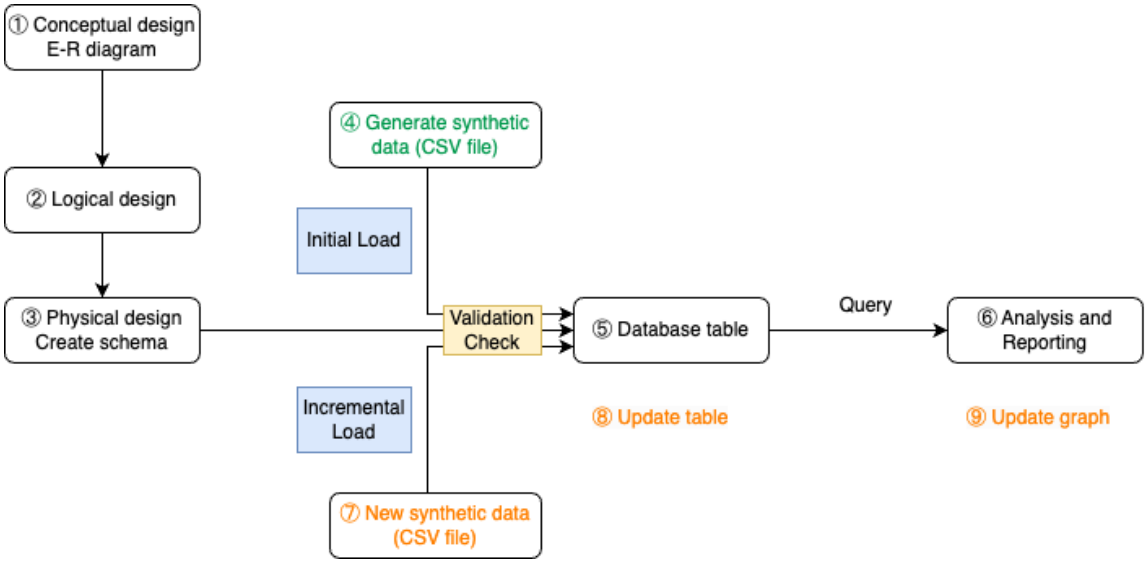
Data Management Group Assignment (IB9HP0)

Group 2

Contents

0.0.1 Executive Summary

This report simulates the end-to-end data management for an e-commerce platform in the US from database design, workflow management to data analysis that provides insights into the platform's key performance metrics.



During the design phase, we started with analysing business objectives to identity key participants (customers, sellers, products, shippers) and types of data that is crucial for insights and the platform's operations. Those elements are conceptually mapped out in an Entity-Relationship (E-R) diagram that is subsequently converted into database schema via logical and physical design steps.

The workflow management automates the process of data validation and importation incorporated rigorous rules and checks to ensure data integrity. If critical checks are failed, the

process is stopped for human intervention. For less severe checks, warnings are given, and errors are captured in error logs while the process resumes.

For the data analysis, we execute queries to retrieve, manipulate, and mine the data stored in the database to extract insights in four aspects of the business: Platform Overview, Sales Performance, Top Products, and Customer Satisfaction.

New data upload will automatically trigger the end-to-end process of validation, importation to analysis with corresponding database and dashboards updated accordingly.

0.0.2 Part 1: Database Design and Implementation

0.0.2.1 Task 1.1: E-R Diagram Design

Conceptual Design

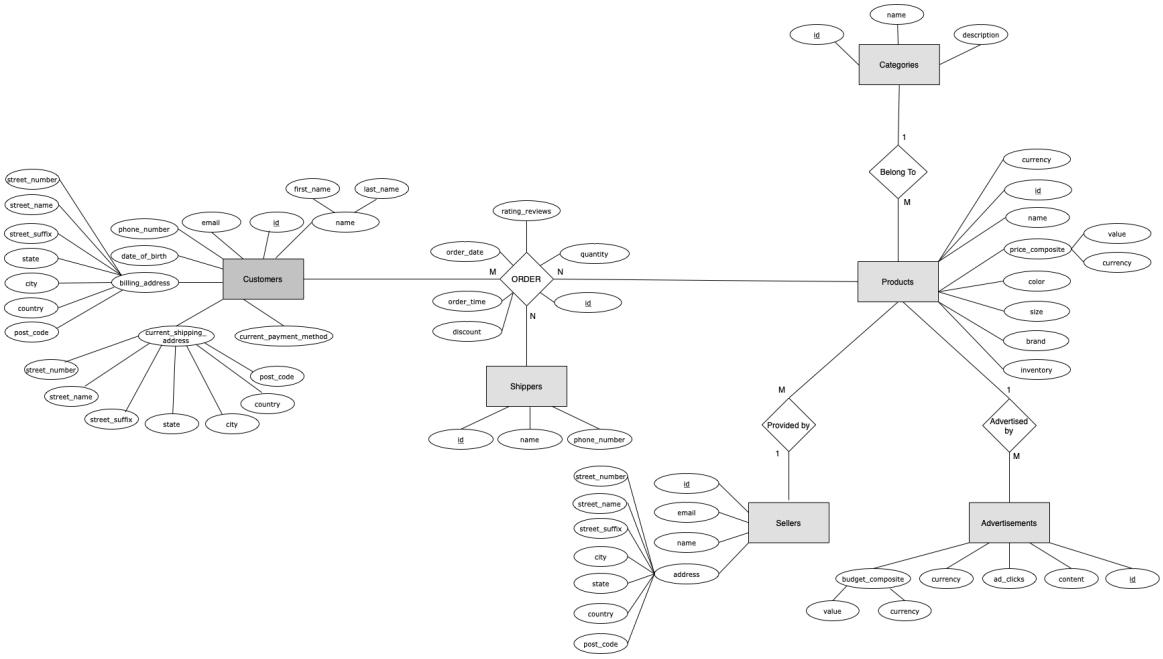


Figure 1: ER Diagram

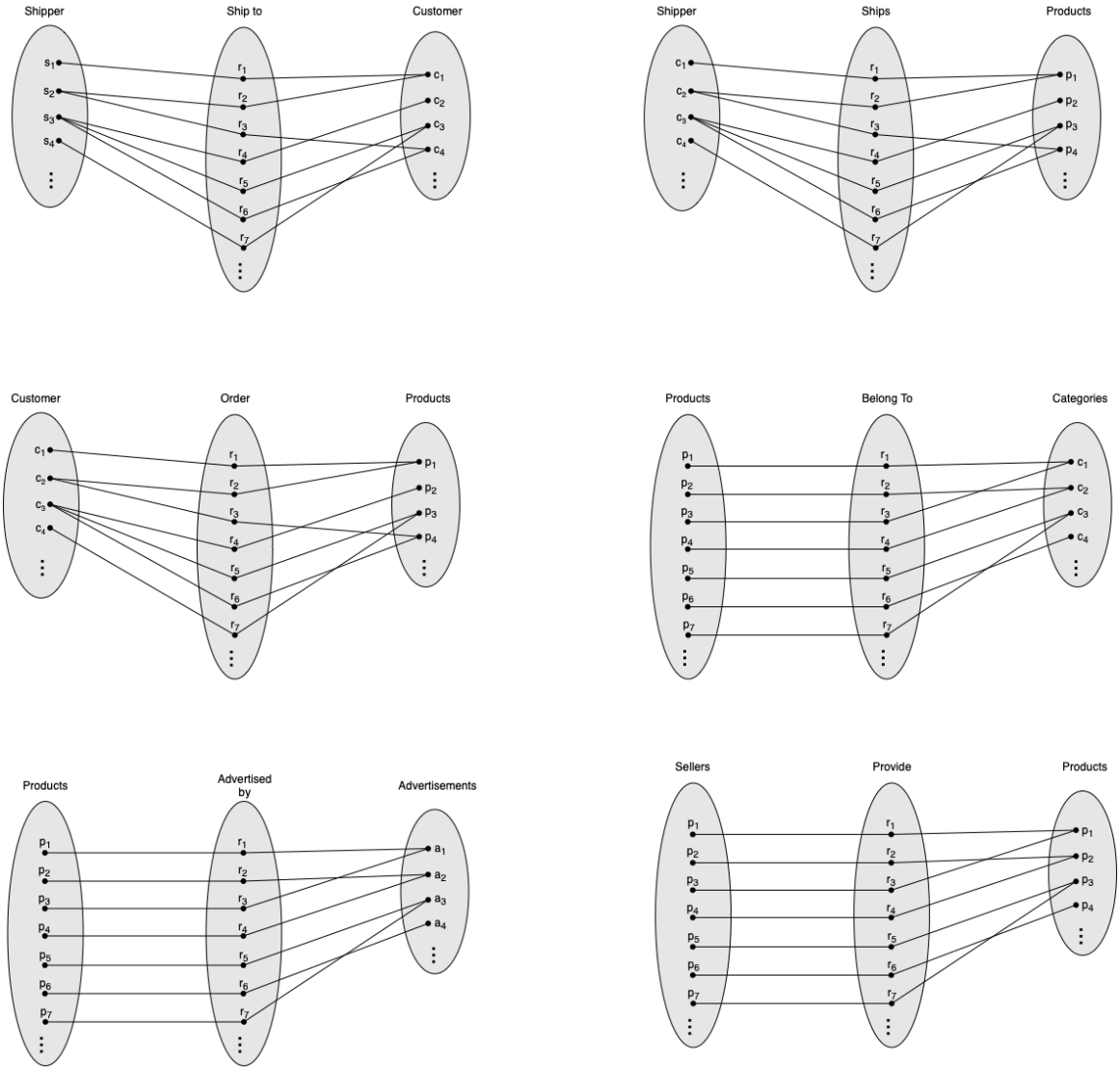


Figure 2: Entity Relationship Sets

Our database conceptual design is based on the following assumptions & justification:

