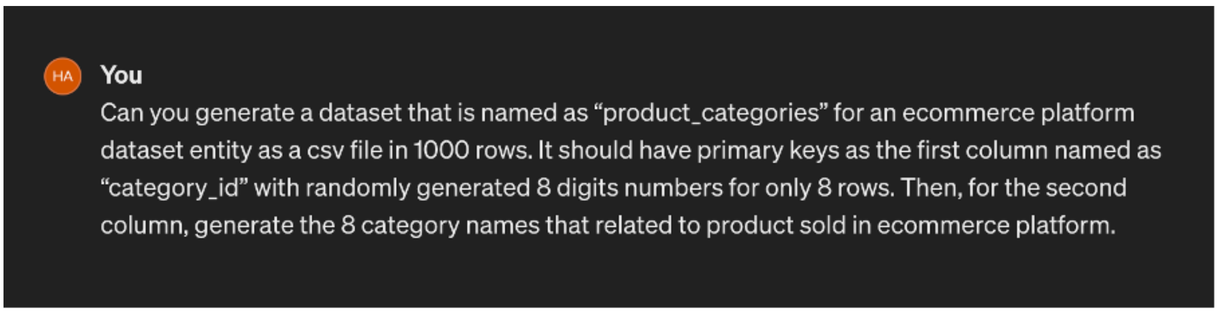


Figure 12: Product Categories Prompt.

Promotions:
For the "Promotions" entity, we focused on generating realistic discount codes and rates, alongside logically constrained start, and end dates for the discounts. See the following prompt:

> HA **You**
> Can you generate a dataset that is named as "Promotions" for an ecommerce platform dataset entity as a csv file in 1000 rows. It should have primary keys as the first column named as "promotion_id" with randomly generated 8 digits numbers, the second and third column should be discount_start date and discount_end_date, dates in format of DD/MM/YYYY, the discount period should only last for less than 30 days, and the discount end date should be after the discount start date. The fourth column should be discount_code, make it realistic and consist with letters. The fifth column should be discount_rate, which it can only be less than 20%, make it in percentage format. Generate 1000 rows in total.

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. See the following prompt:

> HA **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, necessitating their simultaneous generation 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. See the following prompt:

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. See the following prompt:

You

Create a "transactions" dataset for an e-commerce platform with a primary key named "transaction_id" consisting of unique 6-digit numbers as csv file in 629 rows. Each transaction must include the "customer_id" and "order_id" from the existing Customers and Orders datasets, respectively but need only 629 rows. Generate a "receipt_invoice" number for each entry by using this format - 3characters-8digits. The "transaction_method" should be limited to "Credit Card", "Bank Transfer", or "PayPal". Ensure the "transaction_date" aligns with the corresponding "order_date" from the order dataset.

Figure 16: Transactions Prompt.

The following 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

After uploading the data files, the rows and columns of each dataset was displayed using the below code:

```
data_files <- list.files("data_uploads/Dataset/")

suffix <- "-Table 1"

# Rename files
for (file in data_files) {
  # Create a new filename
  new_filename <- paste0("data_uploads/Dataset/", gsub(suffix, "", file))
  file <- paste0("data_uploads/Dataset/", file)
  # Rename the file
  file.rename(from = file, to = new_filename)
}
```

To import the data sets into the SQLite database, we used the below code:

```
data_files <- list.files("data_uploads/Dataset/")

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

# To display the rows and columns of each dataset and
# To import each csv file into the database table
for (file in data_files) {
  this_filepath <- paste0("data_uploads/Dataset/", file)
  this_file_contents <- readr::read_csv(this_filepath)

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

  #To print the number of columns and rows of each dataset
  print(paste0("The file: ",file,
               " has: ",
               format(number_of_rows,big.mark = ","),
               " rows and ",
               number_of_columns," columns"))

  table_name <- gsub(".csv","",file)

  #Writing the csv file contents to the database and
```

```
  #creating the table with the table_name
  RSQLite::dbWriteTable(db_connection,table_name,this_file_contents,overwrite=TRUE)

  #To list the database tables
  RSQLite::dbListTables(db_connection)
}

RSQLite::dbDisconnect(db_connection)
```

Populate SQL database:
While constructing the schema for our SQL database, we focused on defining the data types and structures that would best represent the nature of each field. Data types such as VARCHAR were chosen to impose limits on the length of text-based fields. Meanwhile, for numerical fields, INT and DECIMAL types were used to accurately capture integer and floating-point numbers, respectively. By carefully aligning data types with the intended data structure, we create a framework that ensures data is stored in an organised manner.

Customers Data Validation Code:

In the above code snippets, referential integrity is effectively 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, such as 'products', which references 'product_categories' and 'suppliers'; and 'transactions', which links back to 'customers' and 'orders'. These foreign key constraints ensure that relationships between tables are strictly maintained, preventing orphaned records and preserving the logical integrity of the database schema. Each foreign key declaration not only enforces referential integrity but also delineates the relational architecture of the database, illustrating a well-structured and coherent relationship model among the data entities.

