Database Design Based on Zara's Workflow

Table of contents

Introduction

This report is an extensive commentary on our undertaken project to simulate Zara's real-world e-commerce environment. The project has been segmented into four such tasks: (1) Database design and implementation, (2) Data generation and management, (3) Data pipeline generation, and (4) Data analysis and reporting with Quarto in R. Each of our segmented tasks have further been divided into subtasks, we aim to execute, to help properly encapsulate a real-world e-commerce data environment.

Part 1 Database Design and Implementation

1.1 E-R Diagram Design

The ecommerce database design discussed focuses solely on the core workflow of the ecommerce store, excluding features like favorite lists and saved shipping addresses. Operational assumptions were made regarding international shipping, the entry of shipping addresses with each order, per-product sales discounts, a 2-day payment settlement period, a 30-day refund window, and a no-exchange policy for purchased products.

The finalized database comprises 8 entities: Customer, Product, Product Category, Supplier, Sale, Invoice, Payment, and Refund. Figure 1 below is the ER diagram which illustrates their attributes and database relationships, associations between entities are detailed in the relationship sets shown in figures 2-7.

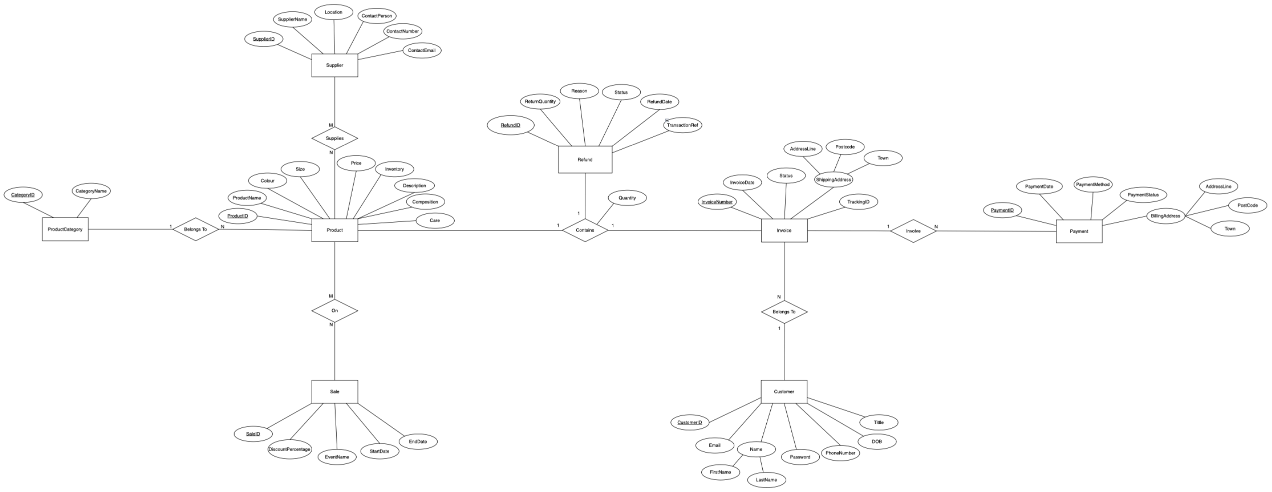


Figure 1: ER Diagram Based on Zara's Workflow

Figure 2 represents the one-to-many relationship between Customer and Invoice, where a customer may have multiple invoices, but each invoice belongs to only one customer.

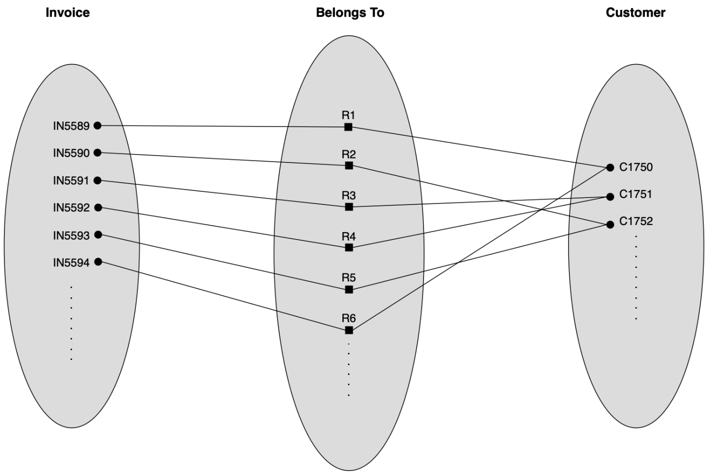


Figure 2: Relationship Set of Invoice and Customer

Figure 3 depicts the one-to-many relationship between Invoice and Payment, where an invoice may involve multiple payments, with each payment record linked to only one invoice.

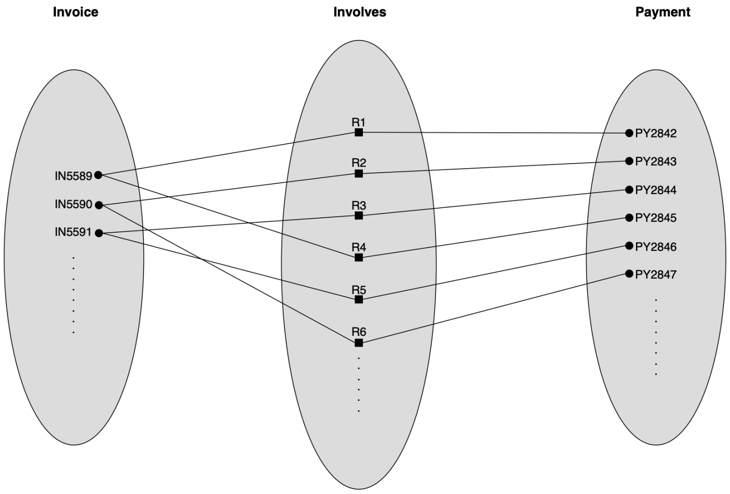


Figure 3: Relationship Set of Invoice and Payment

In Figure 4, the many-to-one relationship between Product and Product Category is shown, with one product belonging to one category, which can contain multiple products.

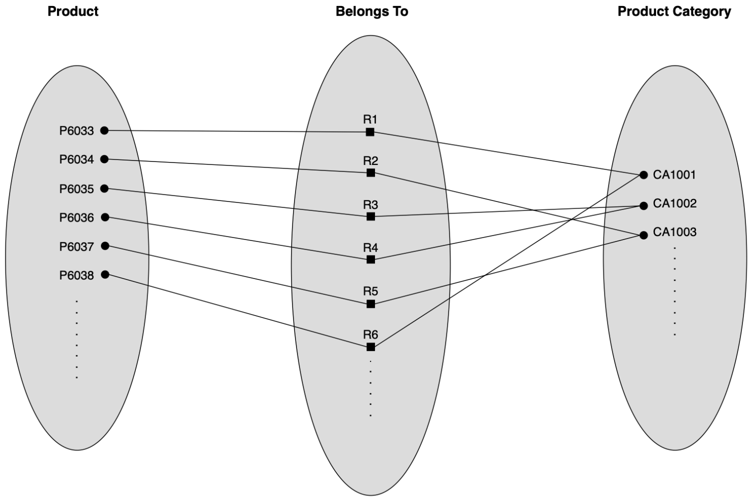


Figure 4: Relationship Set of Product and Category

Figure 5 illustrates the many-to-many relationship between Product and Sale, where a product may be on sale multiple times, and during a sale event, multiple products can be discounted.

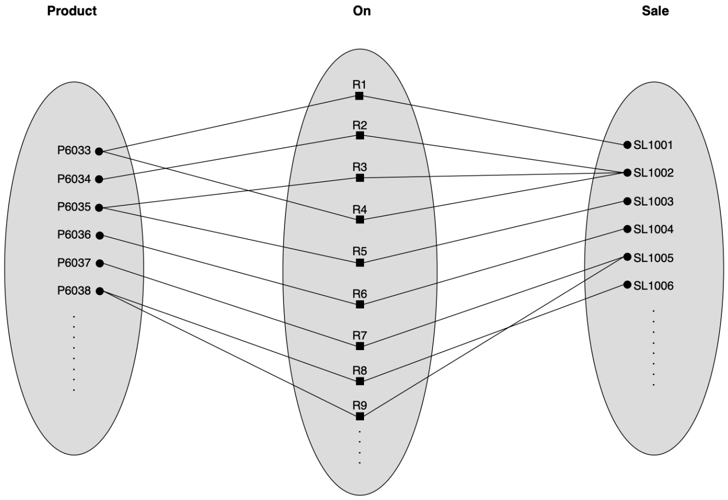


Figure 5: Relationship Set of Product and Sale

The many-to-many relationship between Product and Supplier is represented in Figure 6, where each supplier may provide multiple products, and each product may be supplied by multiple suppliers.

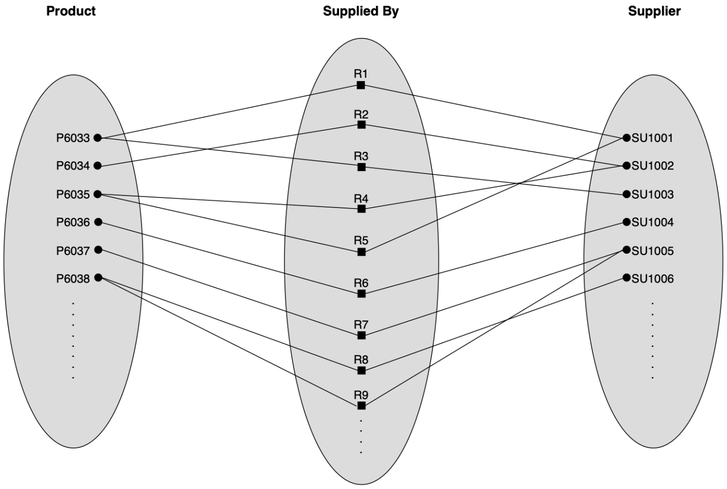


Figure 6: Relationship Set of Product and Supplier

Figure 7 demonstrates a ternary relationship between Product, Invoice, and Refund, with a cardinality of 1:1:1, where each product in an invoice can have only one refund request, each refund request is associated with one product and one invoice, and each refund request for a product is linked to only one invoice.

