Table of contents

1 Part 1 - Database Design and Implementation **1**
- 1.1 Task 1.1: E-R Diagram Design . . . 1
- 1.2 Task 1.2: SQL Database Schema Creation . . . 1

2 Part 2: Data Generation and Management **4**
- 2.1 Task 2.1: Synthetic Data Generation . . . 4
- 2.2 Task 2.2: Data Import and Quality Assurance . . . 8
  - 2.2.1 Check Referential Integrity . . . 20

3 Part 3: Data Pipeline Generation **23**
- 3.1 Task 3.1: GitHub Repository and Workflow Setup . . . 23
- 3.2 Task 3.2: GitHub Actions for Continuous Integration . . . 23

4 Part 4: Data Analysis and Reporting with Quarto in R **23**
- 4.1 Task 4.1: Advanced Data Analysis in R . . . 23
- 4.2 Task 4.2: Comprehensive Reporting with Quarto . . . 23


```
library(readr)
library(RSQLite)
library(tibble)
library(dplyr)
library(lubridate)
library(DBI)
library(assertthat)
library(purrr)
library(ggplot2)
library(RSQLite)
library(DBI)
```

1 Part 1 - Database Design and Implementation

1.1 Task 1.1: E-R Diagram Design

1.2 Task 1.2: SQL Database Schema Creation

```
#setwd("/cloud/project/")
print(getwd())
```

```r
#connect to the SQLite database
my_connection <- RSQLite::dbConnect(RSQLite::SQLite(),
                                    "../database/ecommerce_database_v1.db")


dbExecute(my_connection,
                "CREATE TABLE IF NOT EXISTS CUSTOMERS
                (
                    customer_id VARCHAR(255) NOT NULL PRIMARY KEY,
                    first_name VARCHAR(255) NOT NULL,
                    last_name VARCHAR(255),
                    username VARCHAR(255),
                    gender TEXT,
                    date_of_birth DATE NOT NULL,
                    email VARCHAR(255) UNIQUE,
                    phone VARCHAR(20) UNIQUE,
                    street_name VARCHAR(255),
                    city VARCHAR(255),
                    country VARCHAR(255),
                    zip_code VARCHAR(20),
                    account_created_date TIMESTAMP,
                    premium_subscription INTEGER
                );"
          )

dbExecute(my_connection,
                "CREATE TABLE IF NOT EXISTS PRODUCT_CATEGORY
                (
                    category_id VARCHAR(255) NOT NULL PRIMARY KEY,
                    cat_name VARCHAR(255)
                );"
          )

dbExecute(my_connection,
                "CREATE TABLE IF NOT EXISTS SUPPLIERS
                (
                    supplier_id VARCHAR(255) NOT NULL PRIMARY KEY,
                    supplier_name VARCHAR(255),
                    supplier_address VARCHAR(500),
                    supplier_phone VARCHAR(20),
                    supplier_email VARCHAR(255) UNIQUE
                );"
```

```
          )

dbExecute(my_connection,
                "CREATE TABLE IF NOT EXISTS PRODUCTS
                (
                    product_id VARCHAR(255) NOT NULL PRIMARY KEY,
                    product_name VARCHAR(255),
                    price REAL,
                    stock_quantity INTEGER NOT NULL,
                    category_id VARCHAR(255) NOT NULL,
                    supplier_id VARCHAR(255) NOT NULL,
                    FOREIGN KEY(category_id) REFERENCES
                          PRODUCT_CATEGORY(category_id),
                    FOREIGN KEY(supplier_id) REFERENCES SUPPLIERS(supplier_id)
                );"
          )

dbExecute(my_connection,
                "CREATE TABLE IF NOT EXISTS GIFT_CARD
                (
                gift_card_id VARCHAR(50) NOT NULL PRIMARY KEY,
                gift_card_code VARCHAR(50),
                detail INTEGER,
                status VARCHAR(50)
                );"
          )
dbExecute(my_connection,
                "CREATE TABLE IF NOT EXISTS ORDERS
                (
                    order_id VARCHAR(255) NOT NULL PRIMARY KEY,
                    customer_id VARCHAR(255),
                    product_id VARCHAR(255),
                    gift_card_id VARCHAR(255),
                    payment_method TEXT,
                    quantity INTEGER,
                    order_timestamp TIMESTAMP,
                    payment_timestamp TIMESTAMP,
                    order_status VARCHAR(50) NOT NULL,
                    shipment_id VARCHAR(255),
                    FOREIGN KEY(customer_id) REFERENCES CUSTOMERS(customer_id),
                    FOREIGN KEY(product_id) REFERENCES PRODUCTS(product_id),
                    FOREIGN KEY(shipment_id) REFERENCES SHIPMENT(shipment_id),
```

```
                    FOREIGN KEY(gift_card_id) REFERENCES GIFT_CARD(gift_card_id)
                );"
            )
dbExecute(my_connection,
                "CREATE TABLE IF NOT EXISTS SHIPMENT
                (
                shipment_id VARCHAR(255) NOT NULL PRIMARY KEY,
                dispatch_timestamp DATETIME,
                delivered_timestamp DATETIME,
                status VARCHAR(50) NOT NULL
                );"
            )

#Check if the tables are created

dbGetQuery(my_connection,
    sprintf("SELECT name FROM sqlite_master WHERE type='table';")
  )


# dbDisconnect(my_connection)
```

