



MSBA305: Data Processing Framework - Sect. 1 SP23-24

Term Project

**Optimizing Inventory Management through Relational Database Integration and Predictive Analytics at Linkers**

**Presented to :**

Prof. Elie Semaan Nasser

**Prepared by:** Team 3

James Abbas

Ali Annan

Romanos Rizk

Sasha Nasser

# 

[**Problem Statement: 3**](#_j5chi4cl0zaw)

[**Literature Review: 4**](#_orlr4tte0oc7)

[**Proposed Solution: 6**](#_46ew0ggstmmz)

[**Methodology: 7**](#_t0zedrftbjhh)

[General Preprocessing: 7](#_qgxlvaw443ug)

[DataFrame Creation: 8](#_9vo17ohcc5vy)

[Customer Table: 8](#_kzsx67tfv0lk)

[Order Table: 9](#_eohv6aclxu45)

[Product Table: 9](#_dgfnnnj3r2bv)

[Product Categories: 10](#_bm4lyeckq4yr)

[MySQL: 11](#_t1uaiafrg48s)

[Business Rules: (Extracted From the Company) 11](#_1eg4jigdchuy)

[Identify Entities and Attributes: 12](#_4iyvsdbu2vc7)

[Define Relationships: 16](#_vsz461eswq2p)

[Define the Table Constraints According to the Attributes and to the Business Rules 17](#_ds9hnlkcttdw)

[Create an Entity-Relationship (ER) Diagram: 21](#_p26dp8am09xr)

[Create the Database Schema: 26](#_hwmsktwkdnvi)

[Business Scenario Tests to Test the Database: 27](#_aja3zeh65ii5)

[**Prediction Model: 31**](#_hhukl834dlfu)

[**Visualization Dashboard: 31**](#_p1ur9g9bqh98)

[**Limitations & Recommendations: 34**](#_ysd87ek0iro5)

[Recommendations: 34](#_zamzkl8q0yet)

[Limitations: 35](#_z0vg4ldnvr7b)

[**Results and Discussions: 35**](#_ecj87ffbwrni)

[**References 36**](#_nntlinfye1c1)

[**Appendix: 37**](#_l1dnhaoe8v0o)

[Detailed Schema: 37](#_3c82j5oco85d)

[Python Code: 46](#_hb2jiz6lpkao)

# 

# Problem Statement:

Wholesale appliance companies face significant challenges in accurately forecasting the inventory needed to meet market demands due to limited access to direct customer data. This data, available only to retailers, contains essential insights into market trends and evolving appliance specifications that influence customer purchasing decisions. Wholesale companies typically rely on order transactions from retailers, which provide limited details such as pricing, quantity, and appliance specifications. Without direct access to granular customer data, wholesalers struggle to make precise predictions, leading to potential inventory imbalances, including shortages and overstocking. Additionally, the existing literature primarily addresses the business-to-customer aspect of the supply chain, neglecting the business-to-business relationships that are crucial for wholesalers. This oversight leaves a gap in guidance for wholesalers who need effective strategies to navigate these challenges. It is vital for wholesale companies to bridge this information gap and develop informed and responsive inventory management practices that align with dynamic market trends and customer preferences across all types of appliances.

Established in 1993, Linkers is a company that developed and expanded its activities to become through the years a highly recognized wholesale company in Lebanon in the household appliances and consumer electronics fields. However, up until this day Linkers has not incorporated any kind of database management system into their business. They continue to input all inventory and order information into Excel manually. Wholesale appliance companies like Linkers need forecasting the appliances sold to later meet market demands. Without direct access to customer data Linkers struggles to extract data into analytical data stores such as Python. Linkers needed to design a database to track orders from their shops. After speaking to Linkers it was clear that the company was in need of a relational database in which to store its customer and sales data. However, they did not know how this database should look or take shape. They listed certain needs from their database.

**At linkers:**

**People can place an order of multiple products. They need to record where the order was placed from, who by, at what time and how much. Finally, they want to track the name and price of products and the quantity of each product for each order.**

By leveraging an open source database such as MySQL, we proposed a database layout that would meet their future needs. This required the assembly of entities between specific tables and the forming of connections between MySQl and analytical stores such as Python. This approach will prevent any problems when extracting data specific to multiple tables and meet the needs of the company going forward.

# Literature Review:

The problem statement highlights the difficulties wholesale appliance companies face due to limited customer data access, resulting in inaccurate inventory forecasting and potential stock imbalances. The research article by Chuning Deng and Yongji Liu (2021) proposes a solution using a Deep Learning-Based Inventory Management and Demand Prediction Optimization Method that leverages Long Short-Term Memory (LSTM) models to enhance inventory forecasting accuracy. This method transforms order data into a supervised learning problem, allowing wholesalers to predict demand trends with an accuracy that significantly surpasses traditional models, reducing inventory costs by about 25% (Deng & Liu, 2021). Such an approach directly addresses the core challenges identified in the problem statement by providing an effective strategy for wholesalers to manage inventory dynamically and responsively, even without direct customer insights.In essence, the integration of LSTM into inventory management offers a robust tool for wholesalers to navigate the complexities of supply chain demands, aligning closely with the need for better predictive accuracy and operational efficiency in inventory management as outlined in the problem statement (Deng & Liu, 2021).