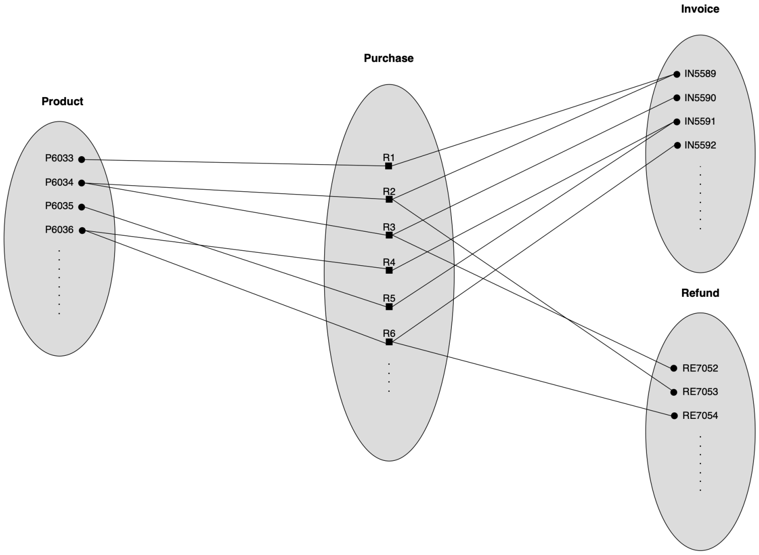


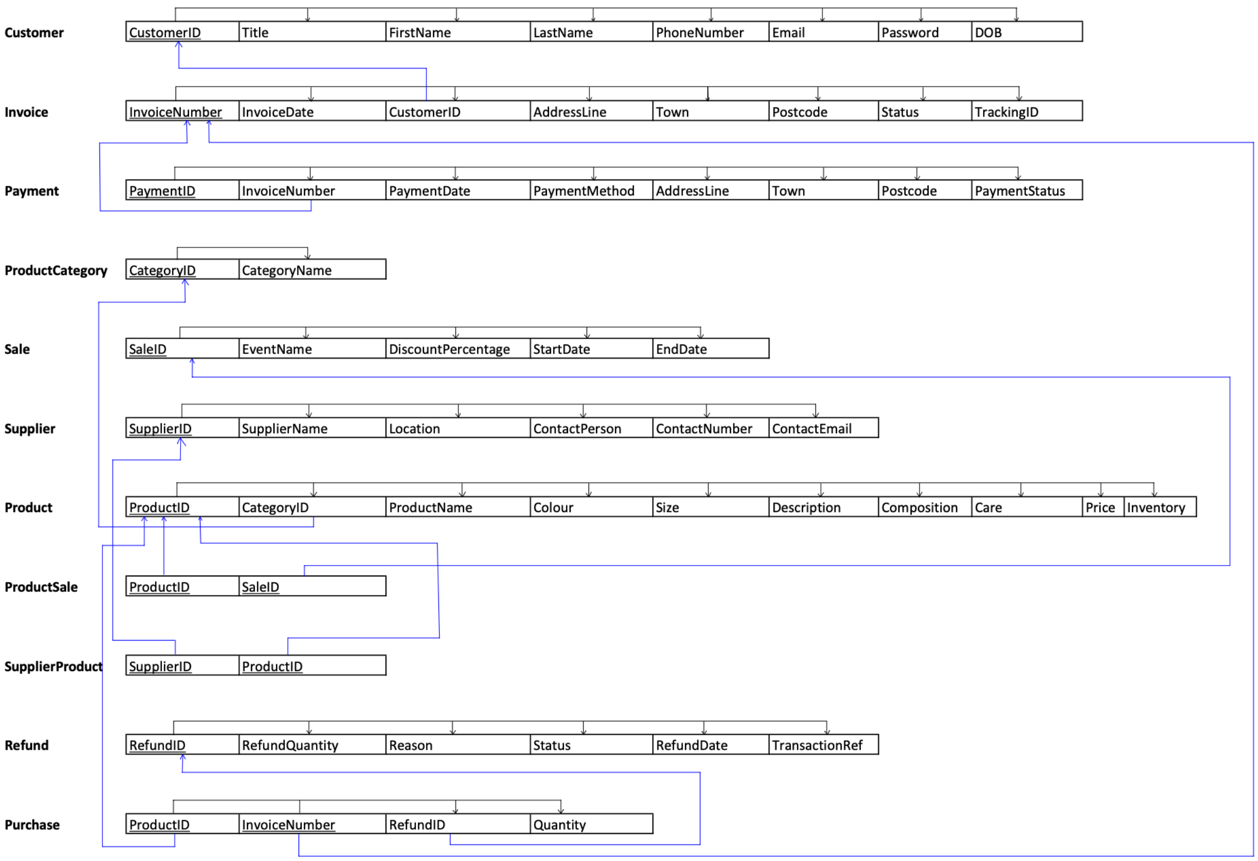
Figure 7: Relationship Set of Product, Invoice and Refund

1.2 SQL Database Schema Creation

The schema was designed based on the ER diagram and normalised to 3NF to avoid data redundancy. Normalisation of the schema is visualized in Figure 8. The black line shows the functional dependencies of attributes in each table while the blue line highlights the referential integrity. Relationships are established using primary keys and foreign keys in the schema. Primary keys are defined for each table and foreign key references ensure that the tables are linked with each other as in the ERD design. Not null constraints are also included to ensure that data stored in the database contain essential information such as shipping address and customer email. Suitable data types are also clearly defined in the schema to ensure high data quality. Moreover, to ensure data integrity and consistency, a constraint "Status_Check" is defined and applied on attributes including invoice status, payment status and refund status. The constraint restricts the allowable values for the column, for instance, payment status can only be either "Payment Successful" and "Payment Declined". This help enhance data quality.

SQL DDL was used to create tables based on the schema. The following shows the DDL for creating the Invoice table. It demonstrates how the primary key, foreign key, not null constraints, and Status_Check constraints was defined. The full SQL script was attached in the appendix. The script was run to create the database "zara.db".

```sql
CREATE TABLE IF NOT EXISTS `Invoice` (
  `InvoiceNumber` VARCHAR(10) PRIMARY KEY,
  `InvoiceDate` DATE NOT NULL,
  `CustomerID` VARCHAR(10) NOT NULL,
  `AddressLine` VARCHAR(255) NOT NULL,
  `Town` VARCHAR(255) NOT NULL,
  `Postcode` VARCHAR(50) NOT NULL,
  `Status` VARCHAR(255) NOT NULL,
  `TrackingID` VARCHAR(255),
  FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID),
  CONSTRAINT Status_Check CHECK (Status IN (”Completed”, ”Shipped”, ”Preparing for Shipment”,
   ↪  ”Processing”, ”Cancelled”, ”Pending for Payment”))
);
```



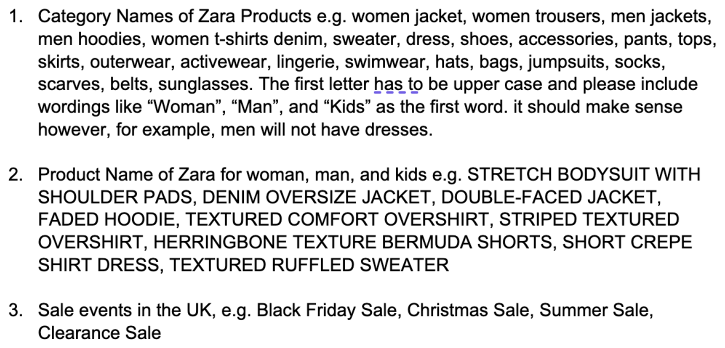
Customer CustomerID Title FirstName LastName PhoneNumber Email Password DOB
Invoice InvoiceNumber InvoiceDate CustomerID AddressLine Town Postcode Status TrackingID
Payment PaymentID InvoiceNumber PaymentDate PaymentMethod AddressLine Town Postcode PaymentStatus
ProductCategory CategoryID CategoryName
Sale SaleID EventName DiscountPercentage StartDate EndDate
Supplier SupplierID SupplierName Location ContactPerson ContactNumber ContactEmail
Product ProductID CategoryID ProductName Colour Size Description Composition Care Price Inventory
ProductSale ProductID SaleID
SupplierProduct SupplierID ProductID
Refund RefundID RefundQuantity Reason Status RefundDate TransactionRef
Purchase ProductID InvoiceNumber RefundID Quantity

Figure 8: Visualisation of Normalisation

Part 2 Data Generation and Management

2.1 Synthetic Data Generation

After finalising our ER diagram and schema, data was generated for use in the project. Mockaroo was used to generate the synthetic data to be inserted into the database. For each table, the name of tables, fields, data types were specified. The custom list data type was used to generate data such as product name and category name where LLM was incorporated at the same time to ensure data quality. Figure 9 shows some of the prompts used (full list is in Appendix). From this, the synthetic data is able to properly mimics the types of data which a company like Zara would be processing. Importantly at this stage, different formulas were also used to ensure data produced aligns with real-world operations and matches with the assumptions made. For instance, a formula was used in the “Status” field in “Invoice” table to ensure that “TrackingID” only appears when “Status” is either “Completed” or “Shipped”. Full list of formulas used can be seen in Appendix. It is also ensured that the foreign key fields were linked with the data in respective tables. The final datasets were exported as .csv’s and imported into the Posit Cloud project for the creation of the database.



1. Category Names of Zara Products e.g. women jacket, women trousers, men jackets, men hoodies, women t-shirts denim, sweater, dress, shoes, accessories, pants, tops, skirts, outerwear, activewear, lingerie, swimwear, hats, bags, jumpsuits, socks, scarves, belts, sunglasses. The first letter has to be upper case and please include wordings like “Woman”, “Man”, and “Kids” as the first word. it should make sense however, for example, men will not have dresses.

2. Product Name of Zara for woman, man, and kids e.g. STRETCH BODYSUIT WITH SHOULDER PADS, DENIM OVERSIZE JACKET, DOUBLE-FACED JACKET, FADED HOODIE, TEXTURED COMFORT OVERSHIRT, STRIPED TEXTURED OVERSHIRT, HERRINGBONE TEXTURE BERMUDA SHORTS, SHORT CREPE SHIRT DRESS, TEXTURED RUFFLED SWEATER

3. Sale events in the UK, e.g. Black Friday Sale, Christmas Sale, Summer Sale, Clearance Sale

Figure 9: LLM Prompts Used in Data Generation

2.2 Data Import and Quality Assurance

The synthetic data was then integrated into the database system. Tables were first created based on the schema. To guarantee that the data can be inserted into the database without any errors., data validation was done before inserting the data into the database. Following higlights the key steps done in the pre-insertion stage:

- Uniqueness Constraints: Primary keys in the database were retrieved and cross-referenced with those of the new data to identify and bypass records violating the unique primary key constraint.
- Not Null Constraints: Given that there are fields in the schema marked as “NOT NULL,” records that fail to satisfy the constraint were bypassed. This ensures that only entries with complete and valid data are processed further
- Data Type Check: Specific attention was given to fields designated as numeric within the schema.
- Format Check: Formats of email and date were also verified before insertion to ensure data consistency and accuracy in the database.
- Duplicates: Duplicates with same values in all fields within each table were removed to eliminate redundant entries.

After the checks, the data was inserted into the predefined tables. Following demonstrates the code for inserting the “Invoice” table.

```
#Validate Invoice
if (”Invoice.csv” %in% all_files) {
  this_file_path <- paste0(file_path,”Invoice.csv”)
  file_content <- readr::read_csv(this_file_path)

  # bypass rows with na values in not null columns
  no_na_invoice <- c(”InvoiceNumber”, ”InvoiceDate”, ”CustomerID”, ”AddressLine”, ”Town”, ”Postcode”,
↪  ”Status”)
  valid_invoice <- file_content[complete.cases(file_content[,no_na_invoice]),]
  # bypass rows that has repeated primary key
  invoice_pk <- RSQLite::dbGetQuery(database_connection, ”SELECT InvoiceNumber FROM Invoice;”)
  valid_invoice <- valid_invoice %>% filter(!(InvoiceNumber %in% invoice_pk$InvoiceNumber))

  # save and remove invalid customer data
  invalid_invoicedate <- !sapply(valid_invoice$InvoiceDate, check_date_format)
  valid_invoice <- valid_invoice[!invalid_invoicedate,]

  # check if invoice status is within allowable value
  valid_invoice <- valid_invoice[valid_invoice$Status %in% c(”Completed”, ”Shipped”, ”Preparing for
↪   Shipment”, ”Processing”, ”Cancelled”, ”Pending for Payment”),]

  # remove duplicated
  valid_invoice <- valid_invoice[!duplicated(valid_invoice),]

  # adjust format of data
  valid_invoice$InvoiceDate <- as.character(as.Date(valid_invoice$InvoiceDate,format = ”%d/%m/%Y”))

  RSQLite::dbWriteTable(database_connection, ”Invoice”, valid_invoice, overwrite = F, append = T)
}
```

Referential integrity of the database was maintained through the establishment of foreign key constraints:

```
Foreign key constraints for table Invoice :
  id seq    table       from         to on_update on_delete match
1  0   0 Customer CustomerID CustomerID NO ACTION NO ACTION  NONE


Foreign key constraints for table Payment :
  id seq   table          from            to on_update on_delete match
1  0   0 Invoice InvoiceNumber InvoiceNumber NO ACTION NO ACTION  NONE


Foreign key constraints for table Product :
  id seq    table       from         to on_update on_delete match
1  0   0 Category CategoryID CategoryID NO ACTION NO ACTION  NONE


Foreign key constraints for table ProductSale :
  id seq   table      from        to on_update on_delete match
1  0   0    Sale    SaleID    SaleID NO ACTION NO ACTION  NONE
2  1   0 Product ProductID ProductID NO ACTION NO ACTION  NONE
```

```
Foreign key constraints for table SupplierProduct :
  id seq    table       from         to on_update on_delete match
1  0   0  Product  ProductID  ProductID NO ACTION NO ACTION  NONE
2  1   0 Supplier SupplierID SupplierID NO ACTION NO ACTION  NONE
```

```
Foreign key constraints for table Purchase :
  id seq   table          from            to on_update on_delete match
1  0   0  Refund      RefundID      RefundID NO ACTION NO ACTION  NONE
2  1   0 Invoice InvoiceNumber InvoiceNumber NO ACTION NO ACTION  NONE
3  2   0 Product     ProductID     ProductID NO ACTION NO ACTION  NONE
```

Final data in database:

Table 1: 5 records

CustomerID	Title	FirstName	LastName	PhoneNumber	Email	Password	DOB
C1750	Ms	Twyla	Wroth	+440384790371	twroth0@nih.gov	dE1%J/33pU	1973-11-09
C1752	Mrs	Jourdan	Gever	+444737177236	jgever2@wiley.com	xE0+k,DcbF	1986-08-11
C1754	Rev	Claudio	McKune	+446358521457	cmckune4@wix.com	iI7/X4)3b1	1977-05-07
C1756	Dr	Sloane	Larder	+440385673326	slarder6@gov.uk	lB2%iU9g7,r9of	2007-12-28
C1758	Dr	Barbie	Meric	+446203713535	bmeric8@paginegialle.it	cQ2”kDp{N?eb	1981-07-13

