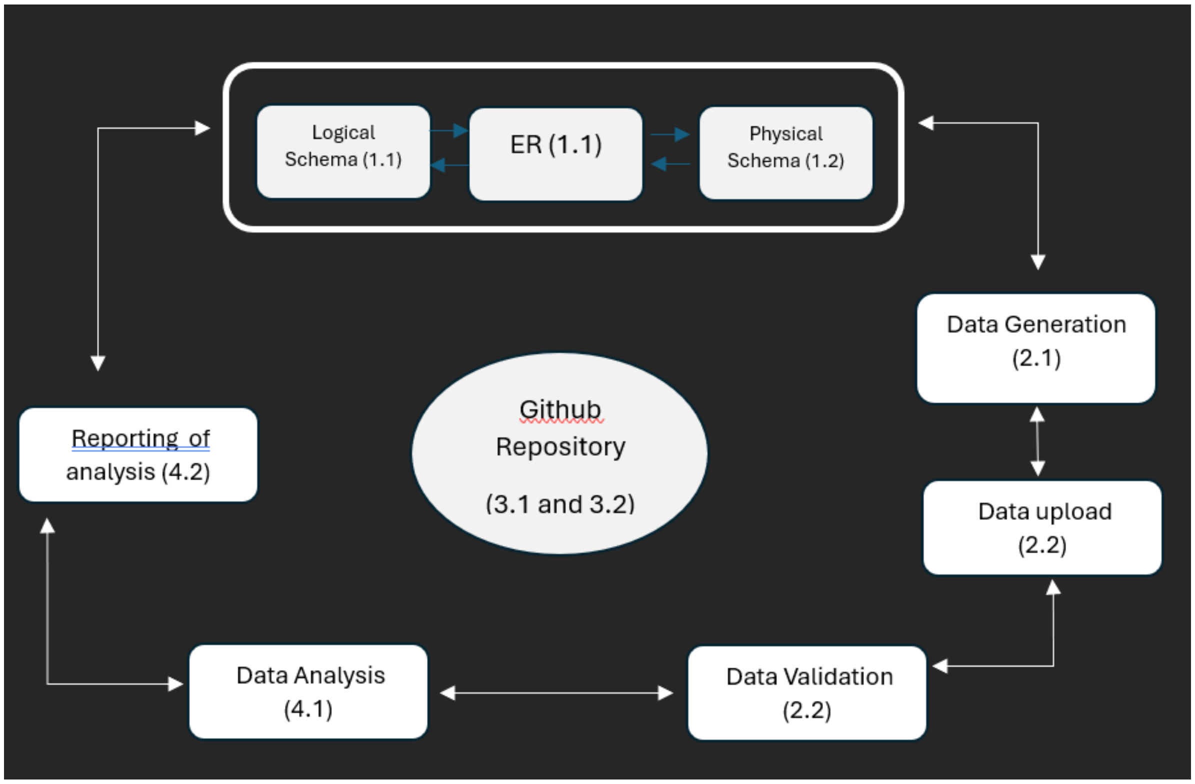
# Data Management Group 19

## Introduction

We project focuses on creating a robust e-commerce database system through database design, SQL schema creation, synthetic data generation and data quality check and validation using R. We have leveraged Github for version control and automation of processing through designing workflow. We have performed advanced data analysis and comprehensive reporting enabling actionable insights for non-technical stakeholders.

Below is the structure of workflow we followed for the project. This process is done in iteration to get best results.

The Link to the Github repository where we have all the files, workflow , data and supporting - https://github.com/Lunarfi/G19



## Part 1 Database Design and Implementation

We have created a database design for a e-commerce company and implemented it by creating physical schema.

### 1.1 E-R Diagram Design

#### Logical Schema

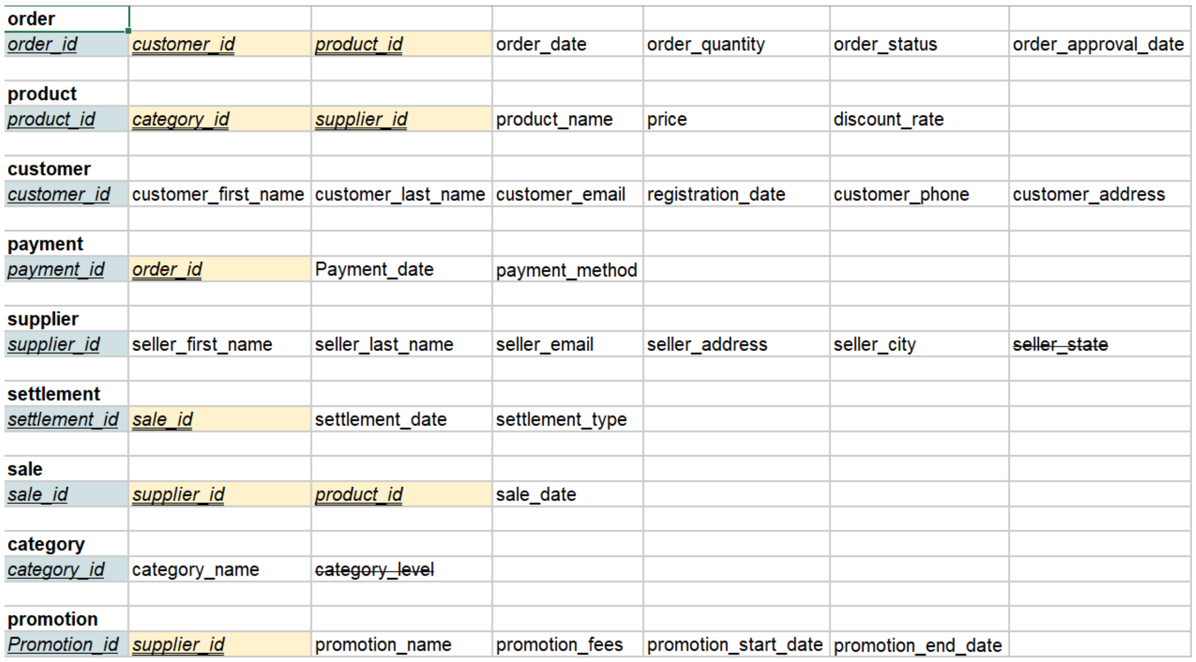
Based on our understanding and usage of different e-commerce, we created a logical schema(LS) on excel first regarding entities we want to include and there relevant attributes.

After many iteration, we created LS for the following entities (Refer file Database logical schema design) :

1. Customer entity - Customer will purchase and rate the product
2. Supplier entity - Supplier will sell and promote the product
3. Category - There can be several category and some categories can be sub categories of one another.

4. Product - Product listed by a supplier and belongs to a category
5. Order - customer will order a product
6. Sale - supplier will make a sale of a product
7. payment - customer pays for the order
8. settlement - supplier receives settlement for the sale
9. Promotion - supplier promotes for better reach to all customers

The below snip is of tab Updated table list based on which we updated our ER diagram and physical schema in next steps.

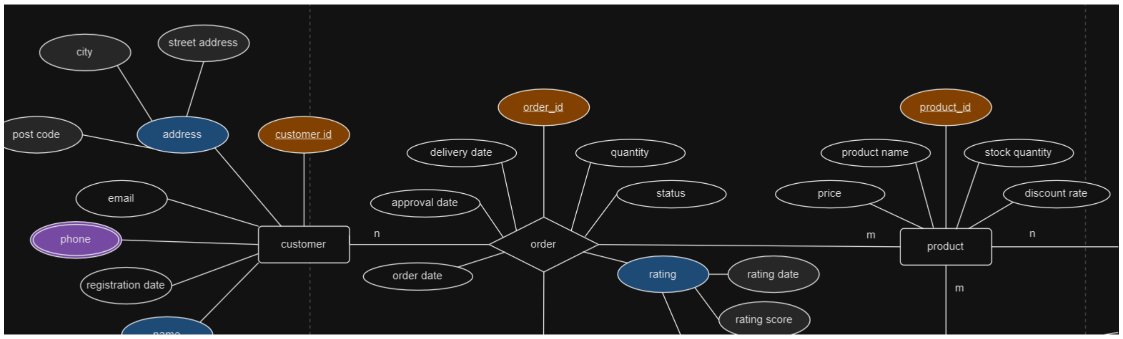


| | | | | | | |
|---|---|---|---|---|---|---|
| **order** | | | | | | |
| *order_id* | *customer_id* | *product_id* | order_date | order_quantity | order_status | order_approval_date |
| | | | | | | |
| **product** | | | | | | |
| *product_id* | *category_id* | *supplier_id* | product_name | price | discount_rate | |
| | | | | | | |
| **customer** | | | | | | |
| *customer_id* | customer_first_name | customer_last_name | customer_email | registration_date | customer_phone | customer_address |
| | | | | | | |
| **payment** | | | | | | |
| *payment_id* | *order_id* | Payment_date | payment_method | | | |
| | | | | | | |
| **supplier** | | | | | | |
| *supplier_id* | seller_first_name | seller_last_name | seller_email | seller_address | seller_city | ~~seller_state~~ |
| | | | | | | |
| **settlement** | | | | | | |
| *settlement_id* | *sale_id* | settlement_date | settlement_type | | | |
| | | | | | | |
| **sale** | | | | | | |
| *sale_id* | *supplier_id* | *product_id* | sale_date | | | |
| | | | | | | |
| **category** | | | | | | |
| *category_id* | category_name | ~~category_level~~ | | | | |
| | | | | | | |
| **promotion** | | | | | | |
| *Promotion_id* | *supplier_id* | promotion_name | promotion_fees | promotion_start_date | promotion_end_date | |

## E-R Diagram

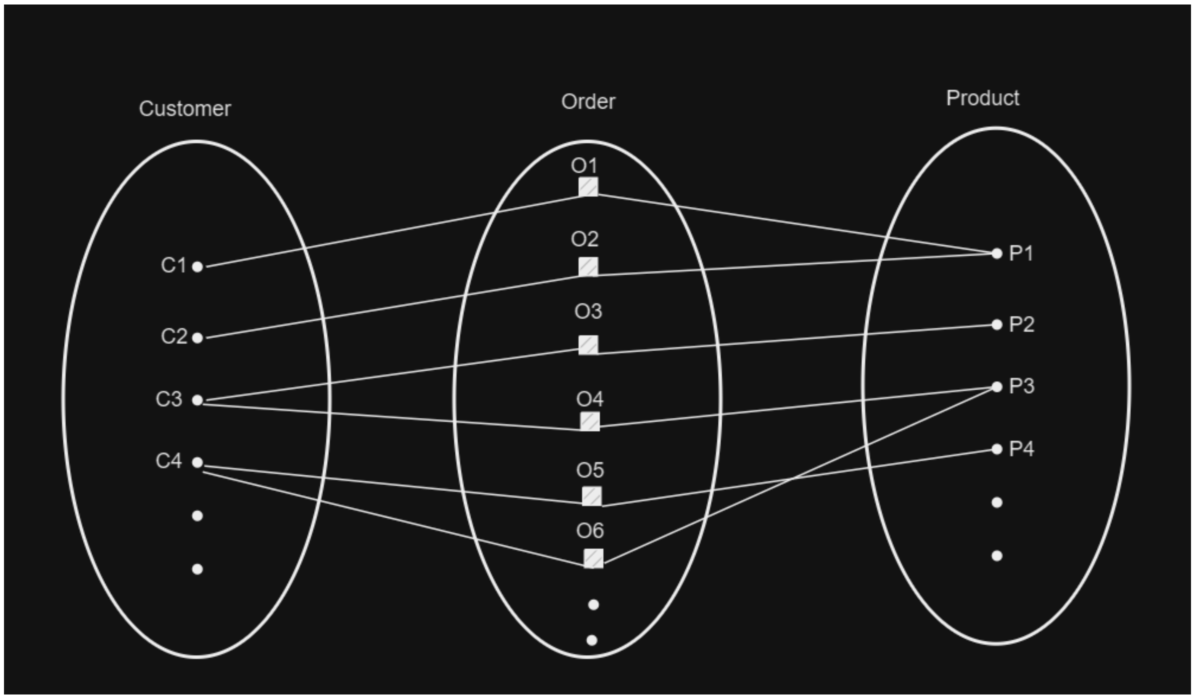
We made our E-R diagram based on logical schema and for every iteration revised the ER. Below are the snips of final E-R diagram.

- Customer order Product - There is a many-to-many relationship between customer and product. As customer can order many products and a product can be ordered by many customers. Customer (customer_id) orders a product (product_id), hence a order_id is a primary key with customer_id and product_id is foreign key in order table to differentiate all orders.

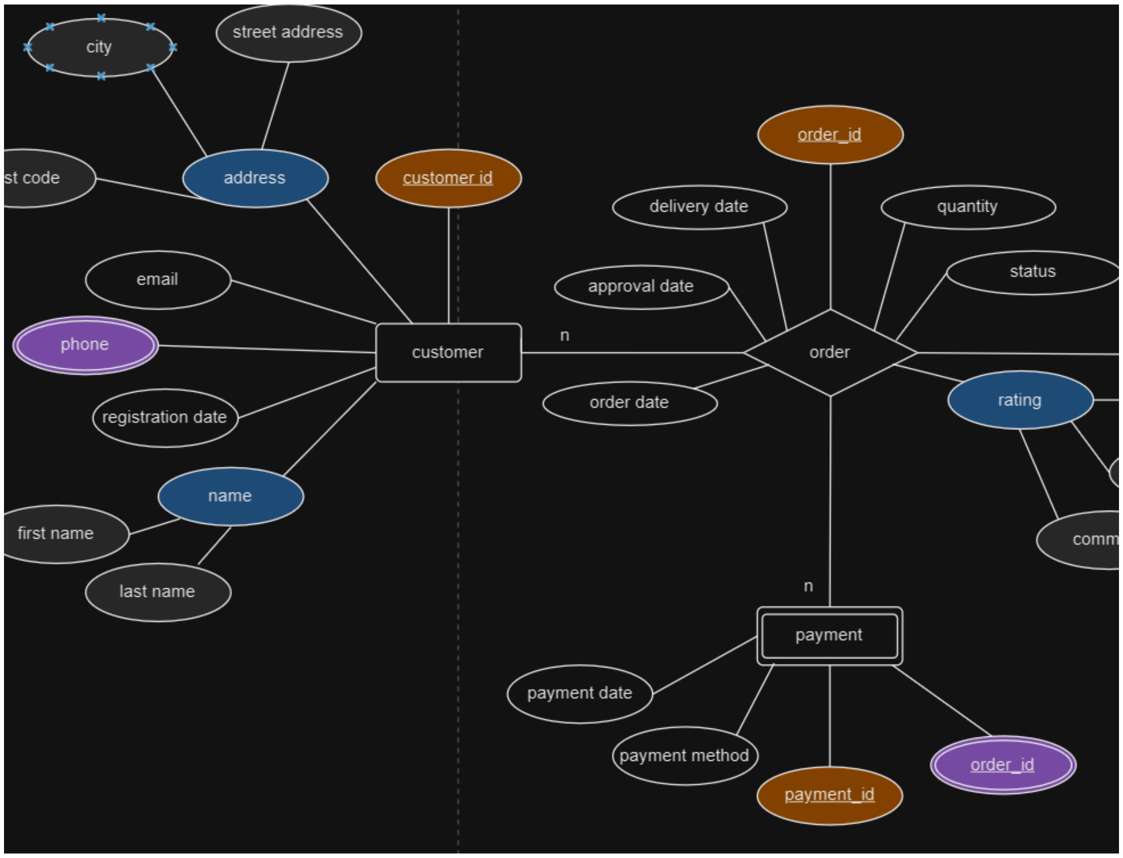


Below are the assumption and relationship set for customer orders product:

1. 1 customer can buy many products and 1 product can be bought by multiple customers.
2. 1 customer can make multiple orders.
3. 1 order contain only 1 product, 1 order will belong to 1 customer.
4. 1 order can have multiple quantity of same product.
5. 1 customer can only give 1 rating per order.

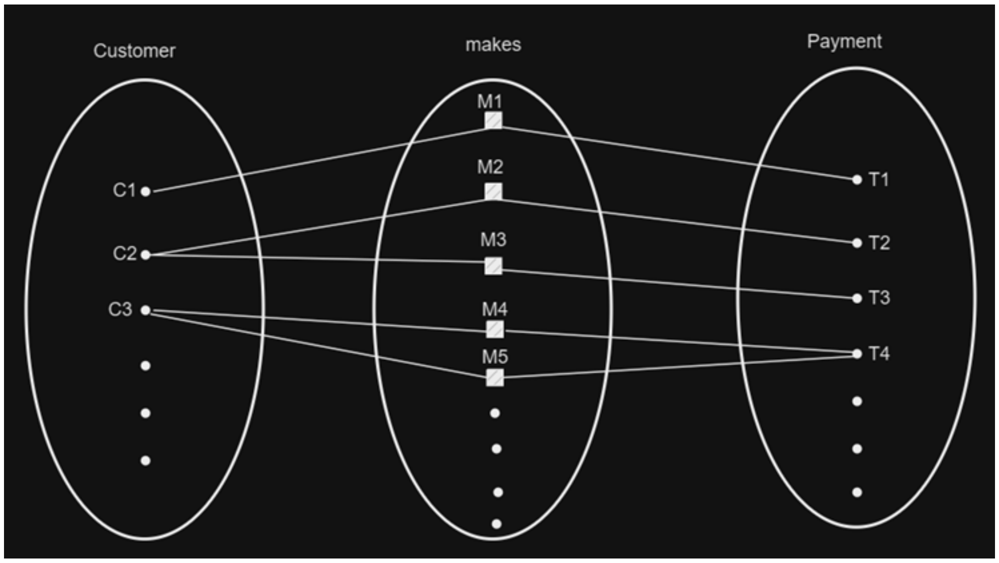


- Customer makes payment - There is one to many relationship between customer and payment. Payment is a weak entity here because without the customer, a payment record might lack context or be meaningless.Customer (customer_id) makes payment (payment_id) for order(order_id).

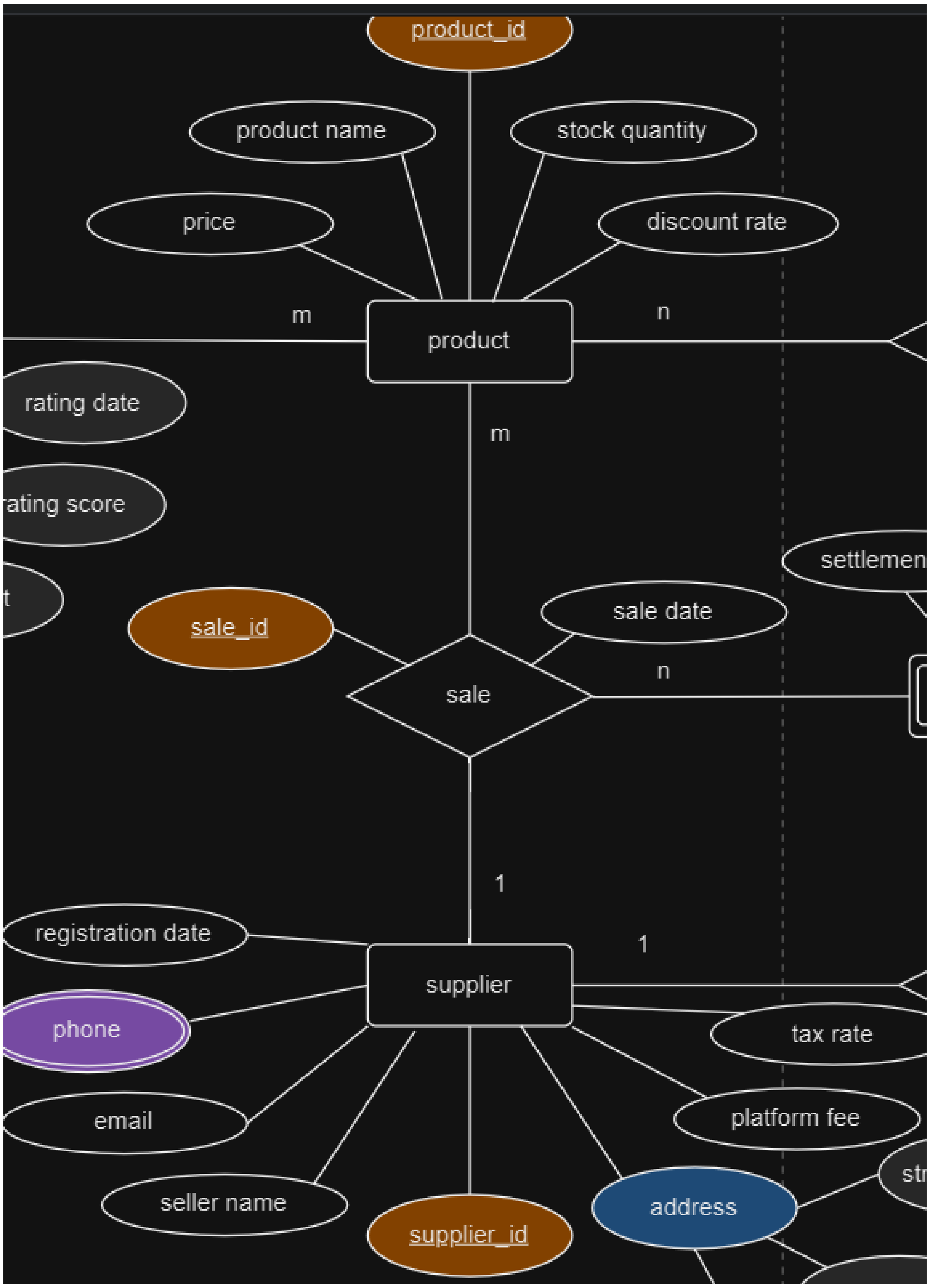


Below are the assumption and relationship set for customer makes payment:

1. 1 customer can make multiple payments.
2. 1 payment belong to only 1 customer.
3. 1 customer can make 1 payment for multiple orders.
4. 1 order will belong to only 1 payment, so partial payment of order cannot be made.
5. Payment is a weak entity since it cannot exist without the customer entity.