Tang et al. (2023) address the challenge of inventory forecasting in the complex, dynamic environment of cross-border e-commerce, a crucial issue also highlighted in your problem statement regarding wholesale appliance companies. This study leverages artificial intelligence (AI) models, particularly the Extreme Gradient Boosting (XGBoost), to enhance inventory forecasting accuracy by processing intensive data streams. The research demonstrates that AI-driven models can significantly outperform traditional forecasting methods, providing more accurate predictions and effectively reducing inventory misalignments such as overstocking or shortages (Tang et al., 2023). The use of AI in inventory forecasting aligns with the needs identified in your problem statement by offering a sophisticated approach to handle the variability and unpredictability of market demands. By incorporating machine learning techniques, Tang et al. (2023) provide a methodological framework that could potentially be adapted by wholesale companies to improve their inventory management practices. This adaptation can lead to better alignment with market trends and customer preferences, thus enhancing overall operational efficiency and responsiveness in the supply chain.

Scott Curtiss (2024) identifies significant inefficiencies in the traditional approaches to wholesale forecasting, arguing for the adoption of channel-specific forecasting to improve accuracy and responsiveness. Curtiss critiques the one-size-fits-all forecasting model that fails to account for the distinct behaviors and needs of different customer segments, leading to supply-demand mismatches and financial losses. This approach aligns with the broader industry recognition that differentiated strategies are needed for diverse market segments to maintain competitiveness and profitability. Channel-specific forecasting addresses these issues by creating individual demand forecasts for distinct customer groups, which allows wholesalers to tailor inventory and operational strategies more effectively. This method is particularly advantageous in managing the complexities of modern supply chains, where consumer demand is increasingly volatile and influenced by a wide range of factors, including seasonal trends and promotional activities. Furthermore, Curtiss (2024) discusses the strategic benefits of this forecasting approach in fostering better collaboration with retail partners. By encouraging retailers to share critical sales data, wholesalers can refine their forecasts further, enhancing overall supply chain efficiency and customer satisfaction. This collaborative approach not only optimizes inventory levels but also strengthens relationships and trust between wholesalers and their clients, essential for long-term business success.

In the article "Effective Inventory Forecasting: Tools and Techniques for Manufacturing Leaders," Damini (2023) emphasizes the critical role of inventory forecasting in manufacturing, outlining various tools and techniques that leaders can use to enhance forecasting effectiveness and drive operational efficiency. The author highlights the importance of understanding customer demand through predictive analytics and leveraging modern technologies such as artificial intelligence (AI) and machine learning (ML) for more accurate demand projections. Damini (2023) advocates for the integration of historical data analysis with real-time data inputs to adjust inventory levels dynamically, thus avoiding overstocking and stockouts. The article discusses several advanced forecasting models, including time series analysis, regression models, and AI-driven predictions that can handle complex data patterns and provide insights into future demand with a higher degree of accuracy. Moreover, the article points out the benefits of collaborative forecasting with suppliers and customers, which not only improves forecast accuracy but also strengthens supply chain relationships and operational resilience (Damini, 2023). This collaborative approach, combined with the use of sophisticated forecasting tools, allows manufacturing leaders to optimize their inventory levels, reduce costs, and meet customer demands effectively.

# Proposed Solution:

To overcome the challenges of variability in the market, one has to stay ahead. To compete, companies must now predict or forecast sales of their products. When one can safely know how much of their inventory will be sold in the upcoming weeks, months or years then one can restock at strategic points in the sale year. For example, if one were to know or predict the current demand of televisions, then choosing the correct restocking amount of appliances becomes an area to compete. Less restocking means cheaper transportation costs throughout the year. Furthermore, knowing demand allows one to better prevent stock outs or ensure that any customer who comes in looking for a particular appliance will find one.

To compete in this way of course requires big data, on past sales, prices, customer type and appliance. With so much data available out there it is imperative to store this data all in one place such as a database. MySQL is an open-source relational database management system (RDBMS) that allows users to store, manage, and retrieve data. It uses Structured Query Language (SQL) to interact with the database, making it efficient for managing large volumes of data. MySQL is known for its scalability, reliability, and speed, making it a popular choice for businesses and organizations of all sizes. It offers various features such as different data types, integration capabilities, security features, and compatibility with different technologies, making it a versatile and powerful tool for managing databases.

To sum up , using MySQL can be highly beneficial for a small business like linkers due to its scalability, reliability, and cost-effectiveness. Additionally, MySQL's open-source nature means there are no licensing fees, making it a cost-effective choice for small businesses with limited budgets. Our proposed solution is to design this database around Linkers’ company.

# Methodology:

Preprocessing the data was the first step. To eventually use MySQL and input Linkers’ data into a well organized database first we had to ensure all the data inputs were useful and readable. To ensure this preprocessing involved several steps for us.

## General Preprocessing:

1. **Transforming Column Names:**
   * To begin our preprocessing, we decided to standardize the column names in our dataframe to make them more consistent. This helps in enhancing the readability of our dataset. Following convention also ensures smooth integration as well as compatibility of our dataframe into MySQL.
   * The methods we used to achieve this normalization of our columns were the following:

* Stripping White Space (leading or trailing)
* Adding Underscores ‘\_’ Between Words
* Lowercasing Words
* Uppercasing the First Letter

1. **Missing Values:**
   * Missing values can lead to biased results and errors in analysis. We identified 3 columns that contain missing values.

* ‘Discount’: We replaced all missing values with 0. After our discussions with the company, discounts were only given starting orders that are greater than 5 items.
* ‘Payment Terms’: All missing values were replaced with the mode of the column.
* ‘Tot\_qty’: In the column for the total quantity, all missing values were dropped.

1. **Converting ‘Date’ to Date Time Type:**
   * To begin with, our order ‘Date’ values were stored as strings and needed to be converted. Storing dates as date/time variables allows for more meaningful date-based analyses.
   * It enables sorting, filtering, and aggregating data based on dates, which is often crucial for time-series analysis and trend identification of sales.
2. **Normalizing Categorical Columns:**

* Diving into our categorical columns we need to ensure consistency while working with these entries throughout the database. To do that, some standardizations were applied.
* Lowercasing Words
* Stripping White Spaces

1. **Illogical Values:**

* Some entries in the dataset contained illogical values (either negative values or 0) when it comes to the quantity and prices columns.
* This is why rows with entries of 0 and negative values in the ‘Tot\_qty’, ‘Unit\_price’, ‘Total\_price’ columns were dropped.