2 Part 2: Data Generation and Management

2.1 Task 2.1: Synthetic Data Generation

```
## Find all files matching the pattern
customer_files <- list.files(path = "../datasets"
                             ,pattern = "CUSTOMERS.*\\.csv$",full.names = TRUE)
category_files <- list.files(path = "../datasets"
                             ,pattern = "CATEGORY.*\\.csv$",full.names = TRUE)
gift_card_files <- list.files(path = "../datasets"
                             ,pattern = "GIFT_CARDS.*\\.csv$",full.names = TRUE)
suppliers_files <- list.files(path = "../datasets"
                             ,pattern = "SUPPLIERS.*\\.csv$",full.names = TRUE)
products_files <- list.files(path = "../datasets"
                             ,pattern = "PRODUCTS.*\\.csv$",full.names = TRUE)

customers_df <- readr::read_csv(customer_files[1])
```

```r
gift_card_df <- readr::read_csv(gift_card_files[1])
suppliers_df <- readr::read_csv(suppliers_files[1])
category_df <- readr::read_csv(category_files[1])
products_df <- readr::read_csv(products_files[1])



#Sample Customers

sample_size <- floor(0.2 * nrow(products_df))
sampled_product_ids <- sample(products_df$product_id,
                              size = sample_size, replace = FALSE)
sampled_products_df <- products_df[products_df$product_id %in%
                                   sampled_product_ids, ]


#Sample Products

sample_size <- floor(0.2 * nrow(customers_df))
sampled_customer_ids <- sample(customers_df$customer_id,
                               size = sample_size, replace = FALSE)
sampled_customers_df <- customers_df[customers_df$customer_id %in%
                                     sampled_customer_ids, ]


generate_orders_data <- function(n = 1000) {
  set.seed(123)

  orders_df <- tibble(
    order_id = sprintf("%s-%04d", "ORD", 1:n),
    customer_id = sample(sampled_customers_df$customer_id, n, replace = TRUE),
    product_id = sample(sampled_products_df$product_id, n, replace = TRUE),
    gift_card_id = sample(c(NA, gift_card_df$gift_card_id), n, replace = TRUE),
    payment_method = sample(c("Credit Card", "Debit Card", "PayPal",
                              "Gift Card"),n, replace = TRUE),
    quantity = sample(1:5, n, replace = TRUE),
    order_timestamp = sample(seq(as.POSIXct('2024/02/01')
                      ,as.POSIXct('2024/02/29'), by="day"), n, replace = TRUE),
    payment_timestamp = order_timestamp + hours(sample(1:72, n, replace = TRUE)),
    order_status = sample(c("Processing", "Shipped", "Delivered",
                            "Cancelled","Pending Payment","Out for Delivery")
                          , n, replace = TRUE),
```

```
  )

  # Augment the orders data frame with supplier_id using left_join
  orders_df <- orders_df %>%
    left_join(sampled_products_df %>% select(product_id, supplier_id)
              , by = "product_id") %>%
    select(order_id, customer_id, product_id, gift_card_id
           , payment_method, quantity, order_timestamp, payment_timestamp
           , order_status, supplier_id)

  return(orders_df)
}

# Generate orders data
orders_df <- generate_orders_data(n = 1000)

generate_shipment_ids <- function(df) {
  # Create a unique identifier for each group
  df <- df %>%
    mutate(date_only = as.Date(order_timestamp)) %>%
    group_by(customer_id, supplier_id, date_only) %>%
    mutate(shipment_group_id = cur_group_id()) %>%
    ungroup() %>%
    mutate(shipment_id = sprintf("SHIP%05d", shipment_group_id)) %>%
    select(-shipment_group_id, -date_only) # Clean up the extra columns

  df
}

# Apply the function to your data frame
orders_df <- generate_shipment_ids(orders_df)

  orders_df <- orders_df %>%
    mutate(shipment_id = if_else(order_status %in%
                                 c("Cancelled","Pending Payment"), NA_character_,
                                    as.character(shipment_id)),
           payment_method = if_else(order_status == "Pending Payment"
                                        ,NA_character_,payment_method)) %>%
    mutate(supplier_id = NULL)
```

```
#Shipment Table
```

```r
shipment_df <- orders_df %>%
  mutate(
    # Dispatch date could be the same as the order date or a day after
    dispatch_timestamp = order_timestamp + days(sample(0:1, n()
                                                       , replace = TRUE)),

    # Delivered date should be after the dispatch date;
    #here I assume delivery takes between 2 to 5 days
    delivered_timestamp = dispatch_timestamp + days(sample(2:5, n()
                                                           , replace = TRUE)),

    # Randomly assign a delivery status
    status = if_else(order_status == "Processing","Ready for Dispatch"
                     ,if_else(order_status == "Shipped","In Transit"
                     ,if_else(order_status == "Out for Delivery",order_status
                    ,if_else(order_status == "Delivered",order_status,"NA"))))
  ) %>%
  # Select only the relevant columns for the shipment table
  select(shipment_id, dispatch_timestamp, delivered_timestamp, status) %>%
  # Remove duplicate rows to ensure unique shipments
  distinct()

shipment_df <- na.omit(shipment_df)


shipment_df <- shipment_df %>%
  mutate(
    # Assign NA to dispatch_timestamp if status is 'Ready for Dispatch'
    dispatch_timestamp = if_else(status == "Ready for Dispatch"
                                 , NA_Date_, dispatch_timestamp),
    delivered_timestamp = if_else(status == "Ready for Dispatch"
                                  , NA_Date_, delivered_timestamp),

    # 'In Transit' status should have a dispatch date but no delivery date
    dispatch_timestamp = if_else(status == "In Transit"
                       , Sys.Date() - days(sample(1:5, 1)), dispatch_timestamp),
    delivered_timestamp = if_else(status == "In Transit"
                       , NA_Date_, delivered_timestamp),

    # 'In Transit' status should have a dispatch date but no delivery date
    dispatch_timestamp = if_else(status == "Out for Delivery"
                     , Sys.Date() - days(sample(1:5, 1)), dispatch_timestamp),
```

```
      delivered_timestamp = if_else(status == "Out for Delivery"
                      , NA_Date_, delivered_timestamp),

      # If status is 'Delivered', both dates should be in the past,
      #with delivered after dispatched
      dispatch_timestamp = if_else(status == "Delivered" &
                                     is.na(dispatch_timestamp)
                      , Sys.Date() - days(sample(6:10, 1)), dispatch_timestamp),
      delivered_timestamp = if_else(status == "Delivered"
              , dispatch_timestamp + days(sample(1:5, 1)), delivered_timestamp)
    )

write_csv(orders_df,"../datasets/ORDERS.csv")

write_csv(shipment_df,"../datasets/SHIPMENTS.csv")
```

2.2 Task 2.2: Data Import and Quality Assurance

1.CUSTOMERS

```
ingest_customer_data <- function(df) {

  my_connection <- RSQLite::dbConnect(RSQLite::SQLite()
                                      , "../database/ecommerce_database_v1.db")
  # Data validation

  #email check
  valid_email <- grepl("^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}$"
                       , df$email)
  df <- df[valid_email, ]

  #gender check
  valid_genders <- c("Male", "Female", "Other")
  df <- df[df$gender %in% valid_genders, ]


  # Data type checks (adjust according to your data frame)
  df$date_of_birth <- as.Date(df$date_of_birth,format = "%d/%m/%y")
  df$account_created_date <- as.Date(df$account_created_date
                                     ,format = "%d/%m/%y")
  df$premium_subscription <- as.integer(df$premium_subscription)
```

```
  # Check for null values in NOT NULL columns
  required_columns <- c("customer_id", "first_name", "date_of_birth")
  df <- df[!rowSums(is.na(df[required_columns])) > 0, ]

  # Insert validated data into the database
  for(i in 1:nrow(df)){

      #Check for duplicate records based on the primary key
   existing_ids <- dbGetQuery(my_connection
         , sprintf("SELECT customer_id FROM CUSTOMERS WHERE customer_id = '%s'"
                                ,   df$customer_id[i]))
    if(nrow(existing_ids) > 0) {
      cat(sprintf("Skipping duplicate entry for customer_id: %s\n"
                  , df$customer_id[i]))
      next
    }

    insert_query <- sprintf("INSERT INTO CUSTOMERS (customer_id, first_name
    , last_name, username, gender, date_of_birth, email, phone, street_name
    , city, country, zip_code, account_created_date, premium_subscription)
    VALUES ('%s', '%s', '%s', '%s', '%s', '%s', '%s', '%s'
    , '%s', '%s', '%s', '%s', '%s', %d)",
    df$customer_id[i], df$first_name[i], df$last_name[i], df$username[i]
    , df$gender[i], df$date_of_birth[i],df$email[i], df$phone[i]
    , df$street_name[i], df$city[i], df$country[i], df$zip_code[i]
    , df$account_created_date[i], df$premium_subscription[i])
    tryCatch({
    dbExecute(my_connection, insert_query)
      cat(sprintf("Successfully inserted row: %d\n", i))
    }, error = function(e) {
      cat(sprintf("Error in inserting row: %d, Error: %s\n", i, e$message))
    })
      }

      # Close the database connection
      dbDisconnect(my_connection)
    }

for(file in customer_files) {
  df <- readr::read_csv(file)
  ingest_customer_data(df)
```

```
}
```

```
my_connection <- RSQLite::dbConnect(RSQLite::SQLite()
                                    , "../database/ecommerce_database_v1.db")
dbGetQuery(my_connection,"SELECT * FROM CUSTOMERS LIMIT 10;")
```

```
                 customer_id first_name last_name     username gender
1  01HQZS38KRC38NFNQR9QF1MTBZ       Poul  Jellings  pjellingsdv   Male
2  01HQZS38KT99V41AM8FFX4GZH7       Rolf   Crocket   rcrocketdw   Male
3  01HQZS38KW6A30TWWP40YR785F     Rockey   Lapwood   rlapwooddx   Male
4  01HQZS38KY9JB7X0RFWGEQESF5      Junia    Bayles    jbaylesdy Female
5  01HQZS38M0RSRWM1K83TZFG06K     Sydney Gillhespy sgillhespydz   Male
6  01HQZS38M3KZFS9R4CYZ8F2QNY     Johnny   Tidbold   jtidbolde0   Male
7  01HQZS38M5ZTYQRT6KQW75RQTS     Edward Strethill estrethille1  Other
8  01HQZS38M7XNA31ACXPJBC78ME       Walt Goulborne wgoulbornee2   Male
9  01HQZS38M9XY7AN2TSG9KTAARY     Bertie    Ratter    brattere3   Male
10 01HQZS38MC1ZX8SFB5WR3V2H66   Gerianne Meininger gmeiningere4 Female
   date_of_birth                        email        phone
1     1992-12-11 pjellingsdv@reverbnation.com 277-129-0314
2     1990-04-21        rcrocketdw@uol.com.br 755-108-4849
3     1992-09-20       rlapwooddx@latimes.com 563-846-2198
4     1999-02-13          jbaylesdy@hc360.com 809-987-6451
5     1990-05-15      sgillhespydz@cdbaby.com 881-340-2239
6     1990-08-04      jtidbolde0@china.com.cn 634-193-3056
7     1998-03-14       estrethille1@goo.ne.jp 716-684-1496
8     1997-02-01         wgoulbornee2@ihg.com 285-539-0816
9     1990-11-13      brattere3@bloomberg.com 455-678-8574
10    1992-10-18       gmeiningere4@amazon.de 302-279-5654
                  street_name       city        country zip_code
1       3 Stone Corner Street   Aberdeen United Kingdom     AB39
2           547 Fordem Avenue    Glasgow United Kingdom       G4
3              97 4th Avenue  Edinburgh United Kingdom      EH9
4            3922 Vahlen Way Birmingham United Kingdom      B12
5          60256 Russell Park  Liverpool United Kingdom      L74
6             5 Huxley Center      Upton United Kingdom     DN21
7              24 Ramsey Road    Kirkton United Kingdom     KW10
8             474 Lunder Lane    Wootton United Kingdom      NN4
9  4691 Weeping Birch Parkway     London United Kingdom     SW1E
10         15 Hanover Terrace   Brampton United Kingdom     NR34
   account_created_date premium_subscription
1            2023-04-01                    0
2            2023-12-15                    0
```