- Customers are unique and identified by their ID. Each customer must have their first name, last name and email address whereas it is optional to provide phone number or date of birth. For simplification, we assume no multivalued attributes; thus, only one billing address, current shipping address and current payment method are captured in the database.
- Products are organised into unique categories, which are identified by category ID. Each product belongs to only one category whilst one category can include multiple products, forming the 1:N relationship between Categories and Products entities.

- Identical items sold by different sellers are considered different products and have unique product ID. Thus, one product can have only one seller, but one seller may distribute multiple products, forming the 1:N relationship between Sellers and Products entities. This conceptual design is important to track price and inventory for each product sold by each seller. Additionally, each product can also have name, colour, size and brand.
- Sellers are unique and identified by their ID. We capture sellers' email, name and address.
- Advertisements (unique and identified by ID) are specific for each product, but one product can have multiple advertisements, forming the 1:N relationship between Products and Advertisements entities. Each advertisement must specify budget and captures content and the number of ad clicks.
- Multiple customers can order multiple products that will be delivered to them by a selection of shippers (e.g., depending on the warehouse location), forming the ternary M:N relationship among the three entities. Each time a customer commits an order of a specific product, which will be delivered by an assigned shipper, a unique order ID will be generated.
- Each order captures quantity, date and time, and discount rate applicable specifically to the order. A customer can only leave a review on a product if he/she has ordered it. Thus, rating review must be specific to an order, forming an attribute of the Orders relationship.
- None of derived attributes are included to ensure physical schema comprise of only atomic values thus in normalised form.

Logical Design

- **Customers** (id, date_of_birth, first_name, last_name, phone_number, email, billing_address_street_number, billing_address_street_name, billing_address_street_suffix, billing_address_city, billing_address_state, billing_address_country, billing_address_postcode, current_shipping_address_street_number, current_shipping_address_street_name, current_shipping_address_street_suffix, current_shipping_address_city, current_shipping_address_state, current_shipping_address_country, current_shipping_address_postcode, current_payment_method)
- **Shippers** (id, name, phone_number)
- **Products** (id, seller_id, category_id, name, color, size, brand, price, currency, inventory)
- **Order** (id, customer_id, product_id, shipper_id, order_date, order_time, quantity, discount, rating_review)
- **Seller** (id, name, email, address_street_number, address_street_name, address_street_suffix, address_city, address_state, address_country, address_postcode)
- **Advertisements** (id, product_id, content, ad_clicks, budget, currency)
- **Categories** (id, name, description)

Figure 3: Logical Design

The logical design of the database is converted from the conceptual ER diagram on the following methodologies:

- Composite attributes (customer name, customer and seller address, product price and advertisement budget) are registered in individual fields using outer layer of the attribute to ensure that physical database only capture atomic value.
- Each entity is converted into one table. Except for the Orders relationship, all other cardinality relationships are 1:N, thus not converted into table.
- The ternary M:N relationship among Customers, Products and Shippers is converted into the Orders table. Primary key (ID) of Customers, Products and Shippers entities, together with order date and time forming a composite key for the Orders relationship table. Each record of the Orders relationship is also unique by an order ID.

0.0.2.2 Task 1.2: SQL Database Schema Creation

Our logical scheme is in Third Normal Form as a result of meticulously abiding to the described principles in the conceptual and logical design phases.

```
# 1. set up the connection
my_db <- RSQLite::dbConnect(RSQLite::SQLite(),"../database/ecommerce.db")
```

```
# 2. link to SQL file to write the schema
sqlite3 "../database/ecommerce.db" < "../main/ecommerce.sql"
```

The code in **ecommerse.sql** file are shown below:

```
DROP TABLE IF EXISTS `customers` ;
DROP TABLE IF EXISTS `products` ;
DROP TABLE IF EXISTS `shippers` ;
DROP TABLE IF EXISTS `orders` ;
DROP TABLE IF EXISTS `advertisements` ;
DROP TABLE IF EXISTS `sellers` ;
DROP TABLE IF EXISTS `categories` ;
```

The table creation sequence starts with tables that do not have foreign keys.

```

------------- CREATE TABLE -------------

-- Customer Schema
CREATE TABLE IF NOT EXISTS `customers` (
  'id' INT PRIMARY KEY,
  'first_name' VARCHAR(250) NOT NULL,
  'last_name' VARCHAR(250) NOT NULL,
  'email' TEXT NOT NULL,
  'phone_number' VARCHAR(20),
  'date_of_birth' DATE,
  'billing_address_street_number' TEXT,
  'billing_address_street_name' TEXT,
  'billing_address_street_suffix' TEXT,
  'billing_address_city' TEXT,
  'billing_address_state' TEXT,
  'billing_address_country' TEXT,
  'billing_address_postcode' TEXT,
  'current_shipping_address_street_number' TEXT,
  'current_shipping_address_street_name' TEXT,
  'current_shipping_address_street_suffix' TEXT,
  'current_shipping_address_city' TEXT,
  'current_shipping_address_state' TEXT,
  'current_shipping_address_country' TEXT,
  'current_shipping_address_postcode' TEXT,
  'current_payment_method' VARCHAR(250)
);

-- Sellers Schema
CREATE TABLE IF NOT EXISTS `sellers` (
```

```
  'id' INT PRIMARY KEY,
  'name' VARCHAR(250) NOT NULL,
  'email' TEXT,
  'address_street_number' TEXT,
  'address_street_name' TEXT,
  'address_street_suffix' TEXT,
  'address_city' TEXT,
  'address_state' TEXT,
  'address_country' TEXT,
  'address_postcode' TEXT
);

-- Categories Schema
CREATE TABLE IF NOT EXISTS `categories` (
  'id' INT PRIMARY KEY,
  'name' VARCHAR(250) NOT NULL,
  'description' TEXT
);

-- Products Schema
CREATE TABLE IF NOT EXISTS `products` (
  'id' INT PRIMARY KEY,
  'seller_id' INT NOT NULL,
  'category_id' INT NOT NULL,
  'name' VARCHAR(60) NOT NULL,
  'color' VARCHAR(60) NOT NULL,
  'size' VARCHAR(5),
  'brand' VARCHAR(250),
  'price' NUMERIC NOT NULL,
  'currency' CHAR(3) NOT NULL,
  'inventory' INT NOT NULL,
  FOREIGN KEY ('seller_id')
    REFERENCES sellers ('id'),
  FOREIGN KEY ('category_id')
    REFERENCES categories ('id')
);

-- Shipper Schema
CREATE TABLE IF NOT EXISTS `shippers` (
  'id' INT PRIMARY KEY,
  'name' CHAR(25) NOT NULL,
  'phone_number' VARCHAR(25) NOT NULL
);

-- Order Schema : create after 3 main tables
```

```sql
CREATE TABLE IF NOT EXISTS `orders` (
  'id' INT PRIMARY KEY,
  'customer_id' INT NOT NULL,
  'product_id' INT NOT NULL,
  'shipper_id' INT NOT NULL,
  'order_date' DATE NOT NULL,
  'order_time' TIMESTAMP NOT NULL,
  'quantity' INT NOT NULL,
  'discount' DECIMAL(3,2) NOT NULL,
  'rating_review' INT,
  FOREIGN KEY ('customer_id')
    REFERENCES customers ('id'),
  FOREIGN KEY ('product_id')
    REFERENCES products ('id'),
  FOREIGN KEY ('shipper_id')
    REFERENCES shippers ('id')
);

-- Ads Schema
CREATE TABLE IF NOT EXISTS `advertisements` (
  'id' INT PRIMARY KEY,
  'product_id' INT NOT NULL,
  'content' TEXT,
  'ad_clicks' INT,
  'budget' DECIMAL(10,2),
  'currency' CHAR(3),
  FOREIGN KEY ('product_id')
    REFERENCES products ('id')
);
```

0.0.2.3 Part 2: Data Generation and Management

0.0.2.4 Task 2.1: Synthetic Data Generation

Synthetic data is generated using Mockaroo leveraging advanced field settings or Mockaroo specified coding language.