Table 2: 5 records

PaymentID	InvoiceNumber	PaymentDate	PaymentMethod	AddressLine	Town	Postcode	PaymentStatus
PY3342	IN5589	2019-02-09	Google Pay	Room 1022	Hereford	KA1 1AA	Payment Successful
PY3343	IN5590	2020-10-29	Debit Card	PO Box 70604	Newcastle	YO1 1AA	Payment Successful
PY3344	IN5591	2020-09-18	Apple Pay	PO Box 14617	Aberdeen	WC2A 1AA	Payment Successful
PY3345	IN5592	2021-07-18	PayPal	Apt 538	Leeds	FY2 1AA	Payment Successful
PY3346	IN5593	2023-10-08	Credit Card	Suite 18	Belfast	ML2 1AA	Payment Successful

Table 3: 5 records

InvoiceNumber	InvoiceDate	CustomerID	AddressLine	Town	Postcode	Status	TrackingID
IN5589	2019-02-07	C2098	Apt 104	Dumfries	GU14 6QJ	Completed	TG1077
IN5590	2020-10-28	C1901	Room 1058	Cambridge	YO15 5DD	Completed	TV2539
IN5591	2020-09-17	C1838	12th Floor	Drogheda	YO14 5DD	Completed	TI2678
IN5592	2021-07-17	C2011	PO Box 38671	Dundee	KT27 7RQ	Completed	TV3553
IN5593	2023-10-06	C1958	Apt 1524	London	NW9 6XE	Completed	TE5142

Table 4: 5 records

PaymentID	InvoiceNumber	PaymentDate	PaymentMethod	AddressLine	Town	Postcode	PaymentStatus
PY3342	IN5589	2019-02-09	Google Pay	Room 1022	Hereford	KA1 1AA	Payment Successful
PY3343	IN5590	2020-10-29	Debit Card	PO Box 70604	Newcastle	YO1 1AA	Payment Successful
PY3344	IN5591	2020-09-18	Apple Pay	PO Box 14617	Aberdeen	WC2A 1AA	Payment Successful
PY3345	IN5592	2021-07-18	PayPal	Apt 538	Leeds	FY2 1AA	Payment Successful
PY3346	IN5593	2023-10-08	Credit Card	Suite 18	Belfast	ML2 1AA	Payment Successful

Table 5: 5 records

CategoryID	CategoryName
CA1001	Woman Jackets
CA1002	Woman Hoodies
CA1003	Woman Sweaters
CA1004	Woman Pants
CA1005	Woman Outerwear

Table 6: 5 records

SaleID	EventName	DiscountPercentage	StartDate	EndDate
SL1001	Customer Appreciation Week Sale	0.40	2024-03-02	2024-03-26
SL1002	Last Chance to Save Sale	0.15	2022-01-03	2022-01-20
SL1003	Mega Sale	0.50	2020-02-11	2020-02-24
SL1004	End of Season Sale	0.20	2023-11-23	2023-12-07
SL1005	Clearance Sale	0.15	2019-03-11	2019-03-24

Table 7: 5 records

SupplierID	SupplierName	Location	ContactPerson	ContactNumber	ContactEmail
SU1001	Stylish Creations Inc.	Marrakech	Daloris Kilfeder	+86 182 798 4067	dkilfeder0@gov.uk
SU1002	Fashion Forward Co.	Tel Aviv	Rosalie Le Fleming	+1 212 310 3662	rle1@xrea.com
SU1003	Trendy Threads Ltd.	Rio de Janeiro	Percival Jelkes	+351 692 599 0631	pjelkes2@yandex.ru
SU1004	Chic Couture Enterprises	Bangkok	Dolph McConnal	+63 361 771 9867	dmcconnal3@moonfruit.com
SU1005	Glamour Galore Corp.	Shanghai	Emmey Bucklee	+386 990 961 2146	ebucklee4@hp.com

Table 8: 5 records

ProductID	CategoryID	ProductName	Colour	Size	Description	Composition	Care	Price	Inventory
P6033	CA1010	EMBROIDERED OUTERWEAR	Indigo	3XL	indigo trendy embroidered outerwear for man	60% wool 40% acrylic	lay flat to dry	59.83	2614658
P6034	CA1005	TEXTURED OUTERWEAR	Aquamarine	3XL	aquamarine luxurious textured outerwear for woman	95% acrylic 5% spandex	Gentle cycle	10.38	4948375
P6035	CA1013	COMPACT SWEATERS	Crimson	S	crimson stylish compact sweaters for kids	95% acrylic 6% spandex	Gentle cycle	8.76	5737676
P6036	CA1007	BOXY-FIT HOODIES	Puce	M	puce luxurious boxy-fit hoodies for man	70% viscose 30% nylon	Spot clean with damp cloth	14.01	4060270
P6037	CA1001	FADED JACKETS	Teal	M	teal trendy faded jackets for woman	60% wool 40% acrylic	lay flat to dry	78.28	268696

Table 9: 5 records

ProductID	SaleID
P6183	SL1001
P6174	SL1002
P6229	SL1003
P6064	SL1004
P6166	SL1005

Table 10: 5 records

SupplierID	ProductID
SU1001	P6033
SU1002	P6034
SU1003	P6035
SU1004	P6036
SU1005	P6037

Table 11: 5 records

RefundID	RefundQuantity	Reason	Status	RefundDate	TransactionRef
RE7052	3	Item didn't fit properly	Waiting for Approval	2021-09-02	NA
RE7053	1	Item didn't fit properly	Rejected	2022-05-19	NA
RE7054	2	Changed mind about purchase	Rejected	2019-02-10	NA
RE7055	9	Wrong color received	Rejected	2022-01-17	NA
RE7056	7	Item didn't match online photos	Processing	2021-01-09	TR7587

Table 12: 5 records

ProductID	InvoiceNumber	RefundID	Quantity
P6151	IN5876	NA	2
P6232	IN5872	NA	1
P6046	IN5957	NA	8
P6119	IN5726	NA	7
P6182	IN5725	RE7241	3

This approach not only streamlined the data import process but also guaranteed the data's consistency and reliability, forming a strong basis for integration with a GitHub repository and further analysis.

Part 3 Data Pipeline Generation

Github Repository: https://github.com/Stephsour/DM_group23

The GitHub automation workflow was achieved through the creation of a YAML file. This workflow is triggered whenever a "push" has been made to the main code branch. The "job" (consisting of multiple steps) executed in the workflow is set to run on the latest Ubuntu environment.

The workflow is in place to ensure that whenever new data is appended/pushed to the database it goes automatically through the workflow (YAML) file particularly the R scripts attached to undergo the functions presented. The YAML file consists of three such R scripts: load_data.R, referential_integrity.R, and plots.R. Load_data.R processes/validates various CSV files (data in the New Data folder) before inserting it into the existing SQLite database, to ensure duplicate entries and illogical inconsistencies don't appear. Referential_integrity.R connects to the existing SQLite database, retrieves the list of tables present with its foreign key constraints, and cross-checks the relationships and structures of these tables. Finally, plots.R is designed to conduct a myriad of visualisations covering aspects of sales performance, returns, and payment methods to help derive business insights. These scripts run simultaneously to ensure addition of new data, data validation, database updates, and a constant churn of new data analysis whenever data is pushed.

Part 4 Data Analysis:

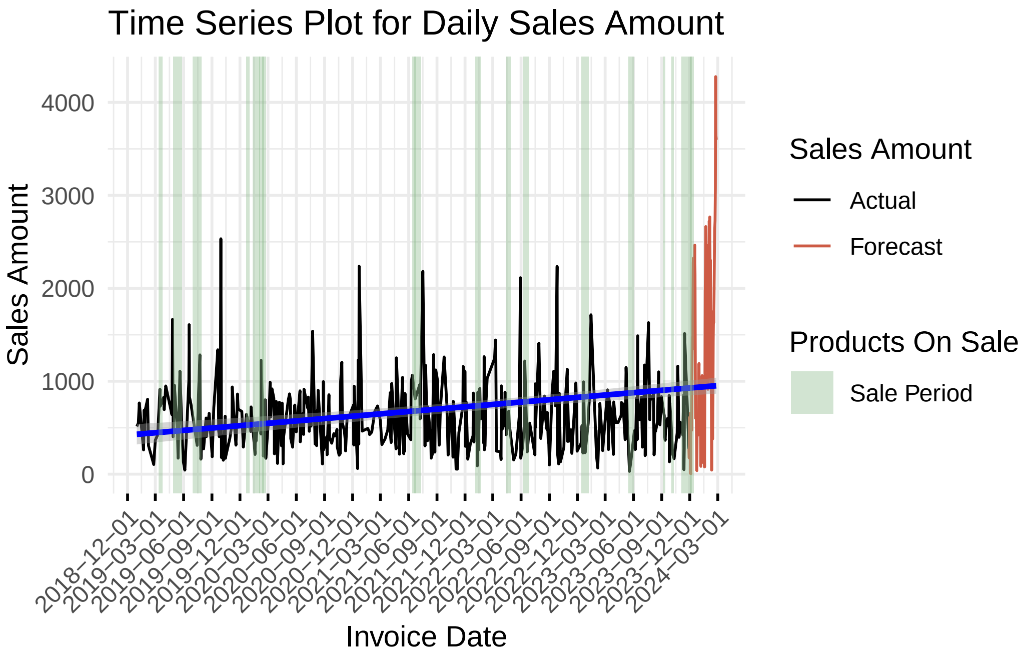
This part analyses sales data from the fictional ZARA clothing e-commerce and visualises various aspects of sales trends, refunds, suppliers, and transaction summaries, aiming to provide insights into the site's performance.

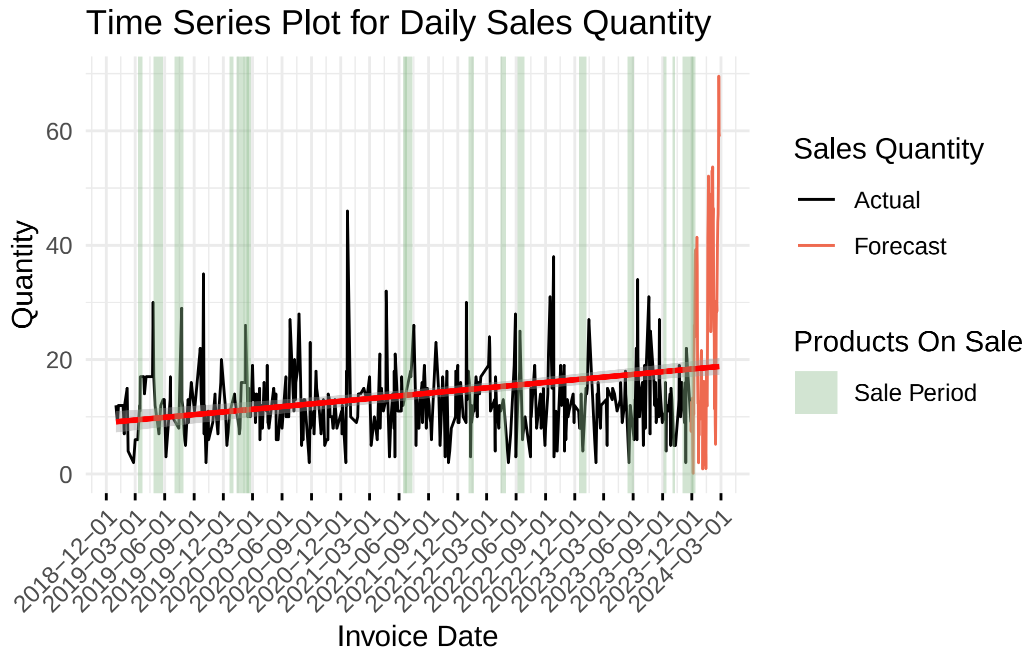
4.1 Data Retrieval and Preparation:

A connection was established to the ZARA database and retrieving relevant sales data, which can be updated instantly alongside Github actions. The data was processed to calculate selling prices, considering discount periods, and organised for further analysis. The preparation process was highlighted in the code below:

4.2 Time Series Analysis:

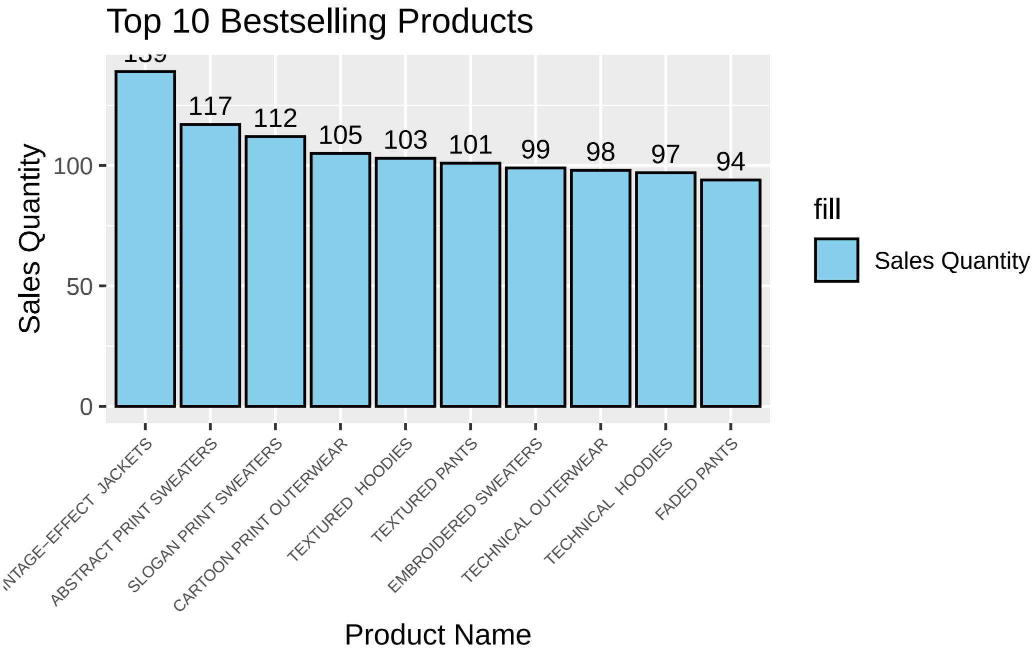
The visual representation of the daily sales amount and quantity over time combine past sales data with forecasted values. It makes use of linear regression to identify trends and a random walk model to predict future sales information. Importantly, this method is most accurate in predicting short-term sales with stationary and noisy data, so the forecasting technique should be altered once the data pattern differs.

