

Data Management Group Assignment (IB9HP0)

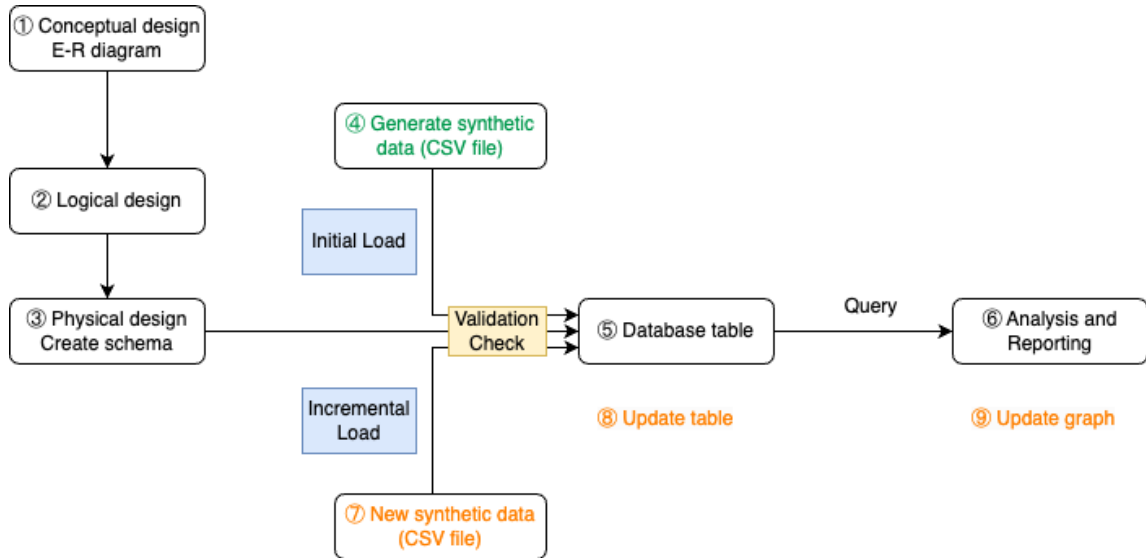
Group 2

Contents

1	Executive Summary	1
2	Part 1: Database Design and Implementation	2
2.1	Task 1.1: E-R Diagram Design	2
2.2	Task 1.2: SQL Database Schema Creation	6
3	Part 2: Data Generation and Management	9
3.1	Task 2.1: Synthetic Data Generation	9
3.2	Task 2.2: Data Import and Quality Assurance	9
4	Part 3: Data Pipeline Generation	19
4.1	Task 3.1: GitHub Repository and Workflow Setup	19
4.2	Task 3.2: GitHub Actions for Continuous Integration	19
5	Part 4: Data Analysis and Reporting with Quarto in R	22
6	Conclusion	32

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.

2 Part 1: Database Design and Implementation

2.1 Task 1.1: E-R Diagram Design

Conceptual Design

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

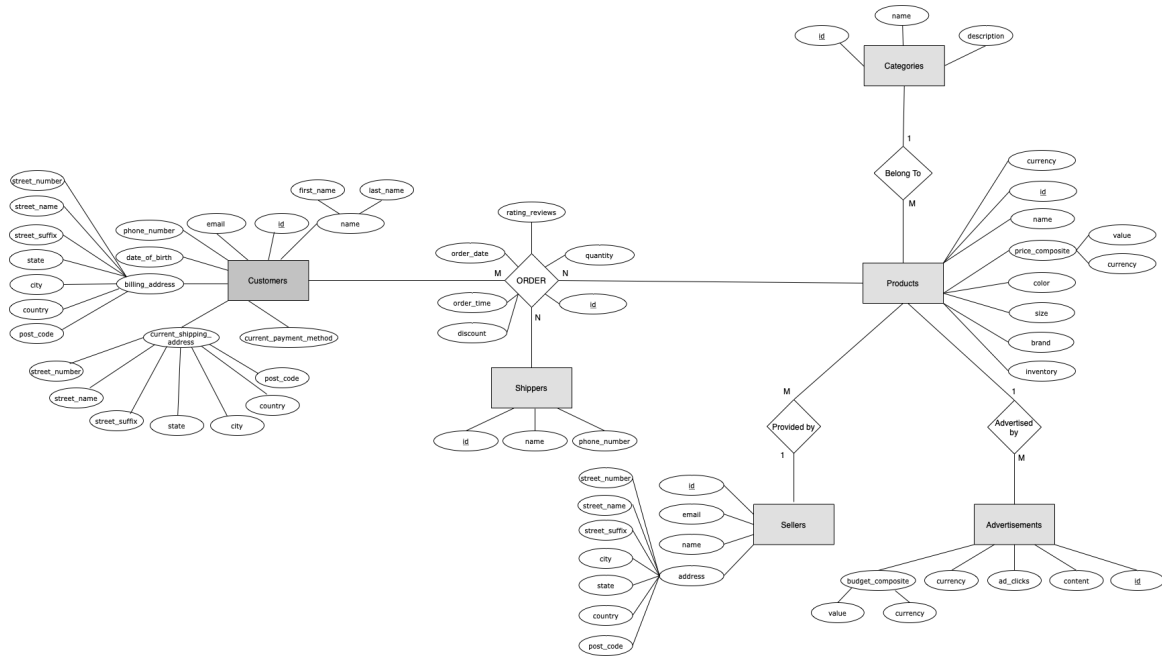


Figure 1: ER Diagram

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.

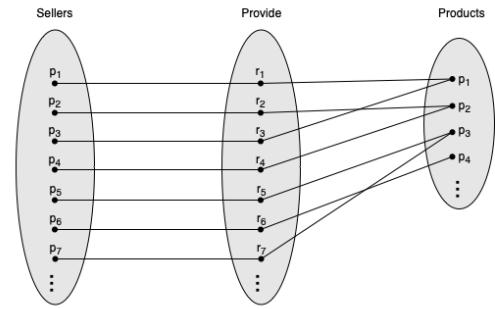
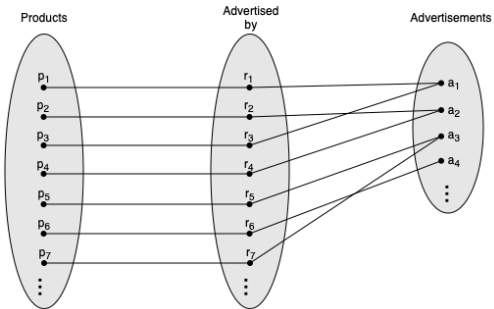
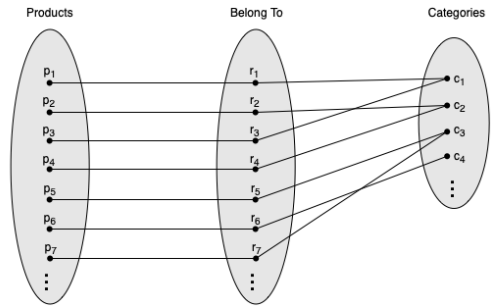
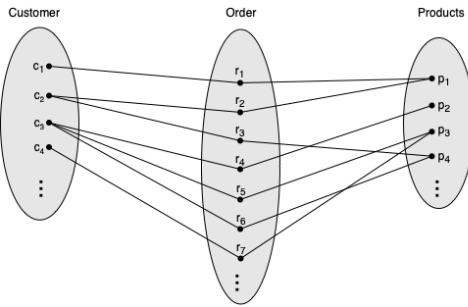
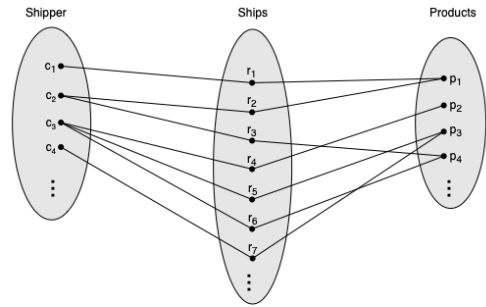
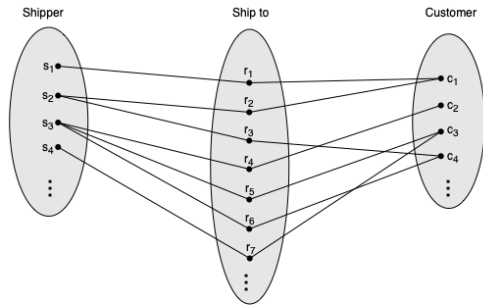


Figure 2: Entity Relationship Sets

- 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.