Data Validation-Quarto:
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.

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.

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 (+44 7XXX-XXX-XXX). Results suggest the correctness of each phone number, confirming that they align with the format.

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.

Email Format Verification:
We executed a script to ensure that email addresses within out data set follow a certain format of firstname.lastname@email.com. The test ensures that all email formats are correct following the specified pattern.

Data Pipeline Generation

3.1 Github Repository and Workflow Setup

ETL (Extract, transform, load)

```
#install.packages("dplyr")
#install.packages("ggplot2")
#install.packages("tidyr")
#install.packages("lubridate")
#install.packages("readxl")
#install.packages("scales")

library(dplyr)
```

```

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```r
library(ggplot2)
library(tidyr)
library(lubridate)
```

```
Attaching package: 'lubridate'

The following objects are masked from 'package:base':

    date, intersect, setdiff, union
```

```r
library(readxl)
library(scales)
```

3.2 Github Actions for Continuous Integration

Data Analysis

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

In our ecommerce platform, we extract key characteristics such as total orders, total revenue, best-selling products, and least-selling products, along with identifying top-selling suppliers. We analyze these results across different time series, including seasonally and quarterly, to uncover patterns and trends. Additionally, we examine total revenue by region and assess transaction methods used by various customers. This comprehensive analysis allows us to generate more valuable insights, enhancing our understanding of market dynamics and customer preferences.

```r
# Reading required database
library(readr)
```

```
Attaching package: 'readr'

The following object is masked from 'package:scales':

    col_factor
```

```r
library(dplyr)
customers <- read_csv("data_uploads/customers.csv")
```

```
Rows: 1000 Columns: 6

-- Column specification --------------------------------------------------------
Delimiter: ","
chr (6): customer_id, first_name, last_name, email, password, phone

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
order_details <- read_csv("data_uploads/order_details.csv")
```

```
Rows: 1000 Columns: 8
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (6): order_detail_id, order_id, product_id, shipping_Address, city, post...
dbl (2): product_price, order_quantity

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
orders <- read_csv("data_uploads/orders.csv")
```

```
Rows: 629 Columns: 6
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (3): estimated_delivery_date, order_date, order_status
dbl (3): order_id, customer_id, total_price

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
product_categories <- read_csv("data_uploads/product_categories.csv")
```

```
Rows: 8 Columns: 2
-- Column specification --------------------------------------------------------
Delimiter: ","
```

```
chr (2): category_id, category_name

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
products <- read_csv("data_uploads/products.csv")
```

```
Rows: 1000 Columns: 8
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (6): product_id, category_id, supplier_id, product_name, product_sku, pr...
dbl (2): price, product_quantity

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
promotion <- read_csv("data_uploads/promotions.csv")
```

```
Rows: 1000 Columns: 5
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (5): promotion_id, discount _start_date, discount_end_date, discount_cod...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
suppliers <- read_csv("data_uploads/suppliers.csv")
```

```
Rows: 1000 Columns: 3
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (2): supplier_email, supplier_name
dbl (1): supplier_id

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
transactions <- read_csv("data_uploads/transactions.csv")
```

```
Rows: 629 Columns: 6
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (3): receipt_invoice, transaction_method, transaction_date
dbl (3): transaction_id, customer_id, order_id

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

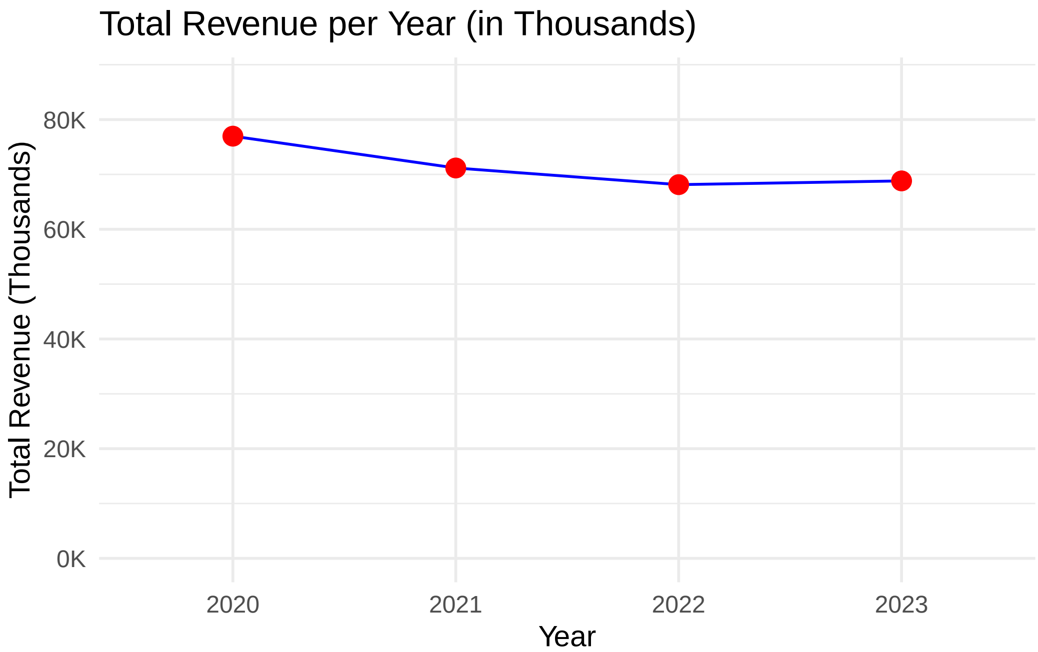
```
#Standardise date format
orders$order_date <- as.Date(orders$order_date, format= "%d/%m/%Y")
orders$estimated_delivery_date <- as.Date(orders$estimated_delivery_date, format = "%d/%m/%Y
```