3	2023-11-30	0
4	2023-07-09	0
5	2023-06-08	1
6	2024-02-26	1
7	2023-04-12	0
8	2024-03-03	1
9	2023-09-12	1
10	2024-01-26	1

2. PRODUCT_CATEGORY

```
ingest_product_category <- function(df) {

  my_connection <- RSQLite::dbConnect(RSQLite::SQLite()
                                      , "../database/ecommerce_database_v1.db")

  # Check for null values in NOT NULL columns
  required_columns <- c("category_id", "cat_name")
  df <- df[!rowSums(is.na(df[required_columns])) > 0, ]

  # Insert validated data into the database
  for(i in 1:nrow(df)){
    # Check for duplicate records based on the primary key
    existing_ids <- dbGetQuery(my_connection, sprintf("SELECT category_id FROM
                PRODUCT_CATEGORY WHERE category_id = '%s'", df$category_id[i]))
    if(nrow(existing_ids) > 0) {
      cat(sprintf("Skipping duplicate entry for category_id: %s\n"
                  , df$category_id[i]))
      next
    }

    insert_query <- sprintf("INSERT INTO PRODUCT_CATEGORY
                            (category_id, cat_name) VALUES ('%s', '%s')",
                            df$category_id[i], df$cat_name[i])
    tryCatch({
      dbExecute(my_connection, insert_query)
      cat(sprintf("Successfully inserted row: %d\n", i))
    }, error = function(e) {
      cat(sprintf("Error in inserting row: %d, Error: %s\n", i, e$message))
    })
  }

    dbDisconnect(my_connection)
```

```
}

for(file in category_files) {

  df <- readr::read_csv(file)
  ingest_product_category(df)
}
```

```
my_connection <- RSQLite::dbConnect(RSQLite::SQLite(),
                                    "../database/ecommerce_database_v1.db")
dbGetQuery(my_connection,"SELECT * FROM PRODUCT_CATEGORY LIMIT 10;")
```

```
                  category_id   cat_name
1  01HQZSYXN5D9YD5YEVE62CZY5T    Jewelry
2  01HQZSYXN2NFNR8NP0JDJJ4EGE      Music
3  01HQZSYXN3Y1HWZHXWRT8QBN1F   Clothing
4  01HQZSYXN8GVDME3KSR2V3CWSY       Home
5  01HQZSYXN9NDEKZ0KDTXG7GWAR       Baby
6  01HQZSYXN8HS73RN25WQHFRVS9     Garden
7  01HQZSYXN69EZ5NYSTKN55ABQ6   Outdoors
8  01HQZSYXN577K9HSBRRVY2QSMT       Kids
9  01HQZSYXN7EQ2BMKM5RZH0274J Automotive
10 01HQZSYXN28M6P8R3N3Y74SSF1      Books
```

SUPPLIERS

```
ingest_suppliers <- function(df) {

  my_connection <- RSQLite::dbConnect(RSQLite::SQLite()
                                      , "../database/ecommerce_database_v1.db")

  # Email format validation
  valid_email <- grepl("^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}$",
                       df$supplier_email)
  df <- df[valid_email, ]

  # Check for null values in NOT NULL columns
  required_columns <- c("supplier_id", "supplier_name")
  df <- df[!rowSums(is.na(df[required_columns])) > 0, ]
```

```
  for(i in 1:nrow(df)){
    # Check for duplicate records based on the primary key
    existing_supplier_ids <- dbGetQuery(my_connection
              , sprintf("SELECT supplier_id FROM SUPPLIERS
                        WHERE supplier_id = '%s'", df$supplier_id[i]))
    if(nrow(existing_supplier_ids) > 0) {
      cat(sprintf("Skipping duplicate entry for supplier_id: %s\n"
                  , df$supplier_id[i]))
      next
    }

    insert_query <- sprintf("INSERT INTO SUPPLIERS (supplier_id, supplier_name,
                            supplier_address, supplier_phone, supplier_email)
                            VALUES ('%s', '%s', '%s', '%s', '%s')",
                            df$supplier_id[i], df$supplier_name[i],
                            df$supplier_address[i], df$supplier_phone[i],
                            df$supplier_email[i])
    tryCatch({
      dbExecute(my_connection, insert_query)
      cat(sprintf("Successfully inserted row: %d\n", i))
    }, error = function(e) {
      cat(sprintf("Error in inserting row: %d, Error: %s\n", i, e$message))
    })

  }
    dbDisconnect(my_connection)

}

for(file in suppliers_files) {

  df <- readr::read_csv(file)
  ingest_suppliers(df)

}
```

GIFT CARDS

```
ingest_gift_card_data <- function(df) {

  my_connection <- RSQLite::dbConnect(RSQLite::SQLite(),
                                      "../database/ecommerce_database_v1.db")
```

```
  # Validate 'gift_card_id' and 'gift_card_code' for null values
  required_columns <- c("gift_card_id", "gift_card_code", "status")
  df <- df[!rowSums(is.na(df[required_columns])) > 0, ]

  # Ensure 'detail' is an integer
  df$detail <- as.numeric(df$detail)

  # Insert validated data into the database
  for(i in 1:nrow(df)){
    # Check for duplicate records based on the primary key
    existing_ids <- dbGetQuery(my_connection, sprintf("SELECT gift_card_id FROM
                    GIFT_CARD WHERE gift_card_id = '%s'", df$gift_card_id[i]))
    if(nrow(existing_ids) > 0) {
      cat(sprintf("Skipping duplicate entry for gift_card_id: %s\n",
                  df$gift_card_id[i]))
      next
    }

    insert_query <- sprintf("INSERT INTO GIFT_CARD (gift_card_id,
              gift_card_code, detail, status) VALUES ('%s', '%s', %f, '%s')",
          df$gift_card_id[i], df$gift_card_code[i], df$detail[i], df$status[i])
    tryCatch({
      dbExecute(my_connection, insert_query)
      cat(sprintf("Successfully inserted row: %d\n", i))
    }, error = function(e) {
      cat(sprintf("Error in inserting row: %d, Error: %s\n", i, e$message))
    })
  }
    dbDisconnect(my_connection)

}

for(file in gift_card_files) {

  df <- readr::read_csv(file)
  ingest_gift_card_data(df)

}
```

```
my_connection <- RSQLite::dbConnect(RSQLite::SQLite()
                                    , "../database/ecommerce_database_v1.db")
dbGetQuery(my_connection,"SELECT * FROM GIFT_CARD;")
```