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

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

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

```
1 DROP TABLE IF EXISTS `customers` ;
2 DROP TABLE IF EXISTS `products` ;
3 DROP TABLE IF EXISTS `shippers` ;
4 DROP TABLE IF EXISTS `orders` ;
5 DROP TABLE IF EXISTS `advertisements` ;
6 DROP TABLE IF EXISTS `sellers` ;
7 DROP TABLE IF EXISTS `categories` ;
```

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

```
1
2 ----- CREATE TABLE -----
3
4 -- Customer Schema
5 CREATE TABLE IF NOT EXISTS `customers` (
6   'id' INT PRIMARY KEY,
7   'first_name' VARCHAR(250) NOT NULL,
8   'last_name' VARCHAR(250) NOT NULL,
9   'email' TEXT NOT NULL,
10  'phone_number' VARCHAR(20),
11  'date_of_birth' DATE,
12  'billing_address_street_number' TEXT,
13  'billing_address_street_name' TEXT,
14  'billing_address_street_suffix' TEXT,
15  'billing_address_city' TEXT,
16  'billing_address_state' TEXT,
17  'billing_address_country' TEXT,
18  'billing_address_postcode' TEXT,
19  'current_shipping_address_street_number' TEXT,
20  'current_shipping_address_street_name' TEXT,
21  'current_shipping_address_street_suffix' TEXT,
22  'current_shipping_address_city' TEXT,
```

```

23     'current_shipping_address_state' TEXT,
24     'current_shipping_address_country' TEXT,
25     'current_shipping_address_postcode' TEXT,
26     'current_payment_method' VARCHAR(250)
27 );
28
29 -- Sellers Schema
30 CREATE TABLE IF NOT EXISTS `sellers` (
31     'id' INT PRIMARY KEY,
32     'name' VARCHAR(250) NOT NULL,
33     'email' TEXT,
34     'address_street_number' TEXT,
35     'address_street_name' TEXT,
36     'address_street_suffix' TEXT,
37     'address_city' TEXT,
38     'address_state' TEXT,
39     'address_country' TEXT,
40     'address_postcode' TEXT
41 );
42
43 -- Categories Schema
44 CREATE TABLE IF NOT EXISTS `categories` (
45     'id' INT PRIMARY KEY,
46     'name' VARCHAR(250) NOT NULL,
47     'description' TEXT
48 );
49
50 -- Products Schema
51 CREATE TABLE IF NOT EXISTS `products` (
52     'id' INT PRIMARY KEY,
53     'seller_id' INT NOT NULL,
54     'category_id' INT NOT NULL,
55     'name' VARCHAR(60) NOT NULL,
56     'color' VARCHAR(60) NOT NULL,
57     'size' VARCHAR(5),
58     'brand' VARCHAR(250),
59     'price' NUMERIC NOT NULL,
60     'currency' CHAR(3) NOT NULL,
61     'inventory' INT NOT NULL,
62     FOREIGN KEY ('seller_id')
63         REFERENCES sellers ('id'),
64     FOREIGN KEY ('category_id')
65         REFERENCES categories ('id')
66 );
67

```

```

68 -- Shipper Schema
69 CREATE TABLE IF NOT EXISTS `shippers` (
70     'id' INT PRIMARY KEY,
71     'name' CHAR(25) NOT NULL,
72     'phone_number' VARCHAR(25) NOT NULL
73 );
74
75 -- Order Schema : create after 3 main tables
76 CREATE TABLE IF NOT EXISTS `orders` (
77     'id' INT PRIMARY KEY,
78     'customer_id' INT NOT NULL,
79     'product_id' INT NOT NULL,
80     'shipper_id' INT NOT NULL,
81     'order_date' DATE NOT NULL,
82     'order_time' TIMESTAMP NOT NULL,
83     'quantity' INT NOT NULL,
84     'discount' DECIMAL(3,2) NOT NULL,
85     'rating_review' INT,
86     FOREIGN KEY ('customer_id')
87         REFERENCES customers ('id'),
88     FOREIGN KEY ('product_id')
89         REFERENCES products ('id'),
90     FOREIGN KEY ('shipper_id')
91         REFERENCES shippers ('id')
92 );
93
94 -- Ads Schema
95 CREATE TABLE IF NOT EXISTS `advertisements` (
96     'id' INT PRIMARY KEY,
97     'product_id' INT NOT NULL,
98     'content' TEXT,
99     'ad_clicks' INT,
100     'budget' DECIMAL(10,2),
101     'currency' CHAR(3),
102     FOREIGN KEY ('product_id')
103         REFERENCES products ('id')
104 );

```


3 Part 2: Data Generation and Management

3.1 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.

3.2 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:

```
1  print("Performing Validation")
2
3  # Get the primary key column names from the database
4  query <- paste("SELECT name FROM pragma_table_info('",table_name,"')
5                WHERE pk = 1;",sep="")
6  primary_key_columns <- dbGetQuery(my_db, query)
7
8  # Get Foreign Key
9  query <- paste("PRAGMA foreign_key_list('",table_name,"');",sep="")
10 foreign_key_columns <- dbGetQuery(my_db, query)
11
12 # ----- 1. Check duplicate primary key within CSV file -----
13 print(paste0("Checking duplicate primary key for: ",variable))
14
15 number_of_rows <- nrow(this_file_contents)
16
17 for (i in primary_key_columns) {
18   if (nrow(unique(this_file_contents[,i]))==number_of_rows) {
19     print(paste("Primary key =",i,": Passed"))
20   }
21   else {
22     stop(paste("Found duplicate record in ", variable,": STOP process!"))
23   }
24 }
25
26
27 # ----- 2. Check data quality and integrity -----
28 print(paste0("Checking integrity for: ",variable))
29
30 # Function to validate email addresses
31 validate_emails <- function(emails) {
32   pattern <- "[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,}$"
33   grepl(pattern, emails)
34 }
35
36 # Function to validate phone numbers
37 validate_phones <- function(phones) {
38   # This is a simple example and might not cover all international formats
39   pattern <- "^1[0-9]{10}$"
40   grepl(pattern, phones)
```

```

41 }
42
43 # Function to validate dates
44 validate_dates <- function(dates) {
45   date_format <- "%Y-%m-%d"
46   dates_parsed <- parse_date_time(dates, orders = date_format)
47   !is.na(dates_parsed)
48 }
49
50 # Function to validate prices
51 validate_prices <- function(prices) {
52   prices > 0
53 }
54
55 # Function to validate currency codes
56 validate_currencies <- function(currencies) {
57   pattern <- "^USD$"
58   grepl(pattern, currencies)
59 }
60
61 # Function to validate payment method
62 validate_payment_method <- function(payment_method){
63   valid_method <- c("Apple Pay", "Google Pay", "Credit Card", "Cash",
64                     "Debit Card", "Bank Transfer", "Cheque")
65   payment_method %in% valid_method
66 }
67
68 # Function to validate ad clicks
69 validate_ad_clicks <- function(ad_clicks) {
70   ad_clicks >= 0
71 }
72
73 # Function to validate discount
74 validate_discount <- function(discount) {
75   (discount >= 0 & discount <= 100)
76 }
77
78 # Function to validate rating_review
79 validate_rating_review <- function(rating_review) {
80   valid_rating <- c(1, 2, 3, 4, 5)
81   is.na(rating_review) | rating_review %in% valid_rating
82 }
83
84 # Function error handling