Foreign key is imported from other datasets by using field type as 'Dataset Column' then connecting to the other datasets where the foreign key comes from.

Field type as 'Formula' or 'Custom List' with dynamic distribution are utilised to set specific rules and conditions for the synthetic data generation as realistic as possible, for example:

- Product names are conditional to category ID; and
- Product size and price ranges are conditional to product name.

The data generation is a part of the dynamic process from data generation to data validation, then data analysis. Data generation is revised if challenges incur during validation and analysis to ensure that the synthetic data simulates the real-world as much as possible.

0.0.2.5 Task 2.2: Data Import and Quality Assurance

Prior to importing into database, data is validated for both the initial load and additional loads to safeguard data integrity. The validation process is first undertaken on tables representing entities on the weaker side of the cardinality relationship, then tables representing the strong side and ending with tables representing the M:N relationship. Key assurance aspects are as follows:

1. Unique Primary Key: An error message would pop up and the data loading would stop if the number of distinct primary key values is lesser than the number of rows, indicating potential of duplicate primary key values.

2. Foreign Key exists as Primary Key in the associated table: Only records that pass the examination can be uploaded into the database. Otherwise, records are not uploaded and will be captured in error logs.

3. Not Null constraints: Fields including Customer Email, Payment Method, Price and Quantity are crucial for the functioning of the platform, thus cannot have Null values.

4. Other value constraint checks: This involves assuring numeric values fall within acceptable ranges, e.g., Rating Review must be Null or integer ranging from 1-5; Discount ranges between 0-100; Price, Quantity and Budget are positive numerical, and Ad Clicks are non-negative numerical.

5. Format constraint check: This covers various variables with pre-defined format, such as email addresses (*@*), phone numbers (1##########), date (YYYY-MM-DD), and currency (USD). As the e-commerce platform currently concentrates only in the US, currency and phone number are restricted to only being from there.

If any of the values present in the above attributes go against the validation rules, a warning statement would pop up, indicating that specific row should be excluded.

The validations are stored in **Validation.R** file are shown below:

```
print("Performing Validation")

# Get the primary key column names from the database
query <- paste("SELECT name FROM pragma_table_info('",table_name,"')
               WHERE pk = 1;",sep="")
primary_key_columns <- dbGetQuery(my_db, query)

# Get Foreign Key
query <- paste("PRAGMA foreign_key_list('",table_name,"');",sep="")
foreign_key_columns <- dbGetQuery(my_db, query)

```

```r
# ------ 1. Check duplicate primary key within CSV file ------
print(paste0("Checking duplicate primary key for: ",variable))

number_of_rows <- nrow(this_file_contents)

for (i in primary_key_columns) {
  if (nrow(unique(this_file_contents[,i]))==number_of_rows) {
    print(paste("Primary key =",i,": Passed"))
  }
  else {
    stop(paste("Found duplicate record in ", variable,": STOP process!"))
  }
}


# ------ 2. Check data quality and integrity ------
print(paste0("Checking integrity for: ",variable))

# Function to validate email addresses
validate_emails <- function(emails) {
  pattern <- "^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,}$"
  grepl(pattern, emails)
}

# Function to validate phone numbers
validate_phones <- function(phones) {
  # This is a simple example and might not cover all international formats
  pattern <- "^1[0-9]{10}$$"
  grepl(pattern, phones)
}

# Function to validate dates
validate_dates <- function(dates) {
  date_format <- "%Y-%m-%d"
  dates_parsed <- parse_date_time(dates, orders = date_format)
  !is.na(dates_parsed)
}

# Function to validate prices
validate_prices <- function(prices) {
  prices > 0
}

# Function to validate currency codes
validate_currencies <- function(currencies) {
```

```r
  pattern <- "^USD$"
  grepl(pattern, currencies)
}

# Function to validate payment method
validate_payment_method <- function(payment_method){
  valid_method <- c("Apple Pay", "Google Pay", "Credit Card", "Cash",
                    "Debit Card", "Bank Transfer", "Cheque")
  payment_method %in% valid_method
}

# Function to validate ad clicks
validate_ad_clicks <- function(ad_clicks) {
  ad_clicks >= 0
}

# Function to validate discount
validate_discount <- function(discount) {
  (discount >= 0 & discount <= 100)
}

# Function to validate rating_review
validate_rating_review <- function(rating_review) {
  valid_rating <- c(1, 2, 3, 4, 5)
  is.na(rating_review) | rating_review %in% valid_rating
}

# Function error handling
validation <- function(this_file_contents,type,column) {
  tmp_table <- this_file_contents
  if (type == 'Email') {
    tmp_table$valid_format <- validate_emails(column)
  } else if (type == 'Phone_numbers') {
    tmp_table$valid_format <- validate_phones(column)
  } else if (type == 'Dates') {
    tmp_table$valid_format <- validate_dates(column)
  } else if (type == 'Prices' || type == 'Budget' || type == 'Quantity') {
    tmp_table$valid_format <- validate_prices(column)
  } else if (type == 'Currencies') {
    tmp_table$valid_format <- validate_currencies(column)
  }
  else if (type == 'payment_method') {
    tmp_table$valid_format <- validate_payment_method(column)
  }
  else if (type == 'ad_clicks') {
```

```
    tmp_table$valid_format <- validate_ad_clicks(column)
  }
  else if (type == 'discount') {
    tmp_table$valid_format <- validate_discount(column)
  }
  else if (type == 'rating_review') {
    tmp_table$valid_format <- validate_rating_review(column)
  }
  if (nrow(tmp_table) >0) {
    for (i in 1:nrow(tmp_table)){
      tmp_row <- tmp_table[i,]
      if (!tmp_row$valid_format) {
        warning(type," Format of ID: ",tmp_row$id, " in ", variable,
                " is incorrect. Please check." )
      }
    }
  }
  if (all(tmp_table$valid_format) == TRUE){
    print(paste(type," Format: Passed!"))
  }
  tmp_table <- tmp_table[tmp_table$valid_format,] # remove row
  tmp_table <- tmp_table[, !names(tmp_table) %in% "valid_format"]
  return(tmp_table)
}

# Perform integrity check
if (table_name == 'customers' && nrow(this_file_contents) >0) {
  this_file_contents <- validation(this_file_contents,'Email',
                                   this_file_contents$email)
  this_file_contents <- validation(this_file_contents,'Phone_numbers',
                                   this_file_contents$phone_number)
  this_file_contents <- validation(this_file_contents,'payment_method',
                                   this_file_contents$current_payment_method)

} else if (table_name == 'orders' && nrow(this_file_contents) >0) {
  this_file_contents <- validation(this_file_contents,'Dates',
                                   this_file_contents$order_date)
  this_file_contents <- validation(this_file_contents,'discount',
                                   this_file_contents$discount)
  this_file_contents <- validation(this_file_contents,'Quantity',
                                   this_file_contents$quantity)
  this_file_contents <- validation(this_file_contents,'rating_review',
                                   this_file_contents$rating_review)
} else if (table_name == 'products' && nrow(this_file_contents) >0) {
  this_file_contents <- validation(this_file_contents,'Prices',
```

```
                                    this_file_contents$price)
  this_file_contents <- validation(this_file_contents,'Currencies',
                                    this_file_contents$currency)

} else if (table_name == 'categories' && nrow(this_file_contents) >0) {

} else if (table_name == 'sellers' && nrow(this_file_contents) >0) {
  this_file_contents <- validation(this_file_contents,'Email',
                                    this_file_contents$email)

} else if (table_name == 'shippers' && nrow(this_file_contents) >0) {
  this_file_contents <- validation(this_file_contents,'Phone_numbers',
                                    this_file_contents$phone_number)
} else if (table_name == 'advertisements' && nrow(this_file_contents) >0) {
  this_file_contents <- validation(this_file_contents,'Currencies',
                                    this_file_contents$currency)
  this_file_contents <- validation(this_file_contents,'Budget',
                                    this_file_contents$budget)
  this_file_contents <- validation(this_file_contents,'ad_clicks',
                                    this_file_contents$ad_clicks)
}

# ------ 3. Check Foreign key ------
if(nrow(this_file_contents) >0) {
  foreign_table <- foreign_key_columns[,'table']
  tmp_table <- this_file_contents
  for (i in foreign_table) {
    foreign_key_ori_column <- foreign_key_columns[
      foreign_key_columns[,'table'] == i,'from']
    foreign_key_dest_column <- foreign_key_columns[
      foreign_key_columns[,'table'] == i,'to']
    print(paste("Checking Foreign key in table",i,
                "column:",foreign_key_ori_column))
    for (j in 1:nrow(this_file_contents)) {
      foreign_key_value <- this_file_contents[j,foreign_key_ori_column]
      query <- paste("SELECT", foreign_key_dest_column," FROM",i," WHERE",
                     foreign_key_dest_column,"=", foreign_key_value, ";")
      result <- dbGetQuery(my_db, query)
      col <- paste("check_",i,sep="")
      if (nrow(result) == 0) {
        warning("Foreign key is missing in row ID =  ",
                this_file_contents[j,primary_key_columns[1,]], " Please check.")
        tmp_table[[col]] <- FALSE
      } else {
        tmp_table[[col]] <- TRUE
```

```
      }
    }
  }
  rows_to_remove <- apply(tmp_table[, grepl("^check", names(tmp_table))], 1,
                          function(row) any(!row))
  tmp_table <- tmp_table[, !grepl("^check", names(tmp_table))] # remove column
  tmp_table <- tmp_table[!rows_to_remove, ] # remove failed row
  this_file_contents <- tmp_table
} else{
  print("No validation check in this table since there's no foreign key")
}

print("Validation Completed")
```