PRODUCTS

```
ingest_products <- function(df) {

  my_connection <- RSQLite::dbConnect(RSQLite::SQLite()
                                      , "../database/ecommerce_database_v1.db")
  # Data type checks
  df$stock_quantity <- as.integer(df$stock_quantity)

  # Check for null values in NOT NULL columns
  required_columns <- c("product_id", "stock_quantity"
                        , "category_id", "supplier_id")
  df <- df[!rowSums(is.na(df[required_columns])) > 0, ]

  for(i in 1:nrow(df)){
    # Check for duplicate records based on the primary key and
    #foreign key constraints
    existing_product_ids <- dbGetQuery(my_connection
    , sprintf("SELECT product_id FROM PRODUCTS WHERE product_id = '%s'"
              , df$product_id[i]))
    if(nrow(existing_product_ids) > 0) {
      cat(sprintf("Skipping duplicate entry for product_id: %s\n"
                  , df$product_id[i]))
      next
    }

    # Construct and execute the insertion query
    insert_query <- sprintf("INSERT INTO PRODUCTS (product_id, product_name,
                            price, stock_quantity, category_id, supplier_id)
                            VALUES ('%s', '%s', %f, %d, '%s', '%s')",
                            df$product_id[i], df$product_name[i], df$price[i]
                  , df$stock_quantity[i], df$category_id[i], df$supplier_id[i])
    tryCatch({
      dbExecute(my_connection, insert_query)
      cat(sprintf("Successfully inserted row: %d\n", i))
    }, error = function(e) {
      cat(sprintf("Error in inserting row: %d, Error: %s\n", i, e$message))
    })
  }
    dbDisconnect(my_connection)
```

```
}

for(file in products_files) {

  df <- readr::read_csv(file)
  ingest_products(df)

}
```

```
my_connection <- RSQLite::dbConnect(RSQLite::SQLite()
                                    , "../database/ecommerce_database_v1.db")
dbGetQuery(my_connection,"SELECT * FROM PRODUCTS LIMIT 10;")
```

```
      product_id                          product_name price stock_quantity
1  5116-vjq-2956             Pampers Swaddlers Diapers    25            222
2  6718-hlo-4759       Huggies Natural Care Baby Wipes    10            424
3  2985-wrf-5782    Similac Pro-Advance Infant Formula    30            229
4  4625-mrp-9938       Philips Avent Soothie Pacifiers     5            216
5  4163-cos-4183           Bumkins Waterproof SuperBib     8            419
6  6949-zmb-6593 Aden + Anais Muslin Swaddle Blankets    20            215
7  8600-uzy-9324                     Gerber Baby Socks     5            431
8  1345-epw-6525  Nuby Mittens with Teething Surfaces     7            162
9  4488-xnr-2917             Hudson Baby Hooded Towels    12            122
10 7706-sdc-6511         Spasilk Soft Terry Washcloths     8            140
                  category_id                supplier_id
1  01HQZSYXN9NDEKZ0KDTXG7GWAR 01HQZS3CHR3Z0C3RDD0QYFT566
2  01HQZSYXN9NDEKZ0KDTXG7GWAR 01HQZS3CHZ74ZQCSDXCS7CBVAC
3  01HQZSYXN9NDEKZ0KDTXG7GWAR 01HQZS3CHX81N7E24DA6H2H5DW
4  01HQZSYXN9NDEKZ0KDTXG7GWAR 01HQZS3CHF5YHQ7PBD8T11XRG1
5  01HQZSYXN9NDEKZ0KDTXG7GWAR 01HQZS3CHWKK9ACW7KQ58MHMZ1
6  01HQZSYXN9NDEKZ0KDTXG7GWAR 01HQZS3CHWKK9ACW7KQ58MHMZ1
7  01HQZSYXN9NDEKZ0KDTXG7GWAR 01HQZS3CHZ74ZQCSDXCS7CBVAC
8  01HQZSYXN9NDEKZ0KDTXG7GWAR 01HQZS3CHR3Z0C3RDD0QYFT566
9  01HQZSYXN9NDEKZ0KDTXG7GWAR 01HQZS3CJ0MY496XC7CYHNBGTJ
10 01HQZSYXN9NDEKZ0KDTXG7GWAR 01HQZS3CHSG3EB7GENNYD7YQ2K
```

ORDER

```
ingest_orders <- function(df) {

  my_connection <- RSQLite::dbConnect(RSQLite::SQLite()
```

```
                                    , "../database/ecommerce_database_v1.db")

# Essential columns for validation
required_columns <- c("order_id", "order_status", "quantity")
df <- df[!rowSums(is.na(df[required_columns])) > 0, ]

for(i in 1:nrow(df)) {
  # Check for duplicate order_id
  existing_ids <- dbGetQuery(my_connection
                , sprintf("SELECT order_id FROM ORDERS WHERE order_id = '%s'"
                          , df$order_id[i]))
  if(nrow(existing_ids) > 0) {
    cat(sprintf("Skipping duplicate entry for order_id: %s\n"
                , df$order_id[i]))
    next
  }

  # Data validation for quantity
  if(!is.numeric(df$quantity[i]) || df$quantity[i] <= 0) {
    cat(sprintf("Skipping entry due to invalid quantity for order_id: %s\n"
                , df$order_id[i]))
    next
  }

  # Insert validated data into the database
  insert_query <- sprintf("INSERT INTO ORDERS (order_id, customer_id,
                          product_id, shipment_id, gift_card_id, payment_method,
                          quantity, order_timestamp, payment_timestamp,
                          order_status) VALUES ('%s', '%s', '%s', '%s', '%s',
                          '%s', %d, '%s', '%s', '%s')",
                          df$order_id[i], df$customer_id[i], df$product_id[i],
                          df$shipment_id[i], df$gift_card_id[i],
                          df$payment_method[i], df$quantity[i],
                          df$order_timestamp[i],
                          df$payment_timestamp[i], df$order_status[i])
  tryCatch({
    dbExecute(my_connection, insert_query)
    cat(sprintf("Successfully inserted row: %d\n", i))
  }, error = function(e) {
    cat(sprintf("Error in inserting row: %d, Error: %s\n", i, e$message))
  })
}
```

```
    dbDisconnect(my_connection)

}


# Assume orders_df is your DataFrame containing orders data
ingest_orders(orders_df)
```

```
my_connection <- RSQLite::dbConnect(RSQLite::SQLite()
                                    , "../database/ecommerce_database_v1.db")
dbGetQuery(my_connection,"SELECT * FROM ORDERS LIMIT 10;")
```

```
   order_id                customer_id    product_id
1  ORD-0001 01HQZS39MGRHCPC7K95E6KTJGB 8929-yyg-4614
2  ORD-0002 01HQZS3A94XFFP2XQZ3P67369X 8601-mlc-2687
3  ORD-0003 01HQZS37FXYKVPXTEHSSXCA45J 7061-iqm-9386
4  ORD-0004 01HQZS3AE1F0DQBSAK3DZ9PT4F 1795-vmr-7994
5  ORD-0005 01HQZS3A4S5CN19MN8GTTXVSFY 0814-ogd-1980
6  ORD-0006 01HQZS380VPJGDC5DS4GYCYRPK 8495-azs-7030
7  ORD-0007 01HQZS38SA77G5023DDM7WT71E 5108-bki-1924
8  ORD-0008 01HQZS37YQ65C5KWAJFDAWYG6R 1397-ful-8708
9  ORD-0009 01HQZS37FXYKVPXTEHSSXCA45J 4149-trx-9479
10 ORD-0010 01HQZS38SA77G5023DDM7WT71E 4363-wmt-3344
                         gift_card_id payment_method quantity order_timestamp
1  de86af99-d41c-4cfc-9a75-50523d6e7350             NA        4      2024-02-07
2  4607a953-6d29-4957-89ca-77ed067b2f7a             NA        1      2024-02-01
3  741a8469-921c-4947-858c-ff000ce30269             NA        5      2024-02-28
4  c650b9fe-1559-4298-92b1-ea89b1ba6009             NA        5      2024-02-10
5  01d9e612-f729-43ac-916a-3cb15ed37000             NA        1      2024-02-16
6  5f5e7589-e52a-4126-b77f-4a945b5eefaf     Debit Card        4      2024-02-08
7  4ccb387e-86bf-4a78-830e-288782a1bfe2     Debit Card        2      2024-02-29
8  15c757bf-ef28-44c2-a980-eacc25425ec2      Gift Card        3      2024-02-06
9  333a7c98-e189-43ea-902b-ac43d37eaccd             NA        1      2024-02-10
10 72500781-3631-4def-947e-745542cc08a5     Debit Card        5      2024-02-11
     payment_timestamp    order_status shipment_id
1  2024-02-07 04:00:00 Pending Payment          NA
2  2024-02-03 10:00:00 Pending Payment          NA
3  2024-03-01 11:00:00 Pending Payment          NA
4  2024-02-11 08:00:00 Pending Payment          NA
5  2024-02-18 21:00:00 Pending Payment          NA
6  2024-02-08 10:00:00       Delivered   SHIP00244
```

```
7  2024-02-29 05:00:00       Cancelled          NA
8  2024-02-06 01:00:00      Processing   SHIP00208
9  2024-02-12 02:00:00 Pending Payment          NA
10 2024-02-12 23:00:00       Cancelled          NA
```