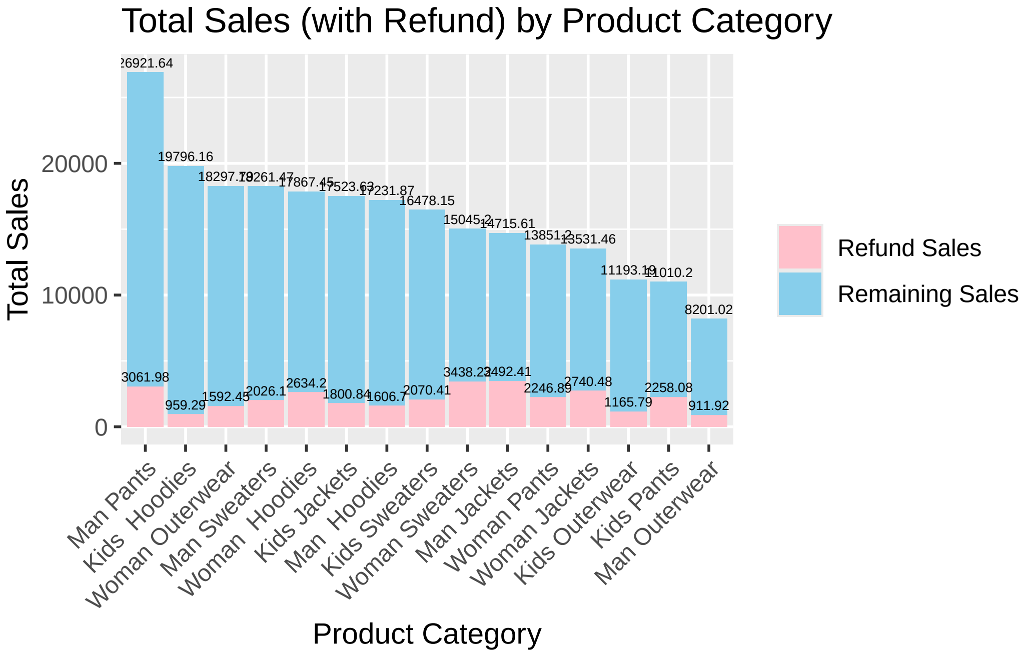


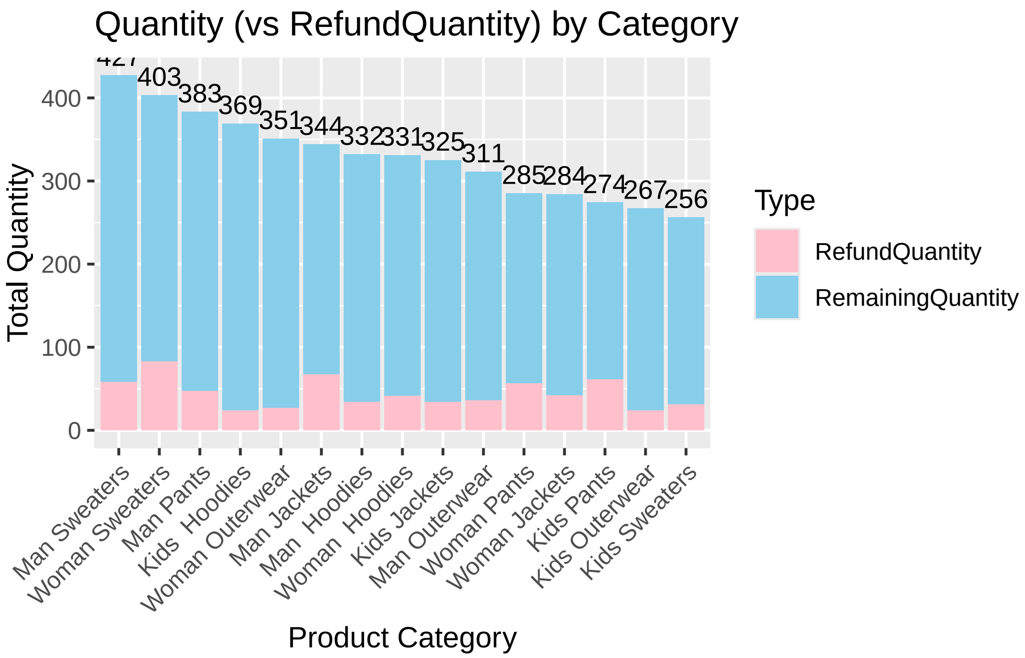


4.3 Product Sales Analysis:

The sales data are analysed in product categories, along with the statistics of the top 10 best-selling products.

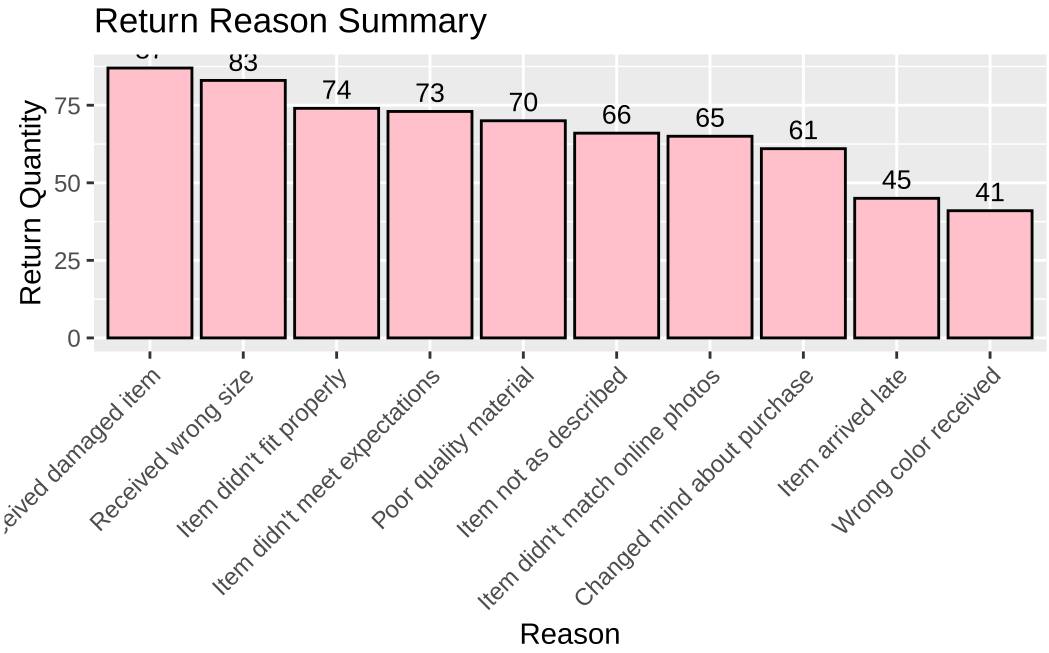


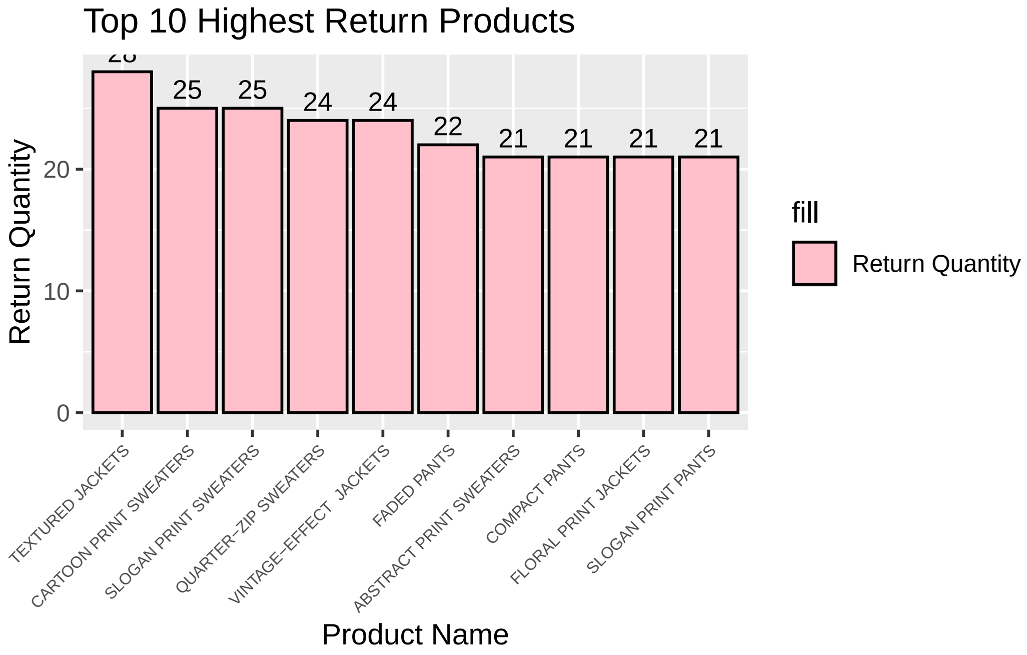




4.4 Return and Refund Analysis:

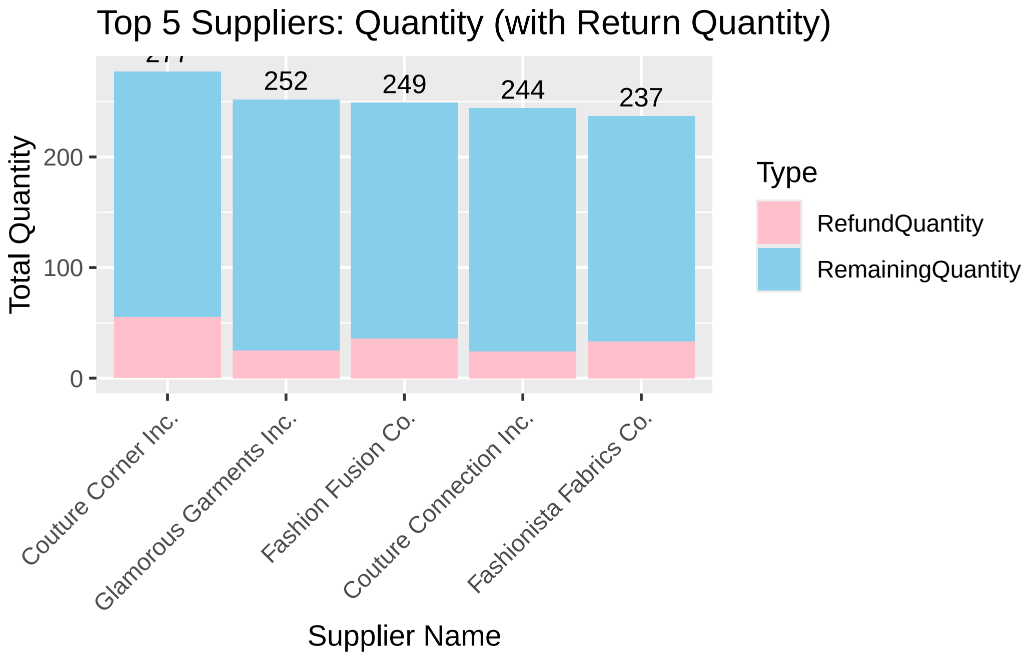
This section investigates a summary of return quantities by reason and identifies products with the 10 highest return rates.