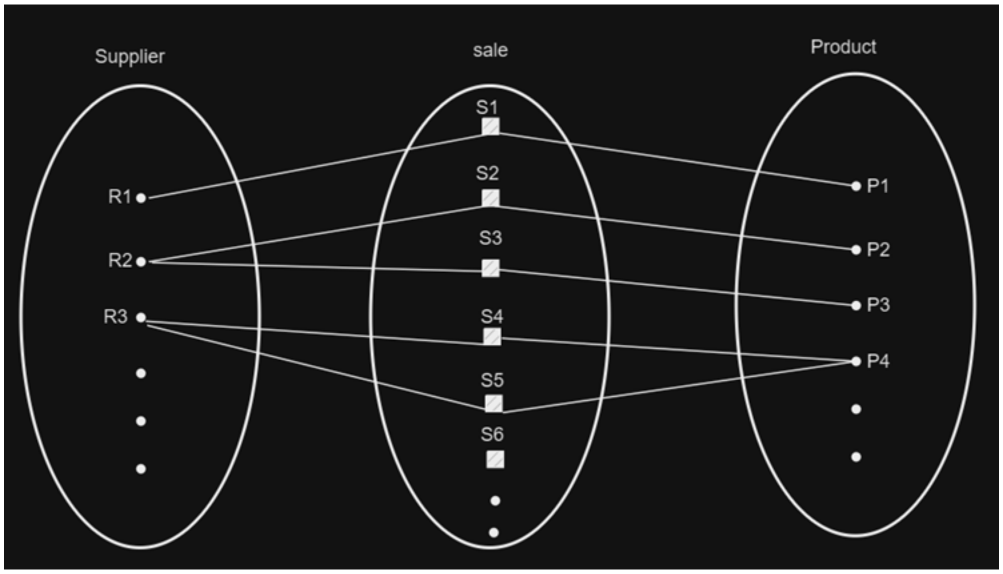


- Supplier sale product - There is one to many relationship between supplier and product. Supplier can sale many product and one product will belong to only one supplier. Supplier (supplier_id) sale a product(product_id). Product_id is a unique id for a product sold by a particular supplier. Each sale has a unique sale_id which related to a product_id which is related to a supplier_id.

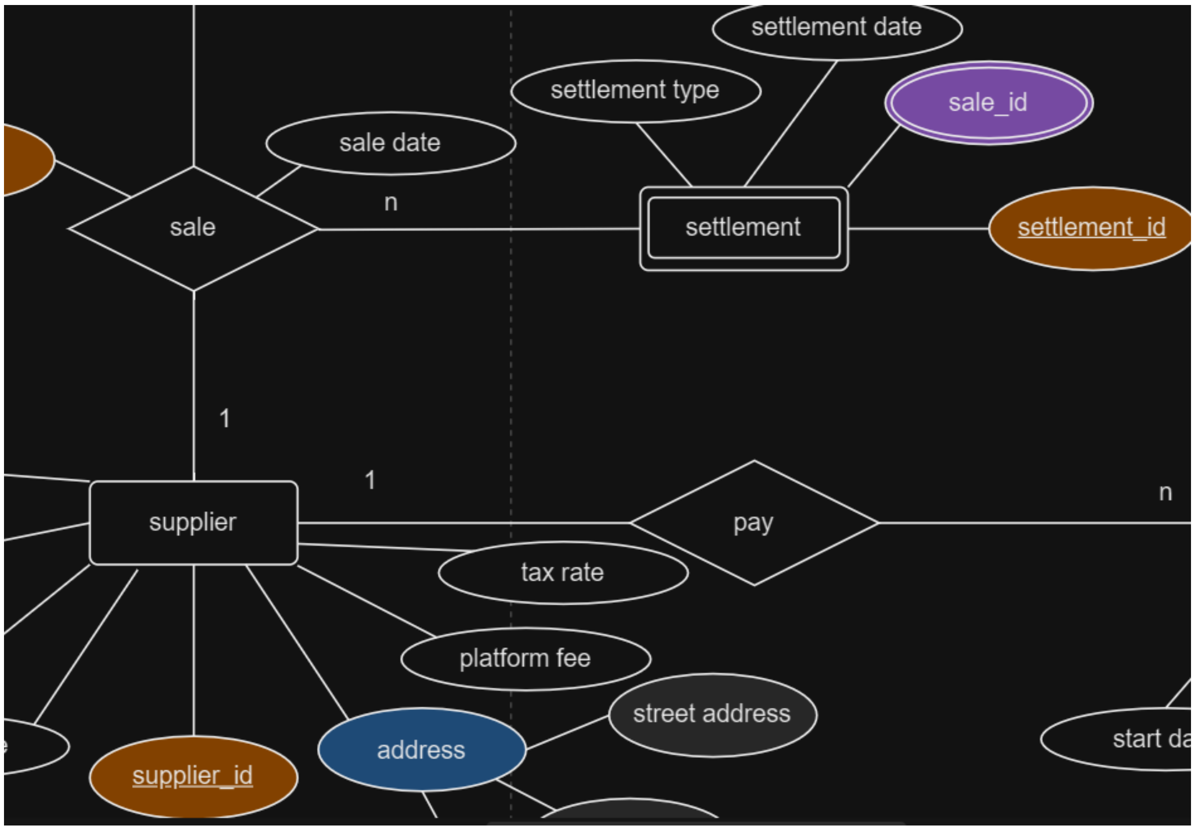

product_id
product name
stock quantity
price
discount rate
m
product
n
rating date
m
ating score
t
settlemen
sale_id
sale date
sale
n
1
registration date
1
supplier
phone
tax rate
email
platform fee
seller name
str
supplier_id
address

Below are the assumption and relationship set for supplier sale product:

1. 1 supplier can sale multiple products, 1 product can only be sold by 1 supplier.
2. 1 sale will contain 1 product, and 1 sale will belong to 1 supplier.
3. 1 supplier can have multiple sale for same product.
4. Same description product sold by different supplier will be considered as different product.

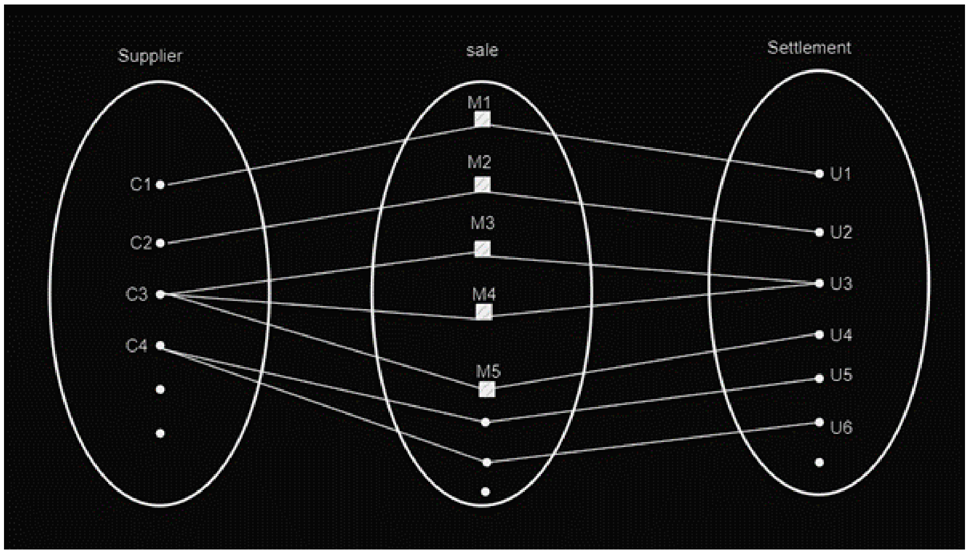


- Supplier sale settlement - There is one to many relationship between supplier and settlement. Settlement is a weak entity here because without the supplier, a settlement record might lack context or be meaningless.Supplier (supplier_id) receives settlement(settlement_id) for sale(sale_id).

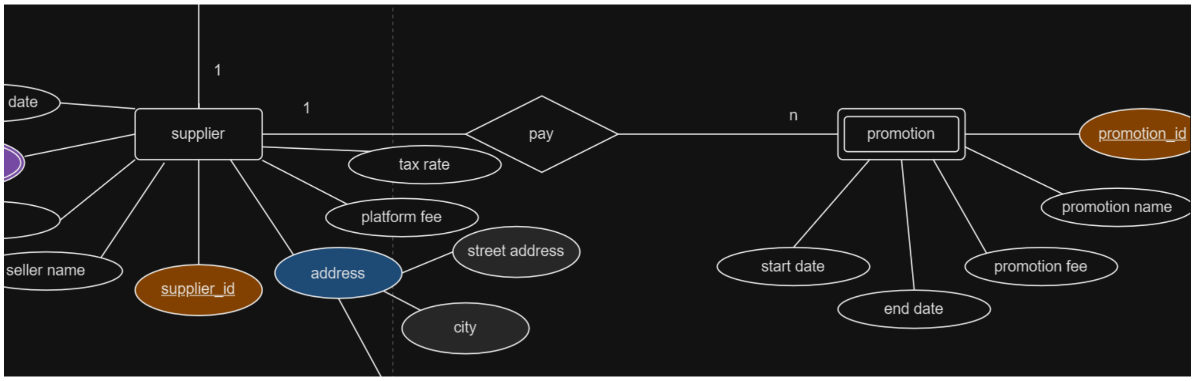


Below are the assumption and relationship set for Supplier sale settlement:

1. 1 supplier can have multiple settlement, 1 settlement belongs to 1 supplier.
2. 1 settlement can belong to multiple sale.
3. 1 sale will belong to only 1 settlement, there is no partial settlement for sale.
4. Settlement is a weak entity since it cannot exist without the supplier and product entity
5. All sales settled in 1 settlement must belong to same supplier

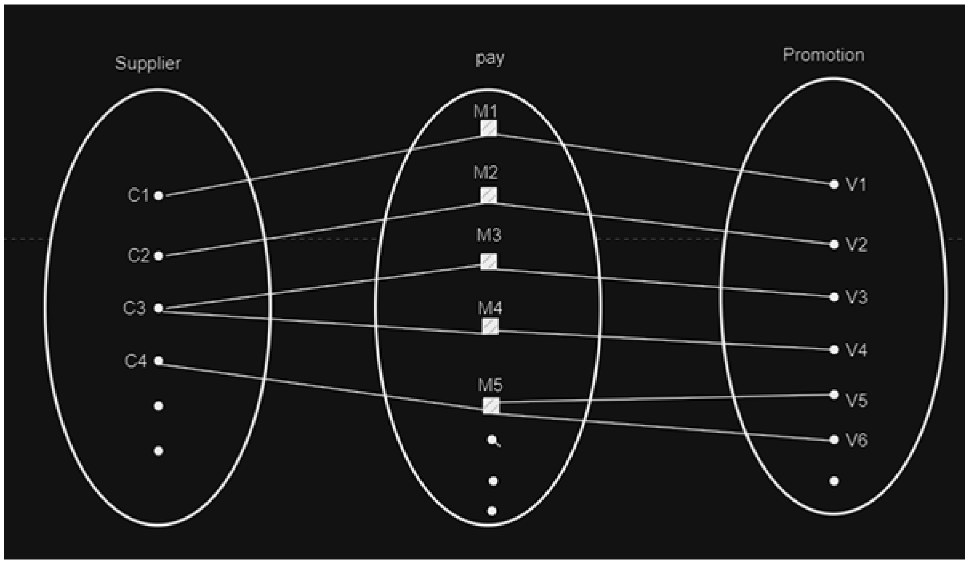


- Supplier pays promotion - There is a one to many relationship between Supplier pays promotion. Promotion is a weak entity here because without the supplier, a Promotion record might lack context or be meaningless.Supplier (supplier_id) pays for promotion(promotion_id).

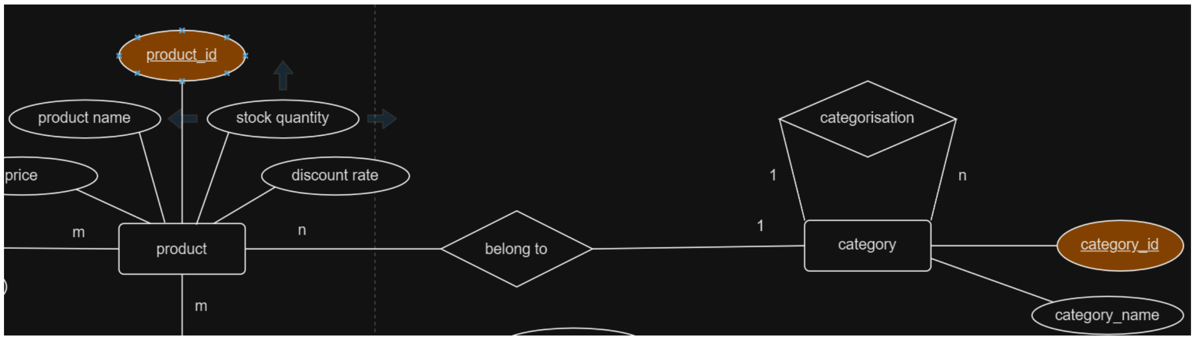


Below are the assumption and relationship set for Supplier pays Promotion:

1. 1 supplier can pay for multiple promotion, 1 promotion can only belong to 1 supplier
2. 1 promotion can be used to advertise multiple products from 1 supplier
3. 1 supplier can pay for multiple promotions together
4. Promotion is a weak entity since it cannot exist without the supplier entity

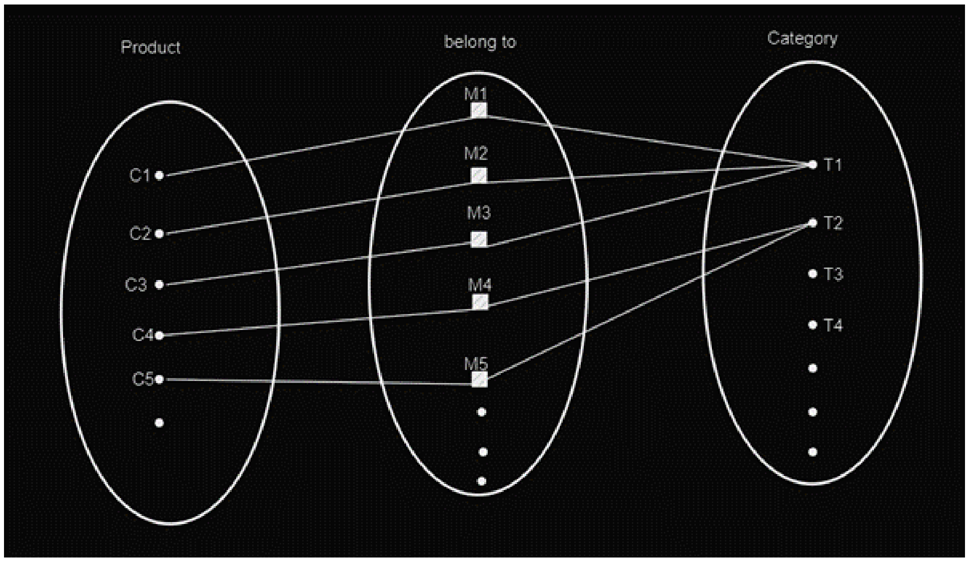


- Product belongs to Category - There is one to many relationship between Product and category. Product_id can fall into only 1 category_id and 1 category_id can have a lot of product_id association.

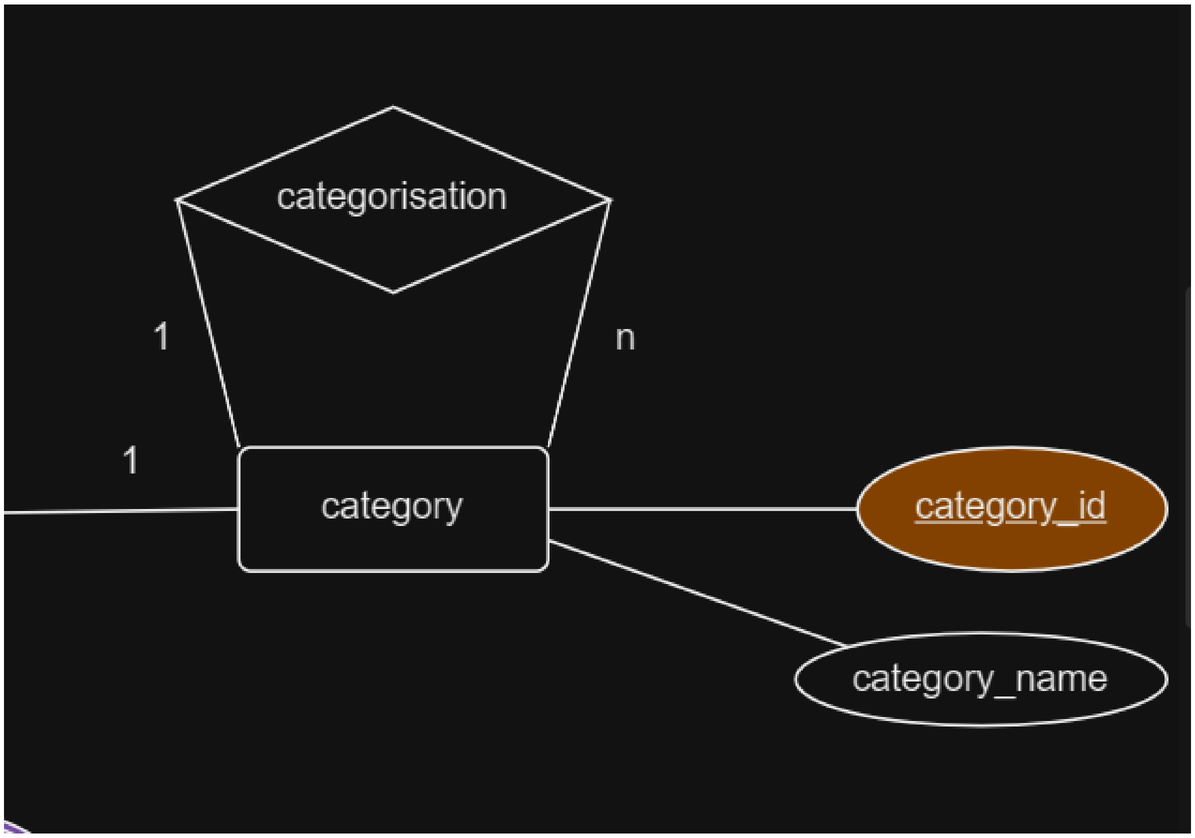


Below are the assumption and relationship set for Product belongs to Category:

1. 1 product belong to one category, 1 category can consist of multiple products.

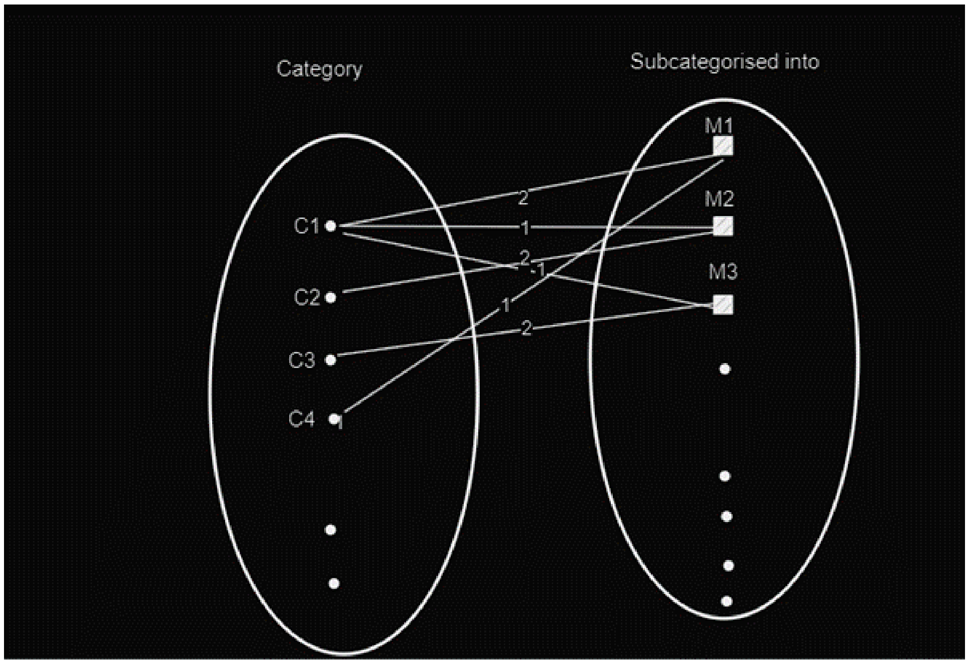


Category and categorization - There is self-referencing (one-to-many) relationship between category and sub-categorization. One category can have many sub-categorization but each categorization will belong to only one parent category.



Below are the assumption and relationship set for Category has a categorization:

1. Each category has one parent category and can have multiple child category.
2. A category is subcategorised into various categorisation that are included in category table.
3. A category can be parent category of one category and sub category of another category.