SHIPMENTS

```
ingest_shipment_data <- function(df) {

  my_connection <- RSQLite::dbConnect(RSQLite::SQLite()
                                      , "../database/ecommerce_database_v1.db")
  # Validate 'shipment_id' and 'status' for null values
  required_columns <- c("shipment_id", "status")
  df <- df[!rowSums(is.na(df[required_columns])) > 0, ]


  # Insert validated data into the database
  for(i in 1:nrow(df)){
    # Check for duplicate records based on the primary key
    existing_ids <- dbGetQuery(my_connection
        , sprintf("SELECT shipment_id FROM SHIPMENT WHERE shipment_id = '%s'",
                  df$shipment_id[i]))
    if(nrow(existing_ids) > 0) {
      cat(sprintf("Skipping duplicate entry for shipment_id: %s\n"
                  , df$shipment_id[i]))
      next
    }

    insert_query <- sprintf("INSERT INTO SHIPMENT (shipment_id,
                        dispatch_timestamp, delivered_timestamp, status)
                        VALUES ('%s', '%s', '%s', '%s')",
                            df$shipment_id[i], df$dispatch_timestamp[i]
                            , df$delivered_timestamp[i], df$status[i])
    tryCatch({
      dbExecute(my_connection, insert_query)
      cat(sprintf("Successfully inserted row: %d\n", i))
    }, error = function(e) {
      cat(sprintf("Error in inserting row: %d, Error: %s\n", i, e$message))
    })
  }
    dbDisconnect(my_connection)
```

```
}

ingest_shipment_data(shipment_df)
```

```
my_connection <- RSQLite::dbConnect(RSQLite::SQLite()
                                    , "../database/ecommerce_database_v1.db")
dbGetQuery(my_connection,"SELECT * FROM SHIPMENT LIMIT 10;")
```

```
   shipment_id dispatch_timestamp delivered_timestamp             status
1    SHIP00297         2024-03-17                  NA         In Transit
2    SHIP00496                 NA                  NA Ready for Dispatch
3    SHIP00131         2024-02-20          2024-02-25          Delivered
4    SHIP00646         2024-03-16                  NA   Out for Delivery
5    SHIP00422         2024-03-17                  NA         In Transit
6    SHIP00236         2024-02-04          2024-02-09          Delivered
7    SHIP00888         2024-03-17                  NA         In Transit
8    SHIP00905         2024-03-17                  NA         In Transit
9    SHIP00658         2024-03-17                  NA         In Transit
10   SHIP00901         2024-03-17                  NA         In Transit
```

2.2.1 Check Referential Integrity

ORDERS customer_id check

```
dbGetQuery(my_connection,
           "SELECT
              DISTINCT o.customer_id as customer_id,
              c.customer_id as customer_id,
              first_name ||' '|| last_name as customer_name
           FROM ORDERS as o
           LEFT JOIN CUSTOMERS as c ON c.customer_id = o.customer_id
           WHERE c.customer_id is NULL
           ;")
```

```
[1] customer_id   customer_id   customer_name
<0 rows> (or 0-length row.names)
```

product_id check

```
dbGetQuery(my_connection,
           "SELECT
              DISTINCT o.product_id as product_id,
              p.product_id as product_id,
              product_name as product_name
           FROM ORDERS as o
           LEFT JOIN PRODUCTS as p ON o.product_id = p.product_id
           WHERE p.product_id is NULL
           ;")
```

```
     product_id product_id product_name
1 5960-rcb-6909       <NA>         <NA>
2 0228-vgx-5140       <NA>         <NA>
3 9167-tzl-8984       <NA>         <NA>
4 2333-eej-0339       <NA>         <NA>
5 9151-uch-7821       <NA>         <NA>
6 5610-pcv-3034       <NA>         <NA>
```

gift_card_id

```
dbGetQuery(my_connection,
           "SELECT
              DISTINCT o.gift_card_id as gif_card_id,
              g.gift_card_id,
              gift_card_code
           FROM ORDERS as o
           LEFT JOIN GIFT_CARD as g ON g.gift_card_id = o.gift_card_id
           WHERE o.gift_card_id is NULL
           ;")
```

```
[1] gif_card_id    gift_card_id   gift_card_code
<0 rows> (or 0-length row.names)
```

shipment_id

```
dbGetQuery(my_connection,
           "SELECT
              DISTINCT o.shipment_id as x,
              s.shipment_id
           FROM ORDERS as o
```

```
            LEFT JOIN SHIPMENT as s ON s.shipment_id = o.shipment_id
            WHERE o.shipment_id is NULL
            ORDER BY o.shipment_id
            ;")
```

```
[1] x           shipment_id
<0 rows> (or 0-length row.names)
```

PRODUCTS supplier_id

```
dbGetQuery(my_connection,
            "SELECT
               DISTINCT p.supplier_id,
               s.supplier_id as a,
               s.supplier_name
            FROM PRODUCTS as p
            LEFT JOIN SUPPLIERS as s ON p.supplier_id = s.supplier_id
            WHERE s.supplier_id is NULL
            ORDER BY p.supplier_id
            ;")
```

```
                 supplier_id    a supplier_name
1 01HQZS3CJJMZ8VE8FSFV12394Q <NA>          <NA>
2 01HQZS3CJSA14X7CFXR9GN7HJJ <NA>          <NA>
3 01HQZS3CK7TNQY984CRWZ2YWYH <NA>          <NA>
4 01HQZS3CP6J1E2W3K754ED8TSV <NA>          <NA>
5 01HQZS3CWAANK3HMDV70KFNRTE <NA>          <NA>
6 01HQZS3CZ808EDV2QSZ7EC6RGQ <NA>          <NA>
7 01HQZS3D2JCXJ0GKKPY6JT5RMM <NA>          <NA>
```

category_id

```
dbGetQuery(my_connection,
            "SELECT
               DISTINCT p.category_id,
               c.category_id as c,
               cat_name
            FROM PRODUCTS as p
            LEFT JOIN PRODUCT_CATEGORY as c ON c.category_id = p.category_id
            WHERE p.category_id is NULL
            ORDER BY p.category_id
            ;")
```

```
[1] category_id c           cat_name
<0 rows> (or 0-length row.names)
```