```

```

85 validation <- function(this_file_contents,type,column) {
86   tmp_table <- this_file_contents
87   if (type == 'Email') {
88     tmp_table$valid_format <- validate_emails(column)
89   } else if (type == 'Phone_numbers') {
90     tmp_table$valid_format <- validate_phones(column)
91   } else if (type == 'Dates') {
92     tmp_table$valid_format <- validate_dates(column)
93   } else if (type == 'Prices' || type == 'Budget' || type == 'Quantity') {
94     tmp_table$valid_format <- validate_prices(column)
95   } else if (type == 'Currencies') {
96     tmp_table$valid_format <- validate_currencies(column)
97   }
98   else if (type == 'payment_method') {
99     tmp_table$valid_format <- validate_payment_method(column)
100  }
101  else if (type == 'ad_clicks') {
102    tmp_table$valid_format <- validate_ad_clicks(column)
103  }
104  else if (type == 'discount') {
105    tmp_table$valid_format <- validate_discount(column)
106  }
107  else if (type == 'rating_review') {
108    tmp_table$valid_format <- validate_rating_review(column)
109  }
110  if (nrow(tmp_table) >0) {
111    for (i in 1:nrow(tmp_table)){
112      tmp_row <- tmp_table[i,]
113      if (!tmp_row$valid_format) {
114        warning(type," Format of ID: ",tmp_row$id, " in ", variable,
115              " is incorrect. Please check." )
116      }
117    }
118  }
119  if (all(tmp_table$valid_format) == TRUE){
120    print(paste(type," Format: Passed!"))
121  }
122  tmp_table <- tmp_table[tmp_table$valid_format,] # remove row
123  tmp_table <- tmp_table[, !names(tmp_table) %in% "valid_format"]
124  return(tmp_table)
125 }
126
127 # Perform integrity check
128 if (table_name == 'customers' && nrow(this_file_contents) >0) {

```

```

129   this_file_contents <- validation(this_file_contents, 'Email',
130                                   this_file_contents$email)
131   this_file_contents <- validation(this_file_contents, 'Phone_numbers',
132                                   this_file_contents$phone_number)
133   this_file_contents <- validation(this_file_contents, 'payment_method',
134                                   this_file_contents$current_payment_method)
135
136 } else if (table_name == 'orders' && nrow(this_file_contents) > 0) {
137   this_file_contents <- validation(this_file_contents, 'Dates',
138                                   this_file_contents$order_date)
139   this_file_contents <- validation(this_file_contents, 'discount',
140                                   this_file_contents$discount)
141   this_file_contents <- validation(this_file_contents, 'Quantity',
142                                   this_file_contents$quantity)
143   this_file_contents <- validation(this_file_contents, 'rating_review',
144                                   this_file_contents$rating_review)
145 } else if (table_name == 'products' && nrow(this_file_contents) > 0) {
146   this_file_contents <- validation(this_file_contents, 'Prices',
147                                   this_file_contents$price)
148   this_file_contents <- validation(this_file_contents, 'Currencies',
149                                   this_file_contents$currency)
150
151 } else if (table_name == 'categories' && nrow(this_file_contents) > 0) {
152
153 } else if (table_name == 'sellers' && nrow(this_file_contents) > 0) {
154   this_file_contents <- validation(this_file_contents, 'Email',
155                                   this_file_contents$email)
156
157 } else if (table_name == 'shippers' && nrow(this_file_contents) > 0) {
158   this_file_contents <- validation(this_file_contents, 'Phone_numbers',
159                                   this_file_contents$phone_number)
160 } else if (table_name == 'advertisements' && nrow(this_file_contents) > 0) {
161   this_file_contents <- validation(this_file_contents, 'Currencies',
162                                   this_file_contents$currency)
163   this_file_contents <- validation(this_file_contents, 'Budget',
164                                   this_file_contents$budget)
165   this_file_contents <- validation(this_file_contents, 'ad_clicks',
166                                   this_file_contents$ad_clicks)
167 }
168
169 # ----- 3. Check Foreign key -----
170 if(nrow(this_file_contents) > 0) {
171   foreign_table <- foreign_key_columns[, 'table']
172   tmp_table <- this_file_contents

```

```

173 for (i in foreign_table) {
174   foreign_key_ori_column <- foreign_key_columns[
175     foreign_key_columns[, 'table'] == i, 'from']
176   foreign_key_dest_column <- foreign_key_columns[
177     foreign_key_columns[, 'table'] == i, 'to']
178   print(paste("Checking Foreign key in table", i,
179     "column:", foreign_key_ori_column))
180   for (j in 1:nrow(this_file_contents)) {
181     foreign_key_value <- this_file_contents[j, foreign_key_ori_column]
182     query <- paste("SELECT", foreign_key_dest_column, " FROM", i, " WHERE",
183       foreign_key_dest_column, "=", foreign_key_value, ";")
184     result <- dbGetQuery(my_db, query)
185     col <- paste("check_", i, sep="")
186     if (nrow(result) == 0) {
187       warning("Foreign key is missing in row ID = ",
188         this_file_contents[j, primary_key_columns[1,]], " Please check.")
189       tmp_table[[col]] <- FALSE
190     } else {
191       tmp_table[[col]] <- TRUE
192     }
193   }
194 }
195 rows_to_remove <- apply(tmp_table[, grepl("^check", names(tmp_table))], 1,
196   function(row) any(!row))
197 tmp_table <- tmp_table[, !grepl("^check", names(tmp_table))] # remove column
198 tmp_table <- tmp_table[!rows_to_remove, ] # remove failed row
199 this_file_contents <- tmp_table
200 } else{
201   print("No validation check in this table since there's no foreign key")
202 }
203
204 print("Validation Completed")

```

The initial load has been performed.

```

1 # Get only Initial file which has the format [table_name].csv
2 all_files <- setdiff(list.files("../data_upload/"),
3   list.files("../data_upload/", pattern = "_"))
4 # Order the files to load to database, to avoid error from foreign key
5 custom_order <- list("customers.csv", "sellers.csv", "categories.csv",
6   "products.csv", "shippers.csv", "orders.csv",
7   "advertisements.csv")
8 all_files <- all_files[order(match(all_files, custom_order))]
9