## 1.2 SQL Database Schema Creation

Once we have understanding and structure of E-R Diagram we move on to creation of Physical schema in SQL and data related assumption for structure of each table.

### Physical Schema

All the physical schema have been defined in file "data_schema.R" where we have created all tables structure in a database using command "CREATE" and defined structure of each attribute of table like how many characters, NOT NULL, date, integer, Float, Primary key and Foreign key references.

1. Order Schema - Order_id is the Primary key, Customer_id from Customer table and product_id from Product table are foreign key which can't be NULL. Order date can also not be NULL.

2. Product Schema - product_id is the Primary key, Category_id from Category table and supplier_id from Supplier table are foreign key which can't be NULL. Price and Discount rate are Floats as they can be in decimal number.

3. Customer Schema - Customer_id is the Primary key for Customer table. Customer email address and customer phone can't be NULL as they are necessary attributes for customer registration.

4. Payment Schema - payment_id is the Primary key, order_id from Order table can't be NULL.

5. Supplier Schema - Supplier_id is the Primary key for Supplier table. Supplier email address and Supplier phone can't be NULL as they are necessary attributes for supplier registration. Platform fees and Tax rate are Floats as they can be expressed in decimal.

6. Settlement Schema - settlement_id is the Primary key, sale_id from Sales table can't be NULL.

7. Sales Schema - sale_id is the Primary key, supplier_id from supplier table and product_id from Product table are foreign key which can't be NULL. Sale date can also not be NULL.

8. Category Schema - category_id is the primary key.

9. Promotion Schema - Promotion_id is the Primary key, Supplier_id from Supplier table can't be NULL.

```r
library(RSQLite)
library(DBI)

# Set database file name
db_file <- "mydatabase.db"

# Check if database file exists
if (file.exists(db_file)) {
  # If exists, delete the file
  unlink(db_file)
}

data <- dbConnect(SQLite(), "mydatabase.db")

# Define the schema for the order table
Orders_schema <- "
CREATE TABLE IF NOT EXISTS Orders(
   order_id VARCHAR(6) PRIMARY KEY,
   product_id VARCHAR(6) NOT NULL,
   customer_id VARCHAR(6) NOT NULL,
   order_date DATE NOT NULL,
   order_quantity INTEGER,
   order_status VARCHAR(20),
   order_approval_date DATE,
   order_delivery_date DATE,
   rating_date DATE,
   rating_score INT,
   rating_comment CHAR,
   FOREIGN KEY ('customer_id') REFERENCES Customer ('customer_id'),
   FOREIGN KEY ('product_id') REFERENCES Product ('product_id')
);"


#schema for product table
Product_schema <- "
CREATE TABLE IF NOT EXISTS 'Product'(
  'product_id' VARCHAR(6) PRIMARY KEY,
  'category_id' VARCHAR(6) NOT NULL,
  'supplier_id' VARCHAR(6) NOT NULL,
  'product_name' CHAR,
  'price' FLOAT,
  'discount_rate' FLOAT,
  FOREIGN KEY ('category_id') REFERENCES Category('category_id'),
  FOREIGN KEY ('supplier_id') REFERENCES Supplier('supplier_id')
);"

#schema for customer table
Customer_schema <- "
CREATE TABLE IF NOT EXISTS 'Customer'(
  'customer_id' VARCHAR(6) PRIMARY KEY,
  'customer_first_name' CHAR(50),
  'customer_last_name' CHAR(50) ,
  'customer_email' VARCHAR(200) NOT NULL,
  'registration_date' DATE,
  'customer_phone' INT(11) NOT NULL UNIQUE,
  'customer_address' VARCHAR ,
  'customer_city' CHAR,
  'customer_postcode' VARCHAR
);"
```

```
# Define the schema for the payment table
Payment_schema <- "
CREATE TABLE IF NOT EXISTS Payment (
   'payment_id' VARCHAR(6) PRIMARY KEY,
   'order_id' VARCHAR(6) NOT NULL,
   'payment_date' DATE,
   'payment_method' CHAR,
   FOREIGN KEY ('order_id') REFERENCES Orders('order_id')
);"

#schema for supplier table
Supplier_schema <- "
CREATE TABLE IF NOT EXISTS 'Supplier'(
'supplier_id' VARCHAR(6) PRIMARY KEY,
'seller_first_name'  TEXT,
'seller_last_name'  TEXT,
'seller_email' VARCHAR(200) NOT NULL,
'seller_address' VARCHAR(200),
'seller_city' CHAR(50) ,
'seller_postcode' VARCHAR,
'seller_phone' INT(11) NOT NULL UNIQUE,
'registration_date' DATE ,
'platform_rate' FLOAT,
'tax_rate' FLOAT
);"


# Define the schema for settlement table
Settlement_schema <- "
CREATE TABLE IF NOT EXISTS Settlement (
   settlement_id VARCHAR(6) PRIMARY KEY,
   sale_id VARCHAR(6) NOT NULL,
   settlement_date DATE,
   settlement_type TEXT,
   FOREIGN KEY ('sale_id') REFERENCES Sales('sale_id')
);"

# Define the schema for sales table
Sales_schema <- "
CREATE TABLE IF NOT EXISTS Sales (
    sale_id VARCHAR(6) PRIMARY KEY,
    supplier_id VARCHAR(6) NOT NULL,
    product_id VARCHAR(6) NOT NULL,
    sale_date DATE NOT NULL,
    FOREIGN KEY ('product_id') REFERENCES Product('product_id'),
    FOREIGN KEY ('supplier_id') REFERENCES Supplier('supplier_id')
);"

# Define the schema for category table
Category_schema <- "
CREATE TABLE IF NOT EXISTS Category (
  category_id VARCHAR(6) PRIMARY KEY,
  category_name VARCHAR(6)
);"


#schema for promotion table
Promotion_schema <- "
CREATE TABLE IF NOT EXISTS 'Promotion'(
  'promotion_id' VARCHAR(6) PRIMARY KEY,
  'supplier_id' VARCHAR(6) NOT NULL,
  'promotion_name' CHAR,
  'promotion_fees' FLOAT ,
  'promotion_start_date' DATE,
  'promotion_end_date' DATE,
  FOREIGN KEY ('supplier_id') REFERENCES Supplier('supplier_id')
);"

# Execute the schema creation queries
dbExecute(data, Orders_schema)
```

```
[1] 0
```

```
dbExecute(data, Product_schema)
```

```
[1] 0
```

```
dbExecute(data, Customer_schema)
```

```
[1] 0
```

```
dbExecute(data, Payment_schema)
```

```
[1] 0
```

```
dbExecute(data, Supplier_schema)
```

```
[1] 0
```

```
dbExecute(data, Settlement_schema)
```

```
[1] 0
```

```
dbExecute(data, Sales_schema)
```

```
[1] 0
```

```
dbExecute(data, Category_schema)
```

```
[1] 0
```

```
dbExecute(data, Promotion_schema)
```

```
[1] 0
```

```
# Close the connection
dbDisconnect(data)
```

All the schema created are normalized up to 3NF form during data creation.

### Data-Related Assumption

We have created the physical schema considering the below data related assumptions for each table. The constraints added are based logical reasoning which we have mentioned in data-related assumption below for each table.

- Customer table

  1. Customer can have only 1 address at any time.
  2. Customer address will be the delivery address as well.
  3. Customer can have 1 phone number at any time.
  4. Customer can have 1 email address at any time.
  5. Customer's name is broken down into first and last name.
  6. There is no balance outstanding or in credit of a customer's account.
  7. Registration date is before or same as 1st order date of a customer (verified in Validation stage 2)

- Supplier table

  1. Supplier can have only 1 address at any time.
  2. Supplier can have 1 phone number at any time.
  3. Supplier can have 1 email address at any time.
  4. Supplier's name is broken down into first and last name.
  5. There is no balance outstanding or in credit of a Supplier's account.
  6. Registration date is before or same as 1st sale date of a supplier(verified in Validation stage 2)
  7. Platform rate charged by e-commerce company can vary for each supplier but will not be more than 25%(verified in Validation stage 1)
  8. Tax rate is same for everyone i.e. 10%(verified in Validation stage 1)

- Product table

  1. Discount is different for each product (product_id) and will not be more than 20%. (verified in Validation stage 1)
  2. Prices of all product listed on e-commerce by supplier will be between min 5 pounds and maximum 100 pounds. (verified in Validation stage 1)
  3. Currency used in only Great Britain pounds (GBP).
  4. Product can have only 1 discount at any time.
  5. All Products are in stock at any point of time.
  6. There can be same product name for different suppliers.

- Order Table

  1. Order date is always after or equal to customer's registration date. (add constraint as a trigger)
  2. Order approval date is always after or equal to order date and payment date. (verified in Validation stage 1)
  3. Order delivery date is always after or equal to order approval date. (verified in Validation stage 1)
  4. No one can order more than 20 quantities in 1 order. (verified in Validation stage 1)
  5. Order status can be only "pending","processing","shipped" or "delivered" (verified in Validation stage 1)
  6. Rating date is always after or equal to order delivery date. (verified in Validation stage 1)
  7. It is not necessary that all customer provides rating and rating comment.
  8. Rating will be on a scale of 1 to 5 only. (verified in Validation stage 1)

- Payment table

  1. Payment date is always after order date. (verified in Validation stage 2)

2. There is only electronic payment option. No Cash on Delivery available. (verified in Validation stage 1)

- Sale table

  1. Sale date is the date when supplier sales i.e., same as when customer order that product.(verified in Validation stage 2)

- Settlement table

  1. Settlement date is always after sale date. (verified in Validation stage 2)
  2. Settlement methods can only be bank transfer, cash, cheque, bank draft, card. (verified in Validation stage 1)

- Promotion table

  1. Promotion start date is always after or equal to supplier's registration date. (verified in Validation stage 2)
  2. Promotion start date is always before promotion end date. (verified in Validation stage 1)
  3. Promotion period cant be more than 30days for a specific promotion_id. (verified in Validation stage 1)
  4. Promotion fees cant be more than 1000 GBP. (verified in Validation stage 1)

- Category table

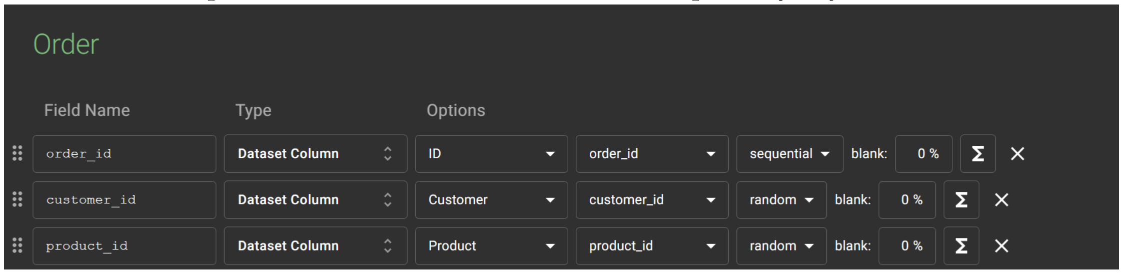
  1. Category name includes parent category and sub-category.

Based on the above defined schema and data condition we perform the later task and validate in Task 2.2 as referred above as well.

## Part 2: Data Generation and Management

### 2.1 Synthetic Data Generation

We generated synthetic data using Mockaroo to simulate an e-commerce environment. We used several prompts to ensure the relationships between entities are accurately depicted (Refer file "Prompts for data generation"). Some Prompts along with code used on Mockaroo are explained below.

- Ensures the foreign keys for each table refer to the primary key of the related tables (e.g. customer_id and product_id on order table refer to the primary key of the latter tables). 

Order

| Field Name | Type | Options | | | | |
|---|---|---|---|---|---|---|
| order_id | Dataset Column | ID | order_id | sequential | blank: | 0 % |
| customer_id | Dataset Column | Customer | customer_id | random | blank: | 0 % |
| product_id | Dataset Column | Product | product_id | random | blank: | 0 % |

- Ensures order_approval_date on order table is null if status is still "pending", and is within 3 days after the order_date.

```
if order_status.casecmp("pending") == 0
  order_approval_date = ""
else
  order_approval_date =  order_date + days(random(0,3))
end
```

- Ensures order_delivery_date on order table has value if status is "delivered" and within 14 days after order_date.

```
if order_status.casecmp("delivered") == 0
  order_delivery_date = order_date + days(random(4,14))
else
 ""
end
```

- Ensures rating_date on order table has value only if status is "delivered".

```
if order_status.casecmp("delivered") == 0
  rating_date = order_date + days(random(15,25))
else
  ""
end
```

- Ensures rating_score on order table ranges between 1-5 and only has value if rating_date is not null.

```
if rating_date.blank? then "" else random(1, 5) end
```

- Ensures rating_comment on order table only has value if rating_score is not null and not all rating_score has rating_comment.

```
comments = [
  "Great product, very satisfied with its performance!",
  "Highly recommended! This product exceeded my expectations.",
  "Good value for money. Does the job well.",
  "Average product, nothing special about it.",
  "Disappointing. Didn't meet my needs as expected.",
  "Quality could be better for the price.",
  "Impressive design, but functionality could be improved.",
  "Easy to use and convenient.",
  "Needs better instructions for setup.",
  "Works perfectly, exactly what I needed!"
]

rating_comment = rating_date.blank? ? "" : comments.sample
```

- Ensures promotion_end_date on promotion table is within 30 days after the promotion_start_date

```
promotion_start_date + days(random(10,30))
```

We generated the following datasets for the initial workflow testing after considering physical schema data structure and Data-related assumptions mentioned in part 1.2.

Customer table - 47 unique customer

Category table - 10 unique category

Orders - 855 unique order id

Payment - 855 unique payment id

Product - 97 unique product id

Promotion - 94 unique promotion id

Sales - 855 unique sale id

Settlement - 855 unique settlement id

Supplier - 47 unique supplier

## 2.2 Data Import and Quality Assurance

### Data Import

First part of data ingestion is importing the data from CSV files using R code "validation.R" that will run on the github workflow. Since multiple files can be uploaded for the same table, a function to search the substring of the file name and check whether it matches the table name is used to ensure all the files are extracted.

```
# Load necessary libraries
library(readr)
# Function to read CSV files from a directory and categorize them into different data frames
read_and_categorize_csv <- function(directory) {
  # Get list of CSV files in the directory
  csv_files <- list.files(directory, pattern = "\\.csv$", full.names = TRUE)

  # Initialize a list to store data frames
  data_frames <- list(
    Category = NULL,
    Customer = NULL,
    Orders = NULL,
    Payment = NULL,
    Product = NULL,
    Promotion = NULL,
    Sales = NULL,
    Settlement = NULL,
    Supplier = NULL
  )

  # Loop through each CSV file
  for (csv_file in csv_files) {
    # Read CSV file into a data frame
    data <- read.csv(csv_file)

    # Extract file name without extension
    file_name <- tools::file_path_sans_ext(basename(csv_file))

    # Determine which data frame to store the data
    if (grepl("Customer", file_name, ignore.case = TRUE)) {
      data_frames$Customer <- rbind(data_frames$Customer, data)
      # Output variable names
      cat("Variables in Customer data frame:\n")
      print(names(data_frames$Customer))
    } else if (grepl("Category", file_name, ignore.case = TRUE)) {
      data_frames$Category <- rbind(data_frames$Category, data)
      # Output variable names
      cat("Variables in Category data frame:\n")
      print(names(data_frames$Category))
    } else if (grepl("Order", file_name, ignore.case = TRUE)) {
      data_frames$Orders <- rbind(data_frames$Orders, data)
      # Output variable names
      cat("Variables in Orders data frame:\n")
      print(names(data_frames$Orders))
    } else if (grepl("Payment", file_name, ignore.case = TRUE)) {
      data_frames$Payment <- rbind(data_frames$Payment, data)
      # Output variable names
      cat("Variables in Payment data frame:\n")
      print(names(data_frames$Payment))
    } else if (grepl("Product", file_name, ignore.case = TRUE)) {
      data_frames$Product <- rbind(data_frames$Product, data)
      # Output variable names
      cat("Variables in Product data frame:\n")
      print(names(data_frames$Product))
    } else if (grepl("Promotion", file_name, ignore.case = TRUE)) {
      data_frames$Promotion <- rbind(data_frames$Promotion, data)
      # Output variable names
      cat("Variables in Promotion data frame:\n")
      print(names(data_frames$Promotion))
    } else if (grepl("Sale", file_name, ignore.case = TRUE)) {
      data_frames$Sales <- rbind(data_frames$Sales, data)
      # Output variable names
      cat("Variables in Sales data frame:\n")
      print(names(data_frames$Sales))
    } else if (grepl("Settlement", file_name, ignore.case = TRUE)) {
      data_frames$Settlement <- rbind(data_frames$Settlement, data)
      # Output variable names
      cat("Variables in Settlement data frame:\n")
      print(names(data_frames$Settlement))
    } else if (grepl("Supplier", file_name, ignore.case = TRUE)) {
      data_frames$Supplier <- rbind(data_frames$Supplier, data)
      # Output variable names
      cat("Variables in Supplier data frame:\n")
      print(names(data_frames$Supplier))
    }
```

```r
  }

  # Return the list of data frames
  return(data_frames)
}

# Directory containing CSV files
directory <- "Data_upload"



# Read CSV files from the directory and categorize them into data frames
data_frames <- read_and_categorize_csv(directory)
```

```
Variables in Category data frame:
[1] "category_id"   "category_name"
Variables in Customer data frame:
[1] "customer_id"         "customer_first_name" "customer_last_name"
[4] "customer_email"      "customer_postcode"   "registration_date"
[7] "customer_phone"      "customer_address"    "customer_city"
Variables in Orders data frame:
 [1] "order_id"            "customer_id"         "product_id"
 [4] "order_date"          "order_quantity"      "order_status"
 [7] "order_approval_date" "order_delivery_date" "rating_date"
[10] "rating_score"        "rating_comment"
Variables in Payment data frame:
[1] "payment_id"     "order_id"       "Payment_date"   "Payment_method"
Variables in Product data frame:
[1] "product_id"    "product_name"  "category_id"   "supplier_id"
[5] "price"         "discount_rate"
Variables in Promotion data frame:
[1] "promotion_id"         "supplier_id"          "promotion_name"
[4] "promotion_fees"       "promotion_start_date" "promotion_end_date"
Variables in Sales data frame:
[1] "sale_id"     "supplier_id" "product_id"  "sale_date"
Variables in Settlement data frame:
[1] "settlement_id"   "sale_id"         "settlement_date" "settlement_type"
Variables in Supplier data frame:
 [1] "supplier_id"       "seller_first_name" "seller_last_name"
 [4] "seller_email"      "seller_postcode"   "seller_address"
 [7] "seller_city"       "seller_phone"      "registration_date"
[10] "platform_rate"     "tax_rate"
```

```r
# Access each data frame by its name
Category <- data_frames$Category
Customer <- data_frames$Customer
Orders <- data_frames$Orders
Payment <- data_frames$Payment
Product <- data_frames$Product
Promotion <- data_frames$Promotion
Sales <- data_frames$Sales
Settlement <- data_frames$Settlement
Supplier <- data_frames$Supplier
```

### Validation Stage 1

After importing the data from CSV using R "validation.R", stage 1 validation is done before the data ingestion to ensure following logics are in place before database is created:

- Primary key is not null
- Primary key is unique
- Foreign key is not null
- Character length is within limit
- Email format is correct
- All data types are correct (integer, float, date, numerical, etc.)
- All values are within limit (tax rate, price, etc.)
- Dates are in chronological order (delivery is after order date, etc.)
- All constraints mentioned in Data related assumption part 1.2 are implying