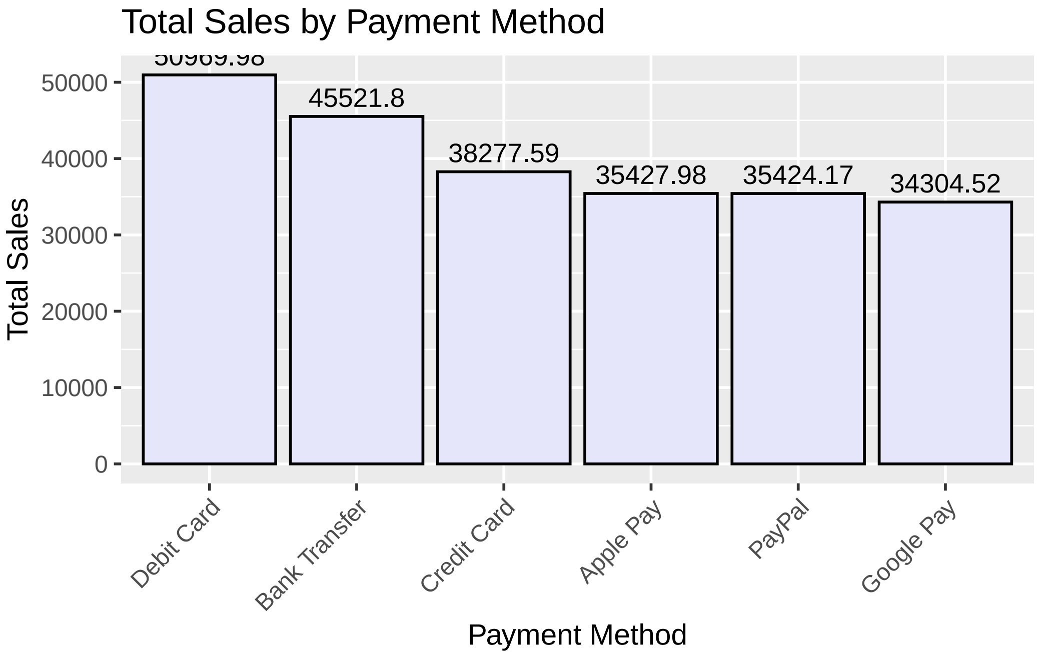
4.5 Sales Summary by Suppliers :

The chart presents insights into the purchasing and returning trends from various vendors for supplier management.



4.6 Transaction method Summary:

The transaction summary section analyses the total sales amount associated with each customer payment method.



4.7 Recommendation from Data Analysis:

1. Sales volume during discount periods does not differ from regular times. In other words, the current promotion strategies should be improved, and several actions can be considered, such as reviewing pricing strategy, monitoring competitor activities, and enhancing marketing efforts.
2. The category of men's pants has the highest profit margin, and the men's sweaters is best-selling; both have significant growth potential. Meanwhile, men's outerwear's sales volume is the lowest; hence, further market research, price optimisation, and feedback analysis are suggested.
3. Regarding refund request reasons, quality control and sizing information should be improved. It is necessary to conduct regular analysis of customer feedback, especially regarding product quality, size accuracy, and overall satisfaction, to address common issues and enhance customer satisfaction and loyalty. Besides, the quality control department and the product management department should corporately address the issues from the most frequently returned products.
4. 'Bank transfers' is the to-go option for customers. Based on those prevalent methods, the checkout process should be optimised to benefit customers.

Difficulties and Resolution

- Business rules and logic: Incorporating complex business rules (e.g., refund interactions, sales event effects) into the ER diagram.
- Contextual understanding: Deep business domain knowledge is crucial for accurate entity and relationship identification, as some business-specific details may not be apparent from the schema alone.
- Generation of data: Default settings of Mockaroo weren't adequate to capture the real-world business scenarios. Various formulas were used to establish connections between datasets to improve data quality.
- Workflow initialisation: Whilst we did configure the workflow to run on each push, there was a host of syntax errors within it which prevented this. Additionally, we had mistakenly already appended the new data to the database, requiring us to revise this so that the process of handling new data could be demonstrated.
- Workflow permissions: The way the GitHub was initially set up prevented any automation having permission to make changes to the main depository. Once this was figured out, the workflow ran flawlessly.

Conclusion

The detailed simulation of an e-commerce environment, seeking inspiration from Zara's fast-fashion model, required careful database planning, data handling, automated workflows, and in-depth analysis. The pipeline generation in our shared GitHub repository was to ensure the automation of actions such addition of new data, validation of data, database updates, and continual churn of new data analysis in a bid to automate the process of new data/files being pushed to the repository to replicate a real working database. Overcoming inherent challenges, including maintaining complex relationships and avoiding things such as data redundancy, laid a robust foundation for improved business insights with operational efficiencies to achieving a self-sufficient functioning database.

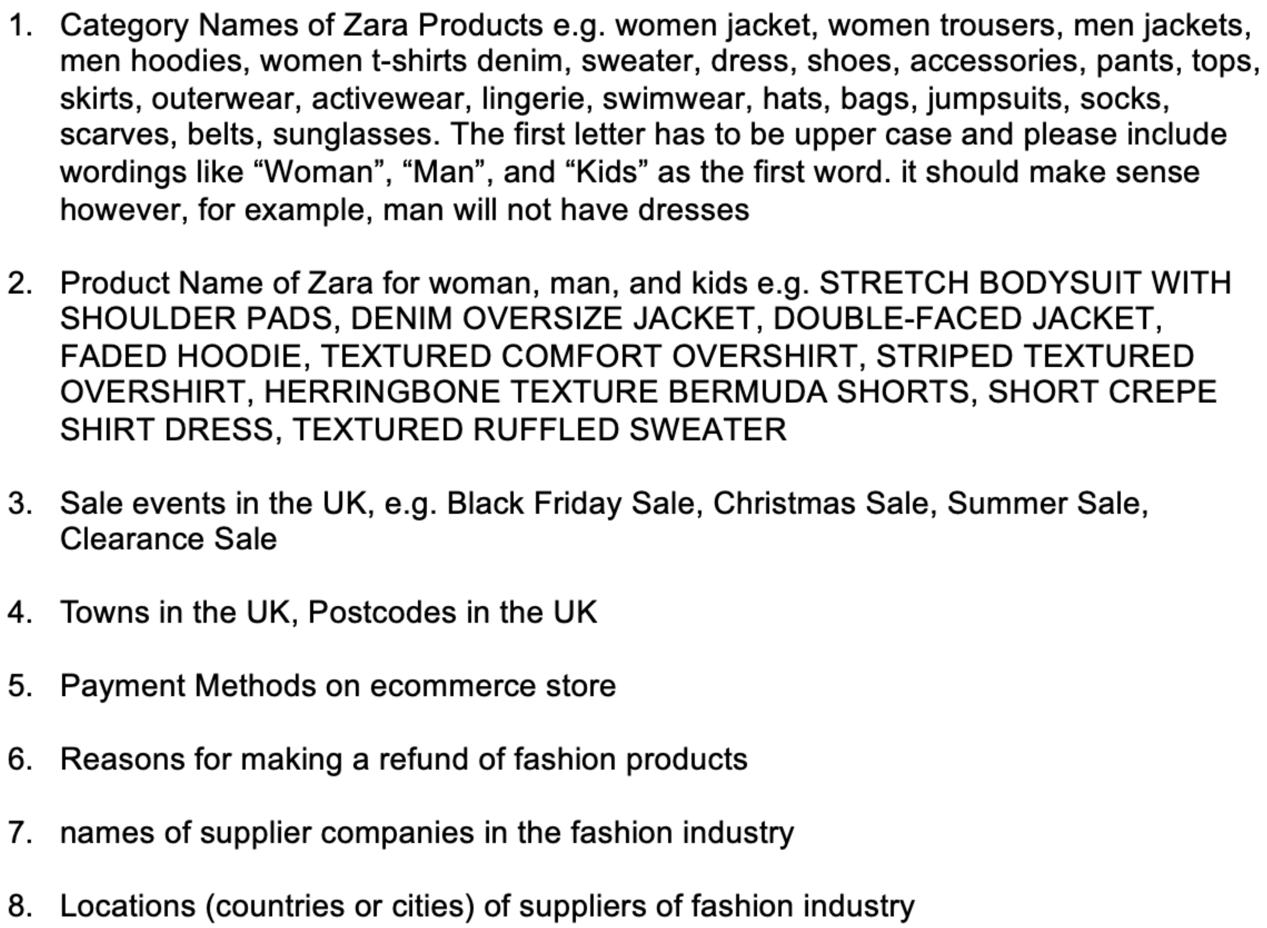
Appendix

1. SQL Data Definition Language (DDL) Script for Database Schema Creation (ddl_script.sql)

```sql
CREATE TABLE IF NOT EXISTS `Customer` (
  `CustomerID` VARCHAR(10) PRIMARY KEY,
  `Title` VARCHAR(50) NOT NULL,
  `FirstName` VARCHAR(255) NOT NULL,
  `LastName` VARCHAR(255) NOT NULL,
  `PhoneNumber` VARCHAR(20),
  `Email` VARCHAR(255) NOT NULL,
  `Password` VARCHAR(255) NOT NULL,
  `DOB` DATE
);

CREATE TABLE IF NOT EXISTS `Invoice` (
  `InvoiceNumber` VARCHAR(10) PRIMARY KEY,
  `InvoiceDate` DATE NOT NULL,
  `CustomerID` VARCHAR(10) NOT NULL,
  `AddressLine` VARCHAR(255) NOT NULL,
  `Town` VARCHAR(255) NOT NULL,
  `Postcode` VARCHAR(50) NOT NULL,
  `Status` VARCHAR(255) NOT NULL,
  `TrackingID` VARCHAR(255),
  FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID),
  CONSTRAINT Status_Check CHECK (Status IN (”Completed”, ”Shipped”, ”Preparing for Shipment”,
   ↪  ”Processing”, ”Cancelled”, ”Pending for Payment”))
);

CREATE TABLE IF NOT EXISTS `Payment` (
  `PaymentID` VARCHAR(10) PRIMARY KEY,
  `InvoiceNumber` VARCHAR(10) NOT NULL,
  `PaymentDate` DATE NOT NULL,
  `PaymentMethod` VARCHAR(255) NOT NULL,
  `AddressLine` VARCHAR(255) NOT NULL,
  `Town` VARCHAR(255) NOT NULL,
  `Postcode` VARCHAR(50) NOT NULL,
  `PaymentStatus` VARCHAR(255) NOT NULL,
  FOREIGN KEY (InvoiceNumber) REFERENCES Invoice(InvoiceNumber),
  CONSTRAINT Status_Check CHECK (PaymentStatus IN (”Payment Successful”, ”Payment Declined”))
);

CREATE TABLE IF NOT EXISTS `ProductCategory` (
  `CategoryID` VARCHAR(10) PRIMARY KEY,
  `CategoryName` VARCHAR(255) NOT NULL
);

CREATE TABLE IF NOT EXISTS `Sale` (
  `SaleID` VARCHAR(10) PRIMARY KEY,
  `EventName` VARCHAR(255) NOT NULL,
  `DiscountPercentage` DECIMAL(10,2) NOT NULL,
  `StartDate` DATE NOT NULL,
  `EndDate` DATE NOT NULL
);
```

```sql
CREATE TABLE IF NOT EXISTS `Supplier` (
  `SupplierID` VARCHAR(10) PRIMARY KEY,
  `SupplierName` VARCHAR(255) NOT NULL,
  `Location` VARCHAR(255) NOT NULL,
  `ContactPerson` VARCHAR(255) NOT NULL,
  `ContactNumber` VARCHAR(20) NOT NULL,
  `ContactEmail` VARCHAR(255) NOT NULL
);

CREATE TABLE IF NOT EXISTS `Product` (
  `ProductID` VARCHAR(10) PRIMARY KEY,
  `CategoryID` VARCHAR(10) NOT NULL,
  `ProductName` VARCHAR(255) NOT NULL,
  `Colour` VARCHAR(50),
  `Size` VARCHAR(50),
  `Description` TEXT,
  `Composition` TEXT,
  `Care` TEXT,
  `Price` DECIMAL(10,2) NOT NULL,
  `Inventory` INT NOT NULL,
  FOREIGN KEY(CategoryID) REFERENCES Category(CategoryID)
);

CREATE TABLE IF NOT EXISTS `ProductSale` (
  `ProductID` VARCHAR(10),
  `SaleID` VARCHAR(10),
  FOREIGN KEY (ProductID) REFERENCES Product(ProductID),
  FOREIGN KEY (SaleID) REFERENCES Sale(SaleID),
  PRIMARY KEY (ProductID, SaleID)
);

CREATE TABLE IF NOT EXISTS `SupplierProduct` (
  `SupplierID` VARCHAR(10),
  `ProductID` VARCHAR(10),
  FOREIGN KEY (SupplierID) REFERENCES Supplier(SupplierID),
  FOREIGN KEY (ProductID) REFERENCES Product(ProductID),
  PRIMARY KEY (SupplierID, ProductID)

);

CREATE TABLE IF NOT EXISTS `Refund` (
  `RefundID` VARCHAR(10) PRIMARY KEY,
  `RefundQuantity` INT NOT NULL,
  `Reason` TEXT NOT NULL,
  `Status` VARCHAR(255) NOT NULL,
  `RefundDate` DATE NOT NULL,
  `TransactionRef` VARCHAR(255),
  CONSTRAINT Status_Check CHECK (Status IN (”Pending”, ”Completed”, ”Rejected”, ”Processing”,
   ↪  ”Approved”, ”Cancelled”, ”Waiting for Approval”, ”On Hold”))
);

CREATE TABLE IF NOT EXISTS `Purchase` (
  `ProductID` VARCHAR(10),
  `InvoiceNumber` VARCHAR(10),
  `RefundID` VARCHAR(10),
  `Quantity` INT NOT NULL,
  FOREIGN KEY (ProductID) REFERENCES Product(ProductID),
  FOREIGN KEY (InvoiceNumber) REFERENCES Invoice(InvoiceNumber),
  FOREIGN KEY (RefundID) REFERENCES Refund(RefundID),
  PRIMARY KEY (ProductID, InvoiceNumber)
);
```