```

```

10 for (variable in all_files) {
11   this_filepath <- paste0("../data_upload/",variable)
12   this_file_contents <- readr::read_csv(this_filepath)
13
14   table_name <- gsub(".csv","",variable)
15
16   # Perform Validation
17   source("../main/Validation.R")
18
19   # convert column date format
20   if (table_name == 'orders') {
21     this_file_contents['order_date'] <- lapply(this_file_contents['order_date'],
22                                               as.character)
23   }
24
25   if (nrow(this_file_contents)>0 ){
26     for (i in 1:nrow(this_file_contents)) {
27       row <- this_file_contents[i, ]
28
29       # Extract primary key values from the row
30       primary_key_values <- paste(names(row)[names(row) %in%
31                                   primary_key_columns],
32                                   row[names(row) %in%
33                                   primary_key_columns],
34                                   sep = "=", collapse = " AND ")
35
36       # Find if the primary key exists
37       query <- paste("SELECT * FROM", table_name, paste("WHERE",
38                                                         primary_key_values))
39       existing_row <- dbGetQuery(my_db, query)
40
41       if (nrow(existing_row) == 0) {
42         # Row is unique, append to the table
43         #print(paste("Append:",primary_key_values))
44         dbWriteTable(my_db, table_name, row, append = TRUE)
45       } else {
46         # Row already exists, update the existing row
47         #print(paste("Update:",primary_key_values))
48         update_query <- paste("UPDATE", table_name,
49                               paste("SET", paste(names(row), "=",
50                                                    paste0("'", row, "'"),
51                                                    collapse = ", "),
52                               "WHERE", primary_key_values))
53         dbExecute(my_db, update_query)

```

```

54     }
55   }
56 }
57 else {
58   print("Nothing to update in database since all
59         rows are not pass the validations")
60 }
61 }

```

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, rating...
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"
```

4 Part 3: Data Pipeline Generation

4.1 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.

4.2 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.)

```
1 # Transformation.R
2 library(readr)
3 library(RSQLite)
```

```

4 library(dplyr)
5 library(ggplot2)
6 library(lubridate)
7
8 # Incremental Load
9 print("Loading CSV file")
10
11 # File format for automation: <table name>_YYYY-MM-DDTHHMMSS.csv
12 current_date <- Sys.Date()
13 print(paste("current date:", current_date))
14 # Get only Incremental file
15 all_files <- list.files("./data_upload", full.names = FALSE, pattern = "_")
16 for (variable in all_files) {
17   # split file name using _ separator
18   file_name <- unlist(strsplit(gsub(".csv","",variable), "_"))
19   table_name <- file_name[1]
20   # Splitting file name using 'T' separator
21   date_time_parts <- unlist(strsplit(file_name[2], "T"))
22   date_str <- date_time_parts[1] # Date string
23   time_str <- date_time_parts[2] # Time string
24   # Parsing date strings into datetime objects using lubridate
25   date_value <- lubridate::ymd(date_str)
26
27   # Get only NEW file that has been loaded into the folder
28   # (and run historical back 2 days)
29   if (date_value >= current_date-1 && date_value <= current_date ) {
30     print(paste("Reading file:",variable))
31     this_filepath <- paste0("./data_upload/",variable)
32     this_file_contents <- readr::read_csv(this_filepath)
33
34     print(paste("Writing table to database:", table_name))
35     my_db <- RSQLite::dbConnect(RSQLite::SQLite(), "./database/ecommerce.db")
36
37     # Perform Validation
38     source("./main/Validation.R")
39
40     # convert column date format
41     if (table_name == 'orders') {
42       this_file_contents['order_date'] <- lapply(
43         this_file_contents['order_date'], as.character)
44     }
45
46     # Validation and Writing on each row to DB
47     if (nrow(this_file_contents)>0 ){

```

```

48     for (i in 1:nrow(this_file_contents)) {
49
50         row <- this_file_contents[i, ]
51
52         # Extract primary key values from the row
53         primary_key_values <- paste(names(row)[names(row) %in%
54                                     primary_key_columns],
55                                     row[names(row) %in% primary_key_columns],
56                                     sep = "=", collapse = " AND ")
57
58         # Find if the primary key exists
59         query <- paste("SELECT * FROM", table_name,
60                       paste("WHERE", primary_key_values))
61         existing_row <- dbGetQuery(my_db, query)
62
63         if (nrow(existing_row) == 0) {
64             # Row is unique, append to the table
65             print(paste("Append:", primary_key_values))
66             dbWriteTable(my_db, table_name, row, append = TRUE)
67         } else {
68             # Row already exists, update the existing row
69             print(paste("Update:", primary_key_values))
70             update_query <- paste("UPDATE", table_name,
71                                  paste("SET", paste(names(row), "=",
72                                                       paste0("'", row, "'"),
73                                                       collapse = ", "),
74                                  "WHERE", primary_key_values))
75             dbExecute(my_db, update_query)
76         }
77     }
78 }
79 else {
80     print("Nothing to update in database since all rows
81           are not pass the validations")
82 }
83 }
84 }
85
86 # Check if the connection object exists and is valid
87 if (exists("my_db") && RSQLite::dbIsValid(my_db)) {
88     # Perform Visualisation
89     source("../main/Visualisation.R")
90     print("Done!")
91     # Disconnect from the database

```