```
# Exclude "cancelled" orders and calculate total revenue per year
orders <- orders %>%
  mutate(
    order_id = as.character(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        76972.
2  2021        71169.
3  2022        68133.
4  2023        68817.
```

```
# 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, gr
  geom_line(color = "blue") +
```

```
  geom_point(color = "red", size = 3) +
  labs(title = "Total Revenue per Year (in Thousands)", x = "Year", y = "Total Revenue (Thous
  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
ggsave(plot = yearly_revenue_plot, filename = "images/yearly_revenue_plot.png", width = 10, h
print(yearly_revenue_plot)
```

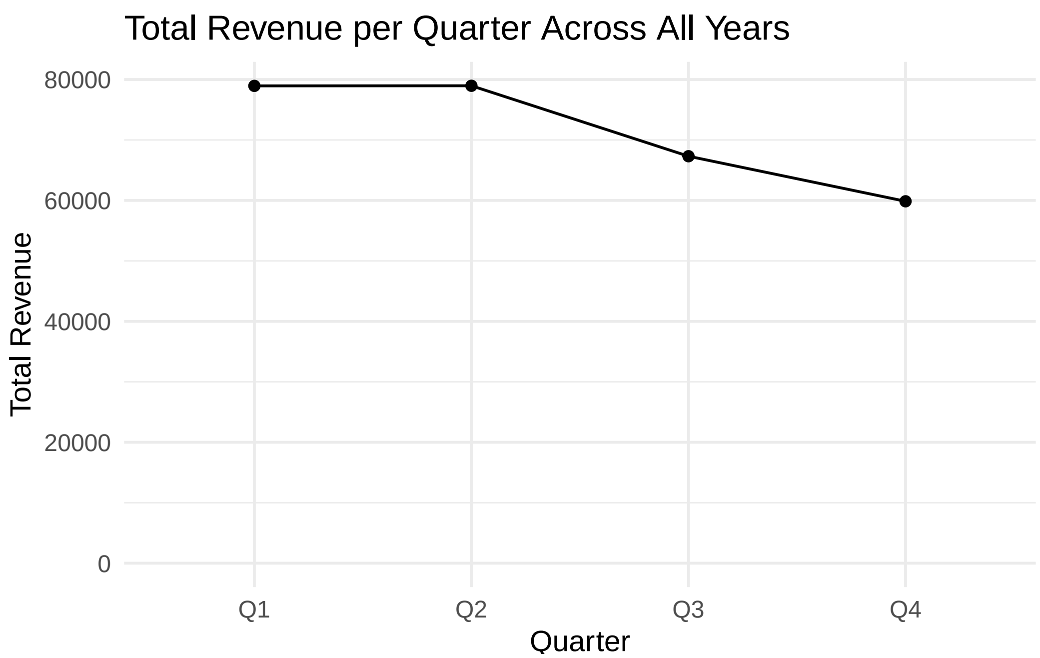


A continuously decreasing pattern is observed in total revenues per year between 2020 and 2023. Starting at a peak in 2020 with a total revenue of more than £90,000, there is a noticeable year-over-year decrease through to 2023, where the revenue has its lowest value at around £74,000.

```
# 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             78950.
2 Q2             78966.
3 Q3             67314.
4 Q4             59860.
```