The initial load has been performed.

```
# Get only Initial file which has the format [table_name].csv
all_files <- setdiff(list.files("../data_upload/"),
                     list.files("../data_upload/", pattern = "_"))
# Order the files to load to database, to avoid error from foreign key
custom_order <- list("customers.csv","sellers.csv","categories.csv",
                     "products.csv","shippers.csv","orders.csv",
                     "advertisements.csv")
all_files <- all_files[order(match(all_files, custom_order))]

for (variable in all_files) {
  this_filepath <- paste0("../data_upload/",variable)
  this_file_contents <- readr::read_csv(this_filepath)

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

  # Perform Validation
  source("../main/Validation.R")

  # convert column date format
  if (table_name == 'orders') {
    this_file_contents['order_date'] <- lapply(this_file_contents['order_date'],
                                               as.character)
  }

  if (nrow(this_file_contents)>0 ){
      for (i in 1:nrow(this_file_contents)) {
        row <- this_file_contents[i, ]

        # Extract primary key values from the row
```

```
        primary_key_values <- paste(names(row)[names(row) %in%
                                               primary_key_columns],
                                    row[names(row) %in%
                                          primary_key_columns],
                                    sep = "=", collapse = " AND ")

        # Find if the primary key exists
        query <- paste("SELECT * FROM", table_name, paste("WHERE",
                                                          primary_key_values))
        existing_row <- dbGetQuery(my_db, query)

        if (nrow(existing_row) == 0) {
          # Row is unique, append to the table
          #print(paste("Append:",primary_key_values))
          dbWriteTable(my_db, table_name, row, append = TRUE)
        } else {
          # Row already exists, update the existing row
          #print(paste("Update:",primary_key_values))
          update_query <- paste("UPDATE", table_name,
                                paste("SET", paste(names(row), "=",
                                                   paste0("'", row, "'"),
                                                   collapse = ", "),
                                      "WHERE", primary_key_values))
          dbExecute(my_db, update_query)
        }
      }
    }
    else {
      print("Nothing to update in database since all
            rows are not pass the validations")
    }
}
```

```
Rows: 50 Columns: 21
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (17): first_name, last_name, email, date_of_birth, billing_address_stree...
dbl  (4): id, phone_number, billing_address_street_number, current_shipping_...

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

[1] "Performing Validation"
[1] "Checking duplicate primary key for: customers.csv"
```

```
[1] "Primary key = id : Passed"
[1] "Checking integrity for: customers.csv"
[1] "Email  Format: Passed!"
[1] "Phone_numbers  Format: Passed!"
[1] "payment_method  Format: Passed!"
[1] "Validation Completed"


Rows: 50 Columns: 10
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (7): name, email, address_street_name, address_street_suffix, address_ci...
dbl (3): id, address_street_number, address_postcode

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


[1] "Performing Validation"
[1] "Checking duplicate primary key for: sellers.csv"
[1] "Primary key = id : Passed"
[1] "Checking integrity for: sellers.csv"
[1] "Email  Format: Passed!"
[1] "Validation Completed"


Rows: 10 Columns: 3
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (2): name, description
dbl (1): id

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


[1] "Performing Validation"
[1] "Checking duplicate primary key for: categories.csv"
[1] "Primary key = id : Passed"
[1] "Checking integrity for: categories.csv"
[1] "Validation Completed"


Rows: 500 Columns: 10
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (5): name, color, size, brand, currency
dbl (5): id, seller_id, category_id, price, inventory
```

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

[1] "Performing Validation"
[1] "Checking duplicate primary key for: products.csv"
[1] "Primary key = id : Passed"
[1] "Checking integrity for: products.csv"
[1] "Prices  Format: Passed!"
[1] "Currencies  Format: Passed!"
[1] "Checking Foreign key in table categories column: category_id"
[1] "Checking Foreign key in table sellers column: seller_id"
[1] "Validation Completed"

Rows: 50 Columns: 3
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (1): name
dbl (2): id, phone_number

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

[1] "Performing Validation"
[1] "Checking duplicate primary key for: shippers.csv"
[1] "Primary key = id : Passed"
[1] "Checking integrity for: shippers.csv"
[1] "Phone_numbers  Format: Passed!"
[1] "Validation Completed"

Rows: 1000 Columns: 9
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl  (7): id, customer_id, product_id, shipper_id, quantity, discount, ratin...
date (1): order_date
time (1): order_time

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

[1] "Performing Validation"
[1] "Checking duplicate primary key for: orders.csv"
[1] "Primary key = id : Passed"
[1] "Checking integrity for: orders.csv"
[1] "Dates  Format: Passed!"
```

```
[1] "discount  Format: Passed!"
[1] "Quantity  Format: Passed!"
[1] "rating_review  Format: Passed!"
[1] "Checking Foreign key in table shippers column: shipper_id"
[1] "Checking Foreign key in table products column: product_id"
[1] "Checking Foreign key in table customers column: customer_id"
[1] "Validation Completed"


Rows: 1000 Columns: 6
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (2): content, currency
dbl (4): id, product_id, ad_clicks, budget

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


[1] "Performing Validation"
[1] "Checking duplicate primary key for: advertisements.csv"
[1] "Primary key = id : Passed"
[1] "Checking integrity for: advertisements.csv"
[1] "Currencies  Format: Passed!"
[1] "Budget  Format: Passed!"
[1] "ad_clicks  Format: Passed!"
[1] "Checking Foreign key in table products column: product_id"
[1] "Validation Completed"
```

0.0.3 Part 3: Data Pipeline Generation

0.0.3.1 Task 3.1: GitHub Repository and Workflow Setup

We first create a new repository for our project on GitHub (https://github.com/pomprodpran/DM-Group-2) This enables collaboration within our group, allowing everyone to work together and make changes to the project through GitHub. After cloning the repository locally, we added our database file, e-commerce data file, report file, and scripts for data validation, transformation, data analysis, and data visualisation to the repository.

Within the '**database**' file, an e-commerce database would be created, and in the '**data upload**' file, we upload all the generated data including advertisements, categories, customers, products, sellers, and shippers table. Within the '**main**' file, we have schema creation, data analysis **visualisations** file, **transformation** and **validation** scripts.

0.0.3.2 Task 3.2: GitHub Actions for Continuous Integration

To set up the workflow, we use the GitHub Actions part to create a new workflow file, then define the workflow as **etl.yaml**.

For the workflow, we specify the actions as follows:

1. Specify that the workflow should run every 2 hours or when changes are pushed to the main branch.

2. Define and build the job (sequence of tasks) to be executed.

3. Check out the code repository into the GitHub Actions runner.

4. Set up the R environment and cache R packages.

5. Install all the packages that we will use.

6. Execute the data **transformation** script from the main directory, checking for data quality and integrity of the new data that we update. (After data transformation and validation, it will trigger the running of the data analysis script, and the new analysis charts will be saved in the folder.)