2. Use of AI (Prompts for Data Generation)



1. Category Names of Zara Products e.g. women jacket, women trousers, men jackets, men hoodies, women t-shirts denim, sweater, dress, shoes, accessories, pants, tops, skirts, outerwear, activewear, lingerie, swimwear, hats, bags, jumpsuits, socks, scarves, belts, sunglasses. The first letter has to be upper case and please include wordings like “Woman”, “Man”, and “Kids” as the first word. it should make sense however, for example, man will not have dresses

2. Product Name of Zara for woman, man, and kids e.g. STRETCH BODYSUIT WITH SHOULDER PADS, DENIM OVERSIZE JACKET, DOUBLE-FACED JACKET, FADED HOODIE, TEXTURED COMFORT OVERSHIRT, STRIPED TEXTURED OVERSHIRT, HERRINGBONE TEXTURE BERMUDA SHORTS, SHORT CREPE SHIRT DRESS, TEXTURED RUFFLED SWEATER

3. Sale events in the UK, e.g. Black Friday Sale, Christmas Sale, Summer Sale, Clearance Sale

4. Towns in the UK, Postcodes in the UK

5. Payment Methods on ecommerce store

6. Reasons for making a refund of fashion products

7. names of supplier companies in the fashion industry

8. Locations (countries or cities) of suppliers of fashion industry

3. R Script for Data Import and Quality Assurance (load_data.R)

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

file_path <- ”Data/”
all_files <- list.files(file_path)


database_connection <- RSQLite::dbConnect(RSQLite::SQLite(), ”zara.db”)
```

```
check_email_format <- function(email_string) {
  pattern <- ”^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,}$”
  return(grepl(pattern, email_string))
}
check_date_format <- function(date) {
  pattern <- ”^^(0?[1-9]|[12][0-9]|3[01])/(0?[1-9]|1[0-2])/(\\d{4})$”
  return(grepl(pattern, date))
}


# Insert Customer Data
if (”Customer.csv” %in% all_files) {
  this_file_path <- paste0(file_path,”Customer.csv”)
  file_content <- readr::read_csv(this_file_path)

  # bypass rows with na values in not null columns
  no_na_cust <- c(”CustomerID”, ”Title”, ”FirstName”, ”LastName”, ”Email”, ”Password”)
  valid_customer <- file_content[complete.cases(file_content[,no_na_cust]),]

  # bypass rows that has repeated primary key
  cust_pk <- RSQLite::dbGetQuery(database_connection, ”SELECT CustomerID FROM Customer;”)
  valid_customer <- valid_customer %>% filter(!(CustomerID %in% cust_pk$CustomerID))

  # save and remove invalid customer data
  invalid_cemail <- !sapply(valid_customer$Email, check_email_format)
  valid_customer <- valid_customer[!invalid_cemail,]

  invalid_DOB <- !sapply(valid_customer$DOB, check_date_format)
  valid_customer <- valid_customer[!invalid_DOB,]

  # remove duplicated rows
  valid_customer <- valid_customer[!duplicated(valid_customer),]

  # adjust format
  valid_customer$DOB <- as.character(as.Date(valid_customer$DOB,format = ”%d/%m/%Y”))
  valid_customer$PhoneNumber <- as.character(paste0(”+”,valid_customer$PhoneNumber))

  RSQLite::dbWriteTable(database_connection, ”Customer”, valid_customer, overwrite = F,append = T)
}

#Validate Invoice
if (”Invoice.csv” %in% all_files) {
  this_file_path <- paste0(file_path,”Invoice.csv”)
  file_content <- readr::read_csv(this_file_path)

  # bypass rows with na values in not null columns
  no_na_invoice <- c(”InvoiceNumber”, ”InvoiceDate”, ”CustomerID”, ”AddressLine”, ”Town”, ”Postcode”,
↪  ”Status”)
  valid_invoice <- file_content[complete.cases(file_content[,no_na_invoice]),]

  # bypass rows that has repeated primary key
  invoice_pk <- RSQLite::dbGetQuery(database_connection, ”SELECT InvoiceNumber FROM Invoice;”)
  valid_invoice <- valid_invoice %>% filter(!(InvoiceNumber %in% invoice_pk$InvoiceNumber))

  # save and remove invalid customer data
  invalid_invoicedate <- !sapply(valid_invoice$InvoiceDate, check_date_format)
  valid_invoice <- valid_invoice[!invalid_invoicedate,]

  # check if invoice status is within allowable value
  valid_invoice <- valid_invoice[valid_invoice$Status %in% c(”Completed”, ”Shipped”, ”Preparing for
↪  Shipment”, ”Processing”, ”Cancelled”, ”Pending for Payment”),]

  # remove duplicated
  valid_invoice <- valid_invoice[!duplicated(valid_invoice),]

  # adjust format of data
  valid_invoice$InvoiceDate <- as.character(as.Date(valid_invoice$InvoiceDate,format = ”%d/%m/%Y”))

  RSQLite::dbWriteTable(database_connection, ”Invoice”, valid_invoice, overwrite = F, append = T)

}

# Validate Payment
if (”Payment.csv” %in% all_files){
  this_file_path <- paste0(file_path,”Payment.csv”)
  file_content <- readr::read_csv(this_file_path)

  # bypass rows with na values in not null columns
  valid_payment <- file_content[complete.cases(file_content),]

  # bypass rows that has repeated primary key
  payment_pk <- RSQLite::dbGetQuery(database_connection, ”SELECT PaymentID FROM Payment;”)
  valid_payment <- valid_payment %>% filter(!(PaymentID %in% payment_pk$PaymentID))

  # save and remove invalid payment records
  invalid_paydate <- !sapply(valid_payment$PaymentDate, check_date_format)
  valid_payment <- valid_payment[!invalid_paydate,]
  # check if payment status in allowable value
  valid_payment <- valid_payment[valid_payment$PaymentStatus %in% c(”Payment Declined”, ”Payment
 ↪  Successful”),]

  # remove duplicated rows
  valid_payment <- valid_payment[!duplicated(valid_payment),]

  # adjust format
  valid_payment$PaymentDate <- as.character(as.Date(valid_payment$PaymentDate,format = ”%d/%m/%Y”))

  RSQLite::dbWriteTable(database_connection, ”Payment”, valid_payment, overwrite = F, append = T)

}

# Validate Sale
if (”Sale.csv” %in% all_files) {
  this_file_path <- paste0(file_path,”Sale.csv”)
  file_content <- readr::read_csv(this_file_path)

  # bypass rows with na values in not null columns
  valid_sale <- file_content[complete.cases(file_content),]

  # bypass rows that has repeated primary key
  sale_pk <- RSQLite::dbGetQuery(database_connection, ”SELECT SaleID FROM Sale;”)
  valid_sale <- valid_sale %>% filter(!(SaleID %in% sale_pk$SaleID))

  # check sale dates format
  invalid_start <- !sapply(valid_sale$StartDate, check_date_format)
  valid_sale <- valid_sale[!invalid_start,]

  invalid_end <- !sapply(valid_sale$EndDate, check_date_format)
  valid_sale <- valid_sale[!invalid_end,]

  # check discount percentage data type
  valid_sale <- valid_sale[is.numeric(valid_sale$DiscountPercentage),]

  # remove duplicated
  valid_sale <- valid_sale[!duplicated(valid_sale),]

  # adjust format
  valid_sale$StartDate <- as.character(as.Date(valid_sale$StartDate,format = ”%d/%m/%Y”))
  valid_sale$EndDate <- as.character(as.Date(valid_sale$EndDate,format = ”%d/%m/%Y”))
  RSQLite::dbWriteTable(database_connection, ”Sale”, valid_sale, overwrite = F, append = T)


}

# Validate Refund
if (”Refund.csv” %in% all_files) {
  this_file_path <- paste0(file_path,”Refund.csv”)
  file_content <- readr::read_csv(this_file_path)

  # bypass rows with na values in not null columns
  no_na_refund <- c(”RefundID”, ”RefundQuantity”, ”Reason”, ”Status”, ”RefundDate”)
  valid_refund <- file_content[complete.cases(file_content[,no_na_refund]),]

  # bypass rows that has repeated primary key
  refund_pk <- RSQLite::dbGetQuery(database_connection, ”SELECT RefundID FROM Refund;”)
  valid_refund <- valid_refund %>% filter(!(RefundID %in% refund_pk$RefundID))

  # check date format
  invalid_refdate <- !sapply(valid_refund$RefundDate, check_date_format)
  valid_refund <- valid_refund[!invalid_refdate,]

  # check data type of refund quantity
  valid_refund <- valid_refund[is.numeric(valid_refund$RefundQuantity),]

  # remove duplicated
  valid_refund <- valid_refund[!duplicated(valid_refund),]

  # adjust format
  valid_refund$RefundDate <- as.character(as.Date(valid_refund$RefundDate,format = ”%d/%m/%Y”))

  RSQLite::dbWriteTable(database_connection, ”Refund”, valid_refund, overwrite = F, append = T)

}

# Validate Supplier
if (”Supplier.csv” %in% all_files) {
  this_file_path <- paste0(file_path,”Supplier.csv”)
  file_content <- readr::read_csv(this_file_path)

  # bypass rows with na values in not null columns
  valid_supplier <- file_content[complete.cases(file_content),]

  # bypass rows that has repeated primary key
  supplier_pk <- RSQLite::dbGetQuery(database_connection, ”SELECT SupplierID FROM Supplier;”)
  valid_supplier <- valid_supplier %>% filter(!(SupplierID %in% supplier_pk$SupplierID))

  # save and remove invalid supplier data
  invalid_supemail <- !sapply(valid_supplier$ContactEmail, check_email_format)
  valid_supplier <- valid_supplier[!invalid_supemail,]

  #remove duplicated
  valid_supplier <- valid_supplier[!duplicated(valid_supplier),]

  RSQLite::dbWriteTable(database_connection, ”Supplier”, valid_supplier, overwrite = F, append = T)