```
quarterly_revenue_all_years_plot <- ggplot(quarterly_revenue, aes(x = quarter, y = total_reve
  geom_line() +
  geom_point() +
  labs(title = "Total Revenue per Quarter Across All Years", x = "Quarter", y = "Total Revenu
  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_year
print(quarterly_revenue_all_years_plot)
```



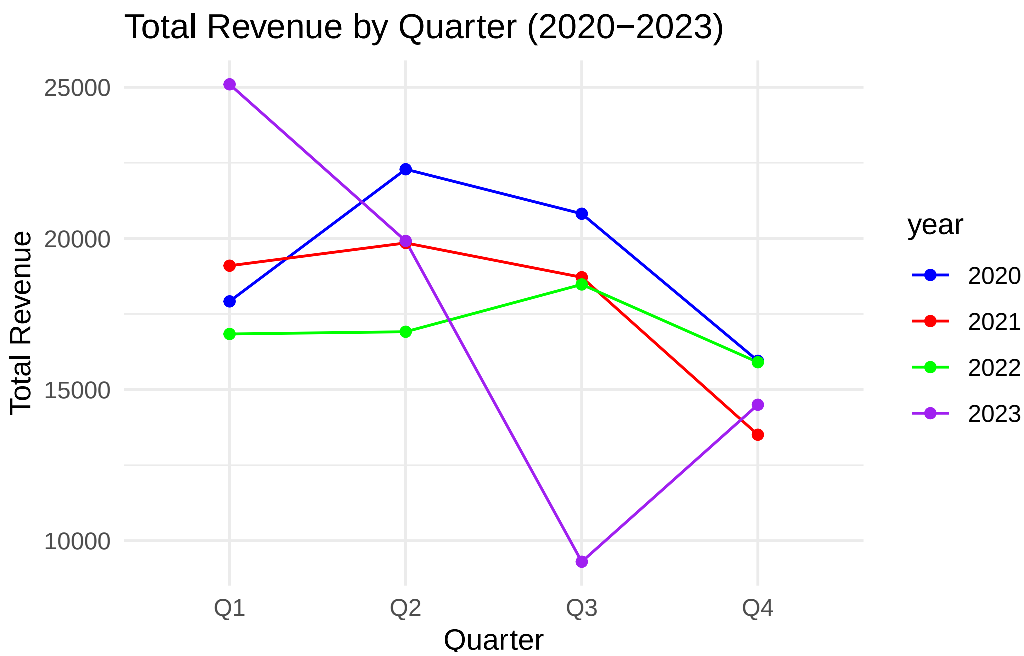
The figure displaying the total revenue per quarter across all years (2020-2023) shows that the sum of the total revenue in all years varies across the quarters, with the highest total revenue recorded in Q2 and the lowest in Q4. It is observed that Q1 has a relatively high total revenue overall, continued 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.

```r
# 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              17917.
 2 2020Q2              22287.
 3 2020Q3              20816.
 4 2020Q4              15951.
 5 2021Q1              19097.
 6 2021Q2              19850.
 7 2021Q3              18715.
 8 2021Q4              13507.
 9 2022Q1              16839.
10 2022Q2              16912.
11 2022Q3              18477.
12 2022Q4              15904.
13 2023Q1              25097.
14 2023Q2              19916.
15 2023Q3               9305.
16 2023Q4              14499.
```

```r
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 quar


# Plot the line chart for quarterly revenue for each year
quarterly_revenue_plot <- ggplot(detailed_quarterly_revenue, aes(x = quarter, y = total_reve
  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" = "
```

```
  theme_minimal()

ggsave(plot = quarterly_revenue_plot, filename = "images/quarterly_revenue_plot.png", width =
print(quarterly_revenue_plot )
```



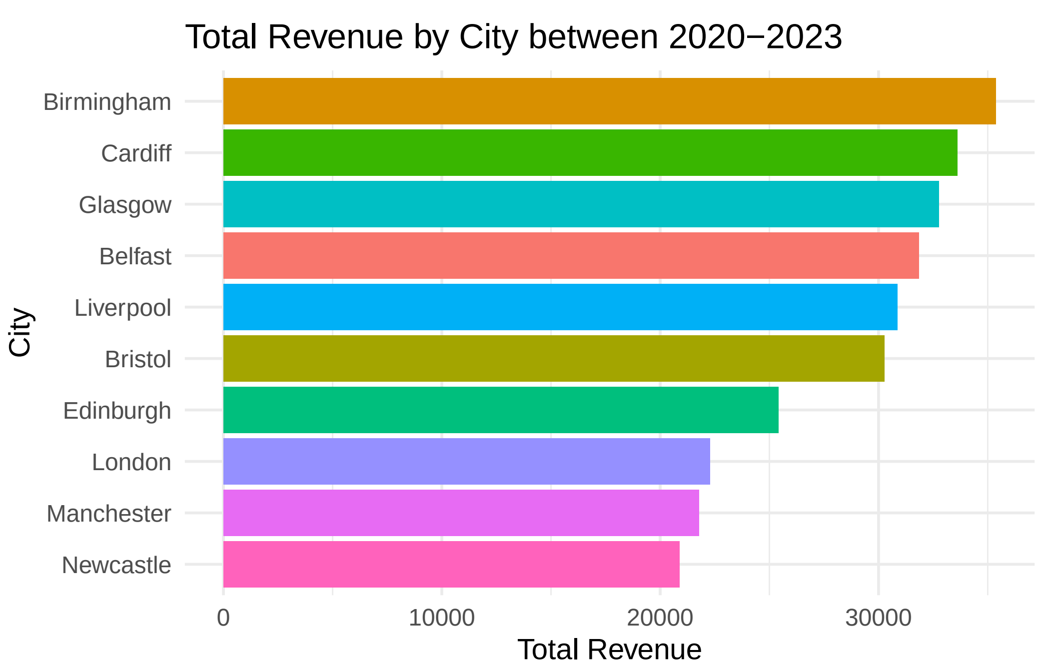
Analysing the total revenue trends from 2020 to 2023, a recurrent increase is observed from Q1 to Q2 each year except in 2023, where a drop is observed from around £25,000 to £20,000. The transition from Q2 to Q3 in 2022 is marked by a significant rise from around £18,000 to £25,000, contrasted by 2023's 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
```