```r
# Transformation.R
library(readr)
library(RSQLite)
library(dplyr)
library(ggplot2)
library(lubridate)

# Incremental Load
print("Loading CSV file")

# File format for automation: <table name>_YYYY-MM-DDTHHMMSS.csv
current_date <- Sys.Date()
print(paste("current date:", current_date))
# Get only Incremental file
all_files <- list.files("./data_upload", full.names = FALSE, pattern = "_")
for (variable in all_files) {
  # split file name using _ separator
  file_name <- unlist(strsplit(gsub(".csv","",variable), "_"))
  table_name <- file_name[1]
  # Splitting file name using 'T' separator
  date_time_parts <- unlist(strsplit(file_name[2], "T"))
  date_str <- date_time_parts[1]  # Date string
  time_str <- date_time_parts[2]  # Time string
  # Parsing date strings into datetime objects using lubridate
  date_value <- lubridate::ymd(date_str)

```

```r
  # Get only NEW file that has been loaded into the folder
  # (and run historical back 2 days)
  if (date_value>= current_date-1 && date_value<= current_date ) {
    print(paste("Reading file:",variable))
    this_filepath <- paste0("./data_upload/",variable)
    this_file_contents <- readr::read_csv(this_filepath)

    print(paste("Writing table to database:", table_name))
    my_db <- RSQLite::dbConnect(RSQLite::SQLite(),"./database/ecommerce.db")

    # Perform Validation
    source("./main/Validation.R")

    # convert column date format
    if (table_name == 'orders') {
      this_file_contents['order_date'] <- lapply(
        this_file_contents['order_date'], as.character)
    }

    # Validation and Writing on each row to DB
    if (nrow(this_file_contents)>0 ){
      for (i in 1:nrow(this_file_contents)) {

        row <- this_file_contents[i, ]

        # Extract primary key values from the row
        primary_key_values <- paste(names(row)[names(row) %in%
                                                 primary_key_columns],
                                    row[names(row) %in% primary_key_columns],
                                    sep = "=", collapse = " AND ")

        # Find if the primary key exists
        query <- paste("SELECT * FROM", table_name,
                       paste("WHERE", primary_key_values))
        existing_row <- dbGetQuery(my_db, query)

        if (nrow(existing_row) == 0) {
          # Row is unique, append to the table
          print(paste("Append:",primary_key_values))
          dbWriteTable(my_db, table_name, row, append = TRUE)
        } else {
          # Row already exists, update the existing row
          print(paste("Update:",primary_key_values))
          update_query <- paste("UPDATE", table_name,
                                paste("SET", paste(names(row), "=",
```

```
                                                    paste0("'", row, "'"),
                                                    collapse = ", "),
                                          "WHERE", primary_key_values))
          dbExecute(my_db, update_query)
        }
      }
    }
    else {
      print("Nothing to update in database since all rows
            are not pass the validations")
    }
  }
}

# Check if the connection object exists and is valid
if (exists("my_db") && RSQLite::dbIsValid(my_db)) {
  # Perform Visualisation
  source("./main/Visualisation.R")
  print("Done!")
  # Disconnect from the database
  RSQLite::dbDisconnect(my_db)
} else {
  # Print a message where the connection object is not found or invalid
  print("Connection object not found or is invalid.")
}
```

When performing incremental loading, the new loaded CSV file format for automation needs to be '[table name]_YYYY-MM-DDTHHMMSS.csv' to read in the data.

Only the newly added files within the past two days will be processed, following the thorough validation process described in Part 2.2.

7. Configure the Git user email and name, adding all files in the database directory to the Git staging area.

8. Finally, commit and push the changes to the main branch.

0.0.4 Part 4: Data Analysis and Reporting with Quarto in R

In accordance with the CRISP-DM, data analysis is designed to assist management with their data-driven decision making on multi-dimensional facets of the business. An effective data analysis should allow management to direct their investments and resources into the areas with most growth and profitability, whilst mitigating potential risks. With that objective in mind, our data analysis is organised into the following key elements of the business:

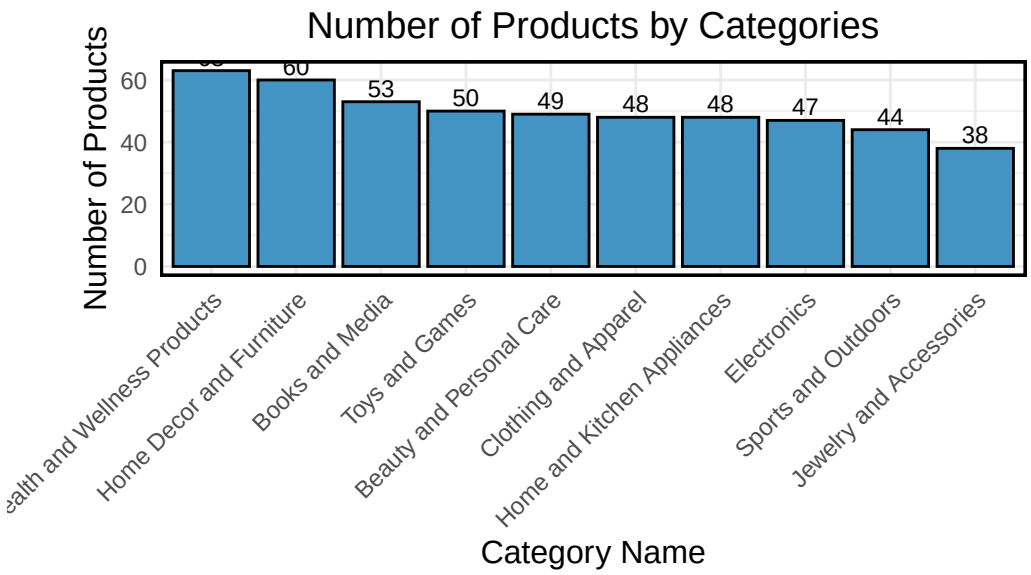
Platform Overview:Designed to provide the management with an overview of the platform participants (customers, sellers, and products), this dashboard addresses fundamental questions on the top selling product categories, customer traffics via ad clicks, and customers' and sellers' geographical location. This aids management in optimising the distribution network, merchandising strategy (i.e., which product to sells) and network expansion. We observe that the Health & Wellness product is the most popular category reflecting the shift in consumers attention towards self-care and wellbeing post the COVID-19 pandemic. Low participants in the Central region may result from the company's deliberate strategy or flash an opportunity for expansion.

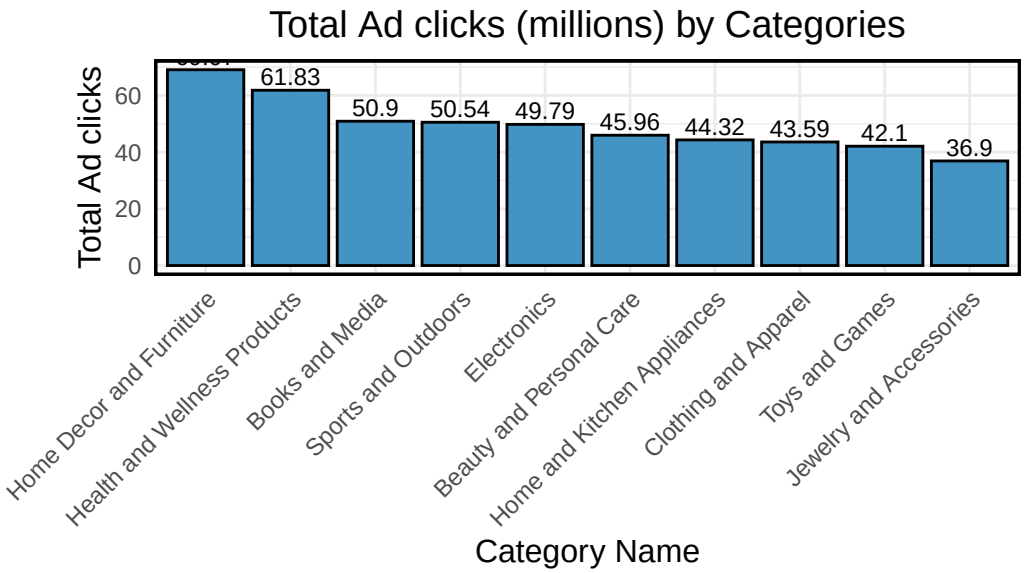
```
[1] 0
```

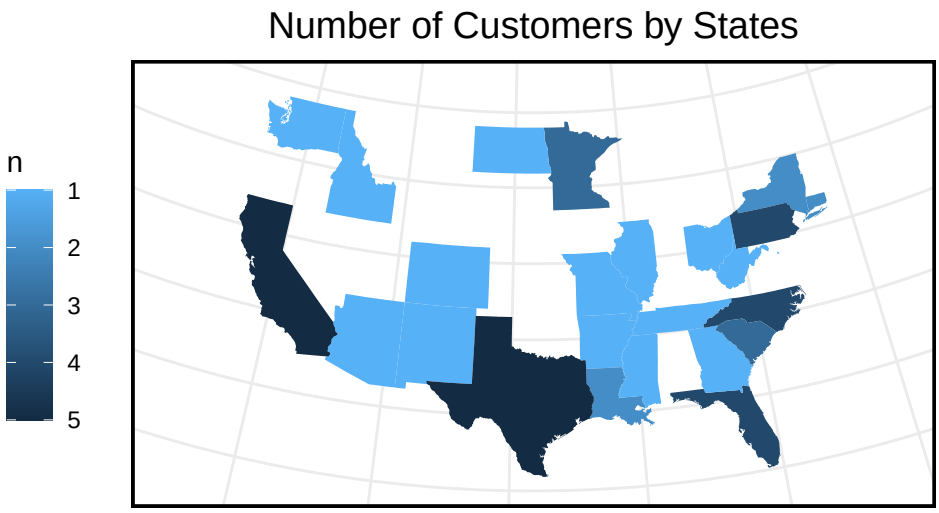
```
[1] 0
```

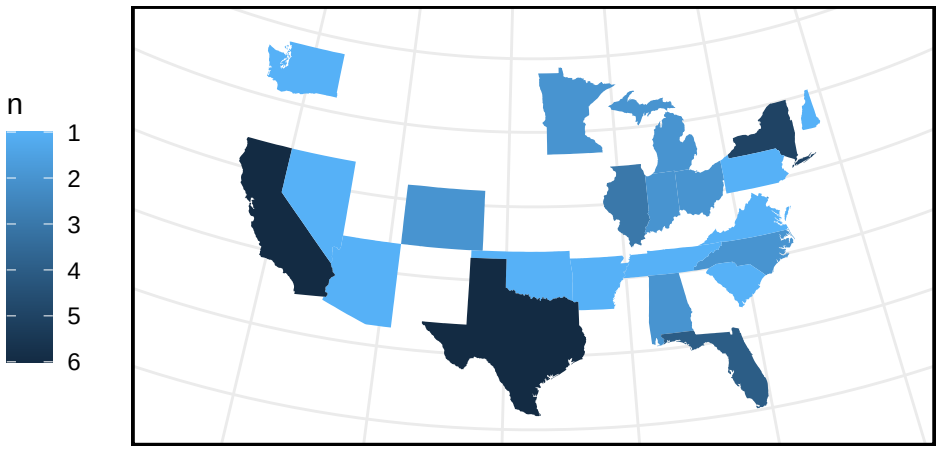
```
[1] 0
```