```

92   RSQLite::dbDisconnect(my_db)
93 } else {
94   # Print a message where the connection object is not found or invalid
95   print("Connection object not found or is invalid.")
96 }

```

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.

5 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:

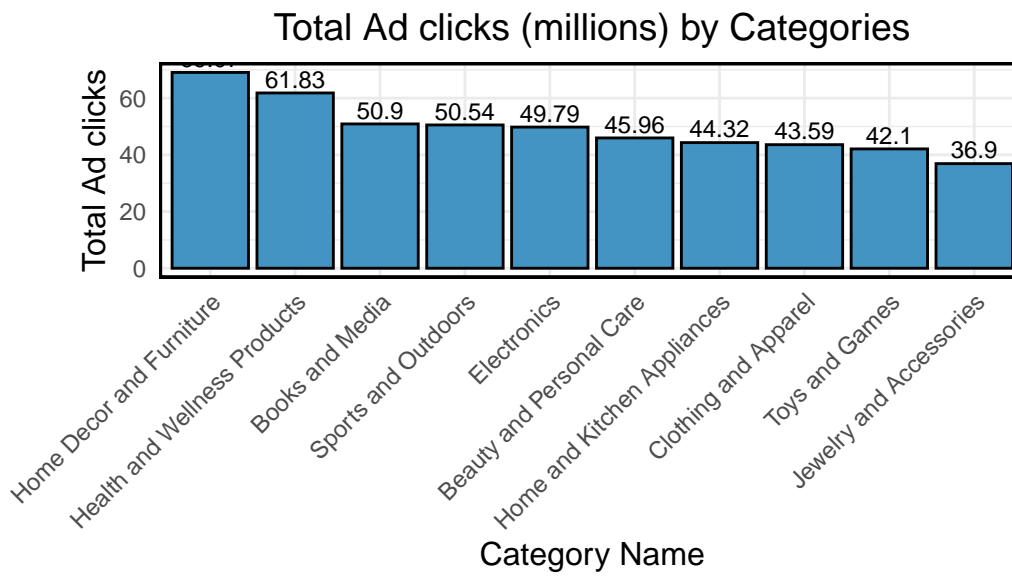
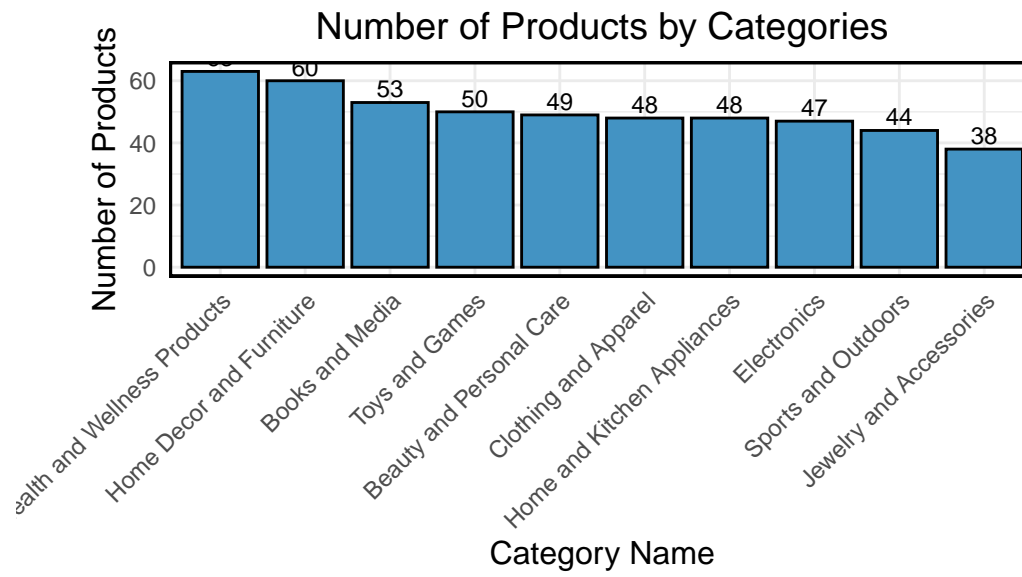
Platform Overview: Designed to provide the management with an overview of the platform participants (customers, sellers, and products), this dashboard addresses fundamental questions on the top selling product categories, customer traffics via ad clicks, and customers' and sellers' geographical location. This aids