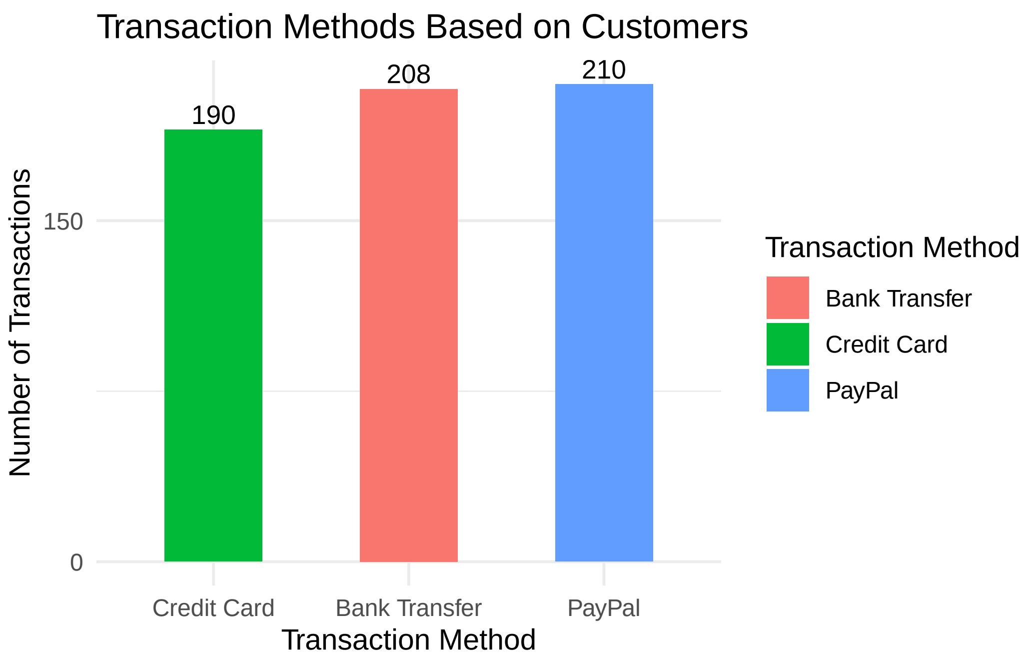
```
   city       total_revenue
   <chr>              <dbl>
 1 Belfast           31856.
 2 Birmingham        35378.
 3 Bristol           30259.
 4 Cardiff           33599.
 5 Edinburgh         25415.
 6 Glasgow           32766.
 7 Liverpool         30870.
 8 London            22275.
 9 Manchester        21785.
10 Newcastle         20887.
```

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



Analysing the total revenue across different cities, Belfast emerges on the top with approximately £41,000, while the other cities exhibit varied results. Cardiff and Birmingham also showcase strong revenues, with around £39,000 and £36,000, respectively. 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.

```r
# Excluding the cancelled orders and calculate the numbers for each transaction method used
transactions <- transactions %>%
  mutate(order_id = as.character(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,
  geom_col(width = 0.5) +
  geom_text(aes(label = number_of_transactions), vjust = -0.3, color = "black", size = 3.5) +
  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", wi
print(transaction_methods_plot)
```