If a table has duplicated primary key, the duplicated value will be removed. If other errors occur, the code will either fix the issue or show an error message notifying where the error exists. If all validation are successful, then the data will be appended into the database.

```r
library(RSQLite)
library(readr)
library(lubridate)

my_connection <- RSQLite::dbConnect(RSQLite::SQLite(),"mydatabase.db")

#define function to check email
isValidEmail <- function(x) {
  grepl("\\<[A-Z0-9._%+-]+@[A-Z0-9.-]+\\.[A-Z]{2,}\\>", as.character(x), ignore.case=TRUE)
}

#define variable to check each table validity
valid_cat = 1
valid_cus = 1
valid_ord = 1
valid_pay = 1
valid_prd = 1
valid_prm = 1
valid_sal = 1
valid_set = 1
valid_sup = 1

#CATEGORY
for (i in 1:nrow(Category))
  #check primary key exists
  if (is.na(Category$category_id[i])) {
    print(paste("Table: Category - Error: category_id is NULL on row",i))
    valid_cat = 0
  }

#check character length
if (nchar(Category$category_id[i]) > 6) {
  print(paste("Table: Category - Error: category_id more than 6 characters on row",i))
  valid_cat = 0
}

#check primary key is unique
Category <- Category[!duplicated(Category$category_id) & !duplicated(Category$category_id, f

if(length(unique(Category$category_id)) != nrow(Category)) {
  print(paste("Table: Category - Error: duplicated category_id"))
  valid_cat = 0
}

if (valid_cat == 1) {
  print("Table: Category - Status: OK")
} else {print("Table: Category - Status: ERROR")}
```

```
[1] "Table: Category - Status: OK"
```

```r
#CUSTOMER
#fix data type
Customer$customer_phone <- as.numeric(Customer$customer_phone)
Customer$registration_date <- mdy(Customer$registration_date)

#start loop
for (i in 1:nrow(Customer))
{
  #check primary key exists
  if (is.na(Customer$customer_id[i])) {
    print(paste("Table: Customer - Error: customer_id is NULL on row",i))
    valid_cus = 0
  }

  #check character length
  if (nchar(Customer$customer_id[i]) > 6) {
    print(paste("Table: Customer - Error: customer_id more than 6 characters on row",i))
    valid_cus = 0
  }

  #check email format
  if (isValidEmail(Customer$customer_email[i]) == FALSE) {
    print(paste("Table: Customer - Error: customer_email format unsupported on row",i))
    valid_cus = 0
  }


  #characters limit
  Customer$customer_first_name[i] <- substring(Customer$customer_first_name[i],1,50)
  Customer$customer_last_name[i] <- substring(Customer$customer_last_name[i],1,50)
  Customer$customer_email[i] <- substring(Customer$customer_email[i],1,200)
}

#check primary key is unique
if(length(unique(Customer$customer_id)) != nrow(Customer)) {
  print(paste("Table: Customer - Error: duplicated customer_id"))
  valid_cus = 0
}

#check email is unique
if(length(unique(Customer$customer_email)) != nrow(Customer)) {
  print(paste("Table: Customer - Error: duplicated customer_email"))
  valid_cus = 0
}

#check phone number is unique
if(length(unique(Customer$customer_phone)) != nrow(Customer)) {
  print(paste("Table: Customer - Error: duplicated customer_phone"))
  valid_cus = 0
}

#final check result
if (valid_cus == 1) {
  print("Table: Customer - Status: OK")
} else {print("Table: Customer - Status: ERROR")}
```

```
[1] "Table: Customer - Status: OK"
```

```
#ORDER
#fix data type
Orders$order_quantity <- as.numeric(Orders$order_quantity)
Orders$rating_score <- as.numeric(Orders$rating_score)
Orders$order_date <- mdy(Orders$order_date)
Orders$order_approval_date <- mdy(Orders$order_approval_date)
Orders$order_delivery_date <- mdy(Orders$order_delivery_date)
Orders$rating_date <- mdy(Orders$rating_date)

#start loop
for (i in 1:nrow(Orders))
{
  #check primary key exists
  if (is.na(Orders$order_id[i])) {
    print(paste("Table: Orders - Error: order_id is NULL on row",i))
    valid_ord = 0
  }

  #check foreign key exists
  if (is.na(Orders$product_id[i])) {
    print(paste("Table: Orders - Error: product_id is NULL on row",i))
    valid_ord = 0
  }

  if (is.na(Orders$customer_id[i])) {
    print(paste("Table: Orders - Error: customer_id is NULL on row",i))
    valid_ord = 0
  }

  #check character length
  if (nchar(Orders$order_id[i]) > 6) {
    print(paste("Table: Orders - Error: order_id more than 6 characters on row",i))
    valid_ord = 0
  }

  if (nchar(Orders$product_id[i]) > 6) {
    print(paste("Table: Orders - Error: product_id more than 6 characters on row",i))
    valid_ord = 0
  }

  if (nchar(Orders$customer_id[i]) > 6) {
    print(paste("Table: Orders - Error: customer_id more than 6 characters on row",i))
    valid_ord = 0
  }

  #check if order quantity is above 20
  if (Orders$order_quantity[i] < 0 || Orders$order_quantity[i] > 20) {
    print(paste("Table: Orders - Error: order_quantity is more than 20 on row",i))
    valid_ord = 0
  }

  #check if order status is valid
  if (!(Orders$order_status[i]%in%c("processing","delivered","pending","shipped"))) {
    print(paste("Table: Orders - Error: order_status is not valid on row",i))
    valid_ord = 0
  }

  #check if status is pending then approval date is null
  if (Orders$order_status[i] == "pending" && is.na(Orders$order_approval_date[i]) == FALSE)
    print(paste("Table: Orders - Error: order_approval_date is not null when status is pendi
    valid_ord = 0
  }

  #check if approval date is after order date
  if (Orders$order_approval_date[i] < Orders$order_date[i] && is.na(Orders$order_approval_da
    print(paste("Table: Orders - Error: order_approval_date is before order_date on row",i))
    valid_ord = 0
  }

  #check if status is delivered then delivery date is not null
  if (Orders$order_status[i] == "delivered" && is.na(Orders$order_delivery_date[i]) == TRUE)
    print(paste("Table: Orders - Error: order_delivery_date is null when status is delivered
    valid_ord = 0
    is.na(Orders$order_delivery_date[2])
  }

  #check if delivery date is after approval date
  if (Orders$order_delivery_date[i] < Orders$order_approval_date[i] && is.na(Orders$order_de
    print(paste("Table: Orders - Error: order_delivery_date is before order_approval_date on
    valid_ord = 0
  }

  #check if rating date is not null then status is delivered
  if (Orders$order_status[i] != "delivered" && is.na(Orders$rating_date[i]) == FALSE) {
    print(paste("Table: Orders - Error: rating_date is not null when status is not delivered
    valid_ord = 0
  }

  #check if rating score is between 1-5
  if ((is.na(Orders$rating_score[i]) == FALSE) && (Orders$rating_score[i] < 0 || Orders$ratin
    print(paste("Table: Orders - Error: rating_score is not between 1-5 on row",i))
    valid_ord = 0
  }

  #check if rating score is not null then rating date is not null
  if (is.na(Orders$rating_score[i]) == FALSE && is.na(Orders$rating_date[i]) == TRUE) {
    print(paste("Table: Orders - Error: rating_date is null when rating_score has value on r
    valid_ord = 0
  }

  #check if rating comment is not null then rating score is not null
  if (nchar(Orders$rating_comment[i]) >= 1  && is.na(Orders$rating_score[i]) == TRUE) {
    print(paste("Table: Orders - Error: rating_score is null when rating_comment has value o
    valid_ord = 0
  }
}

#check primary key is unique
Orders <- Orders[!duplicated(Orders$order_id) & !duplicated(Orders$order_id, fromLast = TRUE
```

```r
if(length(unique(Orders$order_id)) != nrow(Orders)) {
  print(paste("Table: Orders - Error: duplicated order_id"))
  valid_ord = 0
}


#final check result
if (valid_ord == 1) {
  print("Table: Orders - Status: OK")
} else {print("Table: Orders - Status: ERROR")}
```

```
[1] "Table: Orders - Status: OK"
```

```r
#PAYMENT
#fix data type
Payment$Payment_date <- mdy(Payment$Payment_date)

#start loop
for (i in 1:nrow(Payment))
{
  #check primary key exists
  if (is.na(Payment$payment_id[i])) {
    print(paste("Table: Payment - Error: payment_id is NULL on row",i))
    valid_pay = 0
  }

  #check foreign key exists
  if (is.na(Payment$order_id[i])) {
    print(paste("Table: Payment - Error: order_id is NULL on row",i))
    valid_pay = 0
  }

  #check character length
  if (nchar(Payment$payment_id[i]) > 6) {
    print(paste("Table: Payment - Error: payment_id more than 6 characters on row",i))
    valid_pay = 0
  }

  if (nchar(Payment$order_id[i]) > 6) {
    print(paste("Table: Payment - Error: order_id more than 6 characters on row",i))
    valid_pay = 0
  }

  #check if payment method is valid
  if (!(Payment$Payment_method[i]%in%c("venmo","credit_card","debit_card","paypal","apple_pa
    print(paste("Table: Payment - Error: payment_method is not valid on row",i))
    valid_pay = 0
  }
}

#check primary key is unique
Payment <- Payment[!duplicated(Payment$payment_id) & !duplicated(Payment$payment_id, fromLas

if(length(unique(Payment$payment_id)) != nrow(Payment)) {
  print(paste("Table: Payment - Error: duplicated payment_id"))
  valid_pay = 0
}

#final check result
if (valid_pay == 1) {
  print("Table: Payment - Status: OK")
} else {print("Table: Payment - Status: ERROR")}
```

```
[1] "Table: Payment - Status: OK"
```

```
#PRODUCT
#fix data type
Product$price <- as.numeric(Product$price)
Product$discount_rate <- as.numeric(Product$discount_rate)

#start loop
for (i in 1:nrow(Product))
{
  #check primary key exists
  if (is.na(Product$product_id[i])) {
    print(paste("Table: Product - Error: product_id is NULL on row",i))
    valid_prd = 0
  }

  #check foreign key exists
  if (is.na(Product$category_id[i])) {
    print(paste("Table: Product - Error: category_id is NULL on row",i))
    valid_prd = 0
  }

  if (is.na(Product$supplier_id[i])) {
    print(paste("Table: Product - Error: supplier_id is NULL on row",i))
    valid_prd = 0
  }

  #check character length
  if (nchar(Product$product_id[i]) > 6) {
    print(paste("Table: Product - Error: product_id more than 6 characters on row",i))
    valid_prd = 0
  }

  if (nchar(Product$category_id[i]) > 6) {
    print(paste("Table: Product - Error: category_id more than 6 characters on row",i))
    valid_prd = 0
  }

  if (nchar(Product$supplier_id [i]) > 6) {
    print(paste("Table: Product - Error: suplier_id more than 6 characters on row",i))
    valid_prd = 0
  }

  #check if discount is more than 20%
  if (Product$discount_rate[i] < 0 | Product$discount_rate[i] > 20) {
    print(paste("Table: Product - Error: discount rate is not between 0-20 on row",i))
    valid_prd = 0
  }

  #check if price is between 5-100
  if (Product$price[i] < 5 || Product$price[i] > 100) {
    print(paste("Table: Product - Error: price is not between 5-100 on row",i))
    valid_prd = 0
  }
}

#check primary key is unique
Product <- Product[!duplicated(Product$product_id) & !duplicated(Product$product_id, fromLas

if(length(unique(Product$product_id)) != nrow(Product)) {
  print(paste("Table: Product - Error: duplicated product_id"))
  valid_prd = 0
}

#final check result
if (valid_prd == 1) {
  print("Table: Product - Status: OK")
} else {print("Table: Product - Status: ERROR")}
```

```
[1] "Table: Product - Status: OK"
```

```
#PROMOTION

#fix data type
Promotion$promotion_fees <- as.numeric(Promotion$promotion_fees)
Promotion$promotion_start_date <- mdy(Promotion$promotion_start_date)
Promotion$promotion_end_date <- mdy(Promotion$promotion_end_date)

#start loop
for (i in 1:nrow(Promotion))
{
  #check primary key exists
  if (is.na(Promotion$promotion_id[i])) {
    print(paste("Table: Promotion - Error: promotion_id is NULL on row",i))
    valid_prm = 0
  }

  #check foreign key exists
  if (is.na(Promotion$supplier_id[i])) {
    print(paste("Table: Promotion - Error: supplier_id is NULL on row",i))
    valid_prm = 0
  }

  #check character length
  if (nchar(Promotion$promotion_id[i]) > 6) {
    print(paste("Table: Promotion - Error: promotion_id more than 6 characters on row",i))
    valid_prm = 0
  }

  if (nchar(Promotion$supplier_id[i]) > 6) {
    print(paste("Table: Promotion - Error: supplier_id more than 6 characters on row",i))
    valid_prm = 0
  }

  #check end date is after start date
  if (Promotion$promotion_end_date[i] < Promotion$promotion_start_date[i]) {
    print(paste("Table: Promotion - Error: promotion end date is before start date on row",i
    valid_prm = 0
  }

  #check if length of promotion is within 30 days
  if (as.numeric(Promotion$promotion_end_date[i] - Promotion$promotion_start_date[i]) > 30)
    print(paste("Table: Promotion - Error: promotion length is more than 30 days on row",i))
    valid_prm = 0
  }

  #check if promotion fee is more than 1000
  if (Promotion$promotion_fees[i] > 1000) {
    print(paste("Table: Promotion - Error: promotion fee is more than 1000 on row",i))
    valid_prm = 0
  }
}

#check primary key is unique
Promotion <- Promotion[!duplicated(Promotion$promotion_id) & !duplicated(Promotion$promotion

if(length(unique(Promotion$promotion_id)) != nrow(Promotion)) {
  print(paste("Table: Promotion - Error: duplicated promotion_id"))
  valid_prm = 0
}

#final check result
if (valid_prm == 1) {
  print("Table: Promotion - Status: OK")
} else {print("Table: Promotion - Status: ERROR")}
```

```
[1] "Table: Promotion - Status: OK"
```

```
#SALES
for (i in 1:nrow(Sales))
{
  #check primary key exists
  if (is.na(Sales$sale_id[i])) {
    print(paste("Table: Sales - Error: sale_id is NULL on row",i))
    valid_sal = 0
  }

  #check foreign key exists
  if (is.na(Sales$supplier_id[i])) {
    print(paste("Table: Sales - Error: supplier_id is NULL on row",i))
    valid_sal = 0
  }

  if (is.na(Sales$product_id[i])) {
    print(paste("Table: Sales - Error: product_id is NULL on row",i))
    valid_sal = 0
  }

  #check character length
  if (nchar(Sales$sale_id[i]) > 6) {
    print(paste("Table: Sales - Error: sale_id more than 6 characters on row",i))
    valid_sal = 0
  }

  if (nchar(Sales$supplier_id[i]) > 6) {
    print(paste("Table: Sales - Error: supplier_id more than 6 characters on row",i))
    valid_sal = 0
  }

  if (nchar(Sales$product_id[i]) > 6) {
    print(paste("Table: Sales - Error: product_id more than 6 characters on row",i))
    valid_sal = 0
  }
}

#check primary key is unique
Sales <- Sales[!duplicated(Sales$sale_id) & !duplicated(Sales$sale_id, fromLast = TRUE), ]

if(length(unique(Sales$sale_id)) != nrow(Sales)) {
  print(paste("Table: Sales - Error: duplicated sale_id"))
  valid_sal = 0
}

#correct duplicate values
Sales$sale_date <- mdy(Sales$sale_date)
```

```
#final check result
if (valid_sal == 1) {
  print("Table: Sales - Status: OK")
} else {
  print("Table: Sales - Status: ERROR")
}
```

```
[1] "Table: Sales - Status: OK"
```

```
#SETTLEMENT
#fix data type
Settlement$settlement_date <- mdy(Settlement$settlement_date)

for (i in 1:nrow(Settlement))
{
  #check primary key exists
  if (is.na(Settlement$settlement_id[i])) {
    print(paste("Table: Settlement - Error: settlement_id is NULL on row",i))
    valid_set = 0
  }

  #check foreign key exists
  if (is.na(Settlement$sale_id[i])) {
    print(paste("Table: Settlement - Error: sale_id is NULL on row",i))
    valid_set = 0
  }

  #check character length
  if (nchar(Settlement$settlement_id[i]) > 6) {
    print(paste("Table: Settlement - Error: settlement_id more than 6 characters on row",i))
    valid_set = 0
  }

  if (nchar(Settlement$sale_id[i]) > 6) {
    print(paste("Table: Settlement - Error: sale_id more than 6 characters on row",i))
    valid_set = 0
  }

  #check if settlement method is valid
  if (!(Settlement$settlement_type[i]%in%c("bank transfer","card","cheque","bank draft","cas
    print(paste("Table: Settlement - Error: settlement_type is not valid on row",i))
    valid_set = 0
```

```r
  }
}

#check primary key is unique
Settlement <- Settlement[!duplicated(Settlement$settlement_id) & !duplicated(Settlement$settl

if(length(unique(Settlement$settlement_id)) != nrow(Settlement)) {
  print(paste("Table: Settlement - Error: duplicated settlement_id"))
  valid_set = 0
}

#final check result
if (valid_set == 1) {
  print("Table: Settlement - Status: OK")
} else {print("Table: Settlement - Status: ERROR")}
```

```
[1] "Table: Settlement - Status: OK"
```

```r
#SUPPLIER
#start loop
for (i in 1:nrow(Supplier))
{
  #check primary key exists
  if (is.na(Supplier$supplier_id[i])) {
    print(paste("Table: Supplier - Error: supplier_id is NULL on row",i))
    valid_sup = 0
  }

  #check character length
  if (nchar(Supplier$supplier_id[i]) > 6) {
    print(paste("Table: Supplier - Error: supplier_id more than 6 characters on row",i))
    valid_sup = 0
  }

  #check email format
  if (isValidEmail(Supplier$seller_email[i]) == FALSE) {
    print(paste("Table: Supplier - Error: seller_email format unsupported on row",i))
    valid_sup = 0
  }

  #characters limit
  Supplier$seller_first_name[i] <- substring(Supplier$seller_first_name[i],1,50)
  Supplier$seller_last_name[i] <- substring(Supplier$seller_last_name[i],1,50)
  Supplier$seller_email[i] <- substring(Supplier$seller_email[i],1,200)
  Supplier$seller_address[i] <- substring(Supplier$seller_address[i],1,200)
  Supplier$seller_city[i] <- substring(Supplier$seller_city[i],1,50)

  #rate convertion to numeric
  if (is.character(Supplier$platform_rate) == TRUE) {
    Supplier$platform_rate[i] <- parse_number(Supplier$platform_rate[i])
  }

  if (is.character(Supplier$tax_rate) == TRUE) {
  Supplier$tax_rate[i] <- parse_number(Supplier$tax_rate[i])
  }
}

#fix data type
Supplier$seller_phone <- as.numeric(Supplier$seller_phone)
Supplier$registration_date <- mdy(Supplier$registration_date)
Supplier$platform_rate <- as.numeric(Supplier$platform_rate)
Supplier$tax_rate <- as.numeric(Supplier$tax_rate)

#check primary key is unique
Supplier <- Supplier[!duplicated(Supplier$supplier_id) & !duplicated(Supplier$supplier_id, f

if(length(unique(Supplier$supplier_id)) != nrow(Supplier)) {
  print(paste("Table: Supplier - Error: duplicated supplier_id"))
  valid_sup = 0
}

#check email is unique
if(length(unique(Supplier$seller_email)) != nrow(Supplier)) {
  print(paste("Table: Supplier - Error: duplicated seller_email"))
  valid_sup = 0
}

#check phone number is unique
if(length(unique(Supplier$seller_phone)) != nrow(Supplier)) {
  print(paste("Table: Supplier - Error: duplicated seller_phone"))
  valid_sup = 0
}
```

```r
#check platform rate and tax rate value
for (i in 1:nrow(Supplier))
{
  if (Supplier$platform_rate[i] < 0 || Supplier$platform_rate[i] > 25) {
    print(paste("Table: Supplier - Error: platform_rate is not between 0-25 on row",i))
    valid_sup = 0
  }

  if (Supplier$tax_rate[i] != 10) {
    print(paste("Table: Supplier - Error: tax_rate is not equal to 10 on row",i))
    valid_sup = 0
  }
}

#final check result
if (valid_sup == 1) {
  print("Table: Supplier - Status: OK")
} else {print("Table: Supplier - Status: ERROR")}
```

```
[1] "Table: Supplier - Status: OK"
```

```r
#INGESTION IF VALIDATION PASSED
if (valid_cat == 1) {
  RSQLite::dbWriteTable(my_connection, "Category", Category, append = TRUE)}

if (valid_cus == 1) {
  RSQLite::dbWriteTable(my_connection, "Customer", Customer, append = TRUE)}

if (valid_ord == 1) {
  RSQLite::dbWriteTable(my_connection, "Orders", Orders, append = TRUE)}

if (valid_pay == 1) {
  RSQLite::dbWriteTable(my_connection, "Payment", Payment, append = TRUE)}
```

```
Warning: Column names will be matched ignoring character case
```

```r
if (valid_prd == 1) {
  RSQLite::dbWriteTable(my_connection, "Product", Product, append = TRUE)}

if (valid_prm == 1) {
  RSQLite::dbWriteTable(my_connection, "Promotion", Promotion, append = TRUE)}

if (valid_sal == 1) {
  RSQLite::dbWriteTable(my_connection, "Sales", Sales, append = TRUE)}

if (valid_set == 1) {
  RSQLite::dbWriteTable(my_connection, "Settlement", Settlement, append = TRUE)}

if (valid_sup == 1) {
  RSQLite::dbWriteTable(my_connection, "Supplier", Supplier, append = TRUE)}
```

### Validation Stage 2

After the data is ingested, stage 2 validation is run using R "validation_stage_2.R" to ensure three things:

- There are no duplicated primary keys.
- Check referential integrity.
- Check dates across tables are in chronological orders (e.g. customer's first order is after registration date). Invalid records will be removed from the tables.

```
library(RSQLite)
library(readr)

my_connection <- RSQLite::dbConnect(RSQLite::SQLite(),"mydatabase.db")
#check primary key duplication
dup_cat <- "
  WITH
  check_dup AS (
  SELECT category_id, ROW_NUMBER() OVER (PARTITION BY category_id) as id_rnk
  from Category
  )

  DELETE FROM Category
  WHERE category_id in
    (SELECT category_id FROM check_dup WHERE id_rnk > 1)
"

dup_cus <- "
  WITH
  check_dup AS (
  SELECT customer_id, ROW_NUMBER() OVER (PARTITION BY customer_id) as id_rnk
  from Customer
  )

  DELETE FROM Customer
  WHERE customer_id in
    (SELECT customer_id FROM check_dup WHERE id_rnk > 1)
"

dup_ord <- "
  WITH
  check_dup AS (
  SELECT order_id, ROW_NUMBER() OVER (PARTITION BY order_id) as id_rnk
  from Orders
  )

  DELETE FROM Orders
  WHERE order_id in
    (SELECT order_id FROM check_dup WHERE id_rnk > 1)
"

dup_pay <- "
  WITH
  check_dup AS (
  SELECT payment_id, ROW_NUMBER() OVER (PARTITION BY payment_id) as id_rnk
  from Payment
  )

  DELETE FROM Payment
  WHERE payment_id in
    (SELECT payment_id FROM check_dup WHERE id_rnk > 1)
"

dup_prd <- "
  WITH
  check_dup AS (
  SELECT product_id, ROW_NUMBER() OVER (PARTITION BY product_id) as id_rnk
  from Product
  )

  DELETE FROM Product
  WHERE product_id in
    (SELECT product_id FROM check_dup WHERE id_rnk > 1)
"

dup_prm <- "
  WITH
  check_dup AS (
  SELECT promotion_id, ROW_NUMBER() OVER (PARTITION BY promotion_id) as id_rnk
  from Promotion
  )

  DELETE FROM Promotion
  WHERE promotion_id in
    (SELECT promotion_id FROM check_dup WHERE id_rnk > 1)
"

dup_sal <- "
  WITH
  check_dup AS (
  SELECT sale_id, ROW_NUMBER() OVER (PARTITION BY sale_id) as id_rnk
  from Sales
  )

  DELETE FROM Sales
  WHERE sale_id in
    (SELECT sale_id FROM check_dup WHERE id_rnk > 1)
"

dup_set <- "
  WITH
  check_dup AS (
  SELECT settlement_id, ROW_NUMBER() OVER (PARTITION BY settlement_id) as id_rnk
  from Settlement
  )

  DELETE FROM Settlement
  WHERE settlement_id in
    (SELECT settlement_id FROM check_dup WHERE id_rnk > 1)
"

dup_sup <- "
  WITH
  check_dup AS (
  SELECT supplier_id, ROW_NUMBER() OVER (PARTITION BY supplier_id) as id_rnk
  from Supplier
  )

  DELETE FROM Supplier
  WHERE supplier_id in
    (SELECT supplier_id FROM check_dup WHERE id_rnk > 1)
"

dbExecute(my_connection, dup_cat)
```

```
[1] 0
```

```
dbExecute(my_connection, dup_cus)
```

```
[1] 0
```

```
dbExecute(my_connection, dup_ord)
```

```
[1] 0
```

```
dbExecute(my_connection, dup_pay)
```

```
[1] 0
```

```
dbExecute(my_connection, dup_prd)
```

```
[1] 0
```

```
dbExecute(my_connection, dup_prm)
```

```
[1] 0
```

```
dbExecute(my_connection, dup_sal)
```

```
[1] 0
```

```r
dbExecute(my_connection, dup_set)
```

```
[1] 0
```

```r
dbExecute(my_connection, dup_sup)
```

```
[1] 0
```

```r
#check referential integrity
ri_ord <- "
  WITH
  check_ref AS (
  SELECT
    o.order_id,
    c.customer_id,
    p.product_id
  FROM Orders o
  LEFT JOIN Customer c on o.customer_id = c.customer_id
  LEFT JOIN Product p on o.product_id = p.product_id
  )

  DELETE FROM Orders
  WHERE order_id in
    (SELECT order_id FROM check_ref WHERE customer_id IS NULL OR product_id IS NULL)
"

ri_pay <- "
  WITH
  check_ref AS (
  SELECT
    p.payment_id,
    o.order_id
  FROM Payment p
  LEFT JOIN Orders o on p.order_id = o.order_id
  )

  DELETE FROM Payment
  WHERE payment_id in
    (SELECT payment_id FROM check_ref WHERE order_id IS NULL)
"
```

```
ri_prd <- "
  WITH
  check_ref AS (
  SELECT
    p.product_id,
    c.category_id,
    s.supplier_id
  FROM Product p
  LEFT JOIN Category c on p.category_id = c.category_id
  LEFT JOIN Supplier s on p.supplier_id = s.supplier_id
  )

  DELETE FROM Product
  WHERE product_id in
    (SELECT product_id FROM check_ref WHERE category_id IS NULL OR supplier_id IS NULL)
"

ri_sal <- "
  WITH
  check_ref AS (
  SELECT
    s.sale_id,
    u.supplier_id,
    p.product_id
  FROM Sales s
  LEFT JOIN Supplier u on s.supplier_id = u.supplier_id
  LEFT JOIN Product p on s.product_id = p.product_id
  )

  DELETE FROM Sales
  WHERE sale_id IN
    (SELECT sale_id FROM check_ref WHERE supplier_id IS NULL OR product_id IS NULL)
"

ri_set <- "
  WITH
  check_ref AS (
  SELECT
    s.settlement_id,
    l.sale_id
  FROM Settlement s
  LEFT JOIN Sales l on s.sale_id = l.sale_id
  )

  DELETE FROM Settlement
  WHERE settlement_id in
    (SELECT settlement_id FROM check_ref WHERE sale_id IS NULL)
"

ri_prm <- "
  WITH
  check_ref AS (
  SELECT
    p.promotion_id,
    s.supplier_id
  FROM Promotion p
  LEFT JOIN Supplier s on p.supplier_id = s.supplier_id
  )

  DELETE FROM Promotion
  WHERE promotion_id in
    (SELECT promotion_id FROM check_ref WHERE supplier_id IS NULL)
"

dbExecute(my_connection, ri_ord)
```

```
[1] 0
```

```
dbExecute(my_connection, ri_pay)
```

```
[1] 0
```

```
dbExecute(my_connection, ri_prd)
```

```
[1] 0
```

```
dbExecute(my_connection, ri_sal)
```

```
[1] 0
```

```r
dbExecute(my_connection, ri_set)
```

```
[1] 2
```

```r
dbExecute(my_connection, ri_prm)
```

```
[1] 0
```

```r
#check date constraint between 2 tables

#check and fix customer registration date is before first order date
dc_cus <- "
  WITH
  check_order AS (
  SELECT
    c.customer_id,
    MIN(o.order_date) as first_order_date
  FROM Customer c
  LEFT JOIN Orders o on c.customer_id = o.customer_id
  WHERE o.order_date < c.registration_date
  GROUP BY 1
  ORDER BY 1
  )

  UPDATE Customer
  SET registration_date = (SELECT first_order_date FROM check_order)
  WHERE customer_id = (SELECT customer_id FROM check_order)
"

#check and fix supplier registration date is before first order date
dc_sup <- "
  WITH
  check_order AS (
  SELECT
    s.supplier_id,
    min(o.order_date) as first_order_date
  FROM Orders o
  LEFT JOIN Product p on o.product_id = p.product_id
  LEFT JOIN Supplier s on p.supplier_id = s.supplier_id
  WHERE o.order_date < s.registration_date
  GROUP BY 1
  ORDER BY 1
  )

  UPDATE Supplier
  SET registration_date = (SELECT first_order_date FROM check_order)
  WHERE supplier_id = (SELECT supplier_id FROM check_order)
"

#check sale date is equal to order date
dc_sal <- "
  WITH
  check_sal AS (
  SELECT DISTINCT
    s.sale_id,
    o.order_id
  FROM Sales s
  LEFT JOIN Orders o on s.product_id = o.product_id AND s.sale_date = o.order_date
  WHERE o.order_id IS NULL
  )

  DELETE FROM Sales
  WHERE sale_id in
    (SELECT sale_id FROM check_sal) --Delete invalid sales that cannot be mapped to the orde
"

#check payment date is after order date
dc_pay <- "
  WITH
  check_pay AS (
  SELECT
    p.payment_id,
    MAX(o.order_date) as last_order_date
  FROM Payment p
  LEFT JOIN Orders o on p.order_id = o.order_id
  WHERE p.payment_date < o.order_date
  GROUP BY 1
  ORDER BY 1
  )

  UPDATE Payment
  SET payment_date = (SELECT last_order_date FROM check_pay)
  WHERE payment_id = (SELECT payment_id FROM check_pay)
"

#check settlement date is after sale date
dc_set <- "
  WITH
  check_set AS (
  SELECT
    p.settlement_id,
    MAX(s.sale_date) as last_sale_date
  FROM Settlement p
  LEFT JOIN Sales s on p.sale_id = s.sale_id
  WHERE p.settlement_date < s.sale_date
  GROUP BY 1
  ORDER BY 1
  )

  UPDATE Settlement
  SET settlement_date = (SELECT last_sale_date FROM check_set)
  WHERE settlement_id = (SELECT settlement_id FROM check_set)
"

#check supplier registration date is before first promotion date
dc_prm <- "
  WITH
  check_prm AS
  (
  SELECT
    s.supplier_id,
    min(p.promotion_start_date) AS first_promo_date
  FROM Supplier s
  LEFT JOIN Promotion p on s.supplier_id = p.supplier_id
  WHERE p.promotion_start_date < s.registration_date
  GROUP BY 1
  ORDER BY 1
  )

  UPDATE Supplier
  SET registration_date = (SELECT first_promo_date FROM check_prm)
  WHERE supplier_id = (SELECT supplier_id FROM check_prm)
"
```

```r
dbExecute(my_connection, dc_cus)
```

```
[1] 0
```

```r
dbExecute(my_connection, dc_sup)
```

```
[1] 0
```

```r
dbExecute(my_connection, dc_pay)
```

```
[1] 1
```

```r
dbExecute(my_connection, dc_sal)
```

```
[1] 0
```

```r
dbExecute(my_connection, dc_set)
```

```
[1] 0
```

```r
dbExecute(my_connection, dc_prm)
```

```
[1] 0
```

### Automated Report

After the data is validated for the second time, an automated report is generated using R "daily_report.R". This report includes the total order, finance, top sellers and top products.

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

my_connection <- RSQLite::dbConnect(RSQLite::SQLite(),"mydatabase.db")

daily_order <- "
  SELECT
```

```
    date(order_date * 3600 * 24, 'unixepoch') AS day_,
    COUNT(DISTINCT order_id) as total_order,
    COUNT(DISTINCT customer_id) as total_customer,
    COUNT(DISTINCT o.product_id) as total_product,
    SUM(order_quantity) as total_quantity
  FROM Orders o
  LEFT JOIN Product p on o.product_id = p.product_id
  GROUP BY 1
  ORDER BY 1 DESC
  LIMIT 14
"

daily_finance <- "
  SELECT
    date(order_date * 3600 * 24, 'unixepoch') AS day_,
    SUM(order_quantity * p.price * (1-p.discount_rate/100)) as total_gmv,
    SUM(order_quantity * p.price * (1-p.discount_rate/100-s.tax_rate/100)) as total_nmv,
    SUM(order_quantity * p.price * (s.platform_rate/100)) as total_platform_fee,
    SUM(order_quantity * p.price * (1-p.discount_rate/100-s.tax_rate/100-s.platform_rate/100)
  FROM Orders o
  LEFT JOIN Product p on o.product_id = p.product_id
  LEFT JOIN Supplier s on p.supplier_id = s.supplier_id
  GROUP BY 1
  ORDER BY 1 DESC
  LIMIT 14
"

monthly_order <- "
  SELECT
    STRFTIME('%m-%Y', date(order_date * 3600 * 24, 'unixepoch')) AS month_,
    COUNT(DISTINCT order_id) as total_order,
    COUNT(DISTINCT customer_id) as total_customer,
    COUNT(DISTINCT o.product_id) as total_product,
    SUM(order_quantity) as total_quantity
  FROM Orders o
  LEFT JOIN Product p on o.product_id = p.product_id
  GROUP BY 1
  ORDER BY 1 DESC
"

monthly_finance <- "
  SELECT
```

```
    STRFTIME('%m-%Y', date(order_date * 3600 * 24, 'unixepoch')) AS month_,
    SUM(order_quantity * p.price * (1-p.discount_rate/100)) as total_gmv,
    SUM(order_quantity * p.price * (1-p.discount_rate/100-s.tax_rate/100)) as total_nmv,
    SUM(order_quantity * p.price * (s.platform_rate/100)) as total_platform_fee,
    SUM(order_quantity * p.price * (1-p.discount_rate/100-s.tax_rate/100-s.platform_rate/100
  FROM Orders o
  LEFT JOIN Product p on o.product_id = p.product_id
  LEFT JOIN Supplier s on p.supplier_id = s.supplier_id
GROUP BY 1
ORDER BY 1 DESC
"

top_product <- "
  SELECT
    p.product_name,
    COUNT(DISTINCT order_id) as total_order,
    SUM(order_quantity) as total_quantity,
    SUM(order_quantity * p.price * (1-p.discount_rate/100)) as total_gmv
  FROM Orders o
  LEFT JOIN Product p on o.product_id = p.product_id
GROUP BY 1
ORDER BY 2 DESC
LIMIT 10
"

top_product_gmv <- "
  SELECT
    p.product_name,
    COUNT(DISTINCT order_id) as total_order,
    SUM(order_quantity) as total_quantity,
    SUM(order_quantity * p.price * (1-p.discount_rate/100)) as total_gmv
  FROM Orders o
  LEFT JOIN Product p on o.product_id = p.product_id
GROUP BY 1
ORDER BY 4 DESC
LIMIT 10
"

top_seller <- "
  SELECT
    s.supplier_id,
    COUNT(DISTINCT order_id) as total_order,
    SUM(order_quantity) as total_quantity,
    SUM(order_quantity * p.price * (1-p.discount_rate/100-s.tax_rate/100-s.platform_rate/100
  FROM Orders o
  LEFT JOIN Product p on o.product_id = p.product_id
  LEFT JOIN Supplier s on p.supplier_id = s.supplier_id
GROUP BY 1
ORDER BY 2 DESC
LIMIT 10
"

top_seller_settlement <- "
  SELECT
    s.supplier_id,
    COUNT(DISTINCT order_id) as total_order,
    SUM(order_quantity) as total_quantity,
    SUM(order_quantity * p.price * (1-p.discount_rate/100-s.tax_rate/100-s.platform_rate/100
  FROM Orders o
  LEFT JOIN Product p on o.product_id = p.product_id
  LEFT JOIN Supplier s on p.supplier_id = s.supplier_id
GROUP BY 1
ORDER BY 4 DESC
LIMIT 10
"

#show result
print("Daily Order")
```

```
[1] "Daily Order"
```

```
dbGetQuery(my_connection,daily_order)
```

```
         day_ total_order total_customer total_product total_quantity
1  2024-02-25           2              2             2             14
2  2024-02-24           3              3             3             21
3  2024-02-23           1              1             1              6
4  2024-02-22           3              3             3             20
5  2024-02-21           5              5             5             28
6  2024-02-20           4              4             4             16
7  2024-02-19           2              2             2              9
8  2024-02-18           4              4             4             25
9  2024-02-17           2              2             2             11
```

```
10 2024-02-16           3              3             3             18
11 2024-02-15           3              3             3             15
12 2024-02-14           2              2             2             15
13 2024-02-13           2              2             2             17
14 2024-02-12           1              1             1              6
```

```
print("Daily Finance")
```

```
[1] "Daily Finance"
```

```
dbGetQuery(my_connection,daily_finance)
```

```
         day_ total_gmv total_nmv total_platform_fee total_seller_settlement
1  2024-02-25  680.4592  607.5792            78.2768                529.3024
2  2024-02-24 1449.1284 1282.1864           201.0510               1081.1354
3  2024-02-23  448.7760  399.9960            48.7800                351.2160
4  2024-02-22  899.1082  804.3372           173.6277                630.7095
5  2024-02-21  936.1586  832.9206           124.1456                708.7750
6  2024-02-20  811.8824  721.7214           122.5519                599.1695
7  2024-02-19  714.5316  639.2596           127.4100                511.8496
8  2024-02-18 1065.5766  955.6246           129.4283                826.1963
9  2024-02-17  666.5265  595.3695            58.8186                536.5509
10 2024-02-16  550.9846  490.5816           107.5094                383.0722
11 2024-02-15 1381.1378 1238.0718           217.6566               1020.4152
12 2024-02-14  182.0768  163.4928            24.1128                139.3800
13 2024-02-13  555.7665  498.0055            82.1263                415.8792
14 2024-02-12  388.7028  343.5048            67.7970                275.7078
```

```
print("Monthly Order")
```

```
[1] "Monthly Order"
```

```
dbGetQuery(my_connection,monthly_order)
```

```
   month_ total_order total_customer total_product total_quantity
1  12-2023          90             41            59            443
2  11-2023          63             33            44            321
3  10-2023          73             38            50            436
4  09-2023          61             35            49            324
```

```
5   08-2023           78            36            53            449
6   07-2023           66            35            45            396
7   06-2023           68            35            48            372
8   05-2023           76            36            49            445
9   04-2023           67            33            50            413
10  03-2023           78            36            58            414
11  02-2024           59            34            47            347
12  02-2023            2             2             2              9
13  01-2024           74            36            47            381
```

```r
print("Monthly Finance")
```

```
[1] "Monthly Finance"
```

```r
dbGetQuery(my_connection,monthly_finance)
```

```
    month_ total_gmv total_nmv total_platform_fee total_seller_settlement
1   12-2023 17194.547 15326.542           2689.617               12636.926
2   11-2023 15444.970 13738.621           2174.146               11564.475
3   10-2023 19020.331 16959.973           2835.429               14124.544
4   09-2023 15371.071 13710.362           2389.945               11320.418
5   08-2023 20326.165 18112.209           3044.498               15067.711
6   07-2023 15824.265 14093.503           2428.755               11664.748
7   06-2023 16900.193 15045.607           2433.617               12611.990
8   05-2023 20098.097 17895.017           2806.286               15088.731
9   04-2023 18229.462 16245.908           2930.889               13315.018
10  03-2023 18989.717 16895.648           2692.311               14203.337
11  02-2024 14956.076 13340.098           2199.842               11140.256
12  02-2023   264.598   235.576             43.533                 192.043
13  01-2024 15313.778 13618.400           2287.950               11330.450
```

```r
print("Top Product by Number of Order")
```

```
[1] "Top Product by Number of Order"
```

```r
dbGetQuery(my_connection,top_product)
```

```
      product_name total_order total_quantity total_gmv
1       Foundation          61            314 13263.437
```

```
2       Toothbrush          60            316 16628.869
3      Styling Gel          46            251 15893.005
4          Mascara          45            254  5613.813
5           Wallet          43            233  9175.555
6       Hand Cream          42            206  9873.700
7  Facial Cleanser          42            276  5063.638
8          Shampoo          36            198  5886.201
9    Running Shoes          36            210  9937.491
10      Face Cream          35            204  9831.360
```

```r
print("Top Product by GMV")
```

```
[1] "Top Product by GMV"
```

```r
dbGetQuery(my_connection,top_product_gmv)
```

```
    product_name total_order total_quantity total_gmv
1     Toothbrush          60            316 16628.869
2    Styling Gel          46            251 15893.005
3     Foundation          61            314 13263.437
4       Lipstick          31            202 11824.907
5  Running Shoes          36            210  9937.491
6     Hand Cream          42            206  9873.700
7     Face Cream          35            204  9831.360
8    Denim Jeans          32            156  9426.456
9         Wallet          43            233  9175.555
10     Body Wash          27            142  8852.063
```

```r
print("Top Seller by Number of Order")
```

```
[1] "Top Seller by Number of Order"
```

```r
dbGetQuery(my_connection,top_seller)
```

```
   supplier_id total_order total_quantity total_seller_settlement
1       01314q          45            253                3967.950
2       91928z          44            264                7381.747
3       15418t          38            224                6172.986
4       03268l          38            221               10278.569
```

```
5       24853q          34             188                 5842.194
6       39130t          33             172                 6935.794
7       78560p          32             182                 1641.360
8       99243y          30             198                 4493.886
9       50696s          29             160                 4482.903
10      07920r          29             190                 6330.293
```

```r
print("Top Seller by Settlement Amount")
```

```
[1] "Top Seller by Settlement Amount"
```

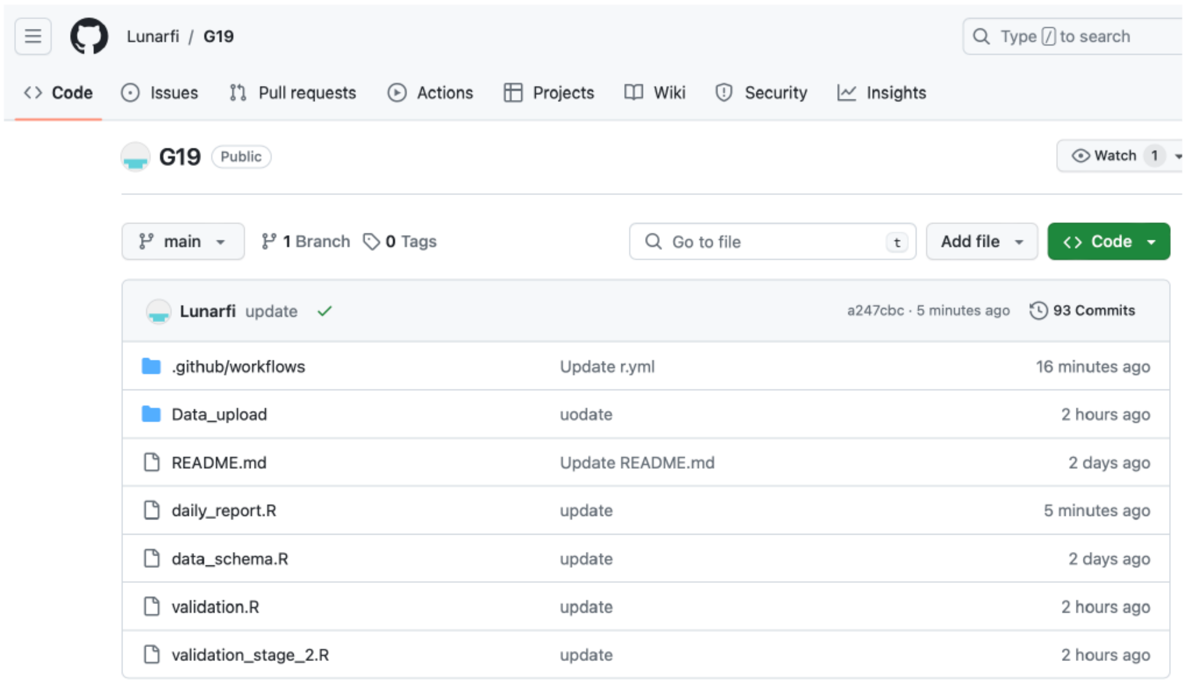
```r
dbGetQuery(my_connection,top_seller_settlement)
```