```
[1] 0
```



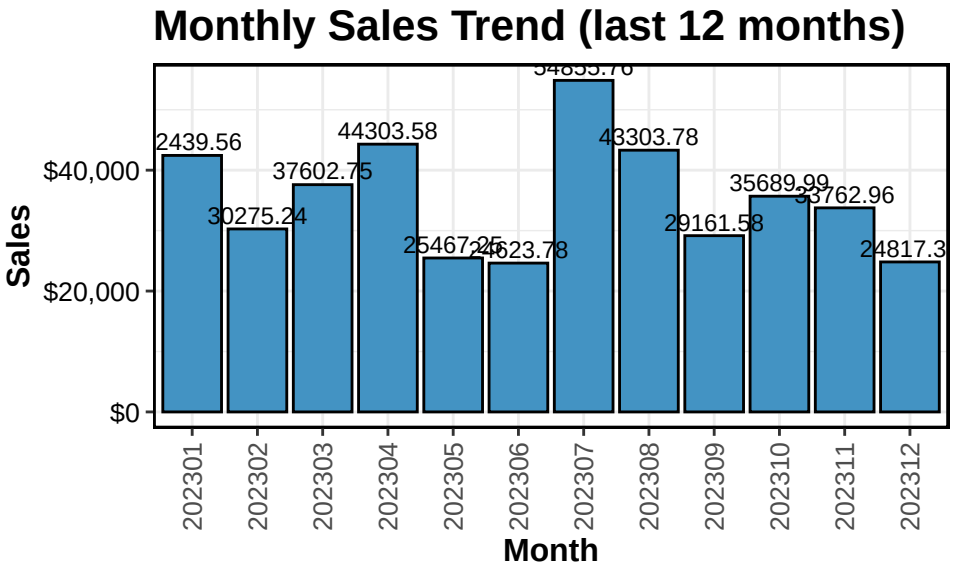






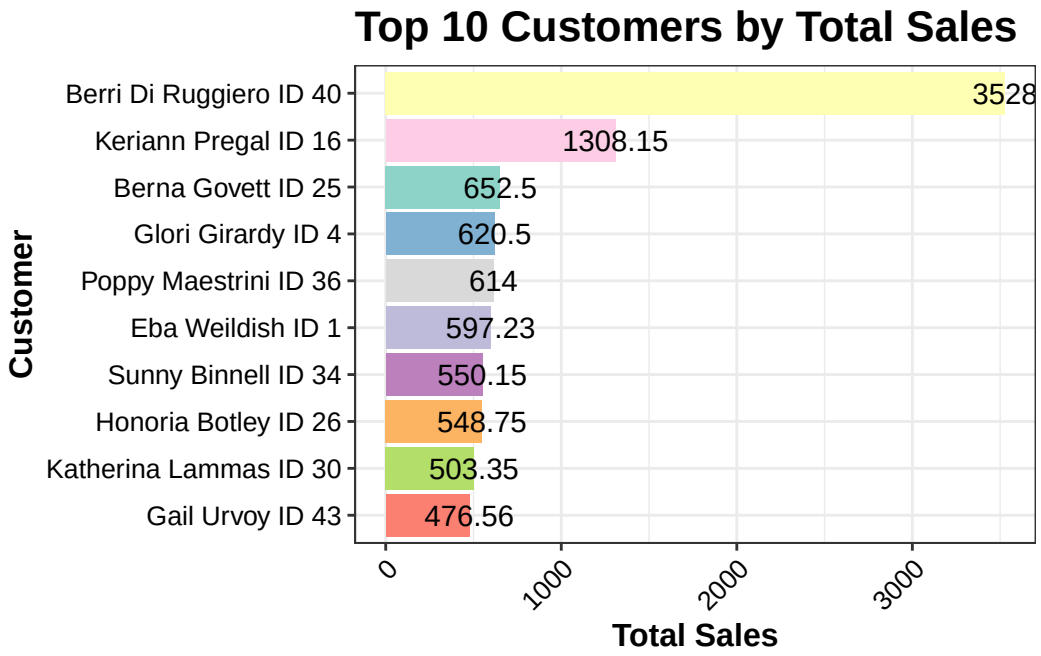
Sales Performance: The dashboard provides key metrics into sales performance and

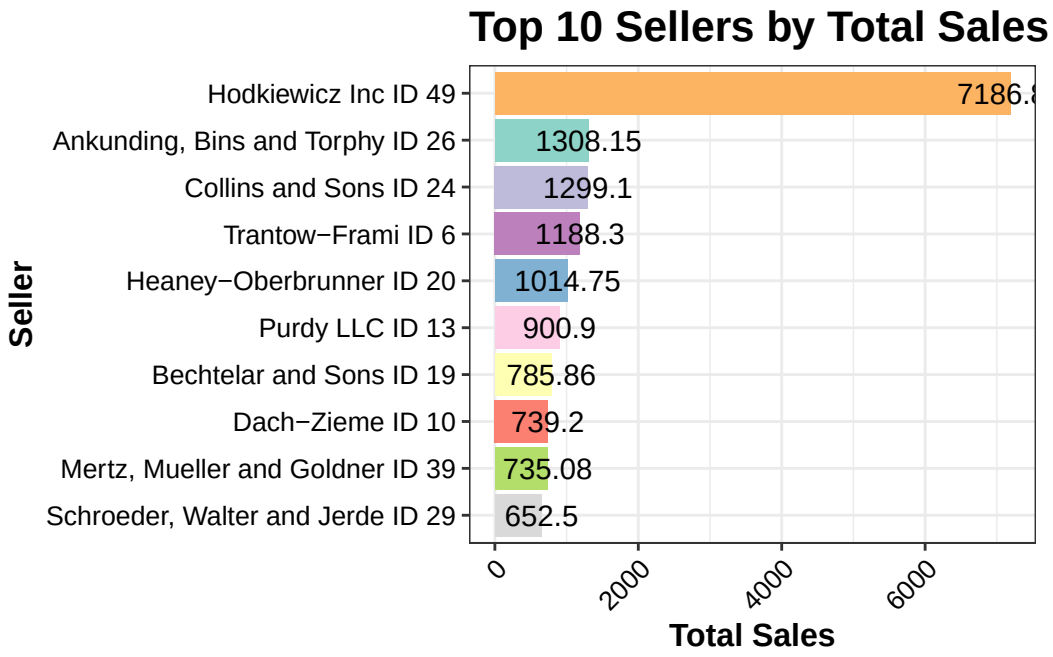
growth potential of the platform. Monthly Sales Trend tracks the live results in the last 12 months, timely highlighting the effectiveness of management's strategies and day-to-day operations. In contrast, the remaining charts capture a snapshot of the overall performance to date, drawing management's attention to the most profitable customers, sellers, and product category. The concentration risk presenting in the top customer and seller of this platform, whilst should be mindful of, indicates growth opportunities on the remaining customers and seller groups.



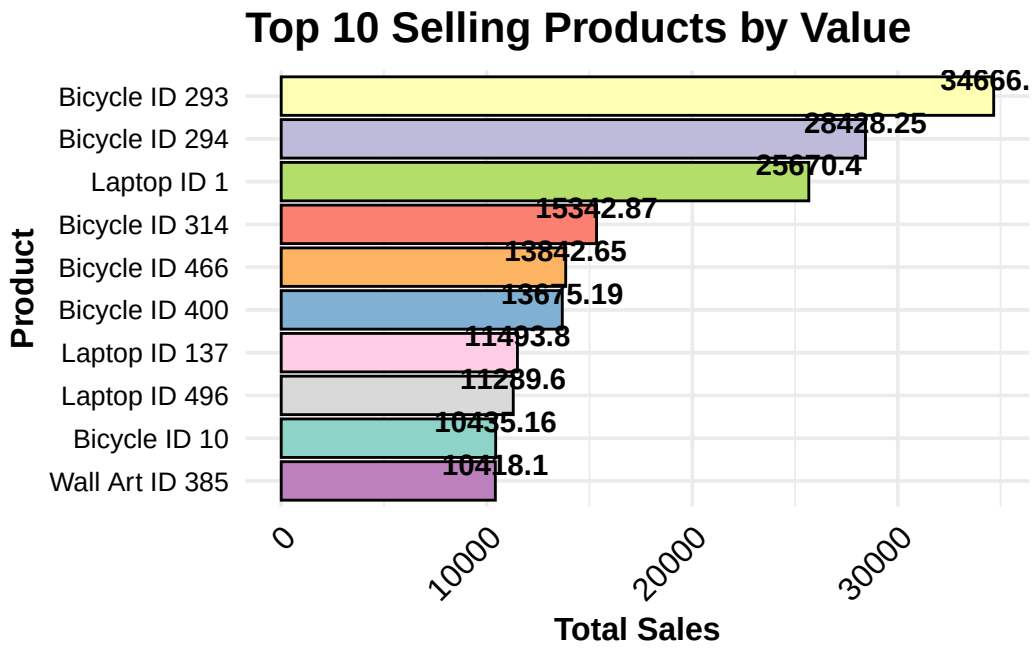
Sales by Categories

Sales values are in USD

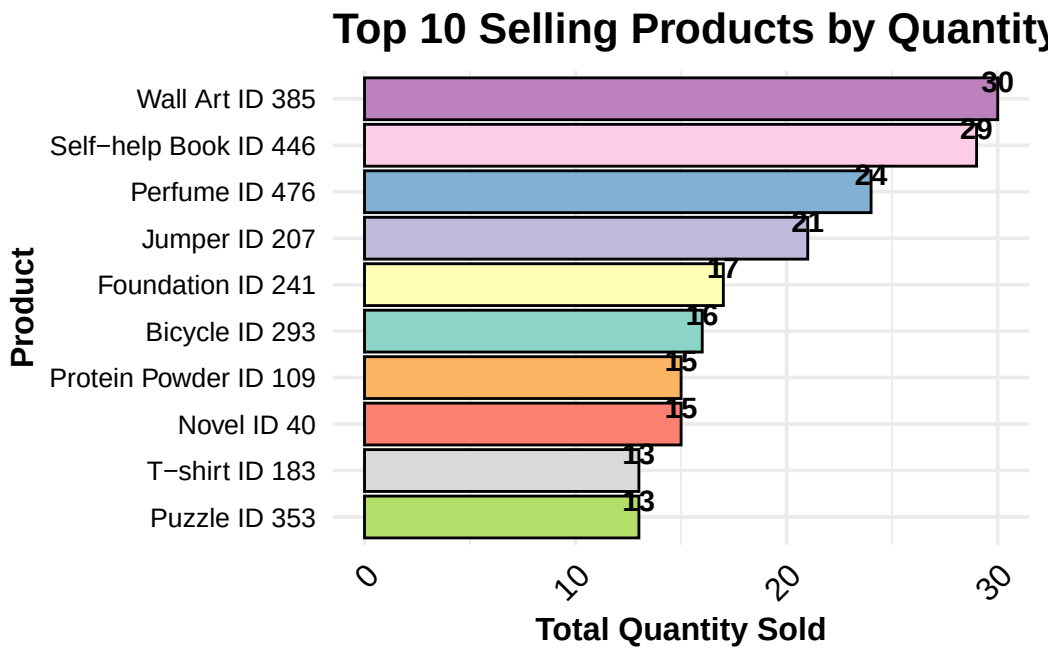


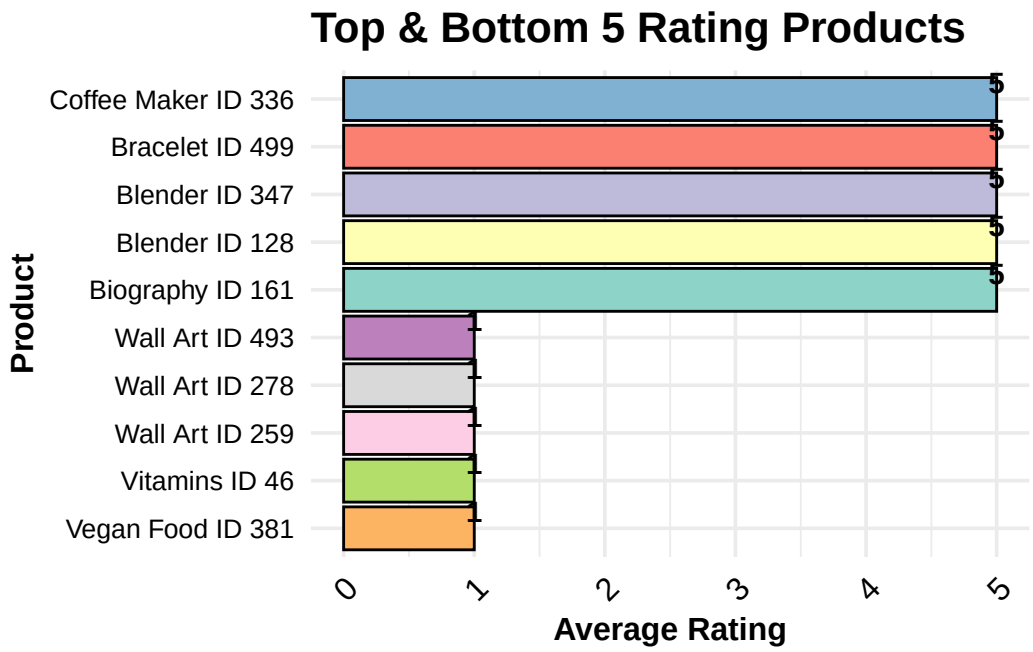


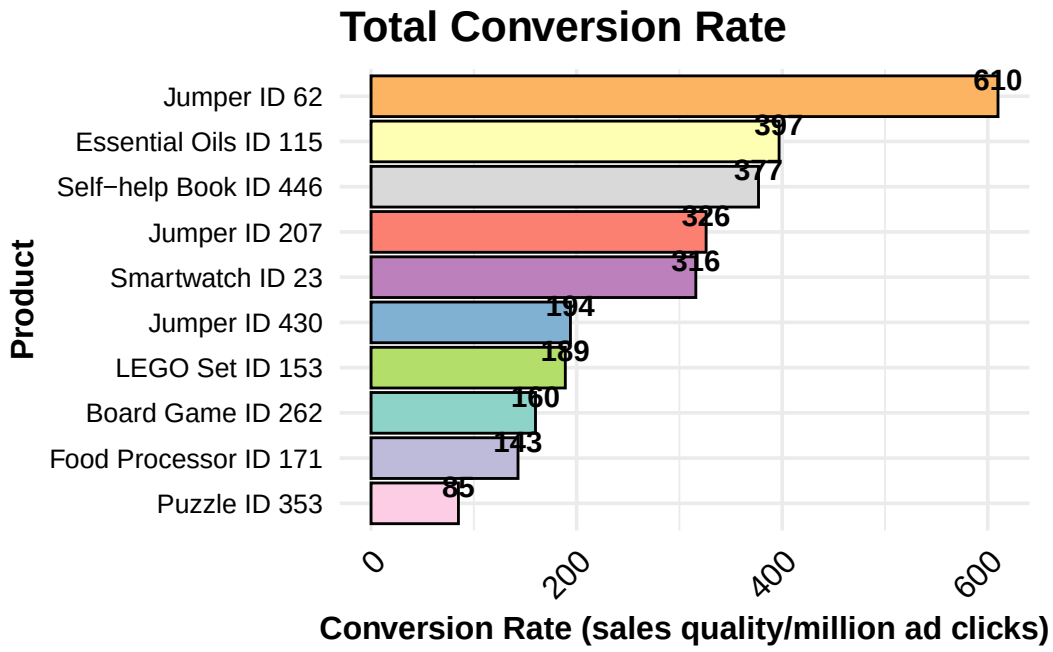
Top Products: Drilling down into the product level, this showcases the top performing products by various metrics: sales value, sales quantum, customer rating and conversion rate (measured by sales quantum per a million of ad clicks), spearheading the platform's marketing, merchandising, and acquisition strategy. Whilst not being the top selling products by quantity, bicycle and laptop generates the most revenue due to its high value propositions.



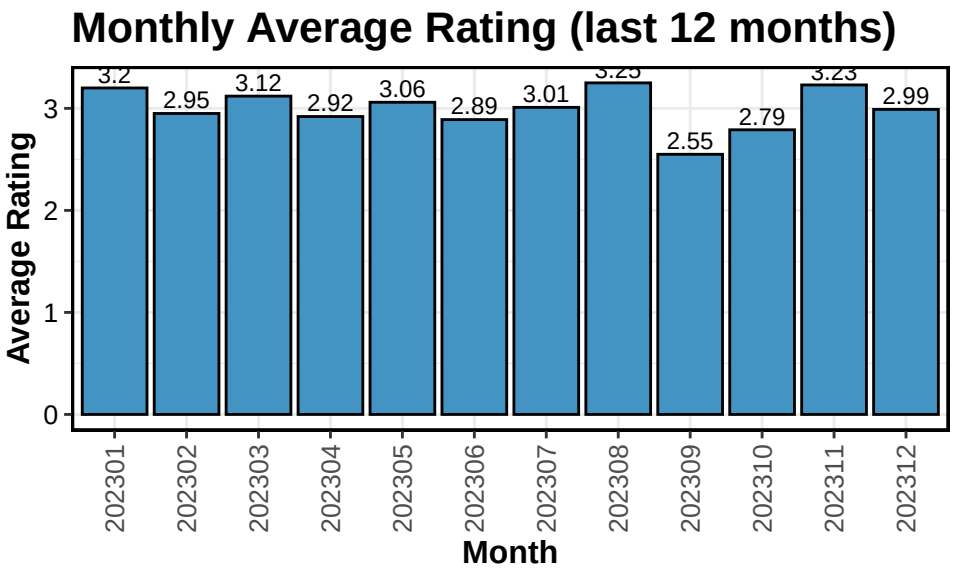