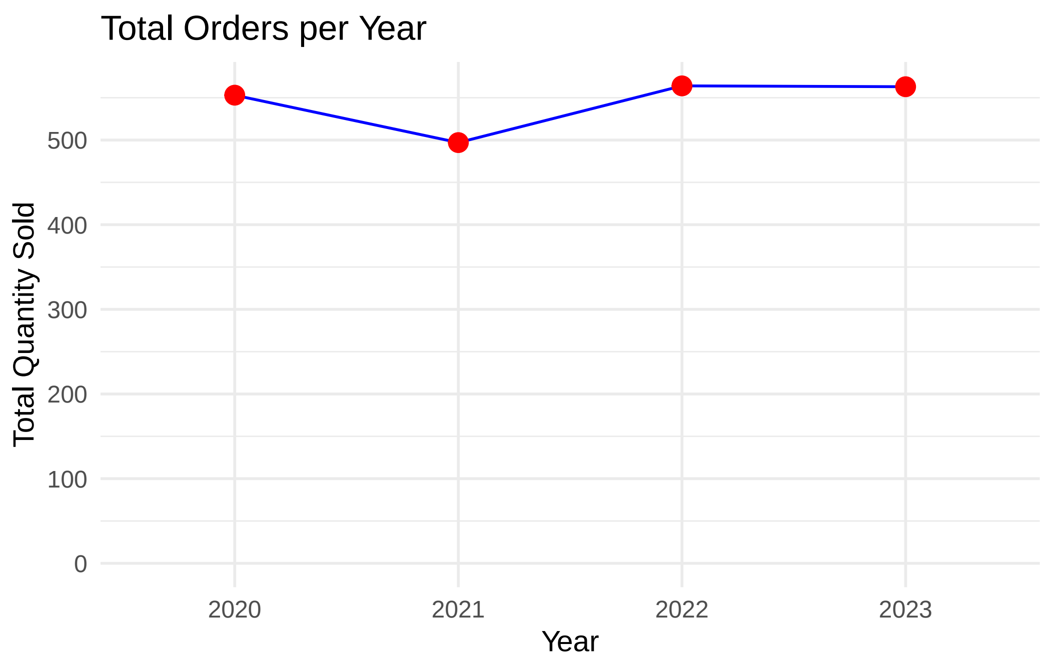
Transaction data from our e-commerce platform reveals a balanced preference for payment methods. PayPal leads marginally with 210 transactions, followed by bank transfers at 208 and credit cards at 190.

```
# Exclude "cancelled" orders and calculate the toal 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 qua
print(yearly_orders_quantity)
```

```
# A tibble: 4 x 2
   year total_quantity
  <dbl>          <dbl>
1  2020            553
2  2021            497
3  2022            564
4  2023            563
```

```
# Plotting the total orders per year
yearly_orders_plot <- ggplot(yearly_orders_quantity, aes(x = as.factor(year), y = total_quan
```

```r
  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, hei
print(yearly_orders_plot)
```



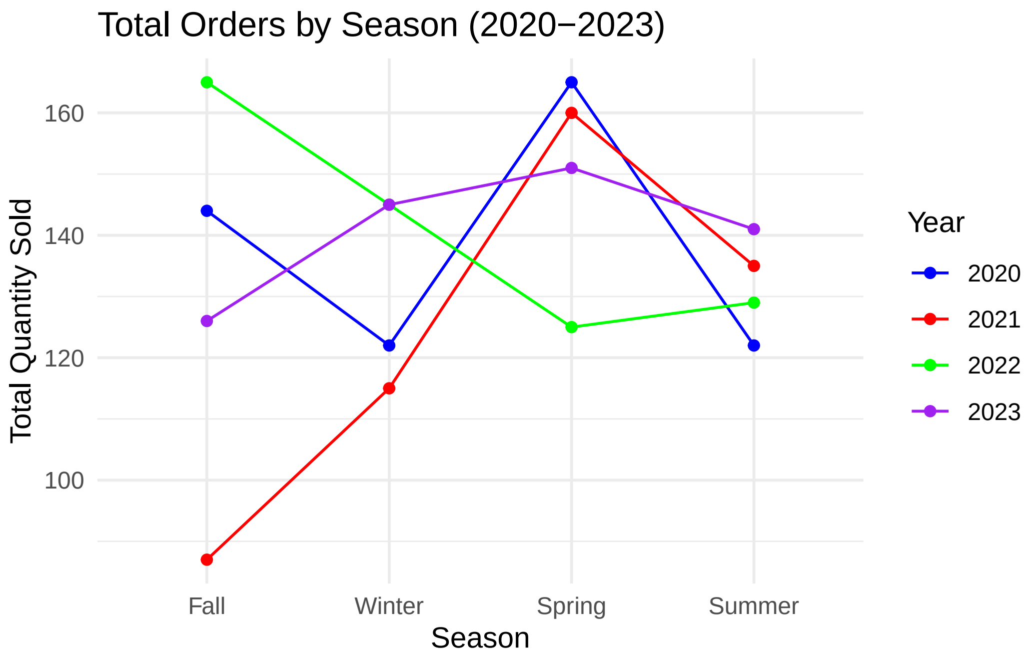
Total order numbers has the lowest value in 2021 and peak value in 2022, starting with a slight decline from 627 orders in 2020 to 567 in 2021, followed by a significant increase to 643 orders in 2022, and a subsequent decrease to 631 orders in 2023.

```r
# 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 qua
print(detailed_seasonal_orders_quantity)
```

```
# A tibble: 16 x 3
    year season total_quantity
   <dbl> <fct>           <dbl>
 1  2020 Fall              144
 2  2020 Winter            122
 3  2020 Spring            165
 4  2020 Summer            122
 5  2021 Fall               87
 6  2021 Winter            115
 7  2021 Spring            160
 8  2021 Summer            135
 9  2022 Fall              165
10  2022 Winter            145
11  2022 Spring            125
12  2022 Summer            129
13  2023 Fall              126
14  2023 Winter            145
15  2023 Spring            151
16  2023 Summer            141
```

```
# Plotting the  seasonal orders for each year
seasonal_orders_plot <- ggplot(detailed_seasonal_orders_quantity, aes(x = season, y = total_
  geom_line() +
  geom_point() +
  scale_color_manual(values = c("2020" = "blue", "2021" = "red", "2022" = "green", "2023" = "
                     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
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 681, suggesting a seasonal peak at that time. 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. Conversely, the summer of 2023 saw a decline to 144 orders, substantially lower than the previous year's 158, indicating a reduction in demand or the impact of other variables.

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

# Calculate total quantity sold for each product and 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 %>%
```

```r
  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: 5 x 2
  product_name          total_quantity_sold
  <chr>                               <dbl>
1 Yoga Mat WZQ                           36
2 Yoga Mat 6ne                           33
3 Whole Grain Bread 4zv                  33
4 Eau de Toilette MZw                    32
5 Premium Blender vXj                    27
```

```r
best_selling_products_plot <- ggplot(top_selling_products_with_names, aes(x = reorder(produc
  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
  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"
print(best_selling_products_plot)
```