1. **Normalizing ‘Description’ Column:**

* Looking into our ‘Description’ column, it contains valuable information regarding product specifications and features that can be extracted and used.
* However, in order to do that, some standardizations of the entries must be done to ensure consistency and facilitate the extraction process.
* Lowercase Words
* Stripping White Spaces
* Removing Punctuations

## DataFrame Creation:

### Customer Table:

1. **Creating Customer Table:**

* In our application of trying to align managerial practices with customer trends, it is essential to easily access data about the customers.
* This is why a Customer DataFrame was created.

1. **Dropping Duplicate Customer ID:**

* Once the DataFrames were created, the rows containing duplicate Customer ID were dropped. In other words, only one record for each Customer ID was left. This was done mainly:
* Due to the Primary Key rule in MySQL which states that the Primary Key cannot be replicated.
* To avoid redundancy of information since the DataFrame would only contain personal information about the customers, which are identical for each Customer ID. (So having 2 instances of the same Customer ID is useless)

1. **Removing Irrelevant Columns:**

* Many columns were removed. Columns that were not relevant or specific to the individual customers as well as columns that are redundant were dropped within these DataFrames.

### Order Table:

1. **Creating Order Table:**

* Tracking the inflow of orders, their quantities and the revenue generated from transactions is a major player in any business. In order for the company to bridge the gap in its information system, it needs to clearly detail and track its order history.
* This is why an Order DataFrame was created.

1. **Removing Irrelevant Columns:**

* Many columns were removed. Columns that were not relevant or specific to the order transactions as well as columns that are redundant were dropped within these DataFrames.

### Product Table:

1. **Creating Product Table:**

* In order for a company to continuously monitor its performance, it needs to maintain and keep track of its product catalog.
* This is why a Product DataFrame was created.

1. **Dropping Duplicate Item Codes:**

* Once the DataFrames were created, the rows containing duplicate Item Codes were dropped. In other words, only one record for each Item Code was left. This was done mainly:
* Due to the Primary Key rule in MySQL which states that the Primary Key cannot be replicated.
* To avoid redundancy of information since the DataFrame would only contain the descriptions and features of the products, which are identical for each Item Code. (So having 2 instances of the same Item Code is useless)

1. **Removing Irrelevant Columns:**

* Many columns were removed. Columns that were not relevant or specific to the product catalog as well as columns that are redundant were dropped within these DataFrames.

### Product Categories:

1. **Creating 24 Product Category DataFrames:**

* By filtering the ‘Family’ column, which contains 24 distinct product categories, we were able to extract 24 distinct DataFrames specific to each product category.
* Standardizing each category name within ‘Family’ and then creating separate DataFrames for each one, makes data manipulation easier and smoother.
* Lowercase Words
* Stripping White Spaces
* Removing Backslashes
* Replacing Spaces Between Words with ‘\_’

1. **Dropping Duplicate Item Codes:**

* Once the DataFrames were created, the rows containing duplicate Item Codes were dropped. In other words, only one record for each Item Code was left. This was done mainly:
* Due to the Primary Key rule in MySQL which states that the Primary Key cannot be replicated.
* To avoid redundancy of information since these DataFrames would only contain the descriptions and features of the products, which are identical for each Item Code. (So having 2 instances of the same Item Code is useless)

1. **Removing Irrelevant Columns:**

* Across the 24 newly created product DataFrames, many columns were removed. Columns that were not relevant or specific to the individual product categories as well as columns that are redundant were dropped within these DataFrames.

1. **Exploding a Column (Description):**
   * The ‘Description’ column holds valuable information about the type of product the customer was ordering. It contains this information as a string.
   * Specific features that are relevant to each individual product category were extracted from the description.
   * This process expands the dataset, making it easier to analyze and extract insights from the nested data.
2. **Fixing the DataTypes:**

* The DataTypes of the extracted columns were adjusted to either integer, float or string in order to generate more meaningful insights from the data.

1. **Dropping ‘Description’ Column:**

* After the feature extraction from the ‘Description’ column, there is a redundancy of information within the DataFrames. To overcome this, the ‘Description’ column was dropped from all 24 DataFrames.

Overall, these preprocessing steps are important for ensuring that the data is clean, complete, and structured in a way that facilitates meaningful analysis and modeling.

## MySQL:

### Business Rules: (Extracted From the Company)

1. A customer must place an order.
2. A customer may place many orders.
3. An order cannot belong to more than 1 customer.
4. An order must contain 1 Item Code.
5. An Item Code may be in many order.
6. A product may be a Wine Cooler
7. A product may be an AC
8. A product may be a DVD
9. A product may be a Sound Bar
10. A product may be a Humidor
11. A product may be a Freezer
12. A product may be a LED
13. A product may be a Cooker
14. A product may be a Dishwasher
15. A product may be a Dispenser
16. A product may be a Dryer
17. A product may be an Electric Oven
18. A product may be a Hood
19. A product may be a Laundry
20. A product may be a Microwave
21. A product may be an Oven
22. A product may be a Top Load
23. A product may be a Table Top
24. A product may be a Twin Tub
25. A product may be a Vacuum
26. A product may be a Multi Door
27. A product may be a Refrigerator
28. A product may be a Hob
29. A product may be a Dap

Designing a database in MySQL for Linkers involves several steps, including identifying the entities and their attributes, creating an entity-relationship (ER) diagram, and defining the database schema.

### Identify Entities and Attributes:

**Customer:**

Represents information about customers who make purchases in your store.