```
Scale for fill is already present.
Adding another scale for fill, which will replace the existing scale.
```







Customer Satisfaction: While average rating reflects customer satisfaction, the percentage of orders left without review indicates customer engagement level. With a high-fixed-cost and low-variable-cost business model, an e-commerce platform depends profoundly on the ability to sustainability grow and expand their customer base. Therefore, these metrics are considered pivotal to their success. Downward trending average rating and increasing percentage of nil review, as in this case, is alarming and requires management's immediate attention.

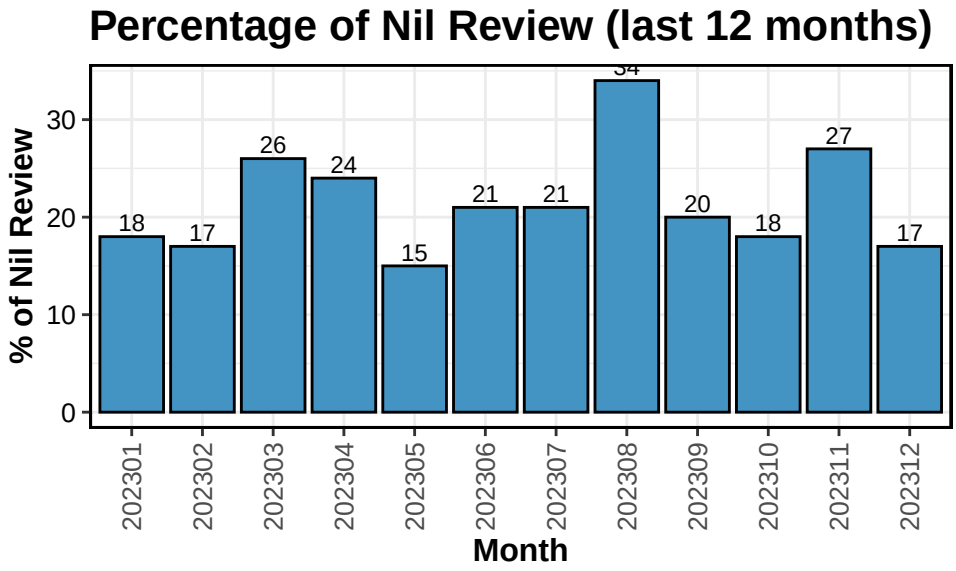