The top 5 selling products are ranked as follow: "Whole Grain Bread 4zv" at 38 units, followed by "Yoga Mat WZQ" at 36, "The Great Adventure EIO" at 35, "Yoga Mat 6ne" at 33, and both "Organic Honey H4w" and "Eau de Toilette MZw" at 32 units each, showcasing a varied consumer interest across food, fitness, and personal care items.

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

```r
  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 the 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>                                  <dbl>
 1 Fall   Whole Grain Bread 4zv                     16
 2 Fall   Summer Dress Dzb                          13
 3 Fall   Yoga Mat 6ne                              12
 4 Fall   The Great Adventure El0                   11
 5 Fall   Hydrating Face Cream 7Y6                  10
 6 Spring Camping Tent gTa                          13
 7 Spring Premium Blender vXj                       12
 8 Spring Organic Honey WKG                         10
 9 Spring Cooking 101 Ttm                            9
10 Spring Leather Wallet 4e6                         9
11 Summer Smartphone X12 Jmx                        15
12 Summer Leather Wallet 509                        14
13 Summer Laptop Pro 15 mgo                         12
14 Summer Yoga Mat WZQ                              12
15 Summer Organic Honey s6H                         10
16 Winter Eau de Toilette MZw                       14
17 Winter Yoga Mat WZQ                              13
18 Winter Smartphone X12 eDX                        11
19 Winter Smartphone X12 xlB                        11
20 Winter Eau de Toilette Jie                       10
```

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. "Organic Honey" is a fall staple, with two varieties selling 13 and 11 units respectively, 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. These patterns suggest a blend of steady demand for consumables and seasonal shifts towards outdoor gear and personal electronics.

```r
# 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>                    <chr>                    <dbl>
1 Smartphone X12 qbk       13142164                     1
2 The Great Adventure zve 14323341                     1
3 Premium Blender OaQ      17434305                     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.

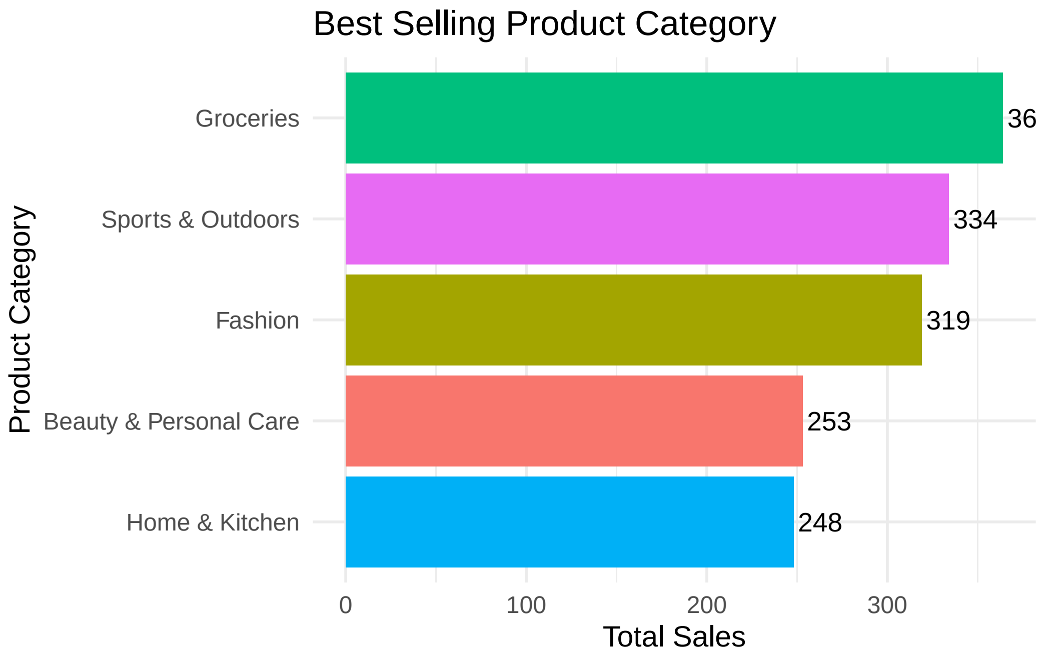
```r
# Filter out cancelled orders
non_cancelled_orders <- orders %>%
  filter(order_status != "cancelled")
```