* **Casa:** Customer's home address.
* **Region:** Customer's region.
* **Customer\_id:** Unique identifier for each customer.
* **Phone\_number:** Customer's phone number.
* **Customer\_loyalty\_indicator:** Indicator for customer loyalty.

**Product:**

Holds details about various products available in your store.

* **Item\_code:** Unique identifier for each product.
* **Product\_class:** Classification of the product.
* **Brand:** Brand of the product.
* **Unit\_price:** Price of the product.
* **Description:** Description of the product.
* **Tot\_qty:** Total quantity of the product in stock.
* **Family:** Product family or category.

**Order:**

Stores information about orders placed by customers.

* **Order\_id**: Unique identifier for each order.
* **Date:** Date of the order.
* **Discount:** Discount applied to the order.
* **Customer\_id**: Identifier of the customer who placed the order.
* **Item\_code:** Identifier of the product in the order.
* **Tot\_qty:** Total quantity of the product in the order.
* **Return\_rate:** Rate of returns for the order.
* **Total\_price:** Total price of the order.

**Wine Cooler:**

Represents details specific to wine coolers.

* **Type:** Type of wine cooler.
* **Capacity:** Capacity of the wine cooler.
* **Item\_code:** Unique identifier for each product.

**Air Conditioner (AC):**

Holds information about air conditioners.

* **Inverter:** Inverter technology availability.
* **BTU:** British Thermal Units, a measure of cooling capacity.
* **Item\_code:** Unique identifier for each product.

**DVD Player:**

Contains information about DVD players.

* **HDMI:** Availability of HDMI port.
* **Item\_code:** Unique identifier for each product.

**Sound Bar:**

Represents details specific to sound bars.

* **Amplifier\_Output:** Amplifier output power.
* **Item\_code:** Unique identifier for each product.

**Humidor:**

Holds information about humidors for storing cigars.

* **Capacity:** Capacity of the humidor.
* **Item\_code:** Unique identifier for each product.

**Freezer:**

Contains information about freezers.

* **Number\_of\_Drawers:** Number of drawers in the freezer.
* **No\_Frost:** Availability of frost-free feature.
* **Freezer\_Type:** Type of freezer.
* **Item\_code:** Unique identifier for each product.

**LED TV:**

Holds details about LED televisions.

* **Size:** Size of the LED TV.
* **Smart:** Availability of smart features.
* **Resolution:** Resolution of the LED TV.
* **Item\_code:** Unique identifier for each product.

**Cooker:**

Represents information about cookers.

* **Dimension:** Dimensions of the cooker.
* **Bottle\_Compartment**: Availability of bottle compartment.
* **Full\_Safety:** Indicates whether the cooker has full safety features.
* **Item\_code:** Unique identifier for each product.

**Dishwasher**:

Contains details about dishwashers.

* **Settings:** Number of settings available.
* **Controls:** Type of controls for the dishwasher.
* **Item\_code:** Unique identifier for each product.

**Dispenser:**

Represents information about dispensers.

* **Number\_of\_Faucets**: Number of faucets in the dispenser.
* **Load\_Type:** Type of load for the dispenser.
* **Item\_code:** Unique identifier for each product.

**Dryer:**

Holds information about dryers.

* **Functionality**: Functionality of the dryer.
* **Size:** Size of the dryer.
* **Item\_code:** Unique identifier for each product.

**Electric Oven:**

Contains details about electric ovens.

* **Size:** Size of the electric oven.
* **Item\_code:** Unique identifier for each product.

**Hood:**

Represents information about hoods.

* **Size:** Size of the hood.
* **Type:** Type of hood.
* **Item\_code:** Unique identifier for each product.

**Laundry Machine:**

Holds details about laundry machines.

* **Speed:** Speed of the laundry machine.
* **Inverter:** Inverter technology availability.
* **Type:** Type of laundry machine.
* **Item\_code:** Unique identifier for each product.

**Microwave Oven:**

Contains details about microwave ovens.

* **Built\_in:** Indicates whether the microwave is built-in.
* **With\_Grill:** Indicates whether the microwave has a grill.
* **With\_Oven:** Indicates whether the microwave has an oven.
* **Control\_Type:** Type of control for the microwave.
* **Item\_code:** Unique identifier for each product.

**Oven:**

Represents information about ovens.

* **Built\_in:** Indicates whether the oven is built-in.
* **With\_Grill:** Indicates whether the oven has a grill.
* **Functionality:** Functionality of the oven.
* **Item\_code:** Unique identifier for each product.

**Top Load Washing Machine:**

Contains details about top load washing machines.

* **Size:** Size of the top load washing machine.
* **Item\_code:** Unique identifier for each top load washing machine.

**Table Top:**

Represents information about table top appliances.

* **Item\_code:** Unique identifier for each table top appliance.
* **Size:** Size of the table top appliance.
* **Color:** Color of the table top appliance.
* **Type:** Type of the table top appliance.

**Twin Tub Washing Machine:**

Holds information about twin tub washing machines.

* **Size:** Size of the twin tub washing machine.
* **Item\_code:** Unique identifier for each top load washing machine.

**Vacuum Cleaner:**

Represents information about vacuum cleaners.

* **Item\_code:** Unique identifier for each vacuum cleaner.
* **Power:** Power of the vacuum cleaner.
* **Type:** Type of vacuum cleaner.
* **Color:** Color of the vacuum cleaner.
* **Bagless:** Indicates whether the vacuum cleaner is bagless.

**Multi-door Refrigerator:**

Contains details about multi-door refrigerators.

* **Item\_code:** Unique identifier for each multi-door refrigerator.
* **Number\_of\_Doors:** Number of doors in the refrigerator.
* **Size:** Size of the refrigerator.
* **Color:** Color of the refrigerator.
* **No\_Frost:** Indicates whether the refrigerator has a frost-free feature.
* **Type:** Type of refrigerator.