management in optimising the distribution network, merchandising strategy (i.e., which product to sell) 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

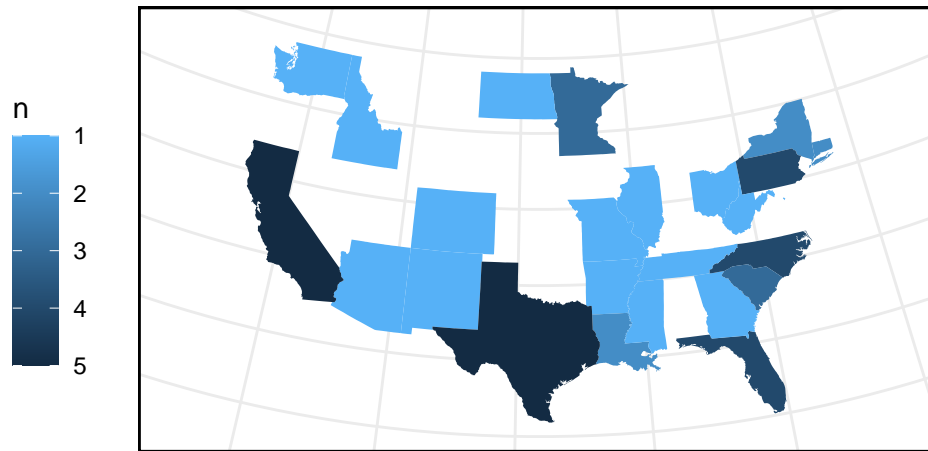
[1] 0

[1] 0

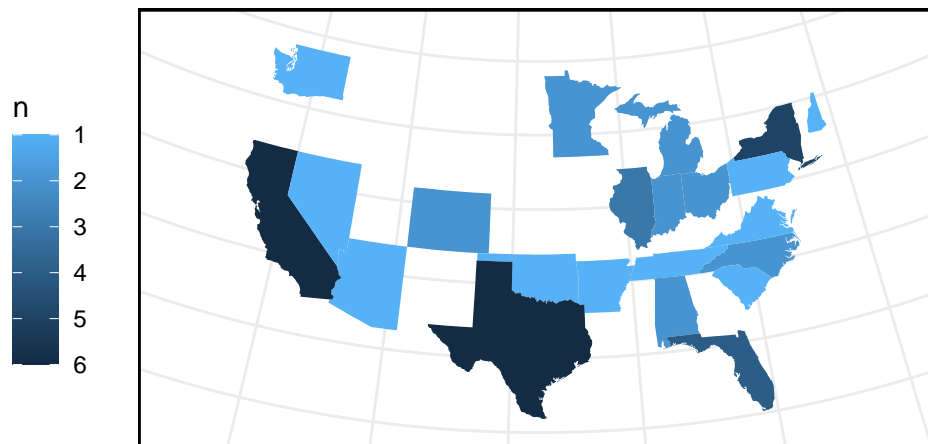
[1] 0



Number of Customers by States

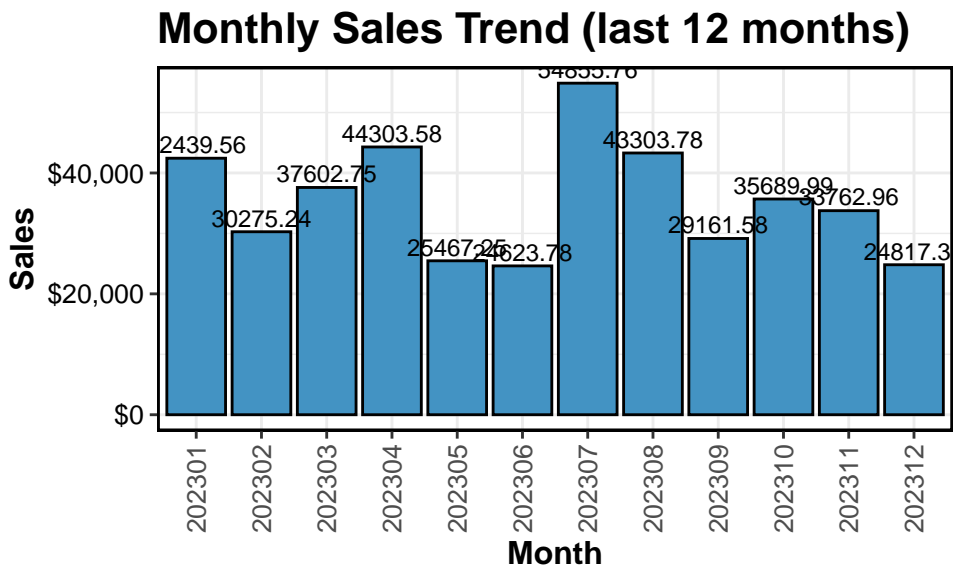


Number of Sellers by States



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.

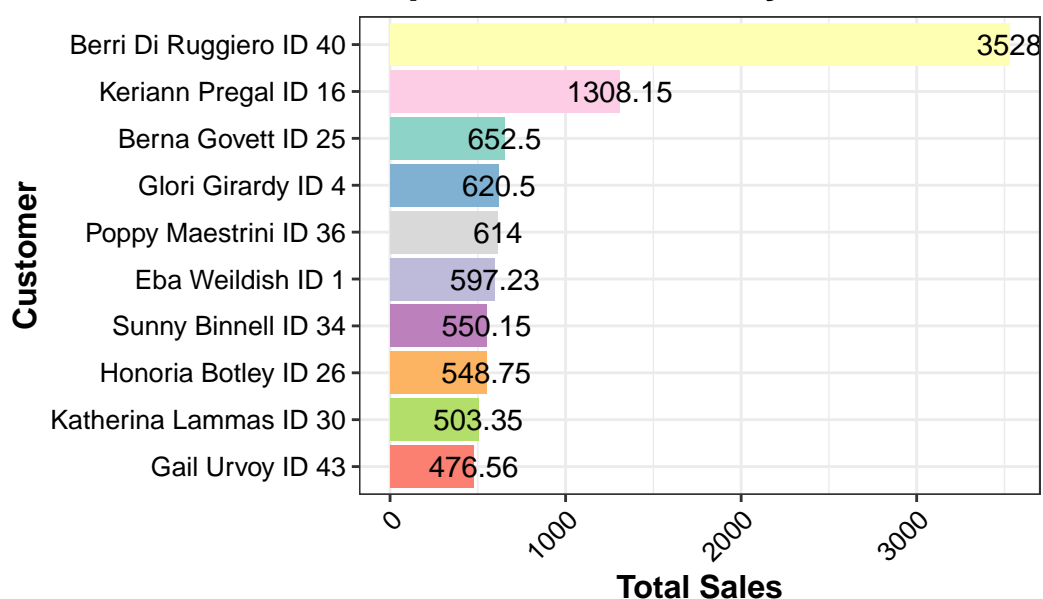


Sales by Categories

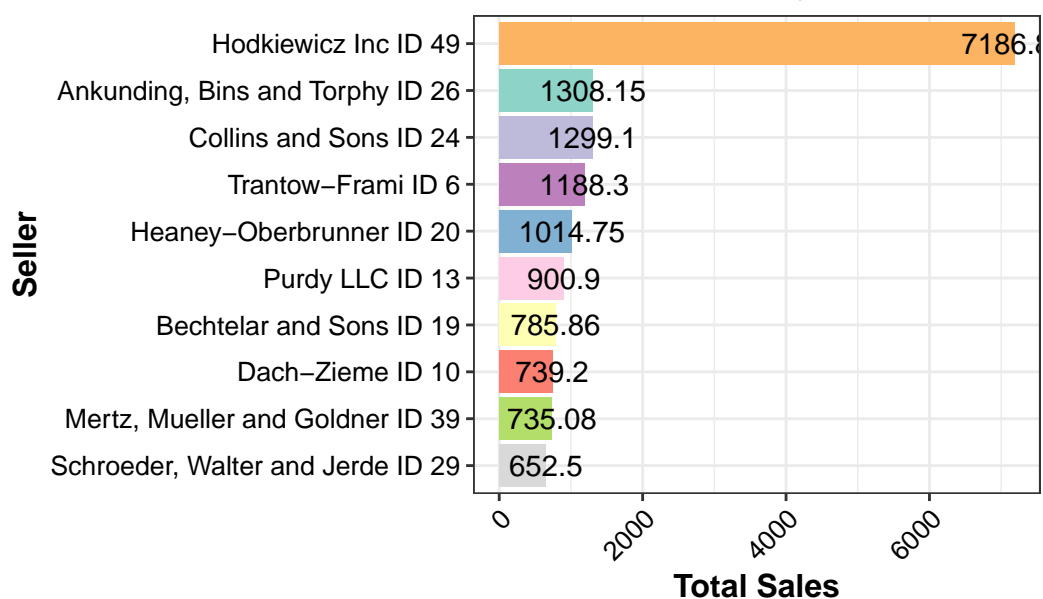


Sales values are in USD

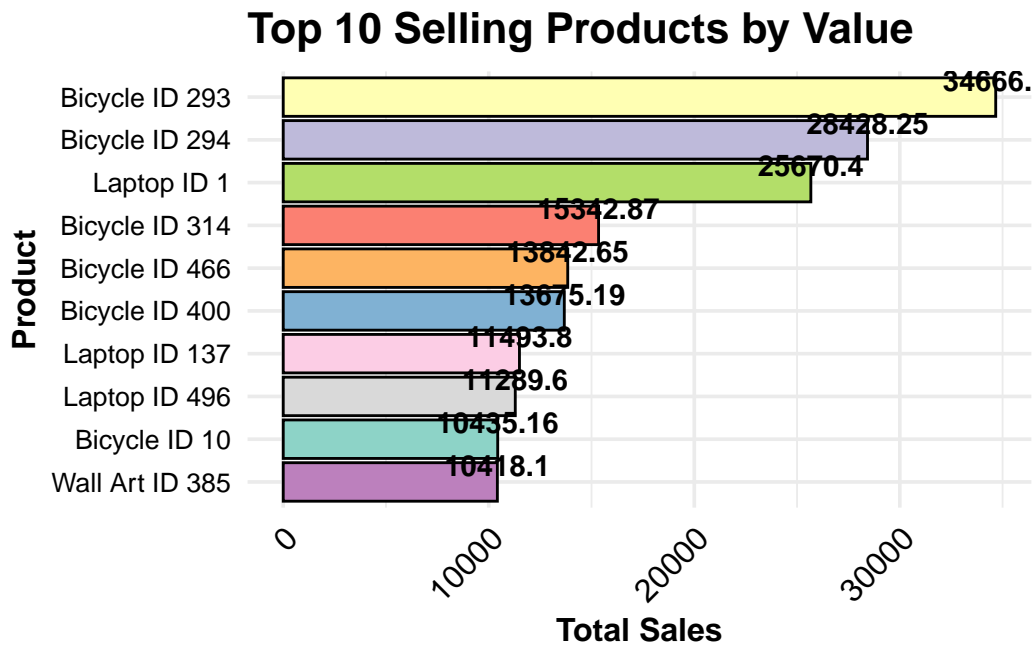
Top 10 Customers by Total Sales



Top 10 Sellers by Total Sales

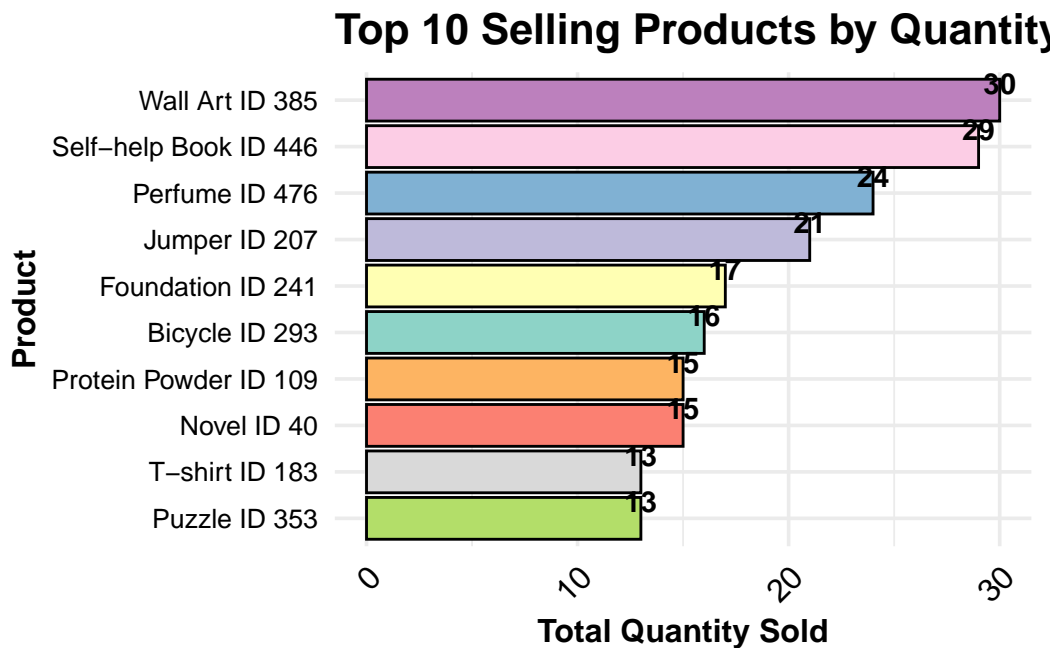