```r
# 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>                        <dbl>
1 Groceries                      364
2 Sports & Outdoors              334
3 Fashion                        319
4 Beauty & Personal Care         253
5 Home & Kitchen                 248
```

```r
# Plot the bar chart
best_selling_categories_plot <- ggplot(best_selling_category, aes(x = total_sales, y = reorde
  geom_bar(stat = "identity") +
  geom_text(aes(label = total_sales, y = reorder(category_name, total_sales)),
            position = position_dodge(width = 0.9),
            hjust = -0.1,
            size = 3.5,
            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.p
print(best_selling_categories_plot)
```



Fashion leads as the top category with 268 total sales, closely followed by Groceries at 265. Sports & Outdoors is also popular with 245 sales. Beauty & Personal Care and Home & Kitchen trail with 208 and 202 sales, respectively.

```
# 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"
    )
  )
```

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

#Join the order details with non_cancelled_orders and products to get category_id for each pr
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.
```

```r
# 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>                        <dbl>
 1 Fall   Groceries                       96
 2 Fall   Fashion                         80
 3 Fall   Sports & Outdoors               73
```

```
 4 Fall   Beauty & Personal Care          67
 5 Fall   Books                           60
 6 Spring Sports & Outdoors               97
 7 Spring Groceries                       95
 8 Spring Fashion                         93
 9 Spring Toys & Games                    81
10 Spring Books                           80
11 Summer Groceries                       95
12 Summer Fashion                         81
13 Summer Sports & Outdoors               79
14 Summer Electronics                     65
15 Summer Home & Kitchen                  58
16 Winter Sports & Outdoors               85
17 Winter Home & Kitchen                  82
18 Winter Beauty & Personal Care          81
19 Winter Groceries                       78
20 Winter Fashion                         65
```

In the fall, Groceries lead the best-selling product category with 120 units, followed by Fashion at 103, and Sports & Outdoors at 83 units. Beauty & Personal Care and Home & Kitchen also perform well with 80 and 74 units respectively. In the spring, Groceries again top the list with 113 units, while Sports & Outdoors rise to a close second with 103 units, overtaking Fashion which is 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.

```r
# Ensure order_date is of Date type and extract year
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") %>%
```

```r
  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.
```

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

suppliers <- suppliers %>%
  mutate(supplier_id = as.character(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()

# Display the best-selling suppliers per year
print(best_selling_suppliers_per_year)
```

```
# A tibble: 5 x 3
   year supplier_name             total_sales
  <dbl> <chr>                           <dbl>
1  2020 Martin-Barnett                     15
2  2021 Spencer  Olson and Turner          10
3  2021 Herrera-Miller                     10
4  2022 Harvey-Hines                       17
5  2023 Perkins  Mathews and Ray           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, a Harvey-Hines at 17 sales among suppliers in 2022, and a return to 15 sales with 'Perkins Mathews and Ray' in 2023.

```r
# 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>                            <dbl>
```

```
1 Fall   Dyer  Pena and Johnson           312
2 Spring Garcia  Taylor and Berry         231
3 Summer Moon  Gillespie and Vargas       402
4 Winter Glover-Russell                   279
```

Across the seasons from 2020 to 2023, 'Dyer Pena and Johnson' was the top supplier in both fall with 390 sales, while 'Casey Group' led in the spring with the highest sales at 388, and 'Moon Gillespie and Vargas' was the top in summer with 402 sales, lastly in the winter 'Glover-Russell'lead with the total sales of 279.

Conclusion and Future Improvements:

Our analysis of the WBS e-commerce platform from 2020 to 2023 reveals a downward trend in annual revenue, with seasonal and quarterly variations. Various analysis conducted on the data on subjects such as on transaction methods and leading cities in revenue generation, revealed a slight preference for PayPal and identified Belfast as an unexpected frontrunner in sales, surpassing major economic centers like London and Manchester. The fluctuating demand across seasons and the identification of best and least-selling products offer strategic insights for the company.To counter the trend of declining revenue and capitalize on market dynamics, a multifaceted approach is needed. This includes product discounts, product diversification, and the enhancement of payment options to spur future growth. Additionally, a comprehensive market analysis should be undertaken to reassess product offerings and explore new market segments. 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 address e-commerce challenges effectively and foster long-term expansion in the market.