**Refrigerator:**

Represents information about refrigerators.

* **Size:** Size of the refrigerator.
* **Color:** Color of the refrigerator.
* **No\_Frost:** Availability of frost-free feature.
* **Number\_of\_Doors:** Number of doors in the refrigerator.
* **Item\_code:** Unique identifier for each refrigerator.
* **Item\_code:** Unique identifier for each product.

**Hob:**

Holds details about hobs**.**

* **Size:** Size of the hob.
* **Color:** Color of the hob.
* **Number\_of\_Burners:** Number of burners in the hob.
* **Safety:** Safety features of the hob.
* **Built\_in:** Indicates whether the hob is built-in.
* **Item\_code:** Unique identifier for each hob.

**DAP (Domestic Appliances):**

Contains details about domestic appliances.

* **Category:** Category of the domestic appliance
* **Item\_code:** Unique identifier for each domestic appliance.

### Define Relationships:

The company’s database manages information on customers, products, orders, and product categories:

* The table named **'customer'** contains information regarding customers, such as their distinctive Customer\_id, contact details, and loyalty indicators. Each customer can place several orders, which are represented by the order table.
* The table named **'order'** holds data about every order, comprising the ‘Order\_id’, date, overall cost, and connected customer via the ‘Customer\_id’ foreign key.
* Tracking of detailed product information such as brand, price, and description is enabled through the **'product'** table.
* Products are classified into different types and each type is represented in a specific table with a foreign key ‘Item\_code’ that references the ‘Item\_code’ in the product table.

The database has a design where the **‘product**’ table is the parent table with all other tables having their foreign keys referencing the ‘Item\_code’. This design ensures that all product information is stored and maintained centrally. Each product in the store is uniquely identified by its Item\_code, which acts as the primary key in the product table.

This database structure ensures a clear hierarchy in which the product table acts as the main repository of product information, while other tables contain additional attributes that are specific to each product category. Therefore, any modifications or updates made to product information can be efficiently managed from a central location, ensuring data consistency and integrity throughout the database.

Product categories such as wine coolers or air conditioners have particular attributes like capacity or BTU, which are stored in their corresponding tables. Furthermore, there are other tables like multi-door and table-top that contain particular attributes for multi-door refrigerators and table-top appliances, respectively. These relationships between tables aid in the efficient organization, retrieval, and analysis of data, which ultimately support the store's operations and interactions with customers.

According to our business rules, the order table in our database acts as a joining table that breaks the many-to-many relationship present between the ‘customers’ and the ‘products’ table.

### Define the Table Constraints According to the Attributes and to the Business Rules

**customer:**

* Each customer is identified uniquely by their customer ID.
* Phone numbers are unique for each customer.

**product:**

* Each product has a unique code.

**order:**

* Each order has a unique order ID.
* Orders are linked to customers.
* Products in orders are linked to the product table.
* Discounts must be between 0 and 100%.
* Quantities in orders must be positive.

**wine\_cooler:**

* Wine coolers are linked to products.
* Capacity of a wine cooler must be greater than zero.

**ac:**

* Air conditioners are linked to products.
* BTU (cooling power) of an air conditioner must be greater than zero.

**dvd:**

* DVD players are linked to products.
* HDMI connection can be either 'yes' or 'no'.

**sound\_bar:**

* Sound bars are linked to products.
* Amplifier output of a sound bar must be non-negative.

**humidor:**

* Humidors are linked to products.
* Capacity of a humidor must be non-negative.

**freezer:**

* Freezers are linked to products.
* Number of drawers in a freezer must be non-negative.
* The freezer type can be either frost-free or not.

**led:**

* LED TVs are linked to products.
* Size of an LED TV must be greater than zero.
* It can be smart ('yes') or not ('no').

**cooker:**

* Cookers are linked to products.
* Bottle compartment must be either available ('yes') or not ('no').
* Full safety feature must be either available ('yes') or not ('no').

**dishwasher:**

* Dishwashers are linked to products.
* Settings (number of settings) must be one or more.

**dispenser:**

* Dispensers are linked to products.
* Number of faucets must be one or more.

**dryer:**

* Dryers are linked to products.
* Size of a dryer drum must be greater than zero.

**electric\_oven:**

* Electric ovens are linked to products.
* Size of an electric oven must be greater than zero.

**hood:**

* Hoods are linked to products.
* Size of a hood must be greater than zero.

**laundry:**

* Laundry appliances are linked to products.
* Size of the appliance must be greater than zero.
* Speed of the appliance must be greater than zero.

**microwave:**

* Microwaves are linked to products.
* Size of a microwave must be greater than zero.
* Microwaves can have built-in features, a grill, and an oven, each of which can be either available ('yes') or not ('no').

**oven:**

* Ovens are linked to products.
* Size of an oven must be greater than zero.
* Ovens can have built-in features and a grill, each of which can be either available ('yes') or not ('no')**.**

**top\_load:**

* Top load washing machines are linked to products.
* Size of a top load washing machine must be greater than zero.

**table\_top:**

* Table tops are linked to products.
* Size of a table top must be greater than zero.

**twin\_tub:**

* Twin tub washing machines are linked to products.
* Size of a twin tub washing machine must be greater than zero.

**vacuum:**

* Vacuum cleaners are linked to products.
* Power of a vacuum cleaner must be greater than zero.
* Bagless features must be either available ('yes') or not ('no').

**multi\_door:**

* Multi-door refrigerators are linked to products.
* Number of doors in a multi-door refrigerator must be one or more.
* Size of a multi-door refrigerator must be greater than zero.
* Multi-door refrigerators can have a frost-free feature, which can be either available ('yes') or not ('no').