3 Part 3: Data Pipeline Generation

3.1 Task 3.1: GitHub Repository and Workflow Setup

3.2 Task 3.2: GitHub Actions for Continuous Integration

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

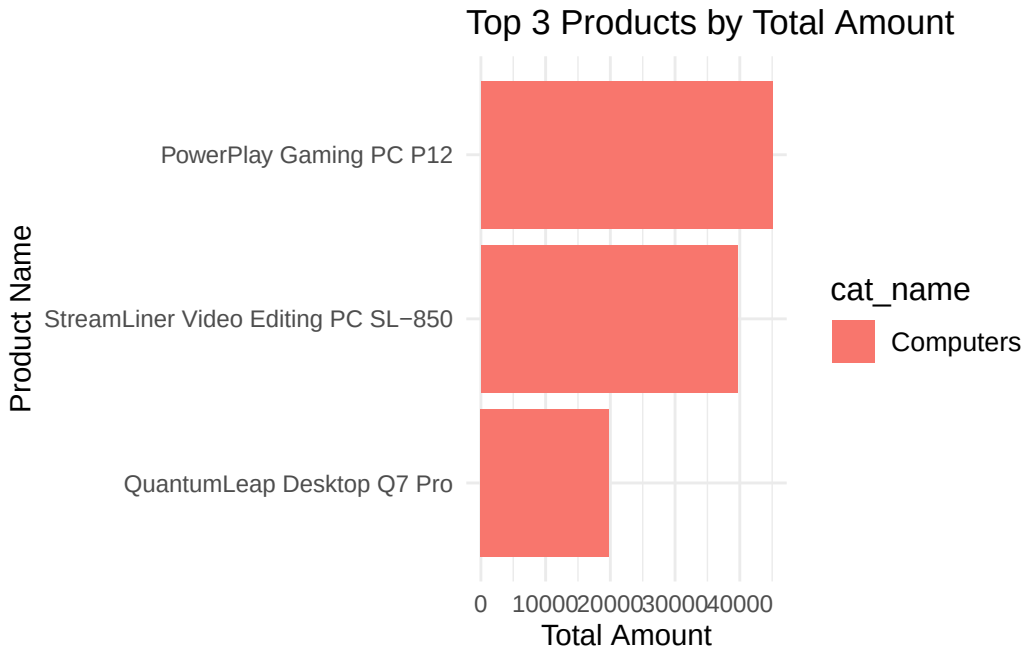
4.1 Task 4.1: Advanced Data Analysis in R

4.2 Task 4.2: Comprehensive Reporting with Quarto

1. Top 10 Products - Overall (Quantity)
2. Top 5 Categories (Quantity)
3. Top 3 Products across categories (Total Amount)

```
# Join orders with products to get category information
orders_with_category <- orders_df %>%
  inner_join(products_df, by = "product_id")

# Calculate total amount for each product
product_amounts <- orders_with_category %>%
  group_by(category_id, product_id, product_name) %>%
  summarise(total_amount = sum(quantity * price, na.rm = TRUE)) %>%
  ungroup()

# Join with category_df to get category names
product_amounts_with_category_name <- product_amounts %>%
  inner_join(category_df, by = "category_id")

# Get overall top 3 products
top_3_products <- product_amounts_with_category_name %>%
  arrange(desc(total_amount)) %>%
  slice_max(total_amount, n = 3) %>%
  ungroup()

# Plot using ggplot2
```

```
ggplot(top_3_products, aes(x = reorder(product_name, total_amount)
                           , y = total_amount, fill = cat_name)) +
  geom_bar(stat = "identity", position = position_dodge()) +
  coord_flip() +
  labs(title = "Top 3 Products by Total Amount",
       x = "Product Name",
       y = "Total Amount") +
  theme_minimal() +
  theme(legend.title = element_text(size = 12),
        legend.text = element_text(size = 10))
```



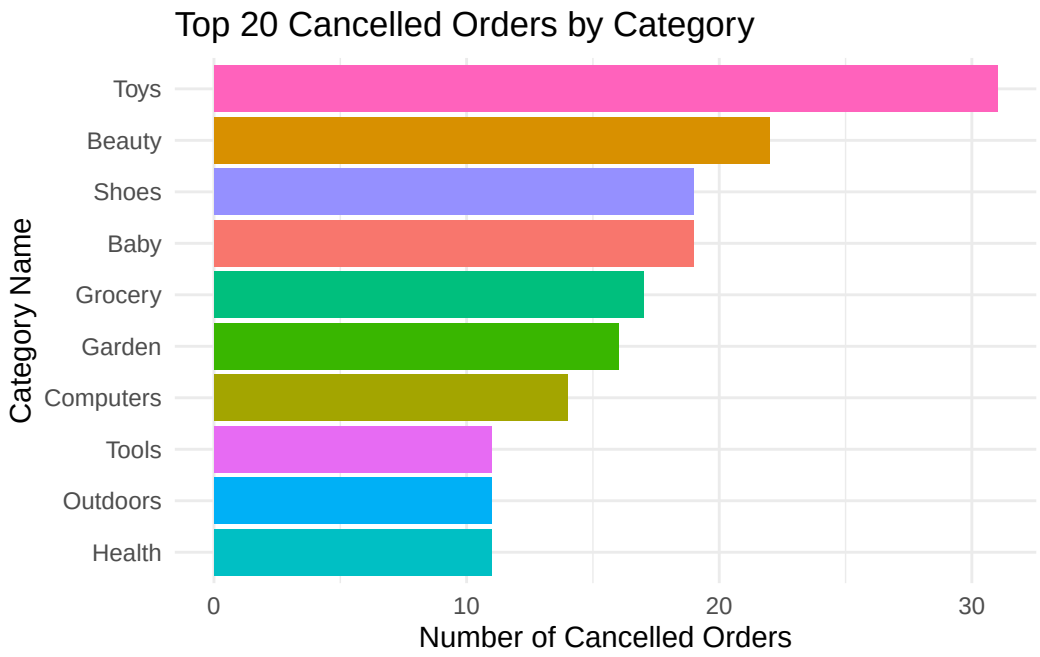
«««< HEAD

======= 4. Average delivery time for orders across top 5 delivery suppliers »»»> d065673478773d9575d62ba440477f02d2d07c48

5. Top 20 Average Spending across customers
6. Top 20 cancelled orders for which category

```
# Join orders with products and then with categories to get category information
orders_with_categories <- orders_df %>%
  inner_join(products_df, by = "product_id") %>%
  inner_join(category_df, by = "category_id")
```

```r
# Filter for cancelled orders and count by category
cancelled_orders_by_category <- orders_with_categories %>%
  filter(order_status == "Cancelled") %>%
  count(cat_name) %>%
  arrange(desc(n)) %>%
  top_n(20, n)

# Visualization
ggplot(cancelled_orders_by_category,
       aes(x = reorder(cat_name, n), y = n, fill = cat_name)) +
  geom_bar(stat = "identity") +
  coord_flip() +
  labs(title = "Top 20 Cancelled Orders by Category",
       x = "Category Name",
       y = "Number of Cancelled Orders") +
  theme_minimal() +
  theme(legend.position = "none")
```



7. Average number of orders across time
8. Scatter plot for revenue across quantity; color by category

SQL version 1. Top 10 Products - Overall (Quantity)

```
dbGetQuery(my_connection,
           "SELECT
              ORDERS.product_id,
              product_name,
              count(quantity) as total_purchase
           FROM ORDERS
           JOIN PRODUCTS ON ORDERS.product_id = PRODUCTS.product_id
           WHERE lower(order_status) in ('shipped','delivered')
           GROUP BY ORDERS.product_id,product_name
           ORDER BY total_purchase desc
           LIMIT 10
           ;")
```

```
      product_id
1  2707-vvd-3653
2  7528-nii-6646
3  0600-bhy-9237
4  6933-gij-4050
5  7270-txk-5871
6  1156-imm-9020
7  2406-xuo-6195
8  2901-cyy-6826
9  2982-wox-0527
10 4202-vwa-5608
                                                                 product_name
1                                                      AquaGuard Water Shoes X1
2                                                 LaughLearn Storybook Projector
3                                                        SweetHarmony Honey Jars
4                                                                  First Aid Kit
5                                                  QuantumLeap Desktop Q7 Pro
6                                                                Iron Supplements
7  EGO Power+ 56-Volt Lithium-Ion Cordless Electric String Trimmer (EGO)
8                          Jack Black Double-Duty Face Moisturizer SPF 20
9                                                  Cozy Cottage Starter Home
10                                SweetSensation Stevia Natural Sweetener
   total_purchase
1               6
2               6
3               5
4               5
5               5
6               4
```

```
7               4
8               4
9               4
10              4
```