```
   supplier_id total_order total_quantity total_seller_settlement
1       032681          38            221               10278.569
2       91928z          44            264                7381.747
3       39130t          33            172                6935.794
4       17202g          28            151                6783.202
5       81867y          22            119                6582.198
6       07920r          29            190                6330.293
7       15418t          38            224                6172.986
8       14559o          26            149                6097.715
9       24853q          34            188                5842.194
10      89782w          23            141                5650.540
```

## Part 3: Data Pipeline Generation

### 3.1 GitHub Repository and Workflow Setup

We have created a GitHub repository for this project which can be accessed through this link: https://github.com/Lunarfi/G19.



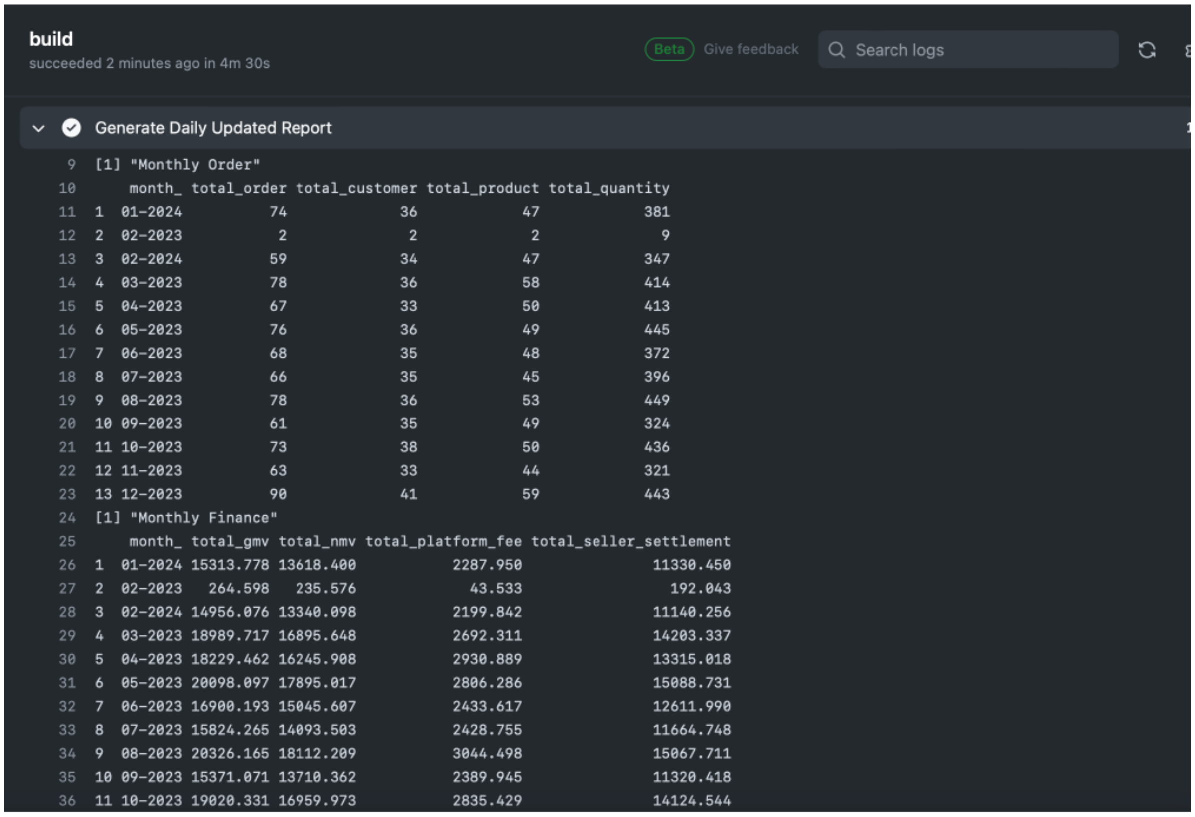
Lunarfi update ✓ a247cbc · 5 minutes ago 93 Commits

| | | |
|---|---|---|
| .github/workflows | Update r.yml | 16 minutes ago |
| Data_upload | uodate | 2 hours ago |
| README.md | Update README.md | 2 days ago |
| daily_report.R | update | 5 minutes ago |
| data_schema.R | update | 2 days ago |
| validation.R | update | 2 hours ago |
| validation_stage_2.R | update | 2 hours ago |

Figure 1: GitHub Repository

## Workflow setup

data_schema.R is the script to create the schema for the database. Then, validation.R is run to check the stage 1 validation and ingest the data into the database. After the data is ingested, validation_stage_2 will run to ensure the duplication and referential integrity of the whole data. After the final validation, an automated report will be generated to show the monthly order, finance, as well as the top sellers and products of the current period.



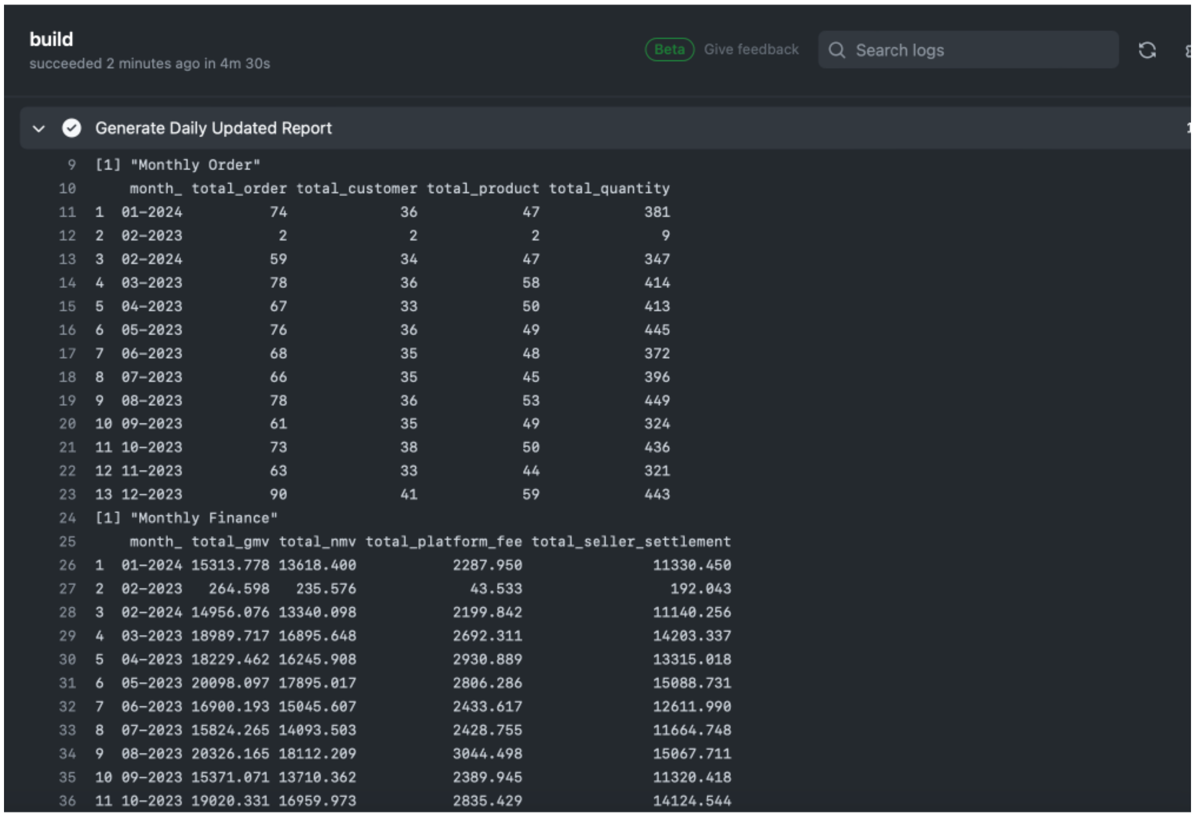
build
succeeded 2 minutes ago in 4m 30s

Beta  Give feedback  Search logs

Generate Daily Updated Report

```
[1] "Monthly Order"
    month_ total_order total_customer total_product total_quantity
1  01-2024          74             36            47            381
2  02-2023           2              2             2              9
3  02-2024          59             34            47            347
4  03-2023          78             36            58            414
5  04-2023          67             33            50            413
6  05-2023          76             36            49            445
7  06-2023          68             35            48            372
8  07-2023          66             35            45            396
9  08-2023          78             36            53            449
10 09-2023          61             35            49            324
11 10-2023          73             38            50            436
12 11-2023          63             33            44            321
13 12-2023          90             41            59            443
[1] "Monthly Finance"
    month_ total_gmv total_nmv total_platform_fee total_seller_settlement
1  01-2024 15313.778 13618.400           2287.950               11330.450
2  02-2023   264.598   235.576             43.533                 192.043
3  02-2024 14956.076 13340.098           2199.842               11140.256
4  03-2023 18989.717 16895.648           2692.311               14203.337
5  04-2023 18229.462 16245.908           2930.889               13315.018
6  05-2023 20098.097 17895.017           2806.286               15088.731
7  06-2023 16900.193 15045.607           2433.617               12611.990
8  07-2023 15824.265 14093.503           2428.755               11664.748
9  08-2023 20326.165 18112.209           3044.498               15067.711
10 09-2023 15371.071 13710.362           2389.945               11320.418
11 10-2023 19020.331 16959.973           2835.429               14124.544
```

Figure 2: Monthly Order and Finance Report



build
succeeded 2 minutes ago in 4m 30s

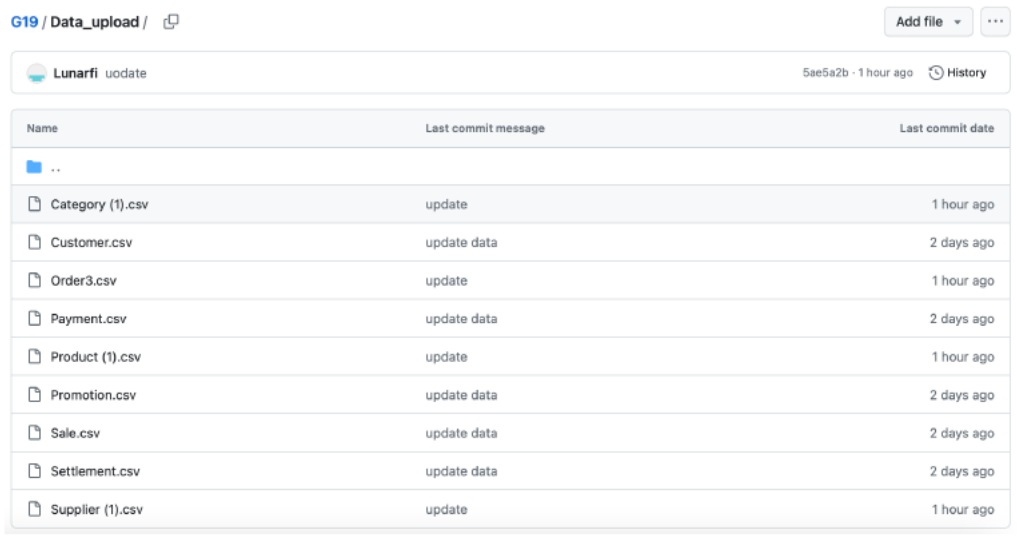
Generate Daily Updated Report

```
[1] "Monthly Order"
     month_ total_order total_customer total_product total_quantity
1  01-2024          74             36            47            381
2  02-2023           2              2             2              9
3  02-2024          59             34            47            347
4  03-2023          78             36            58            414
5  04-2023          67             33            50            413
6  05-2023          76             36            49            445
7  06-2023          68             35            48            372
8  07-2023          66             35            45            396
9  08-2023          78             36            53            449
10 09-2023          61             35            49            324
11 10-2023          73             38            50            436
12 11-2023          63             33            44            321
13 12-2023          90             41            59            443
[1] "Monthly Finance"
     month_ total_gmv total_nmv total_platform_fee total_seller_settlement
1  01-2024 15313.778 13618.400           2287.950               11330.450
2  02-2023   264.598   235.576             43.533                 192.043
3  02-2024 14956.076 13340.098           2199.842               11140.256
4  03-2023 18989.717 16895.648           2692.311               14203.337
5  04-2023 18229.462 16245.908           2930.889               13315.018
6  05-2023 20098.097 17895.017           2806.286               15088.731
7  06-2023 16900.193 15045.607           2433.617               12611.990
8  07-2023 15824.265 14093.503           2428.755               11664.748
9  08-2023 20326.165 18112.209           3044.498               15067.711
10 09-2023 15371.071 13710.362           2389.945               11320.418
11 10-2023 19020.331 16959.973           2835.429               14124.544
```

Figure 2: Monthly Order and Finance Report

Figure 1: Top Sellers and Products

The CSV files as the input for the database is stored in the Data_upload folder.


G19 / Data_upload /

Add file

Lunarfi uodate 5ae5a2b · 1 hour ago History

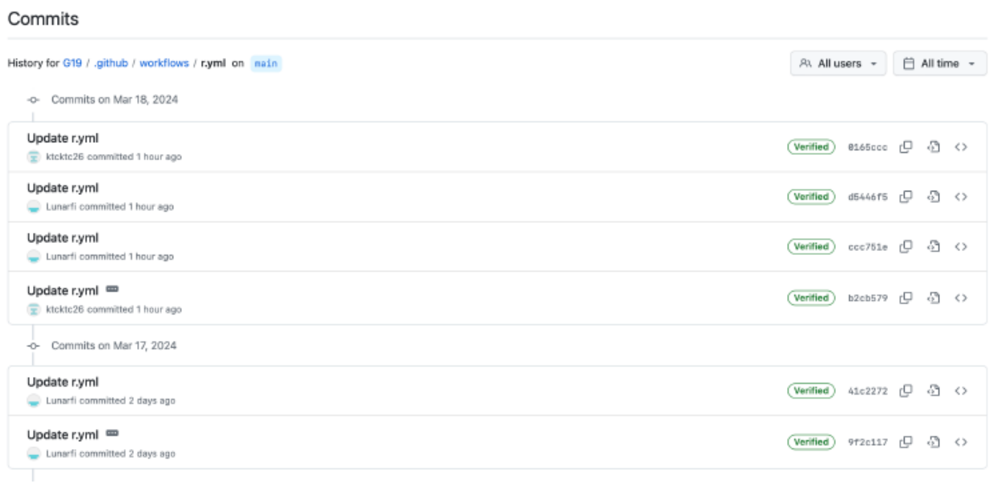
| Name | Last commit message | Last commit date |
|---|---|---|
| .. | | |
| Category (1).csv | update | 1 hour ago |
| Customer.csv | update data | 2 days ago |
| Order3.csv | update | 1 hour ago |
| Payment.csv | update data | 2 days ago |
| Product (1).csv | update | 1 hour ago |
| Promotion.csv | update data | 2 days ago |
| Sale.csv | update data | 2 days ago |
| Settlement.csv | update data | 2 days ago |
| Supplier (1).csv | update | 1 hour ago |

Figure 4: Data Upload Folder

Figure 2: Data Upload Folder

## Version- Control

Every changes to the files in the repository is managed through the version control of Git. For example, several changes made to the YAML file is recorded and can be reverted if needed.


Commits

History for G19 / .github / workflows / r.yml on main

All users · All time

Commits on Mar 18, 2024

| | | |
|---|---|---|
| Update r.yml<br>ktcktc26 committed 1 hour ago | Verified | 0165ccc |
| Update r.yml<br>Lunarfi committed 1 hour ago | Verified | d5446f5 |
| Update r.yml<br>Lunarfi committed 1 hour ago | Verified | ccc751e |
| Update r.yml<br>ktcktc26 committed 1 hour ago | Verified | b2cb579 |

Commits on Mar 17, 2024

| | | |
|---|---|---|
| Update r.yml<br>Lunarfi committed 2 days ago | Verified | 41c2272 |
| Update r.yml<br>Lunarfi committed 2 days ago | Verified | 9f2c117 |

Figure 5: Version Control

Figure 3: Version Control

## 3.2 GitHub Actions for Continuous Integration

We have automated the data validation, ingestion, and basic data analysis using workflow on GitHub. This workflow is set to run every 24 hours or if there is any changes done to the scripts and data.

```
name: workflow

on:
  schedule:
    - cron: '0 0 * * *' # Run every 24 hours
  push:
    branches: [ main ]
    paths:
      - 'Data_upload/**'  # Listen for changes in the Data_upload directory
      - '**.R'            # Listen for changes in all R files
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v2
      - name: Setup R environment
        uses: r-lib/actions/setup-r@v2
        with:
          r-version: '4.3.3'
      - name: Cache R packages
        uses: actions/cache@v2
        with:
          path: ${{ env.R_LIBS_USER }}
          key: ${{ runner.os }}-r-${{ hashFiles('**/lockfile') }}
          restore-keys: |
            ${{ runner.os }}-r-
      - name: Install packages
        if: steps.cache.outputs.cache-hit != 'true'
        run: |
          Rscript -e 'install.packages(c("ggplot2","dplyr", "readr", "RSQLite", "lubridate",
      - name: Delete database file
        run: |
          rm -f mydatabase.db
      - name: Create sql schema
        run: |
          Rscript data_schema.R
      - name: Data validation
        run: |
          Rscript validation.R
      - name: Data validation stage 2
        run: |
          Rscript validation_stage_2.R
      - name: Generate Daily Updated Report
        run: |
          Rscript daily_report.R
      - name: Data Analysis Update
        run: |
          Rscript Data_Analysis.R
      - name: Push changes
        uses: ad-m/github-push-action@v0.6.0
        with:
            github_token: ${{ secrets.MY_TOKEN }}
            branch: main
```

# Part 4: Data Analysis

## 4.1: Advanced Data Analysis in R

Installed all packages needed for data analysis. The code also gets the date and time the code is ran for figure labeling. Refer file "Data_Analysis.R" attached in Github for automated analysis as a part of workflow.

```r
library(readr)
library(dplyr)
library(tidyverse)
library(ggplot2)
library(lubridate)
library(writexl)
this_filename_date <- as.character(Sys.Date())
# format the Sys.time() to show only hours and minutes
this_filename_time <- as.character(format(Sys.time(), format = "%H_%M"))
```

The following code loads all the data from the databases.

```r
# Load from Database
library(DBI)
my_connection <- dbConnect(RSQLite::SQLite(), dbname = "mydatabase.db")

# Load Data from Database
category <- dbGetQuery(my_connection, "SELECT * FROM Category")
customer <- dbGetQuery(my_connection, "SELECT * FROM Customer")
order <- dbGetQuery(my_connection, "SELECT * FROM Orders")
payment <- dbGetQuery(my_connection, "SELECT * FROM Payment")
product <- dbGetQuery(my_connection, "SELECT * FROM Product")
promotion <- dbGetQuery(my_connection, "SELECT * FROM Promotion")
sale <- dbGetQuery(my_connection, "SELECT * FROM Sales")
settlement <- dbGetQuery(my_connection, "SELECT * FROM Settlement")
supplier <- dbGetQuery(my_connection, "SELECT * FROM Supplier")
```

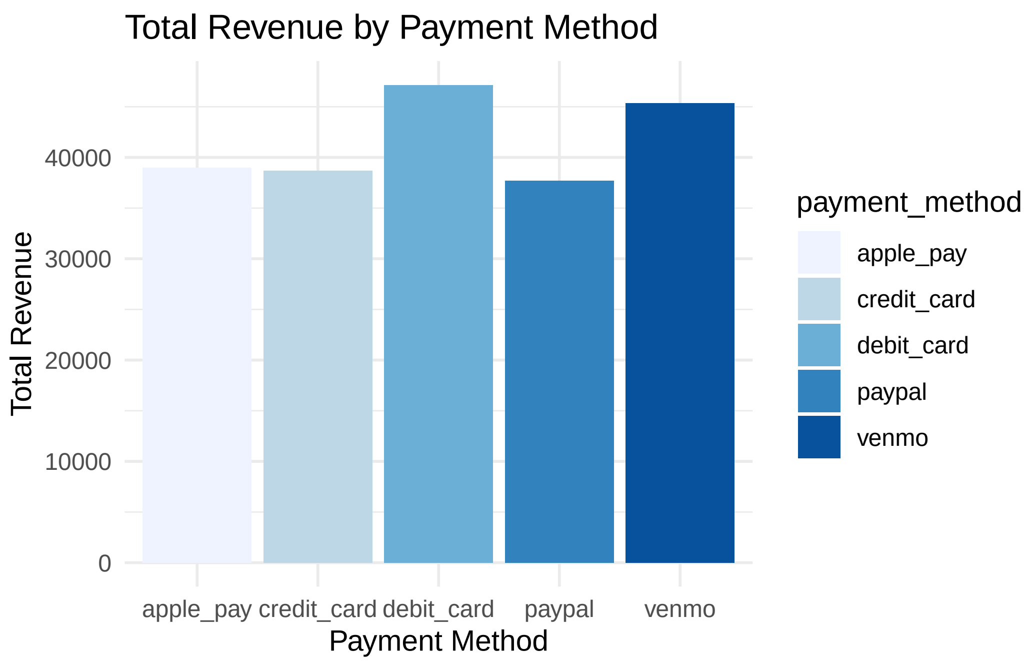
After, we can use various methods of advanced data analysis and visualisation.

Analysis 1:

### Revenue per Payment Type

This is a graph which states different modes of payment method on ‘x axis’ and Total Revenue on ‘y axis’ which helps us interpret which payment types are bringing in the most revenue, facilitating an analysis of the most utilised checkout option and customer preferences. This can be helpful to optimise revenue generation by aligning promotional efforts with preferred payment method.