Top Products: Drilling down into the product level, this showcases the top performing products by various metrics: sales value, sales quantum, customer rating and conversion rate (measured by sales quantum per a million of ad clicks), spearheading the platform's marketing, merchandising, and acquisition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

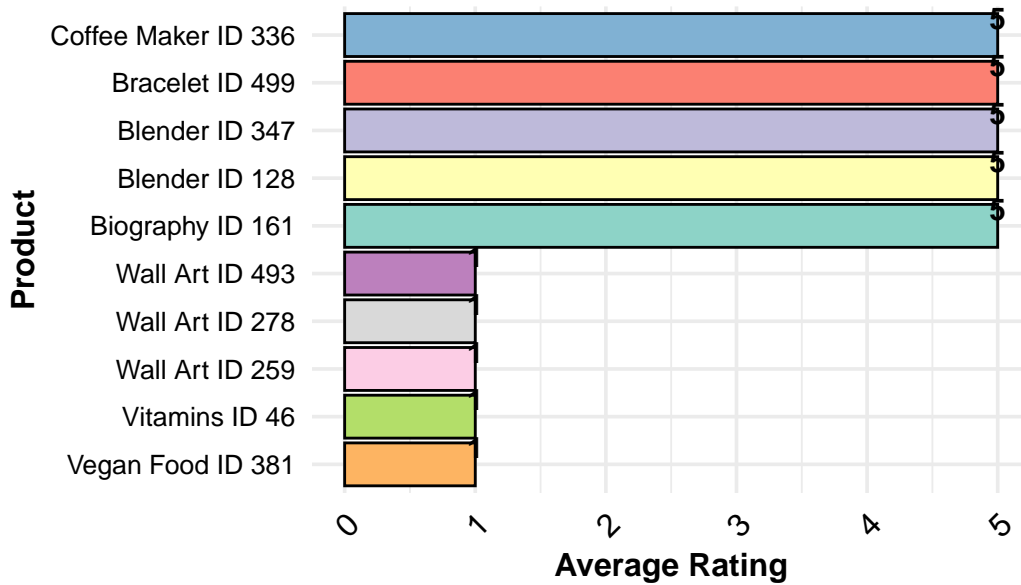


Scale for fill is already present.

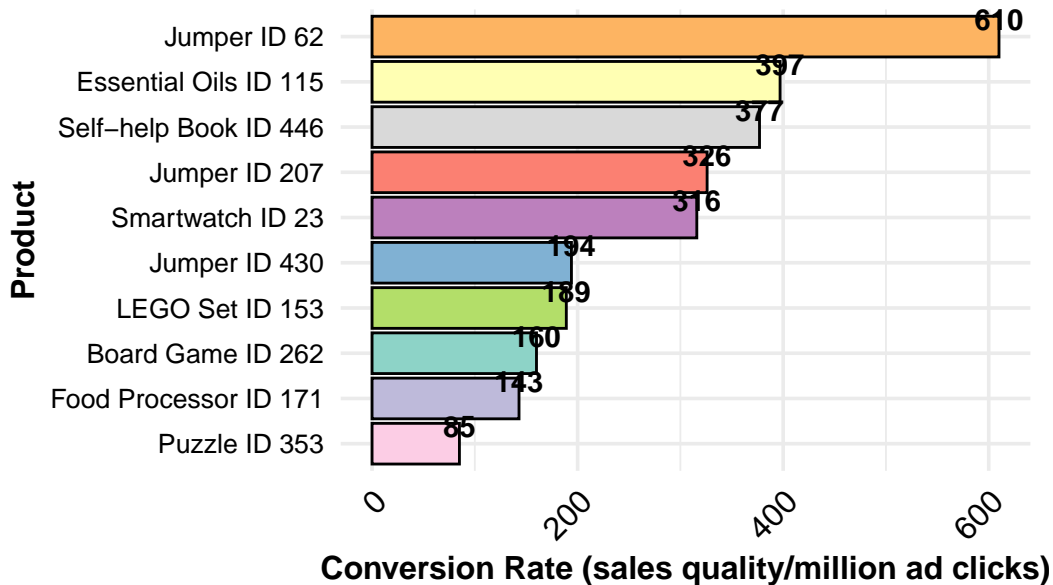
Adding another scale for fill, which will replace the existing scale.



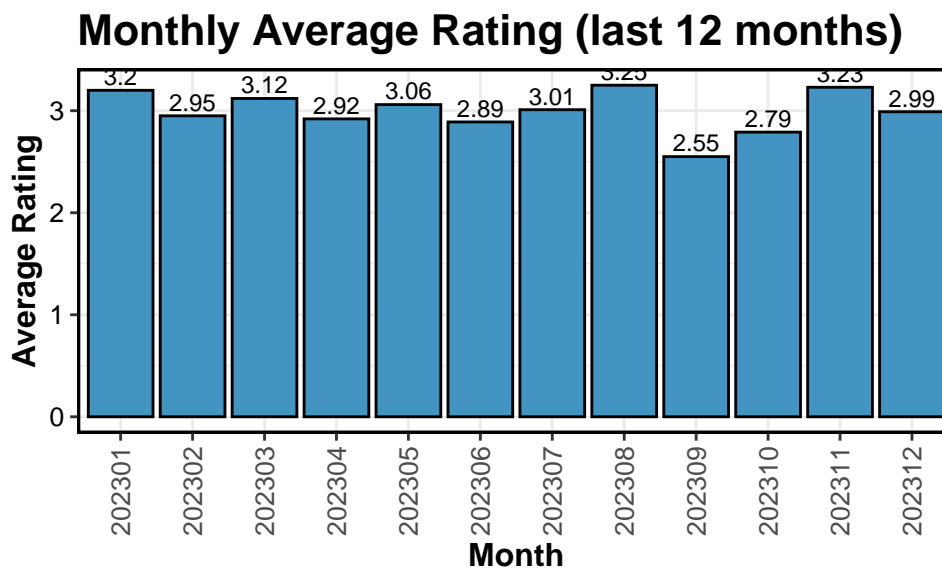
Top & Bottom 5 Rating Products



Total Conversion Rate



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.