2. Top 5 Categories (Quantity)

```
dbGetQuery(my_connection,
           "SELECT
              cat_name as category,
              count(quantity) as total_purchase
           FROM ORDERS
           JOIN PRODUCTS ON ORDERS.product_id = PRODUCTS.product_id
           JOIN PRODUCT_CATEGORY ON PRODUCTS.category_id = PRODUCT_CATEGORY.category_id
           WHERE lower(order_status) in ('shipped','delivered')
           GROUP BY cat_name
           ORDER by total_purchase desc
           LIMIT 5
           ;")
```

```
  category total_purchase
1 Clothing             25
2    Tools             23
3    Shoes             21
4  Grocery             21
5     Kids             20
```

3. Top 3 Products across categories (Total Amount)

```
dbGetQuery(my_connection,
           "WITH product AS (
              SELECT
                p.product_id,
                pc.cat_name,
                p.product_name
              FROM PRODUCTS as p
              JOIN PRODUCT_CATEGORY as pc ON pc.category_id = p.category_id
            ),
            order_amount AS (
              SELECT
                o.product_id AS product_id,
                SUM(o.quantity * p.price) AS total_amount
```

```
              FROM ORDERS as o
              JOIN PRODUCTS as p ON o.product_id = p.product_id
              WHERE LOWER(o.order_status) IN ('shipped', 'delivered')
              GROUP BY o.product_id
            ),
            rnk AS (
              SELECT
                pr.cat_name,
                pr.product_name,
                oa.total_amount,
                ROW_NUMBER() OVER (PARTITION BY pr.cat_name ORDER BY oa.total_amount DESC) AS
              FROM order_amount as oa
              JOIN product as pr ON oa.product_id = pr.product_id
            )
            SELECT
              cat_name,
              product_name,
              total_amount
            FROM rnk
            WHERE rnk IN (1,2,3);")
```

```
      cat_name
1   Automotive
2   Automotive
3   Automotive
4         Baby
5         Baby
6         Baby
7       Beauty
8       Beauty
9       Beauty
10       Books
11       Books
12       Books
13    Clothing
14    Clothing
15    Clothing
16   Computers
17   Computers
18   Computers
19 Electronics
20 Electronics
```

```
21 Electronics
22       Games
23       Games
24       Games
25      Garden
26      Garden
27      Garden
28     Grocery
29     Grocery
30     Grocery
31      Health
32      Health
33      Health
34        Home
35        Home
36        Home
37     Jewelry
38     Jewelry
39     Jewelry
40        Kids
41        Kids
42        Kids
43       Music
44       Music
45       Music
46    Outdoors
47    Outdoors
48    Outdoors
49       Shoes
50       Shoes
51       Shoes
52      Sports
53      Sports
54      Sports
55       Tools
56       Tools
57       Tools
58        Toys
59        Toys
60        Toys
```

```
                                product_name
1                  MotionSense Car Alarm System
2                    NightVision LED Headlights
```

3 BrakeBoss Performance Brakes
4 Nanit Plus Smart Baby Monitor and Wall Mount
5 Chicco KeyFit 30 Infant Car Seat
6 Braun Digital Ear Thermometer
7 Anastasia Beverly Hills Modern Renaissance Eyeshadow Palette
8 Jack Black Double-Duty Face Moisturizer SPF 20
9 Urban Decay All Nighter Setting Spray
10 CodeCrafters: Programming Puzzles & Challenges
11 SurvivalStrategies: Thriving in the Wild
12 CulinaryCreations: World Cuisine Made Easy
13 SummitStride Mountain Gear
14 Wanderlust World Traveler Backpacks
15 StealthMode Camouflage Apparel
16 QuantumLeap Desktop Q7 Pro
17 StreamLiner Video Editing PC SL-850
18 InfinityPad Tablet 12.9" Pro
19 VisionClear 4K Projector
20 AirSync Drone with HD Camera
21 QuickConnect Smart Door Lock
22 The Settlers of Catan
23 Dominion
24 Chess
25 Greenworks Pro 80V Cordless Backpack Leaf Blower (Greenworks)
26 EGO Power+ 56-Volt Lithium-Ion Cordless Electric String Trimmer (EGO)
27 Weber Original Kettle Premium Charcoal Grill (Weber)
28 SweetHarmony Honey Jars
29 SweetSensation Stevia Natural Sweetener
30 SunriseCereals Oatmeal Instant
31 First Aid Kit
32 Blood Glucose Monitoring Kit
33 Iron Supplements
34 EcoWash Low Water Dishwashers
35 EnergyEfficient Insulation Kits
36 HarmonyHome Sound Systems
37 Twilight Sparkle Diamond Necklace
38 Celestial Navigation Compass Necklace
39 Phoenix Feather Fire Opal Brooch
40 LaughLearn Storybook Projector
41 ZoomZoom Race Track
42 WonderWheels Miniature Car Collection
43 ForteFidelity Stereo Amplifier
44 MelodyMap Interactive Songbook
45 SonicScribe Music Composition Software

```
46                                           YETI Hopper Flip Portable Cooler
47                                                    Kelty Discovery 4 Tent
48                   Coleman RoadTrip 285 Portable Stand-Up Propane Grill
49                                         BreezeBlock Breathable Loafers B4
50                                            SportShift Basketball Shoes S3
51                                                  AquaGuard Water Shoes X1
52                                              SurfMaster Wetsuit Full Body
53                                          IronGrip Dumbbell Set Adjustable
54                                         GlideTech Ice Skates Professional
55                                         DustDeputy Shop Dust Collector D3
56                                                    CleanSweep Shop Vac C5
57                                            MultiMate Rotary Tool Kit R400
58                                                 Cozy Cottage Starter Home
59                                            Thomas & Friends Super Station
60                              Mini App-Enabled Programmable Robot Ball
   total_amount
1          1200
2           640
3           450
4          1750
5           750
6           105
7           280
8           216
9           140
10          240
11          100
12           75
13         1320
14          700
15          700
16        11700
17         5400
18         4900
19         1500
20         1500
21          500
22          150
23          120
24          105
25         2250
26         2200
27         1650
```

28	84
29	52
30	36
31	570
32	420
33	270
34	2700
35	1500
36	600
37	5000
38	4200
39	4000
40	450
41	300
42	100
43	1200
44	750
45	200
46	2500
47	1000
48	800
49	780
50	600
51	330
52	800
53	750
54	540
55	1400
56	1350
57	770
58	490
59	420
60	200

4. Average delivery time for orders across top 5 delivery suppliers

```
dbGetQuery(my_connection,
           "SELECT
               sup.supplier_id,
               sup.supplier_name AS supplier_name,
               AVG(julianday(s.delivered_timestamp) - julianday(s.dispatch_timestamp)) AS del
           FROM SHIPMENT AS s
           JOIN ORDERS AS o ON o.shipment_id = s.shipment_id
```

```
            JOIN PRODUCTS AS p ON p.product_id = o.product_id
            JOIN SUPPLIERS AS sup ON sup.supplier_id = p.supplier_id
            WHERE LOWER(s.status) = 'delivered'
            GROUP BY sup.supplier_id, sup.supplier_name
            ORDER BY delivery_time DESC, supplier_name
            LIMIT 5;")
```

```
                 supplier_id                supplier_name delivery_time
1 01HQZS3CPJM2B601GQBYY4P7W4                 Abbott-Harvey             5
2 01HQZS3CZB71YHDPM35V3QBBRB               Abshire-Torphy              5
3 01HQZS3D543D1CTXHV3Y0K6CZN                Armstrong-Fay              5
4 01HQZS3D37TG8J8QEYMQ264V9G               Bailey-Barrows              5
5 01HQZS3D4WJXGYQXWF8JV1DY9J Bartoletti, Hoeger and Rice               5
```