**refrigerator:**

* Refrigerators are linked to products.
* Number of doors in a refrigerator must be one or more.
* Size of a refrigerator must be greater than zero.
* Refrigerators can have a frost-free feature, which can be either available ('yes') or not ('no').

**hob:**

* Hobs are linked to products.
* Number of burners in a hob must be one or more.
* Size of a hob must be greater than zero.
* Hobs can have safety features and can be built-in, each of which can be either available ('yes') or not ('no').

**dap:**

* Domestic appliances are linked to products.
* Category of the domestic appliance.

The constraints applied to the database tables ensure data accuracy, integrity, and consistency, which are crucial for the smooth operation of the business. By enforcing constraints such as:

* **primary keys, unique keys, and foreign keys**, the database maintains the uniqueness of key identifiers and establishes relationships between different entities. For instance, primary keys like ‘Customer\_id’ and ‘Order\_id’ uniquely identify customers and orders respectively, ensuring each customer and order is uniquely identifiable.
* **Unique non-primary keys** like ‘Phone\_number’ ensure that each customer's contact number is unique, avoiding duplicate entries.
* **Foreign keys** establish relationships between tables, allowing for data retrieval and manipulation across related entities.Also, a cascading effect was specified for the foreign keys to ensure that any deletion in the product category (child) tables would be apply to the ‘product’ (parent) table
* **CHECK constraints** that ensure that data entered meets specific criteria, such as positive quantities, valid options (e.g., 'yes' or 'no'), and numeric values greater than zero. These constraints prevent the insertion of incorrect or invalid data, maintaining data quality.

### Create an Entity-Relationship (ER) Diagram:

* + We used a MySQL Workbench to create an ER diagram based on your identified entities, attributes, and relationships.
  + By using rectangles for entities, ovals for attributes, and lines to show relationships between entities. As well as diamond to link two entities.

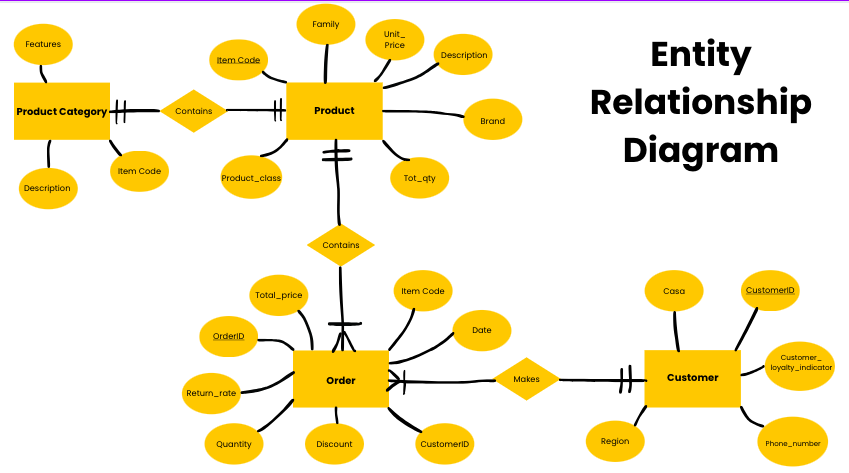
Our ERD diagram follows a standard convention where each entity, such as customers, products, and orders, is represented as a box that encapsulates its unique attributes, including customer names, product descriptions, and order dates. These attributes form the foundation of our database structure.

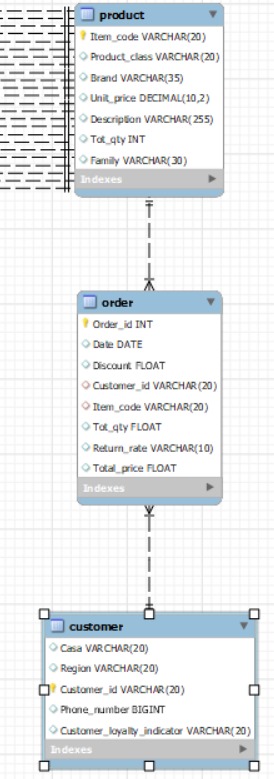
Relationships between entities are depicted using connecting lines that illustrate how they interact. For example, customers can place multiple orders, creating a one-to-many relationship between the customer and order entities. Similarly, products can be included in multiple orders, establishing another one-to-many relationship between products and orders.

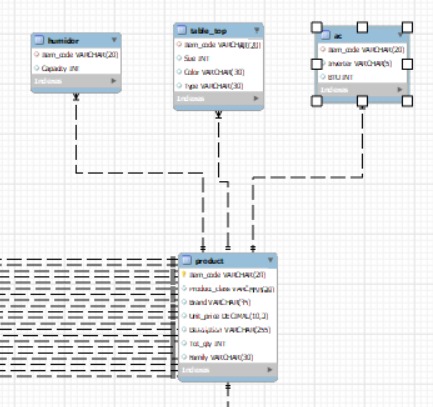
We also included additional details in our cardinalities. For instance, a customer must have at least one order to be considered a customer, but a product can be added to our database without belonging to any order.