```r
#barplot for revenue per payment type
#Highest revenue per customer
merged_order <- order %>%
  inner_join(product, customer, by = "product_id")

# merging orders, customers and priduct dfs with price after discount
final_price_df <- merged_order %>%
  mutate(
    final_price = round(price - (price * (discount_rate / 100)), 2)
  )

#*quantity price of order_id by order_id
final_price_df$full_price <-  (final_price_df$final_price*final_price_df$order_quantity)
lm_order_payment <-  final_price_df %>%
  inner_join(payment, by = "order_id")%>%
  select(order_id, payment_method, full_price)%>%
  group_by(payment_method)%>%
  summarise(total_per_payment_method = sum(full_price))

library(ggplot2)

ggplot(lm_order_payment, aes(x = payment_method, y = total_per_payment_method, fill = paymen
  geom_col() +  # Use geom_col() for pre-summarized data
  labs(title = "Total Revenue by Payment Method",
       x = "Payment Method",
       y = "Total Revenue") +
  theme_minimal()+
  scale_fill_brewer(palette = )
```



```
ggsave(plot = last_plot(),paste0("figures/barplots/payment_methods",
              this_filename_date,"_",
              this_filename_time,".png"))
```

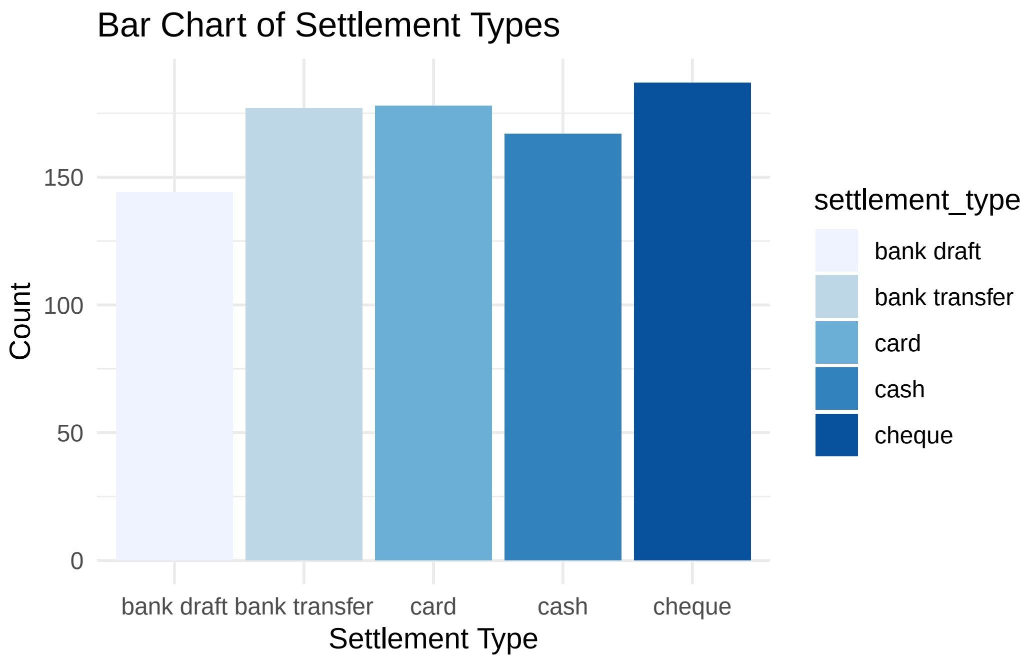
```
Saving 5.5 x 3.5 in image
```

Analysis 2:

**Number of settlements per Settlement Type**

This is a bar graph, which states different modes of settlement on ‘x axis’ and count on ‘y axis’, compares the count of each settlement type preferred by suppliers. This helps visualise the most used method of settlement within the eCommerce company and have operational clarity.

```
#Barplot for count per settlement type
ggplot(settlement, aes(x = settlement_type, fill = settlement_type)) +
  geom_bar() +
  theme_minimal() +
  labs(title = "Bar Chart of Settlement Types",
       x = "Settlement Type",
       y = "Count") +
  scale_fill_brewer(palette = )
```



```
ggsave(plot = last_plot(),paste0("figures/barplots/settlement_types",
            this_filename_date,"_",
            this_filename_time,".png"))
```

```
Saving 5.5 x 3.5 in image
```

Analysis 3:

### Highest Grossing Sellers/Suppliers

This table includes supplier ID , supplier name and total settlement amount which allows us to quickly identify the best sellers in terms of net revenue from sale so that we can employ methods to keep the seller loyal to the platform. For example, give them lower platform rates, better discounted promotion deal etc.

```
# highest grossing sellers/suppliers
supplier_gross_df <- supplier %>%
  inner_join(product, by = "supplier_id")


supplier_gross_df <- supplier_gross_df %>%
  mutate(
    final_seller_price = round(price - (price * ((tax_rate/100) + (discount_rate/100) +(plat

supplier_gross_df <- supplier_gross_df %>% merge(order, by = "product_id")
supplier_gross_df$fullp <- supplier_gross_df$order_quantity*supplier_gross_df$final_seller_p


seller_details <- supplier_gross_df %>% group_by(supplier_id, seller) %>% summarize(total_se
```

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

```
seller_details <- seller_details %>% arrange(desc(total_settlement))
print(seller_details)
```

```
# A tibble: 42 x 3
# Groups:   supplier_id [42]
   supplier_id seller             total_settlement
   <chr>       <chr>                         <dbl>
 1 03268l      Cello Tewkesberry            10279.
 2 91928z      Vilhelmina Clewer             7381.
 3 39130t      Lynnell Linggood              6935.
 4 17202g      Brannon Hurlston              6783.
 5 81867y      Matilde Bondesen              6582.
 6 07920r      Tamas Matysik                 6330.
 7 15418t      Rubia Folk                    6173.
 8 14559o      Loraine Lorence               6098.
 9 24853q      Gerrie McKeachie              5843.
```

```
10 89782w       Oberon Elders                    5651.
# i 32 more rows
```

Analysis 4:

### Highest Revenue per Customer

This is an important table which includes Customer name and Revenue from those customer. Allows us to identify top customers that spend the most at our company. Therefore, we know which customers to retain through various marketing methods. Also can help identify whether changes to our company positively or negatively affect our customer base.

```r
merge2 <- order %>%
  full_join(product, by = "product_id")

merge2$final_price  <- round(merge2$price* (1- (merge2$discount_rate/100)))

merge2$fullprice <- (merge2$final_price)*(merge2$order_quantity)

#merging customer with order and product
cus_order <- merge2%>% full_join(customer, by = "customer_id")


# merging orders, customers and priduct dfs with price after discount
final_price_df <- cus_order %>%
  mutate(final_price = round(price - (price * (discount_rate / 100)), 2)) %>%
  select(order_id,customer_first_name, customer_last_name, fullprice)

# merging customer first name and last name

final_price_df$name <- paste(final_price_df$customer_first_name, final_price_df$customer_las

final_price_df$customer_first_name <- NULL
final_price_df$customer_last_name <- NULL

# checking revenue generated by each customer
rev_per_cm <- final_price_df %>% group_by(name)%>% summarize(sum(fullprice))
rev_per_cm$revenue <- rev_per_cm$`sum(fullprice)`
rev_per_cm$`sum(fullprice)`<- NULL
rev_per_cm <- rev_per_cm %>% arrange(desc(revenue))
```

```r
library(writexl)

write_xlsx(rev_per_cm, path = paste0("figures/tables/revenue_per_customer",
              this_filename_date,"_",
              this_filename_time,".xlsx"),col_names = TRUE)


print(head(rev_per_cm))
```

```
# A tibble: 6 x 2
  name                revenue
  <chr>                 <dbl>
1 Coralyn Ord            7849
2 Myron Blackler         6357
3 Sigfrid Brettor        5884
4 Letizia Bastow         5718
5 Stanfield Fortey       5624
6 Diann Bartholomew      5499
```

Analysis 5:

#### Highest Revenue from Product

This table includes Product id, the category it beongs to and quantity and revenue from sale of those products which shows the best selling products in terms of quantity and revenue. This allows the company to capitalise on current trends to increase conversion rates on products, therefore, increasing revenue.

Furthermore, it allows the company to analyse growth areas as well as support the prosperous areas.

```r
# checking most sold products

ms_products <- merge2 %>% group_by(product_id)%>%
  summarize(revenue = sum(fullprice), quantity = sum(order_quantity))

ms_product_name <- ms_products %>% merge(product, by = "product_id")


ms_product123 <- ms_product_name %>% merge(category, by = "category_id" )%>%
  select(product_name,revenue, quantity, price, discount_rate )
```

```
ms_product_name$final_price <- ms_product_name$price * (1-((ms_product_name$discount_rate)/1

ms_product_df <- ms_product_name %>% arrange(desc(revenue)) %>% select( product_name,final_p

write_xlsx(ms_product_df, path = paste0("figures/tables/top-selling-products",
              this_filename_date,"_",
              this_filename_time,".xlsx"),col_names = TRUE)


print(head(ms_product_name))
```

```
  product_id revenue quantity category_id supplier_id   product_name price
1     00063g     432       12      28916w      84475z Essential Oils 39.12
2     00613n     245       35      25070l      17202g         Wallet  7.39
3     00908q    4095       63      28916w      17202g     Toothbrush 67.46
4     02090u    2790       45      81761s      97584v  Business Suit 62.81
5     02510u    1296       72      92360v      11866m     Foundation 17.97
6     02931a     600       15      92360v      97584v        Mascara 41.59
  discount_rate final_price
1             8     35.9904
2             4      7.0944
3             3     65.4362
4             2     61.5538
5             0     17.9700
6             3     40.3423
```

Analysis 6:

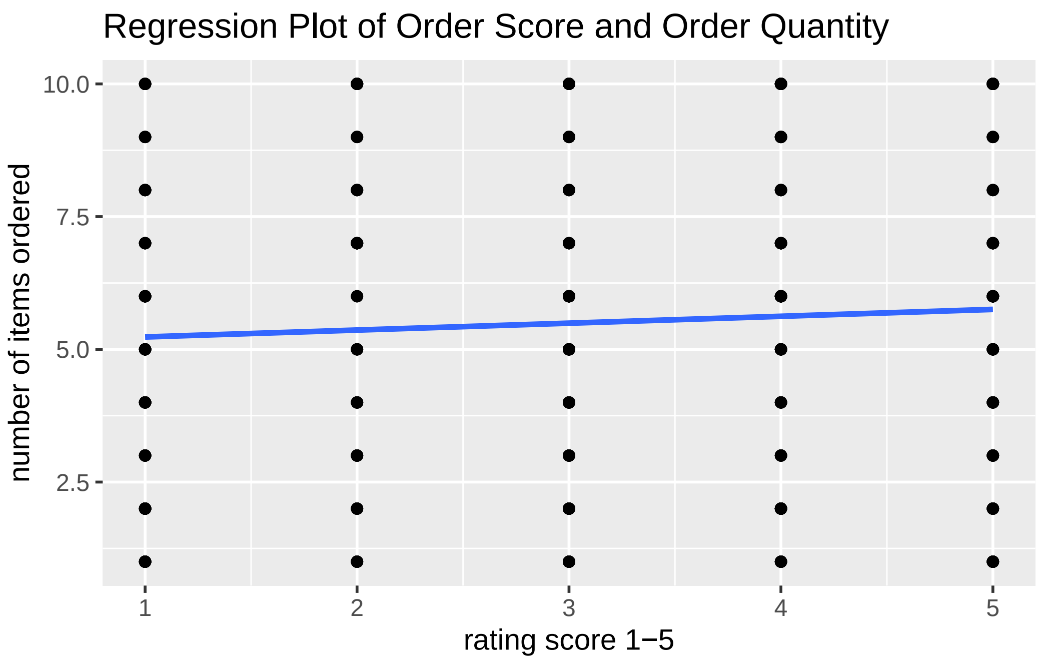
### Regressions: Reviews against Order Quantity

The regression plot allows us to gain insight as to whether reviews influence the number of items ordered by customers. This can be helpful to understand any biasness towards rating by quantity.

```
# reviews regressed against order quantity
review_order_db <- order
```

```r
review_order_plot_data <- review_order_db%>%
  select(order_id, rating_score, order_quantity)%>%
  na.omit()

ggplot(review_order_plot_data, aes(x = rating_score, y = order_quantity)) +
  geom_point() +
  geom_smooth(method = lm, se = FALSE) +
  labs(title = "Regression Plot of Order Score and Order Quantity",
       x = "rating score 1-5",
       y = "number of items ordered")
```

```
`geom_smooth()` using formula = 'y ~ x'
```



```r
ggsave(plot = last_plot(),paste0("figures/regression_plot/score-quantity",
              this_filename_date,"_",
              this_filename_time,".png"))
```

```
Saving 5.5 x 3.5 in image
`geom_smooth()` using formula = 'y ~ x'
```

Analysis 7:

#### Regression: Quantity against Discount Rate

The regression plot allows us to gain insight as to whether the discount rate on an item affects the quantity sold. This can be helpful to define the discount policies if they are having any direct impact on quantity sold which impacts sales.

```r
lmporsche <- lm(quantity~discount_rate , data = ms_product_name)
summary(lmporsche)
```

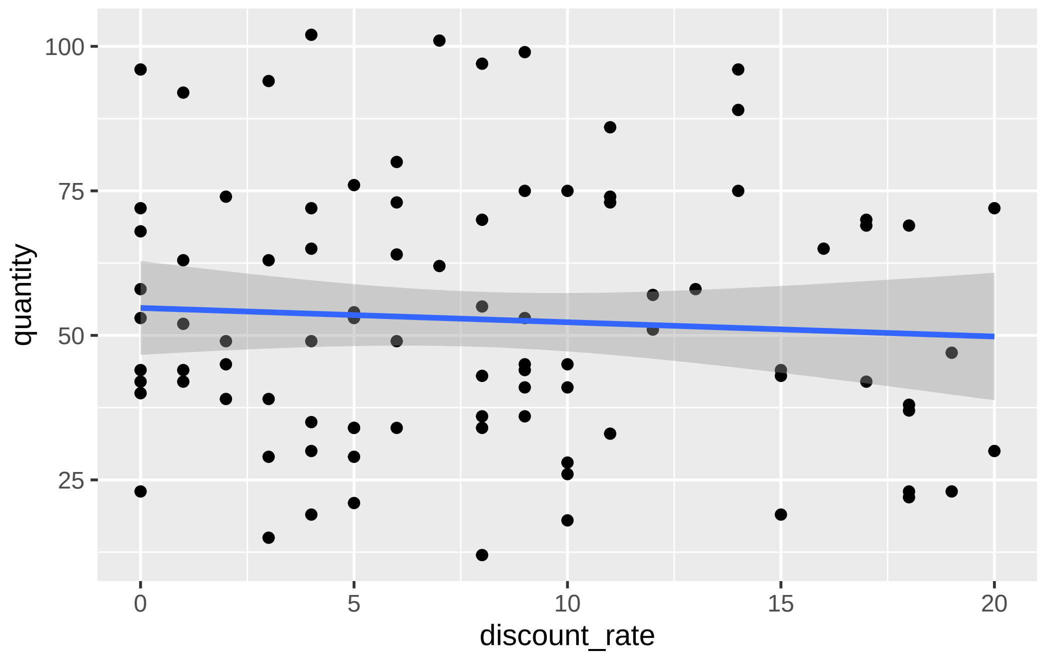
```
Call:
lm(formula = quantity ~ discount_rate, data = ms_product_name)

Residuals:
    Min      1Q  Median      3Q     Max 
-40.770 -16.708  -4.511  18.403  48.241 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)    
(Intercept)    54.7481     4.0859  13.399   <2e-16 ***
discount_rate  -0.2473     0.4152  -0.596    0.553    
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 22.75 on 88 degrees of freedom
  (7 observations deleted due to missingness)
Multiple R-squared:  0.004015,	Adjusted R-squared:  -0.007303 
F-statistic: 0.3548 on 1 and 88 DF,  p-value: 0.553
```

```r
ggplot(data = ms_product_name, mapping = aes(y = quantity, x = discount_rate))+
  geom_point()+ geom_smooth(method = lm, se = TRUE)
```

```
`geom_smooth()` using formula = 'y ~ x'
```

```
Warning: Removed 7 rows containing non-finite outside the scale range
(`stat_smooth()`).
```

```
Warning: Removed 7 rows containing missing values or values outside the scale range
(`geom_point()`).
```



```r
ggsave(plot = last_plot(),paste0("figures/regression_plot/discount-quantity",
              this_filename_date,"_",
              this_filename_time,".png"))
```

```
Saving 5.5 x 3.5 in image
`geom_smooth()` using formula = 'y ~ x'

Warning: Removed 7 rows containing non-finite outside the scale range (`stat_smooth()`).
Removed 7 rows containing missing values or values outside the scale range
(`geom_point()`).
```

Analysis 7:

#### Regression : Price against Orders

The regression plot allows us to analyse whether the price of an order impacts the number of units sold. This can help supplier revise the price of any product if they want to improve sale of that product.

```r
# Regressing price against orders

lm_gt3 <- lm(quantity~price, data = ms_product_name )
summary(lm_gt3)
```

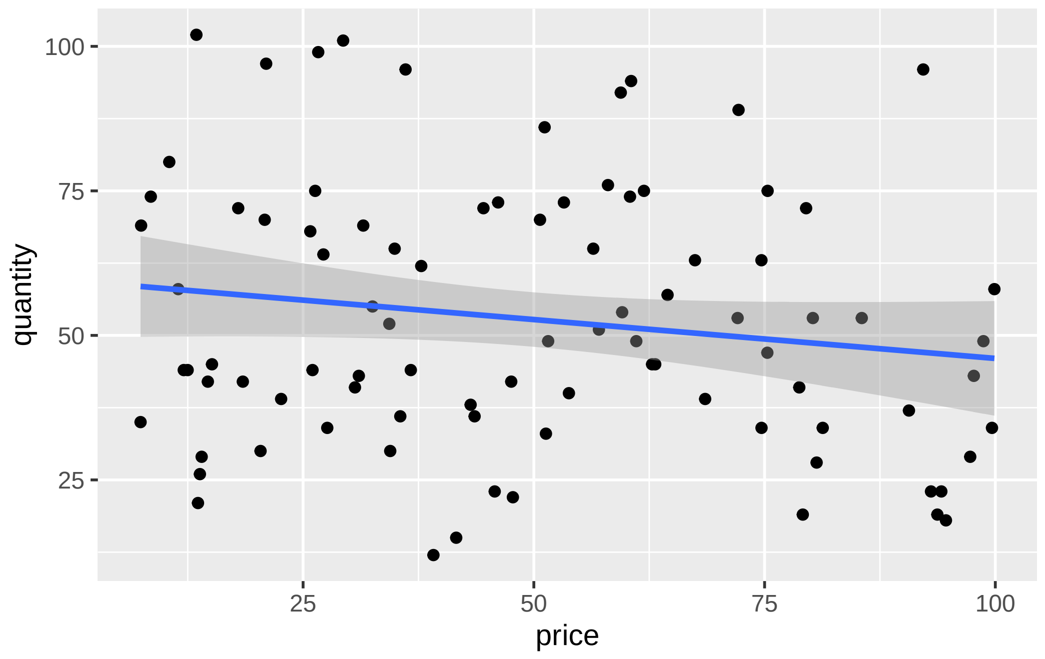
```
Call:
lm(formula = quantity ~ price, data = ms_product_name)

Residuals:
    Min      1Q  Median      3Q     Max 
-42.200 -15.616  -2.589  14.661  48.936 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 59.46074    4.94718  12.019   <2e-16 ***
price       -0.13446    0.08736  -1.539    0.127    
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 22.5 on 88 degrees of freedom
  (7 observations deleted due to missingness)
Multiple R-squared:  0.02622,	Adjusted R-squared:  0.01515 
F-statistic: 2.369 on 1 and 88 DF,  p-value: 0.1273
```

```r
# graph of regression
#add colour as CATEGORY WHEN FIXED!!!
ggplot(data = ms_product_name, mapping = aes(y = quantity, x = price))+
  geom_point()+ geom_smooth(method = lm, se = TRUE)
```

```
`geom_smooth()` using formula = 'y ~ x'
```

```
Warning: Removed 7 rows containing non-finite outside the scale range
(`stat_smooth()`).
```

```
Warning: Removed 7 rows containing missing values or values outside the scale range
(`geom_point()`).
```



```r
ggsave(paste0("figures/regression_plot/price-quantity",
              this_filename_date,"_",
              this_filename_time,".png"))
```

```
Saving 5.5 x 3.5 in image
`geom_smooth()` using formula = 'y ~ x'