}

# Insert the remaining
remaining <- all_files[!(all_files %in% c(”Customer.csv”, ”Invoice.csv”, ”Payment.csv”, ”Sale.csv”,
↪  ”Refund.csv”, ”Supplier.csv”))]
for (file in remaining) {
  this_file_path <- paste0(file_path,file)
  file_content <- readr::read_csv(this_file_path)

  file_content <- file_content[!duplicated(file_content),]

  entity <- gsub(”.csv”,””,file)

  if (entity %in% c(”Product”, ”ProductCategory”)) {
    results <- RSQLite::dbGetQuery(database_connection, paste0(”SELECT * FROM ”,entity,”;”))
    primary_key <- as_tibble(results[,1])
    valid_record <- file_content %>% filter(!(file_content[,1] %in% primary_key))
    if (entity == ”Product” & nrow(valid_record) != 0) {
      valid_record <- valid_record[is.numeric(valid_record$Price),]
      no_na_prod <- c(”ProductID”, ”CategoryID”, ”ProductName”, ”Price”, ”Inventory”)
      valid_record <- valid_record[complete.cases(valid_record[,no_na_prod]),]
    } else {
      valid_record <- valid_record[complete.cases(valid_record),]
    }
  } else if (entity %in% c(”ProductSale”, ”SupplierProduct”, ”Purchase”)) {
    results <- RSQLite::dbGetQuery(database_connection, paste0(”SELECT * FROM ”, entity, ”;”))
    primary_key <- results[,1:2]
    file_content <- as.data.frame(file_content)
    primary_key_combined <- paste(primary_key[,1], primary_key[,2])
    file_content_combined <- paste(file_content[,1], file_content[,2])
    repeated <- sapply(file_content_combined, function(x) any(x == primary_key_combined))
    valid_record <- file_content[!repeated,]
    if (entity == "Purchase" & nrow(valid_record != 0)) {
      valid_record <- valid_record[is.numeric(valid_record$Quantity),]
      no_na_purchase <- c("ProductID", "InvoiceNumber", "Quantity")
      valid_record <- valid_record[complete.cases(valid_record[,no_na_purchase]),]
    } else {
      valid_record <- valid_record[complete.cases(valid_record),]
    }
  }

  RSQLite::dbWriteTable(database_connection, entity, valid_record, overwrite = F, append = T)


}

# post insert check for duplicated entries
duplicated_custemail <- RSQLite::dbGetQuery(database_connection, 'SELECT "Customer email "||Email||" is
 ↪  repeated."
FROM Customer
GROUP BY Email
HAVING COUNT(*) > 1;')
if (nrow(duplicated_custemail) == 0) {
  print("There are no duplicated customer email.")
} else {
  for (i in 1:nrow(duplicated_custemail)){
    print(duplicated_custemail[i,])
  }
}

duplicated_category <- RSQLite::dbGetQuery(database_connection,
                                           'SELECT "Category name "||CategoryName||" is repeated."
FROM ProductCategory
GROUP BY CategoryName
HAVING COUNT(*) > 1;')
if (nrow(duplicated_category) == 0) {
  print("There are no duplicated category name.")
} else {
  for (i in 1:nrow(duplicated_category)){
    print(duplicated_category[i,])
  }
}

duplicated_supplier <- RSQLite::dbGetQuery(database_connection,
                                           'SELECT SupplierName, ContactPerson, ContactNumber,
                                            ↪  ContactEmail||" is repeated."
```

```
FROM Supplier
GROUP BY SupplierName, ContactPerson, ContactNumber, ContactEmail
HAVING COUNT(*) > 1;')
if (nrow(duplicated_supplier) == 0) {
  print(”There are no duplicated suppliers.”)
} else {
  for (i in 1:nrow(duplicated_supplier)){
    print(duplicated_supplier[i,])
  }
}

duplicated_saleevent <- RSQLite::dbGetQuery(database_connection,
                                           'SELECT EventName, DiscountPercentage, StartDate, EndDate||”
                                             ↪  is repeated.”
FROM Sale
GROUP BY EventName, DiscountPercentage, StartDate, EndDate
HAVING COUNT(*) > 1;')
if (nrow(duplicated_saleevent) == 0) {
  print(”There are no duplicated sale events.”)
} else {
  for (i in 1:nrow(duplicated_saleevent)){
    print(duplicated_saleevent[i,])
  }
}

duplicated_transref <- RSQLite::dbGetQuery(database_connection,
                                           'SELECT TransactionRef||” is repeated.”
FROM Refund
WHERE TransactionRef NOT NULL
GROUP BY TransactionRef
HAVING COUNT(*) > 1;')
if (nrow(duplicated_transref) == 0) {
  print(”There are no duplicated refund transaction reference.”)
} else {
  for (i in 1:nrow(duplicated_transref)){
    print(duplicated_transref[i,])
  }
}

duplicated_trackingid <- RSQLite::dbGetQuery(database_connection,
                                             'SELECT TrackingID||” is repeated.”
FROM Invoice
WHERE TrackingID NOT NULL
GROUP BY TrackingID
```

```
HAVING COUNT(*) > 1;')
if (nrow(duplicated_trackingid) == 0) {
  print(”There are no duplicated tracking IDs.”)
} else {
  for (i in 1:nrow(duplicated_trackingid)){
    print(duplicated_trackingid[i,])
  }
}

# post insertion check for entry error
incorrect_sale <- RSQLite::dbGetQuery(database_connection,
                                      'SELECT SaleID || ” has end date before start date.”
FROM Sale
WHERE StartDate > EndDate;')
if (nrow(incorrect_sale) == 0) {
  print(”There are no entry errors in sale event dates.”)
} else {
  for (i in 1:nrow(incorrect_sale)){
    print(incorrect_sale[i,])
  }
}

incorrect_refdate <- RSQLite::dbGetQuery(database_connection,
                                         'SELECT R.RefundID || ” should not be approved as refund is not
                                          ↪  requested within 30 days.”
FROM Refund AS R INNER JOIN Purchase AS P
  ON R.RefundID = P.RefundID INNER JOIN Invoice AS I
  ON P.InvoiceNumber = I.InvoiceNumber
WHERE R.RefundDate > DATE(I.InvoiceDate, ”+30 days”);')
if (nrow(incorrect_refdate) == 0) {
  print(”There are no entry errors in refund date.”)
} else {
  for (i in 1:nrow(incorrect_refdate)){
    print(incorrect_refdate[i,])
  }
}

incorrect_refquantity <- RSQLite::dbGetQuery(database_connection,
                                             'SELECT R.RefundID || ” has entry error as refund quantity
                                              ↪  should be less than or equal to purchase quantity.”
FROM Refund AS R INNER JOIN Purchase as P
  ON R.RefundID = P.RefundID
WHERE R.RefundQuantity > P.Quantity;')
if (nrow(incorrect_refquantity) == 0) {
  print(”There are no entry errors in refund quantity.”)
} else {
  for (i in 1:nrow(incorrect_refquantity)){
    print(incorrect_refquantity[i,])
  }
}

incorrect_refund <- RSQLite::dbGetQuery(database_connection,
                                        'SELECT ”Refund should not be available for ”|| P.ProductID ||”
                                         ↪  in invoice ”||P.InvoiceNumber||”.”
FROM Purchase as P INNER JOIN Invoice as I
  ON P.InvoiceNumber = I.InvoiceNumber
WHERE I.Status != ”Completed” AND P.RefundID NOT NULL;')
if (nrow(incorrect_refund) == 0) {
  print(”There are no invalid refunds.”)
} else {
  for (i in 1:nrow(incorrect_refund)){
    print(incorrect_refund[i,])
  }
}

incorrect_invpstatus <- RSQLite::dbGetQuery(database_connection,
                                            'SELECT I.InvoiceNumber || ” should be cancelled as payment
                                             ↪  is not settled within two days.”
FROM Invoice AS I INNER JOIN Payment as P
  ON I.InvoiceNumber = P.InvoiceNumber
WHERE DATE(I.InvoiceDate, ”+2 days”) < P.PaymentDate;')
if (nrow(incorrect_invpstatus) == 0) {
  print(”There are no entry errors in invoice status.”)
} else {
  for (i in 1:nrow(incorrect_invpstatus)){
    print(incorrect_invpstatus[i,])
  }
}

incorrect_invstatus <- RSQLite::dbGetQuery(database_connection,
                                           'SELECT I.InvoiceNumber||” should be cancelled or pending for
                                            ↪  payment as payment is declined.”
FROM Invoice AS I INNER JOIN Payment AS P
  ON I.InvoiceNumber = P.InvoiceNumber
WHERE (I.Status != ”Cancelled” AND I.Status != ”Pending for Payment”)
  AND P.PaymentStatus = ”Payment Declined”;')
if (nrow(incorrect_invstatus) == 0) {
  print(”There are no entry errors in invoice status.”)
```

```
} else {
  for (i in 1:nrow(incorrect_invstatus)){
    print(incorrect_invstatus[i,])
  }
}
RSQLite::dbDisconnect(database_connection)
```

4. Referential Integrity Check (referential_integrity.R)

```
library(readr)
library(RSQLite)

database_connection <- RSQLite::dbConnect(RSQLite::SQLite(), ”zara.db”)

zara_tables <- RSQLite::dbGetQuery(database_connection, ”SELECT name FROM sqlite_master WHERE
 ↪  type='table'”)$name

fk_constraints_list = list()
for (table_name in zara_tables) {
  fk_constraints <- dbGetQuery(database_connection, paste(”PRAGMA foreign_key_list(”, table_name, ”)”))
  fk_constraints_list[[table_name]] <- fk_constraints
}

for (table_name in zara_tables) {
  if (nrow(fk_constraints_list[[table_name]]) > 0) {
    cat(”Foreign key constraints for table”, table_name, ”:\n”)
    print(fk_constraints_list[[table_name]])
    cat(”\n”)
    cat(”\n”)
  }
}

RSQLite::dbDisconnect(database_connection)
```

5. Data Analysis (Plots.R)

```
library(readr)
library(RSQLite)
library(dplyr)
library(ggplot2)

database_connection <- RSQLite::dbConnect(RSQLite::SQLite(), ”zara.db”)

# Prepare data for plotting the time series data
sales_data <- RSQLite::dbGetQuery(database_connection,
                                  ”SELECT DISTINCT I.InvoiceNumber, I.InvoiceDate,
                                   ↪  P.ProductID,PR.ProductName, P.Quantity,
                                   ↪  R.RefundQuantity,R.Reason, PC.CategoryName, SUP.SupplierName,
                                   ↪  PR.Price AS OriginalPrice, PY.PaymentMethod,PY.PaymentStatus,
    CASE
        WHEN I.InvoiceDate >= S.StartDate AND I.InvoiceDate <= S.EndDate THEN (1 - S.DiscountPercentage)
 ↪  * PR.Price
        ELSE PR.Price
    END AS SellPrice
FROM Invoice AS I
INNER JOIN Purchase AS P ON I.InvoiceNumber = P.InvoiceNumber
LEFT  JOIN Refund AS R ON P.RefundID = R.RefundID
INNER JOIN Product AS PR ON P.ProductID = PR.ProductID
LEFT JOIN SupplierProduct AS SP ON SP.ProductID = PR.ProductID
LEFT JOIN Supplier AS SUP ON SUP.SupplierID = SP.SupplierID
INNER JOIN ProductCategory AS PC ON PC.CategoryID = PR.CategoryID
INNER JOIN Payment AS PY ON PY.InvoiceNumber = I.InvoiceNumber
LEFT JOIN ProductSale AS PS ON PR.ProductID = PS.ProductID
LEFT JOIN Sale AS S ON PS.SaleID = S.SaleID
WHERE I.Status != 'Cancelled'
GROUP BY I.InvoiceNumber, P.ProductID
ORDER BY I.InvoiceNumber
;”)

discount_record <- RSQLite::dbGetQuery(database_connection,
                                       ”SELECT I.InvoiceNumber, I.InvoiceDate, P.ProductID, P.Quantity,
                                        ↪  PR.Price, ROUND(PR.Price * (1 - S.DiscountPercentage),2) AS
                                        ↪  SellPrice
FROM Invoice AS I
INNER JOIN Purchase AS P ON I.InvoiceNumber = P.InvoiceNumber
INNER JOIN Product AS PR ON P.ProductID = PR.ProductID
LEFT JOIN ProductSale AS PS ON PR.ProductID = PS.ProductID
LEFT JOIN Sale AS S ON PS.SaleID = S.SaleID
WHERE I.Status != 'Cancelled'  AND I.InvoiceDate >= S.StartDate AND I.InvoiceDate <= S.EndDate
GROUP BY I.InvoiceNumber, PR.ProductID;”)

for (i in 1:nrow(discount_record)) {
  # Find rows in sales_data where InvoiceNumber and ProductID match
  matching_rows <- sales_data$InvoiceNumber == discount_record[i,]$InvoiceNumber &
    sales_data$ProductID == discount_record[i,]$ProductID

  # Replace matching rows in sales_data with current row from discount_record
  sales_data[matching_rows, ”SellPrice”] <- discount_record[i,”SellPrice”]
}

sales_data$InvoiceDate <- as.Date(sales_data$InvoiceDate)


# Daily Sales and Sales Volume

# Extract daily sales
daily_sales <- sales_data %>% group_by(InvoiceDate) %>% summarise(quantity = sum(Quantity), sale =
 ↪  sum(Quantity * SellPrice))
daily_sales <- daily_sales[order(daily_sales$InvoiceDate, decreasing = T),]