5. Top 20 Average Spending across customers

```
dbGetQuery(my_connection,
           "SELECT
              o.customer_id as customer_id,
              c.first_name ||' '|| c.last_name as customer_name,
              AVG(p.price*o.quantity) as avg_amount,
              SUM(p.price*o.quantity) as total_amount
            FROM ORDERS as o
            JOIN CUSTOMERS as c ON o.customer_id = c.customer_id
            JOIN PRODUCTS as p ON p.product_id = o.product_id
            WHERE LOWER(o.order_status) IN ('shipped', 'delivered')
            GROUP BY o.customer_id,customer_name
            ORDER BY avg_amount DESC
            limit 20
           ;")
```

```
                 customer_id     customer_name avg_amount total_amount
1  01HQZS37ZGCCFYSZQ6CMDAKBS3   Kori Middleweek  4000.0000         4000
2  01HQZS383VMTAR1JSDDY2PRMWA       Evie Durman  3600.0000         3600
3  01HQZS37Z67FFM7M3RANB5VH1Y     Darci Winsper  3000.0000         3000
4  01HQZS39N0SDVJ4RGTRK94RJ3J  Fredericka Eames  1527.0000         3054
5  01HQZS384BHGFPG39RGY9CS9G1     Darrin Peddel  1415.0000         2830
6  01HQZS39N5TG2YHMB2QR02XYMT    Raimund Bromet  1400.0000         1400
7  01HQZS39MGRHCPC7K95E6KTJGB     Benetta Chave  1246.6667         3740
8  01HQZS386YF3Y2F5C4F7SGSC1J  Hyman Dobrowlski  1235.0000         4940
9  01HQZS3A52VT13XS8D6NZDB6MB    Salem Kollasch  1200.0000         1200
```

```
10 01HQZS38314Z9NJ6C0C71PS2FB      Bea Dunseith  1087.5000         4350
11 01HQZS38SW54M5M4FFEB0SF1X3     Liliane Atger  1078.7500         4315
12 01HQZS39FVYFWSK9DP5DE94NX0     Godiva Jerams  1060.0000         2120
13 01HQZS38HFMS3QQJWEKBXJ7WBV         Ab Gaffer   970.0000         2910
14 01HQZS37DA6GSRZDJSGCM3ZJ1F   Sibby Kinneally   925.0000         1850
15 01HQZS3AD0FCQC1MC59ETFR9M9 Ingelbert Barnham   816.6667         2450
16 01HQZS38APRRJFXA6AJYT5RGYH    Chev Mulcaster   775.0000         1550
17 01HQZS39JFYTKB20NX044YPW80      Nydia Heliar   750.0000         1500
18 01HQZS389GH3JKCG6DAB1VJZ56       Paton Flawn   700.0000          700
19 01HQZS38QJBCBRXYQCFV4SN48Q Marissa Bridgland   640.0000         1280
20 01HQZS38TRK2WCNCHSGSPQ98N3       Lian Rodden   635.0000         1270
```

6. Top 20 cancelled orders for which category

```
dbGetQuery(my_connection,
           "SELECT
              cat_name,
              COUNT(o.quantity) as total_cancelled
            FROM ORDERS as o
            JOIN PRODUCTS as p ON p.product_id = o.product_id
            JOIN PRODUCT_CATEGORY as pc on pc.category_id = p.category_id
            WHERE LOWER(order_status) = 'cancelled'
            GROUP BY cat_name
            ORDER BY total_cancelled DESC
           ;")
```

```
      cat_name total_cancelled
1         Toys              15
2       Beauty              12
3        Tools              11
4       Sports              11
5         Kids              11
6      Jewelry              11
7      Grocery              11
8        Shoes              10
9        Games              10
10   Computers              10
11 Electronics               9
12    Outdoors               8
13      Health               8
14      Garden               8
15        Home               7
```

```
16    Clothing              5
17       Books              5
18  Automotive              4
19       Music              2
20        Baby              2
```

7. Average number of orders across time

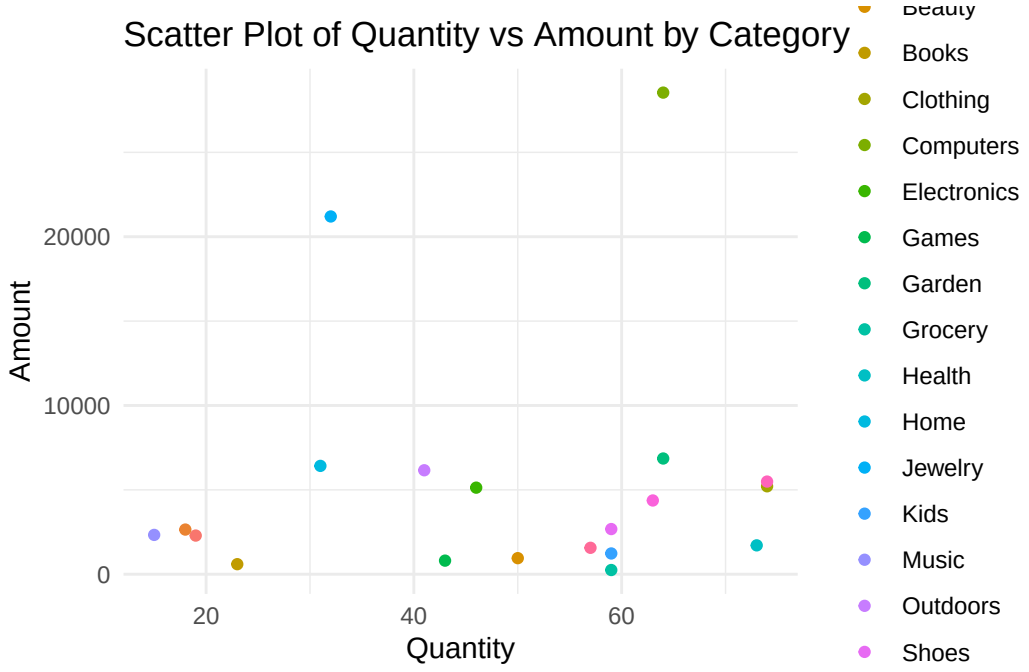
```
dbGetQuery(my_connection,
           "SELECT
              order_timestamp as date,
              SUM(o.quantity) as total_order
            FROM ORDERS as o
            WHERE LOWER(order_status) IN ('shipped', 'delivered')
            GROUP BY order_timestamp
            ORDER BY date
           ;")
```

```
         date total_order
1  2024-02-01          32
2  2024-02-02          32
3  2024-02-03          14
4  2024-02-04          50
5  2024-02-05          39
6  2024-02-06          15
7  2024-02-07          29
8  2024-02-08          28
9  2024-02-09          15
10 2024-02-10          27
11 2024-02-11          24
12 2024-02-12          20
13 2024-02-13          21
14 2024-02-14          57
15 2024-02-15          36
16 2024-02-16          37
17 2024-02-17          35
18 2024-02-18          52
19 2024-02-19          38
20 2024-02-20          40
21 2024-02-21          27
22 2024-02-22          33
23 2024-02-23          34
```

```
24 2024-02-24          65
25 2024-02-25          41
26 2024-02-26          43
27 2024-02-27          45
28 2024-02-28          43
29 2024-02-29          31
```

8. Scatter plot for revenue across quantity; color by category

```
revenue_quantity <- dbGetQuery(my_connection,
           "SELECT
              cat_name,
              SUM(o.quantity) as quantity,
              SUM(p.price * o.quantity) as amount
             FROM ORDERS as o
             JOIN PRODUCTS as p ON p.product_id = o.product_id
             JOIN PRODUCT_CATEGORY as pc on pc.category_id = p.category_id
             WHERE LOWER(order_status) IN ('shipped', 'delivered')
             GROUP BY cat_name
           ;")

ggplot(revenue_quantity, aes(x = quantity, y = amount, color = cat_name)) +
  geom_point() +
  theme_minimal() +
  labs(title = "Scatter Plot of Quantity vs Amount by Category",
       x = "Quantity",
       y = "Amount") +
  theme(legend.position = "right")
```




```
dbDisconnect(my_connection)
```