Warning: Removed 7 rows containing non-finite outside the scale range (`stat_smooth()`).
Removed 7 rows containing missing values or values outside the scale range
(`geom_point()`).
```

### 4.2: Reporting for Stakeholders

After running an analysis on the eCommerce company, we can gain valuable data insights to leverage in the future.

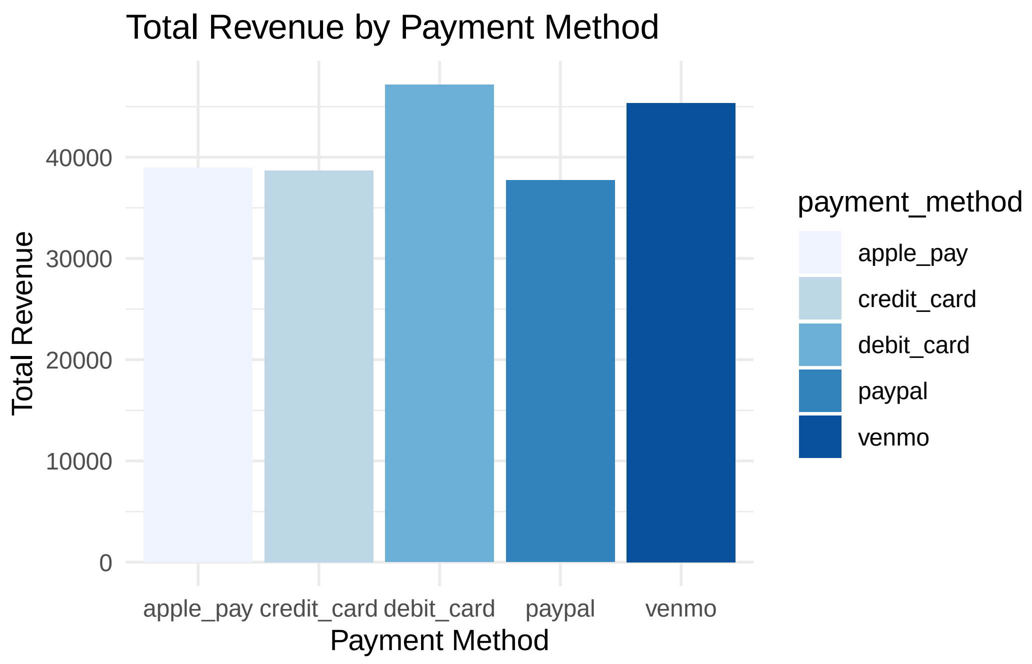
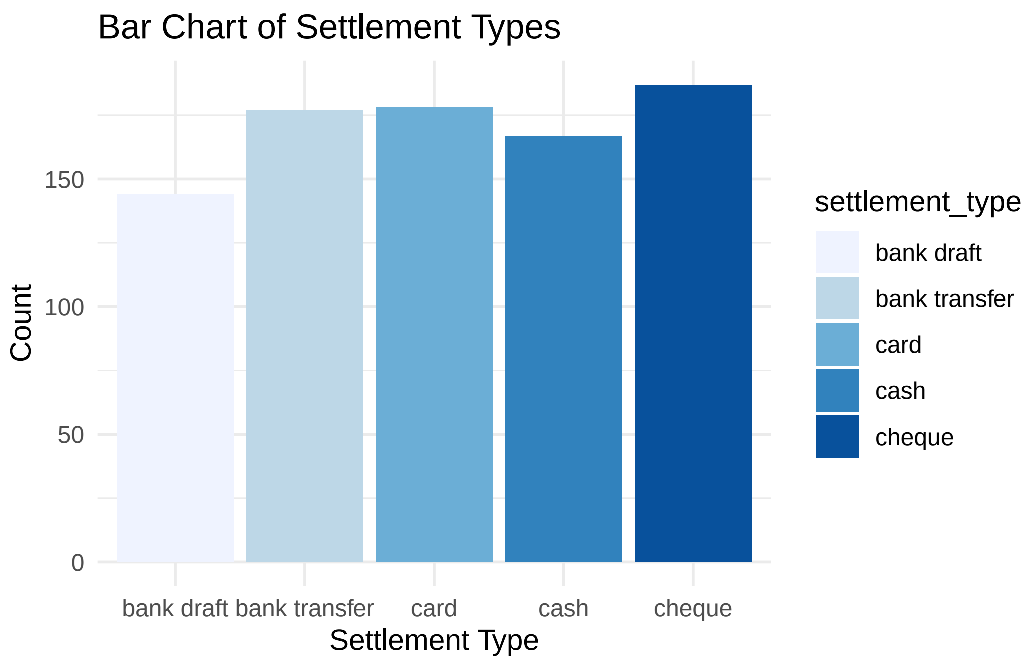


Figure 1 Total Revenue per Payment Method by Customer

We can see that 'debit_card' is the common method among customers bringing in the most revenue. The company can use this data to ensure debit card transactions run smoothly. If not, the company may lose a lot of revenue as customers may go to different e-commerce platform where they get the option for debit card.



```
Saving 5.5 x 3.5 in image
```

Figure 2 Graph to show settlement types used by suppliers

From this graph, we can see that overall settlement types are quite even, displaying minimal deviation. The most common settlement type is 'cheque' with 'bank draft' being the least.

```
`geom_smooth()` using formula = 'y ~ x'
```

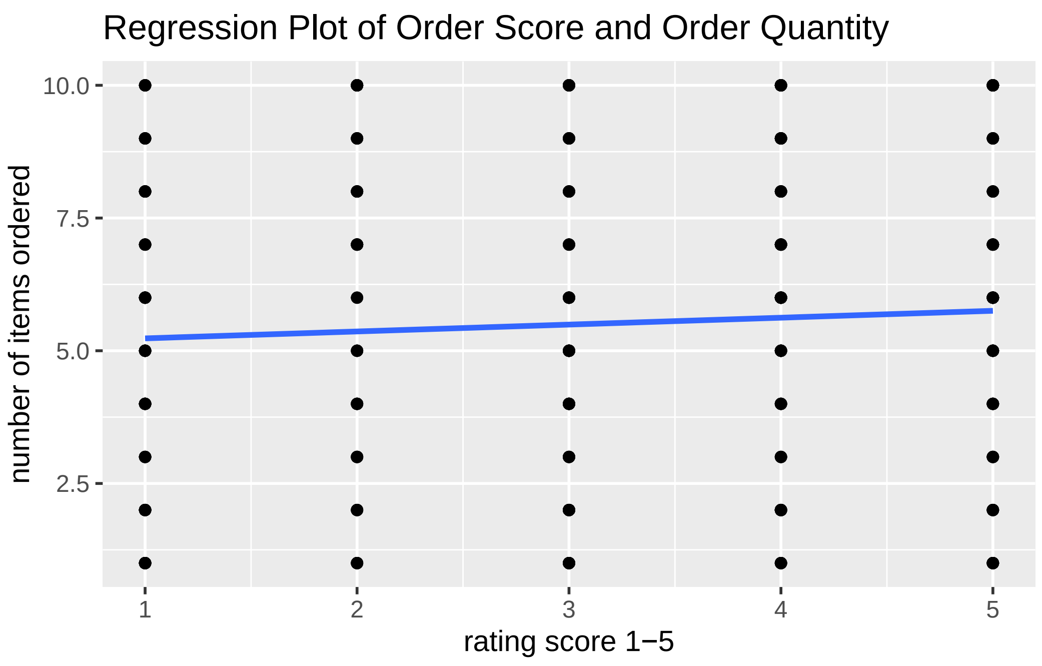


Figure 3: Relation of rating and order quantity

We ran a regression model to see if there are an order's rating score has an impact on the order's quantity (Figure 3).

The results show that there is no significant effect.

```
Call:
lm(formula = quantity ~ price, data = ms_product_name)

Residuals:
    Min      1Q  Median      3Q     Max 
-42.200 -15.616  -2.589  14.661  48.936 
```

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 59.46074    4.94718  12.019   <2e-16 ***
price       -0.13446    0.08736  -1.539    0.127
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 22.5 on 88 degrees of freedom
  (7 observations deleted due to missingness)
Multiple R-squared:  0.02622,   Adjusted R-squared:  0.01515
F-statistic: 2.369 on 1 and 88 DF,  p-value: 0.1273

`geom_smooth()` using formula = 'y ~ x'

Warning: Removed 7 rows containing non-finite outside the scale range
(`stat_smooth()`).

Warning: Removed 7 rows containing missing values or values outside the scale range
(`geom_point()`).
```

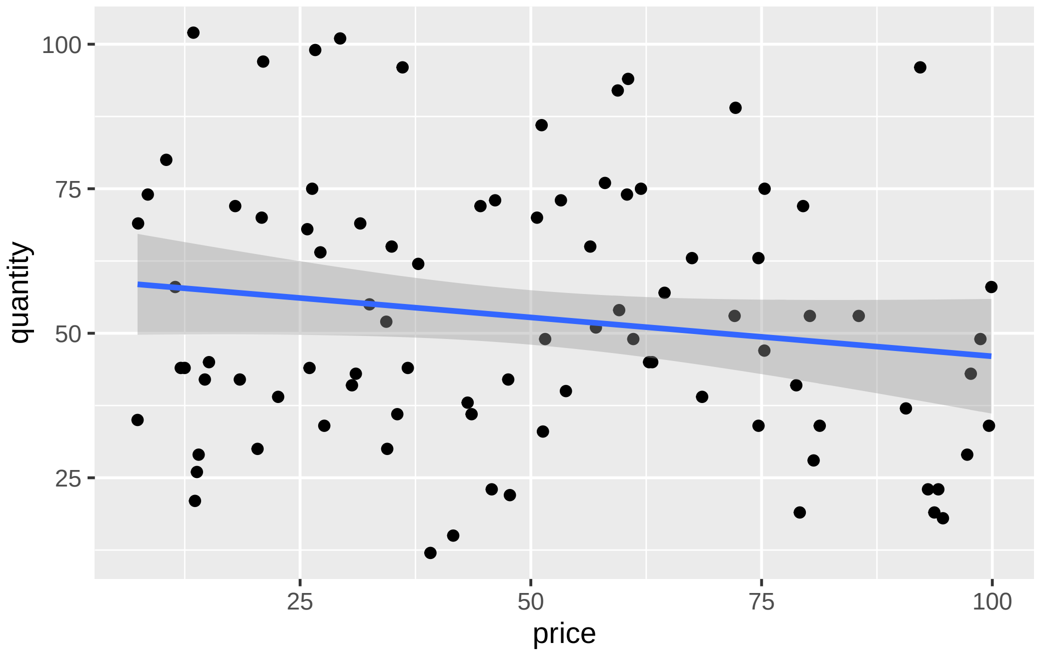


Figure 4: Regression of Price Against Quantity

We also ran a regression model to see if there are an order's rating score has an impact on the order's quantity (Figure 4).

The graph shows a slight positive correlation between price and quantity sold, but it is not significant.

```
Call:
lm(formula = quantity ~ discount_rate, data = ms_product_name)

Residuals:
    Min      1Q  Median      3Q     Max 
-40.770 -16.708  -4.511  18.403  48.241 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)    
(Intercept)    54.7481     4.0859  13.399   <2e-16 ***
discount_rate  -0.2473     0.4152  -0.596    0.553    
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 22.75 on 88 degrees of freedom
  (7 observations deleted due to missingness)
Multiple R-squared:  0.004015,	Adjusted R-squared:  -0.007303 
F-statistic: 0.3548 on 1 and 88 DF,  p-value: 0.553
```

```
`geom_smooth()` using formula = 'y ~ x'
```

```
Warning: Removed 7 rows containing non-finite outside the scale range
(`stat_smooth()`).
```

```
Warning: Removed 7 rows containing missing values or values outside the scale range
(`geom_point()`).
```

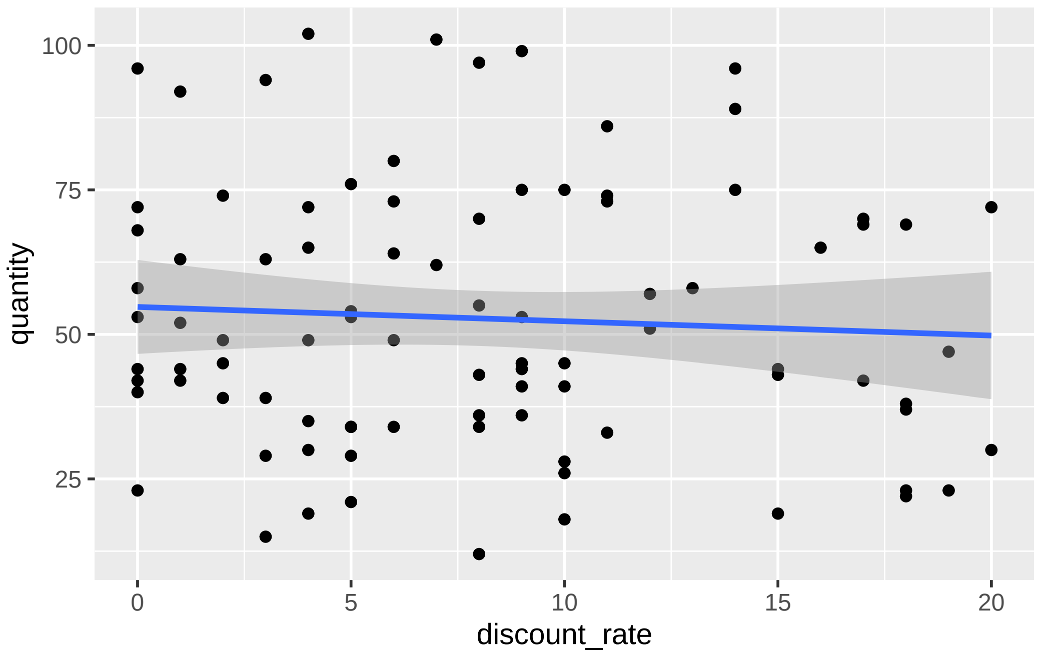
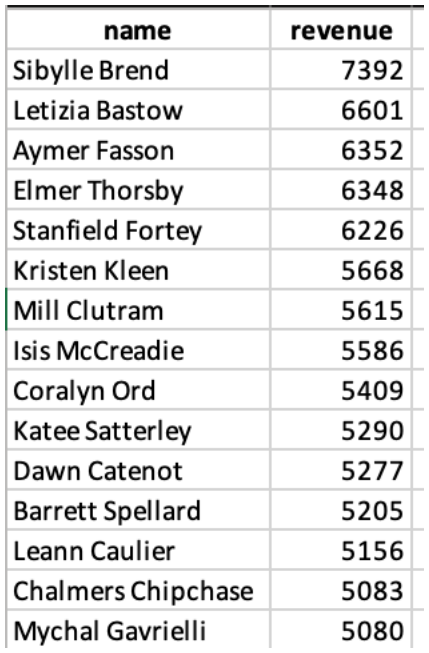


Figure 5: Regression of Discount Against Quantity

Finally we analysed whether the discount rate of a product has an effect on the quantity ordered (Figure 5).

This regression displays a very slight negative correlation. It indicates that as discount rate increases, the quantity of product bought decreases, however, it isn't a significant impact.

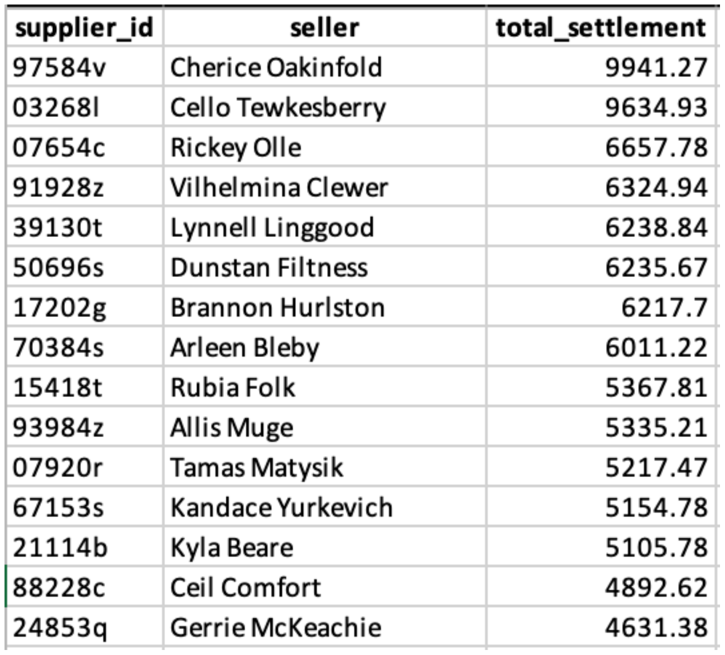
Our analysis also provides excel tables of various customer figures. This allows for files to be easily read and analysed in R. Below are screenshots of sections from all the excel tables produced.



| name | revenue |
|---|---|
| Sibylle Brend | 7392 |
| Letizia Bastow | 6601 |
| Aymer Fasson | 6352 |
| Elmer Thorsby | 6348 |
| Stanfield Fortey | 6226 |
| Kristen Kleen | 5668 |
| Mill Clutram | 5615 |
| Isis McCreadie | 5586 |
| Coralyn Ord | 5409 |
| Katee Satterley | 5290 |
| Dawn Catenot | 5277 |
| Barrett Spellard | 5205 |
| Leann Caulier | 5156 |
| Chalmers Chipchase | 5083 |
| Mychal Gavrielli | 5080 |

Figure 6 details revenue per customer.

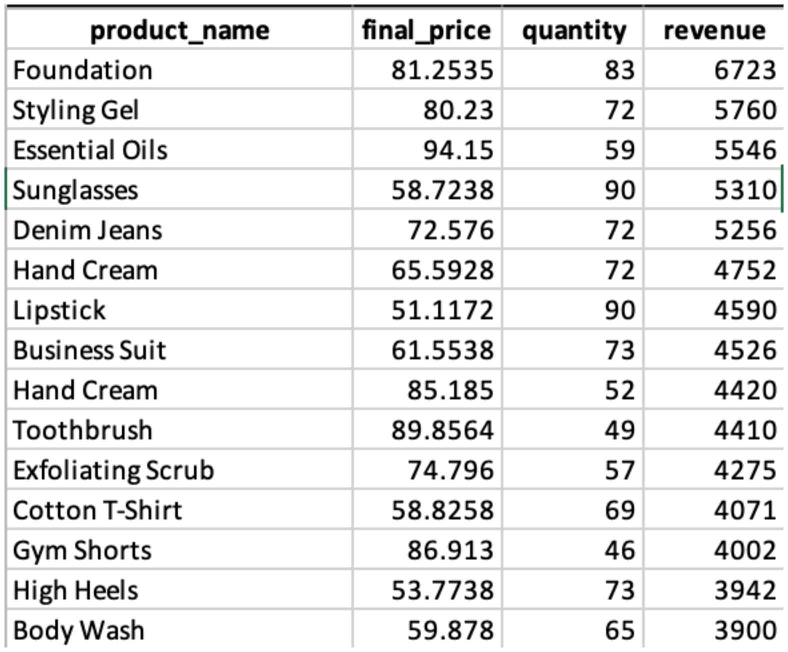
From this data, we can see top 5 customers are shopping spent more than than GBP 6,000. Maybe for the next quarter we should offer than loyalty schemes and offers to retain them and gain customer value.



| supplier_id | seller | total_settlement |
|---|---|---|
| 97584v | Cherice Oakinfold | 9941.27 |
| 03268l | Cello Tewkesberry | 9634.93 |
| 07654c | Rickey Olle | 6657.78 |
| 91928z | Vilhelmina Clewer | 6324.94 |
| 39130t | Lynnell Linggood | 6238.84 |
| 50696s | Dunstan Filtness | 6235.67 |
| 17202g | Brannon Hurlston | 6217.7 |
| 70384s | Arleen Bleby | 6011.22 |
| 15418t | Rubia Folk | 5367.81 |
| 93984z | Allis Muge | 5335.21 |
| 07920r | Tamas Matysik | 5217.47 |
| 67153s | Kandace Yurkevich | 5154.78 |
| 21114b | Kyla Beare | 5105.78 |
| 88228c | Ceil Comfort | 4892.62 |
| 24853q | Gerrie McKeachie | 4631.38 |

Figure 7 (supplier-settlement) is important. The table displays how much each supplier has made.

The two standout sellers on our platform are “Cherice Oakinfold” and “Cello Tewkesberry”, who both have total settlement values exceeding 9000, we can offer them additional discount on promotion of there listed products and also charge less platform fees.



| product_name | final_price | quantity | revenue |
|---|---|---|---|
| Foundation | 81.2535 | 83 | 6723 |
| Styling Gel | 80.23 | 72 | 5760 |
| Essential Oils | 94.15 | 59 | 5546 |
| Sunglasses | 58.7238 | 90 | 5310 |
| Denim Jeans | 72.576 | 72 | 5256 |
| Hand Cream | 65.5928 | 72 | 4752 |
| Lipstick | 51.1172 | 90 | 4590 |
| Business Suit | 61.5538 | 73 | 4526 |
| Hand Cream | 85.185 | 52 | 4420 |
| Toothbrush | 89.8564 | 49 | 4410 |
| Exfoliating Scrub | 74.796 | 57 | 4275 |
| Cotton T-Shirt | 58.8258 | 69 | 4071 |
| Gym Shorts | 86.913 | 46 | 4002 |
| High Heels | 53.7738 | 73 | 3942 |
| Body Wash | 59.878 | 65 | 3900 |

Figure 8 (top-selling-products) details the product, its price, quantity sold and how much total revenue is made from sales of that product.

We can see that ‘foundation’ is currently by far the highest selling product in terms of revenue, at 6723. Therefore, we should probably collborate with more beauty supplier as they attract more customers.

These were the key business findings for the current period.