```
## Figure 14: Percentage of Nil Rating

rating_all <- dbGetQuery(my_db,
                         "SELECT id, product_name, order_date, rating_review, sales
  FROM df_sales")
```

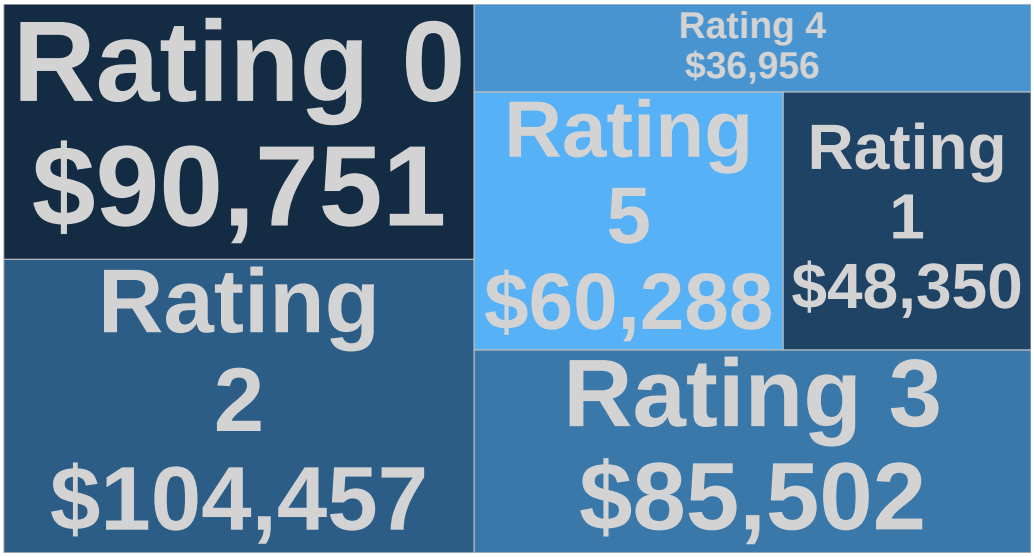
```
Warning: Column `rating_review`: mixed type, first seen values of type integer,
coercing other values of type string
```

```
# Convert order_date to date format
rating_all$order_date <- as.Date(as.character(rating_all$order_date),
                                 format = "%Y-%m-%d")
rating_all <- rating_all %>%
  mutate(year_month = gsub('-','',
                           as.character(format(as.Date(order_date), "%Y-%m"))))

# Calculate total number of orders per months
rating_all_summary <- rating_all %>% group_by(year_month) %>% summarise (n_all = n())

# Filter no rating and convert date format
rating_n <- rating_all %>% filter(rating_review == 0)

# Calculate number of orders with no review
rating_n_summary <- rating_n %>% group_by(year_month) %>% summarise (n_n = n())

# Combine data
rating_n_summary <- merge(rating_all_summary, rating_n_summary)

```

```r
# Calculate nil review rate
rating_n_summary <- rating_n_summary %>%
  mutate(nil_review_rate = round(n_n *100/n_all),2) %>%
  arrange(desc(year_month))

# Take last 12 months
rating_n_summary <- head(rating_n_summary,12)

# Plot monthly sales trend with advanced visualization
(figure.14 <- ggplot(rating_n_summary, aes(x = year_month, y = nil_review_rate)) +
  geom_bar(stat = "identity", fill = "#4393C3", color = "black") +
  labs(title = "Percentage of Nil Review (last 12 months)", x = "Month",
       y = "% of Nil Review") +
  theme_bw() +
  theme(axis.text.y = element_text(size = 10, color = "black"),
        axis.title = element_text(size = 12, color = "black", face = "bold"),
        plot.title = element_text(size = 16, color = "black", face = "bold"),
        legend.position = "none",
        panel.border = element_rect(color = "black", fill = NA, size = 1),
        plot.margin = margin(t = 0.5, r = 0.5, b = 1, l = 1, unit = "cm"),
        axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 1, size = 10)) +
  geom_text(aes(label = nil_review_rate), vjust = -0.3, size = 3, color = "black"))
```



Sales by Rating



Sales values are in USD

Monthly Average Discount (last 12 months)