```

1  ## Figure 14: Percentage of Nil Rating
2
3  rating_all <- dbGetQuery(my_db,
4                          "SELECT id, product_name, order_date, rating_review, sales
5                          FROM df_sales")

```

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

```

1  # Convert order_date to date format
2  rating_all$order_date <- as.Date(as.character(rating_all$order_date),
3                                   format = "%Y-%m-%d")
4  rating_all <- rating_all %>%
5    mutate(year_month = gsub('-', '',
6                               as.character(format(as.Date(order_date), "%Y-%m")))))
7
8  # Calculate total number of orders per months
9  rating_all_summary <- rating_all %>% group_by(year_month) %>% summarise (n_all = n())
10
11 # Filter no rating and convert date format
12 rating_n <- rating_all %>% filter(rating_review == 0)
13
14 # Calculate number of orders with no review
15 rating_n_summary <- rating_n %>% group_by(year_month) %>% summarise (n_n = n())

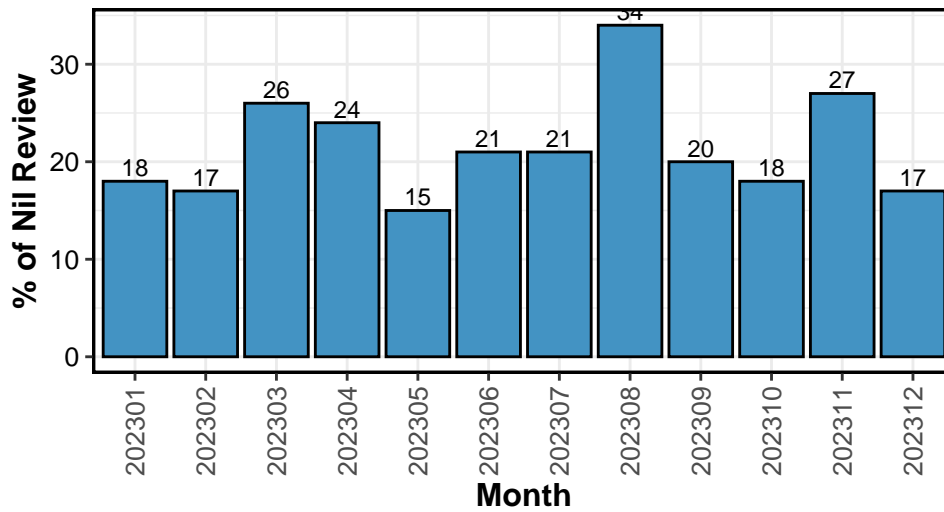
```

```

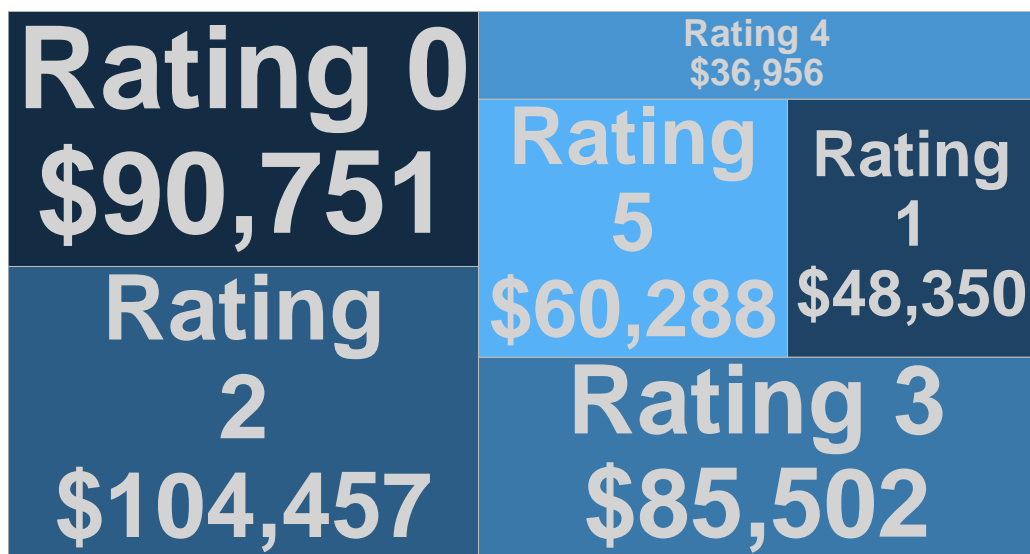
16
17 # Combine data
18 rating_n_summary <- merge(rating_all_summary, rating_n_summary)
19
20 # Calculate nil review rate
21 rating_n_summary <- rating_n_summary %>%
22   mutate(nil_review_rate = round(n_n *100/n_all),2) %>%
23   arrange(desc(year_month))
24
25 # Take last 12 months
26 rating_n_summary <- head(rating_n_summary,12)
27
28 # Plot monthly sales trend with advanced visualization
29 (figure.14 <- ggplot(rating_n_summary, aes(x = year_month, y = nil_review_rate)) +
30   geom_bar(stat = "identity", fill = "#4393C3", color = "black") +
31   labs(title = "Percentage of Nil Review (last 12 months)", x = "Month",
32     y = "% of Nil Review") +
33   theme_bw() +
34   theme(axis.text.y = element_text(size = 10, color = "black"),
35     axis.title = element_text(size = 12, color = "black", face = "bold"),
36     plot.title = element_text(size = 16, color = "black", face = "bold"),
37     legend.position = "none",
38     panel.border = element_rect(color = "black", fill = NA, size = 1),
39     plot.margin = margin(t = 0.5, r = 0.5, b = 1, l = 1, unit = "cm"),
40     axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 1, size = 10)) +
41   geom_text(aes(label = nil_review_rate), vjust = -0.3, size = 3, color = "black"))

```

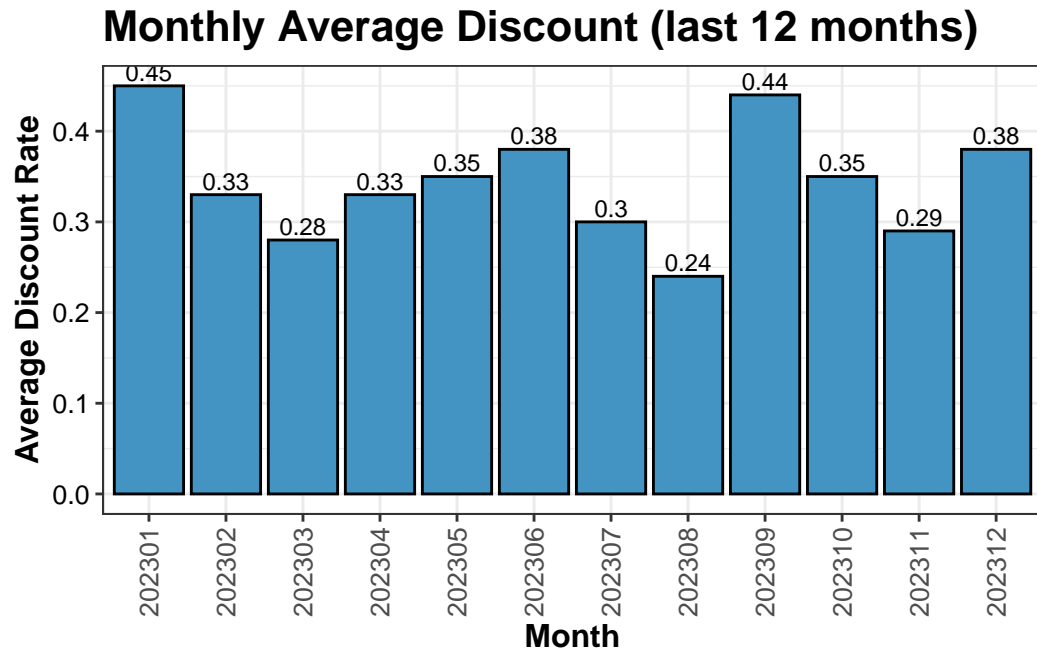
Percentage of Nil Review (last 12 months)



Sales by Rating



Sales values are in USD



6 Conclusion

Simulating a real-world problem allows us to familiarise with challenges we may confront during an end-to-end process of database design, workflow management and data analysis. We recognise the importance of a dynamic approach incorporating feedback loops and incremental improvements built on foundational understanding of the business objectives. We have evolved through this iterative journey embraced with collaborative teamwork to deliver this end-product, further advancing ourselves for tackling the real-world challenges in future.