# Generate forecast prediction
set.seed(10)
prediction_length <- 90
predicted_sale <- numeric(prediction_length)
predicted_sale[1] <- daily_sales$sale[1]
random_sale <- rnorm(prediction_length, mean = 0, sd = sd(diff(daily_sales$sale)))
for (i in 2:prediction_length) {
  predicted_value <- predicted_sale[[i-1]] + random_sale[i]
  if (predicted_value <= 0) {
    predicted_value <- mean(daily_sales$sale)
  }
  predicted_sale[i] <- predicted_value
}
predicted_sale <- as.data.frame(predicted_sale)
predicted_sale$InvoiceDate <- seq.Date(from = daily_sales$InvoiceDate[1], by = ”day”, length.out = 90)

combined_sale <- rbind(data.frame(date = daily_sales$InvoiceDate, value = daily_sales$sale, type =
 ↪  ”Actual”),
                       data.frame(date = predicted_sale$InvoiceDate, value =
                        ↪  predicted_sale$predicted_sale, type = ”Forecast”))


# Extract time periods when products are on sale
sale_date <- RSQLite::dbGetQuery(database_connection, ”SELECT StartDate, EndDate FROM Sale;”)
sale_date$StartDate <-  as.Date(sale_date$StartDate)
sale_date$EndDate <-  as.Date(sale_date$EndDate)
sale.shade <- data.frame(first = sale_date$StartDate,second = sale_date$EndDate, min = -Inf, max = Inf)
```

```
legend_order <- c(”Sales Amount”, ”Products On Sale”)

# Plot Daily Sales Amount
ggplot(combined_sale, aes(x = date, y = value)) +
  geom_line(aes(col = type)) +
  scale_color_manual(name = ”Sales Amount”, values = c(”black”, ”coral3”)) +
  geom_smooth(method = ”lm”,se = T, col = ”blue”) +
  geom_rect(data = sale.shade, aes(x = NULL, y = NULL, xmin = first, xmax = second, ymin = min, ymax =
  ↪  max, fill = ”darkseagreen”), alpha = 0.4) +
  scale_fill_manual(name = ”Products On Sale”, values = ”darkseagreen”, labels = ”Sale Period”) +
  labs(fill = ”Sale Period”)  +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1),axis.ticks.x = element_line(size = 0.5)) +
  scale_x_date(limits = c(min(combined_sale$date), max(combined_sale$date)),breaks = '3 months') +
  labs(title = ”Time Series Plot for Daily Sales Amount”, x = ”Invoice Date”, y = ”Sales Amount”) +
  guides(color = guide_legend(order = 1),fill = guide_legend(order = 2))

set.seed(10)
predicted_quantity <- numeric(prediction_length)
predicted_quantity[1] <- daily_sales$quantity[1]
random_quantity <- rnorm(prediction_length, mean = 0, sd = sd(diff(daily_sales$quantity)))

for (i in 2:prediction_length) {
  predicted_value <- predicted_quantity[[i-1]] + random_quantity[i]
  if (predicted_value <= 0) {
    predicted_value <- mean(daily_sales$quantity)
  }
  predicted_quantity[i] <- predicted_value
}

predicted_quantity <- as.data.frame(predicted_quantity)
predicted_quantity$InvoiceDate <- seq.Date(from = daily_sales$InvoiceDate[1], by = ”day”, length.out =
 ↪  90)

combined_quantity <- rbind(data.frame(date = daily_sales$InvoiceDate, value = daily_sales$quantity,
 ↪  type = ”Actual”),
                           data.frame(date = predicted_quantity$InvoiceDate, value =
                            ↪  predicted_quantity$predicted_quantity, type = ”Forecast”))

ggplot(combined_quantity, aes(x = date, y = value)) +
  geom_line(aes(col = type)) +
  scale_color_manual(name = ”Sales Quantity”, values = c(”black”, ”coral2”)) +
  geom_smooth(method = ”lm”,se = T, col = ”red”) +
  geom_rect(data = sale.shade, aes(x = NULL, y = NULL, xmin = first, xmax = second, ymin = min, ymax =
   ↪  max, fill = ”darkseagreen”), alpha = 0.4) +
  scale_fill_manual(name = ”Products On Sale”, values = ”darkseagreen”, labels = ”Sale Period”) +
  labs(fill = ”Sale Period”)  +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1),axis.ticks.x = element_line(size = 0.5)) +
  scale_x_date(limits = c(min(combined_quantity$date), max(combined_quantity$date)),breaks = '3
   ↪  months') +
  labs(title = ”Time Series Plot for Daily Sales Quantity”, x = ”Invoice Date”, y = ”Quantity”) +
  guides(color = guide_legend(order = 1),fill = guide_legend(order = 2))


# Fill missing values with 0 before merging
sales_data$Quantity[is.na(sales_data$Quantity)] <- 0
sales_data$RefundQuantity[is.na(sales_data$RefundQuantity)] <- 0

#Reason
# sum up RefundQuantity by Reason
refundR_summary <- aggregate(RefundQuantity ~ Reason, data = sales_data, FUN = sum)

# arrange them in order
refundR_summary <- refundR_summary[order(-refundR_summary$RefundQuantity), ]

# create bar chart to count the number of refund reason
ggplot(refundR_summary, aes(x = reorder(Reason, -RefundQuantity), y = RefundQuantity)) +
  geom_bar(stat = ”identity”, fill = ”pink”, color = ”black”) +
  geom_text(aes(label = RefundQuantity), vjust = -0.5, color = ”black”, size = 3.5) +
  labs(title = ”Return Reason Summary”,
       x = ”Reason”,
       y = ”Total Return Quantity”) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))


# Top 10 Bestselling Products
# sum up Quantity by ProductName
quantity_summary <- aggregate(Quantity ~ ProductName, data = sales_data, FUN = sum)

# arrange them in order
quantity_summary <- quantity_summary[order(-quantity_summary$Quantity), ]

# Keep only top 10 products
quantity_summary <- quantity_summary[1:10,]

# create bar chart to count the number of refund ProductName
```

```
ggplot(quantity_summary, aes(x = reorder(ProductName, -Quantity), y = Quantity, fill = ”Quantity”)) +
  geom_bar(stat = ”identity”, color = ”black”, position = ”stack”) +
  geom_text(aes(label = Quantity), vjust = -0.5, color = ”black”, size = 3.5) +
  labs(title = ”Top 10 Bestselling Products”,
       x = ”Product Name”,
       y = ”Sales Quantity”) +
  scale_fill_manual(values = c(”Quantity” = ”skyblue”),
                    labels = c(”Quantity” = ”Sales Quantity”)) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1, size = 5.5))



# Refund by Product
# sum up RefundQuantity by ProductName
refundP_summary <- aggregate(RefundQuantity ~ ProductName, data = sales_data, FUN = sum)

# Keep only top 10 products
refundP_summary <- refundP_summary[order(-refundP_summary$RefundQuantity), ][1:10,]

# create bar chart to count the number of refund ProductName
ggplot(refundP_summary, aes(x = reorder(ProductName, -RefundQuantity), y = RefundQuantity, fill =
 ↪  ”RefundQuantity”)) +
  geom_bar(stat = ”identity”, color = ”black”, position = ”stack”) +
  geom_text(aes(label = RefundQuantity), vjust = -0.5, color = ”black”, size = 3.5) +
  labs(title = ”Top 10 Highest Return Products”,
       x = ”Product Name”,
       y = ”Return Quantity”) +
  scale_fill_manual(values = c(”RefundQuantity” = ”pink”),
                    labels = c(”RefundQuantity” = ”Return Quantity”)) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1, size = 5.5))



# Sales by product category
# sum up sell price by Product Category Name
PCsales_summary <- aggregate(SellPrice * Quantity ~ CategoryName, data = sales_data, FUN = sum)

# sum up refund price by Product Category Name
refund_summary <- aggregate(SellPrice * RefundQuantity ~ CategoryName, data = sales_data, FUN = sum)

# merge the two summaries
PCsales_summary <- merge(PCsales_summary, refund_summary, by = ”CategoryName”, all = TRUE)

# arrange them in order
```

```
PCsales_summary <- PCsales_summary[order(-PCsales_summary$`SellPrice * Quantity`), ]

# plot
ggplot(PCsales_summary, aes(x = reorder(CategoryName, -`SellPrice * Quantity`))) +
  geom_bar(aes(y = `SellPrice * Quantity`, fill = ”RemainingQuantity”), stat = ”identity”) +
  geom_text(aes(y = `SellPrice * Quantity`, label = `SellPrice * Quantity`), vjust = -0.5, color =
  ↪  ”black”, size = 1.8) +
  geom_bar(aes(y = `SellPrice * RefundQuantity`, fill = ”RefundQuantity”), stat = ”identity”) +
  geom_text(aes(y = `SellPrice * RefundQuantity`, label = `SellPrice * RefundQuantity`), vjust = -0.5,
  ↪  color = ”black”, size = 1.8) +
  labs(title = ”Total Sales (with Refund) by Product Category”,
       x = ”Product Category”,
       y = ”Total Sales”,
       fill = ””) +
  scale_fill_manual(values = c(”RemainingQuantity” = ”skyblue”, ”RefundQuantity” = ”pink”),
                    labels = c(”RemainingQuantity” = ”Remaining Sales”, ”RefundQuantity” = ”Refund
                    ↪  Sales”)) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Quantity by Product Category
# sum up Quantity by Product Category Name
PCquantity <- aggregate( Quantity ~ CategoryName, data = sales_data, FUN = sum)

# sum up Refund Quantity by Product Category Name
refund_quantity <- aggregate(RefundQuantity ~ CategoryName, data = sales_data, FUN = sum)

# merge the two summaries
PCquantity <- merge(PCquantity, refund_quantity, by = ”CategoryName”, all = TRUE)

#arrange order
PCquantity <- PCquantity[order(-PCquantity$Quantity, -PCquantity$RefundQuantity), ]

# stacked bar plot for quantity(with return) by category name
ggplot(PCquantity, aes(x = reorder(CategoryName, -Quantity), y = Quantity)) +
  geom_bar(aes(fill = ”RemainingQuantity”), stat = ”identity”) +
  geom_text(aes(label = Quantity), vjust = -0.5, color = ”black”, size = 3.5) +
  geom_bar(aes(y = RefundQuantity, fill = ”RefundQuantity”), stat = ”identity”) +
  labs(title = ”Quantity (vs RefundQuantity) by CategoryName”,
       x = ”Category Name”,
       y = ”Total Quantity”) +
  scale_fill_manual(values = c(”RemainingQuantity” = ”skyblue”, ”RefundQuantity” = ”pink”),
                    name = ”Type”) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  guides(fill = guide_legend(title = ”Type”))
```

```
# Quantity by Suppliers
# sum up Quantity by Supplier Name
SUPquantity <- aggregate(Quantity ~ SupplierName, data = sales_data, FUN = sum)

# sum up Refund Quantity by Supplier Name
SUPrefund_quantity <- aggregate(RefundQuantity ~ SupplierName, data = sales_data, FUN = sum)

# merge the two summaries
SUPquantity <- merge(SUPquantity, SUPrefund_quantity, by = ”SupplierName”, all = TRUE)

#arrange order
SUPquantity <- SUPquantity[order(-SUPquantity$Quantity, -SUPquantity$RefundQuantity), ]

# Keep only top 5 suppliers
SUPquantity <- SUPquantity[1:5,]

# stacked bar plot for quantity(with return) by Supplier name
ggplot(SUPquantity, aes(x = reorder(SupplierName, -Quantity), y = Quantity)) +
  geom_bar(aes(fill = ”RemainingQuantity”), stat = ”identity”) +
  geom_text(aes(label = Quantity), vjust = -0.5, color = ”black”, size = 3.5) +
  geom_bar(aes(y = RefundQuantity, fill = ”RefundQuantity”), stat = ”identity”) +
  labs(title = ”Top 5 Suppliers: Quantity (with Return Quantity)”,
       x = ”Supplier Name”,
       y = ”Total Quantity”) +
  scale_fill_manual(values = c(”RemainingQuantity” = ”skyblue”, ”RefundQuantity” = ”pink”),
                    name = ”Type”) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  guides(fill = guide_legend(title = ”Type”))



# transaction summary
# computing transaction amount in each payment Method
py_method_summary <- aggregate(SellPrice * Quantity ~ PaymentMethod, data = sales_data, FUN = sum)

# in order
py_method_summary <- py_method_summary[order(-py_method_summary$`SellPrice * Quantity`), ]

# plot
ggplot(py_method_summary, aes(x = reorder(PaymentMethod, -`SellPrice * Quantity`), y = `SellPrice *
↪  Quantity`)) +
  geom_bar(stat = ”identity”, fill = ”lavender”, color = ”black”) +
  geom_text(aes(label = `SellPrice * Quantity`), vjust = -0.5, color = ”black”, size = 3.5) +
  labs(title = ”Total Sales by Payment Method”,
       x = ”Payment Method”,
       y = ”Total Sales”) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```