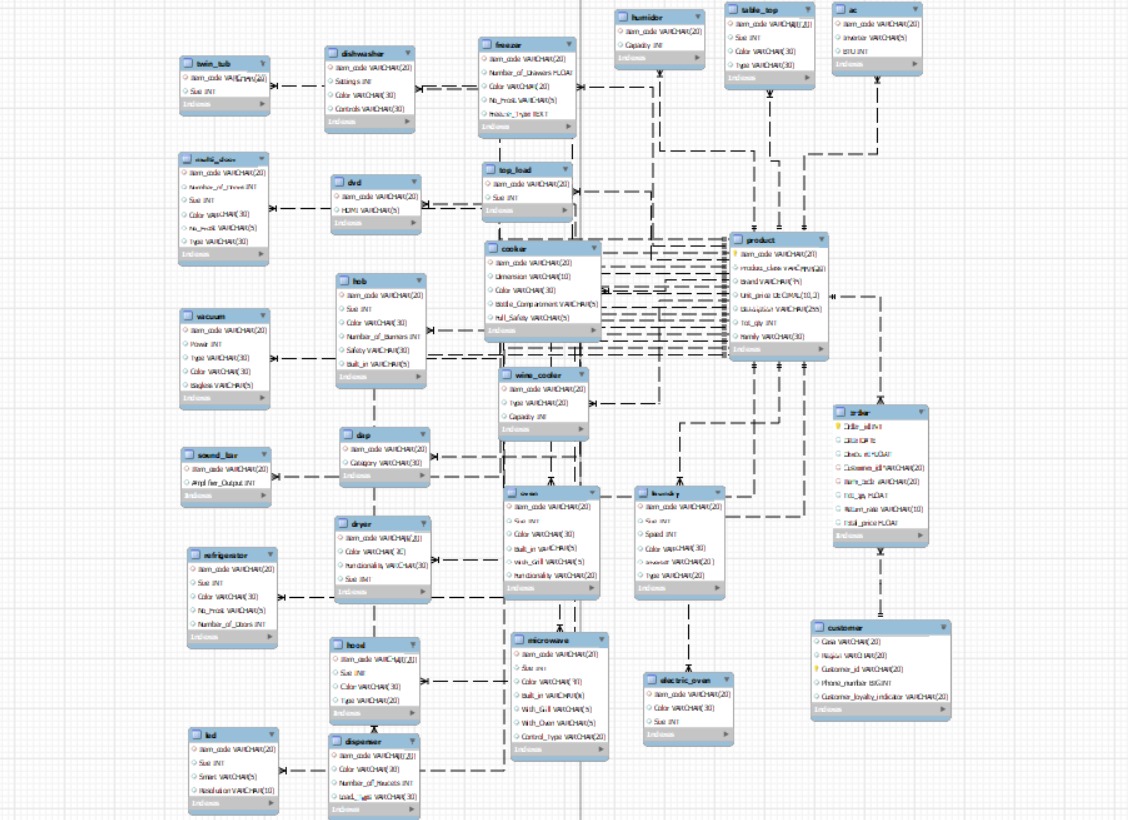
Lastly, The product table serves as a parent table to various product entities, such as microwaves and LEDs, that share common attributes while allowing specialization for unique features of each product type.

.



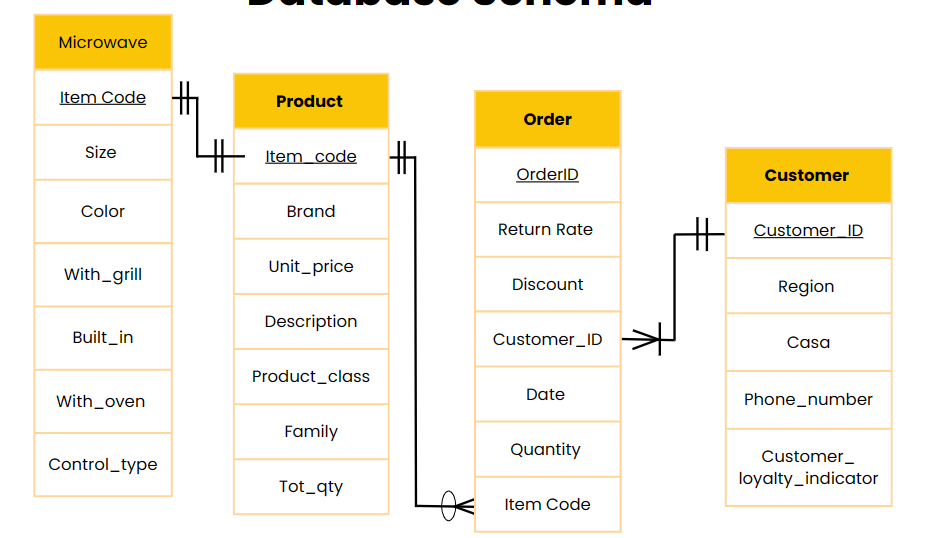
****

****

****

### Create the Database Schema:

Based on our ER diagram, we created the database schema for Linekrs using MySQL. We defined tables for each entity and went about specifying the columns and data types for each attribute as well as the appropriate constraints for each table:



Once the ERD is finalized, the next step is to develop the database schema. A database schema is a blueprint that defines the structure of a database, including its tables, columns, relationships, and constraints. The schema defines the data types for each column, specifies the primary and foreign keys to establish relationships between tables, and sets constraints to ensure data integrity, such as unique constraints or not null constraints.

In our schema, each table represents a specific entity, such as customers, products, and orders, with primary keys uniquely identifying each record.

For example, the Customer\_id serves as the primary key in the customer table, which is linked to the Order table, where it becomes a foreign key, establishing a relationship between customers and their orders. Similarly, the Item\_code acts as the primary key in the Product table, ensuring each product has a unique identifier.

In addition, the parent table 'Products' plays a central role and its primary key is utilized as a foreign key in all of the sub-family categories, ensuring a consistent and structured approach to data management. This allows each sub-family category to inherit the primary key from the 'Products' table, establishing a clear hierarchy within the database, simplifying data retrieval and ensuring data integrity across all product categories.

### Business Scenario Tests to Test the Database:

After the tables have been created successfully we began populating Linkers’ database with real customer data.

Looking at the day-to-day operations of a sales person at Linker’s group, we will present a breakdown of all the activities that actually take place when a customer wants to place a new order:

* A customer contacts a salesperson. What the customer is mostly interested in first is the product category that they are looking for and the possible brands. So this is always what the customer asks about first.
* After zooming into the specified brand for the required product, the next thing that is of interest to the customer is to have an overview of the unit prices for these products.
* After the salesperson gives the customer a rundown of the prices for each item code, the customer would then ask to check the description and specifications of the respective item codes.
* Once the customer agrees on an item to order, the information is entered into the excel file.

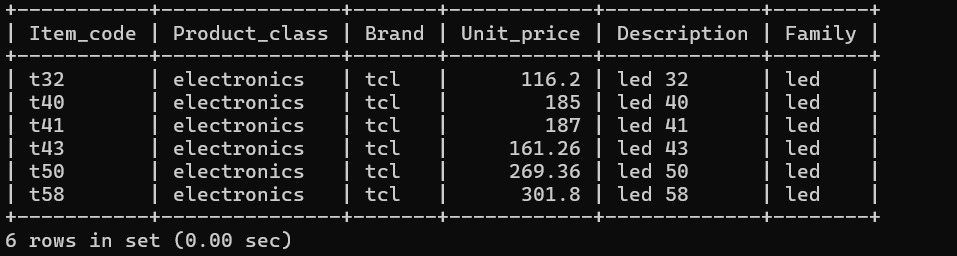
Now after constructing a comprehensive database, this process becomes easier and allows the entry of new data into MySQL which allows for a more consistent flow and analysis of the work.

Below is a rundown of how this process is transformed after implementing the MySQL database into the operations at the company.

**Step 1:**

The customer asks about LED, specifically about the TCL brand.

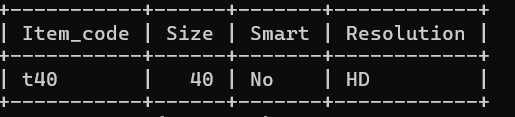
**SQL Query:** SELECT \* FROM product WHERE Family = 'led' AND Brand = 'tcl';



**Step 2:**

Zooming into Item Code t40 after checking its price to have an overview of the specifications.

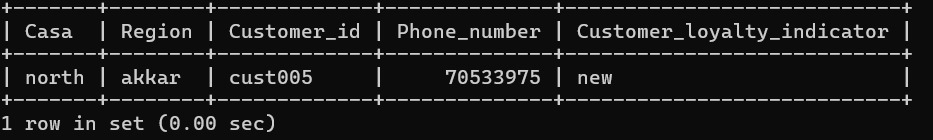
**SQL Query:** SELECT \* FROM led WHERE Item\_code = 't40';



**Step 3:**

After looking at the specifications, the customer chooses Item t40. For the salesperson to create a new order entry, the salesperson asks the customer for his ‘phone\_number’. Through the ‘phone\_number’ he would be able to access the ‘Customer\_id’.

**SQL Query:** SELECT \* FROM customer WHERE Phone\_number = '70533975';



**Step 4:**

Once the salesperson accesses the ‘Customer\_id’, he can then begin the process of creating the new order entry in the ‘order’ table.

**SQL Query:**

INSERT INTO order (Date, Customer\_id, Item\_code, Tot\_qty, Discount, Total\_price)

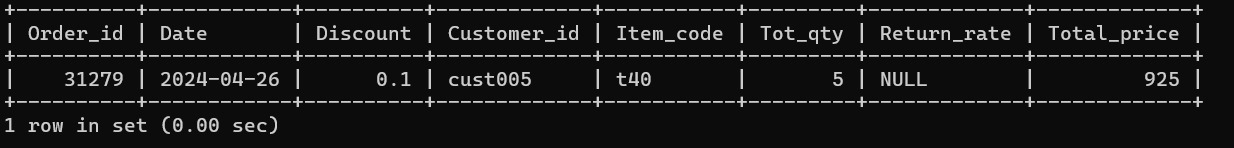
-> VALUES (CURDATE(), 'cust005', 't40', 5, 0.1, (5 \* (SELECT Unit\_price FROM product WHERE Item\_code = 't40')));

**Step 5:**

After creating the new order entry, the salesperson double checks that the order was properly placed for the right customer and following their requirements.

**SQL Query:**

SELECT \* FROM order WHERE Date = CURDATE() AND Customer\_id = 'cust005';



Below are some additional scenarios that could be of interest for management in order to gain certain insights about the customers and some information about possible revenue streams and how the business is doing.

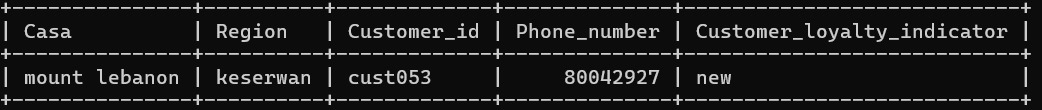
1. **View Customer Information:**

**Scenario 1:** An employee needs to view information about a specific customer.

**SQL Query:**

SELECT \* FROM customer WHERE Customer\_id = 'cust053';

**Output:**



**Scenario 2:** The manager wants to identify the top 10 customers who have placed the most orders.

**SQL Query:**

SELECT c.Customer\_id, COUNT(o.Order\_id) AS Order\_Count

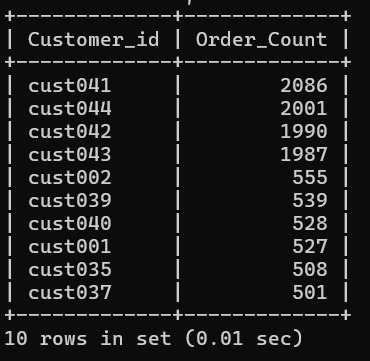
FROM customer c

LEFT JOIN order o ON c.Customer\_id = o.Customer\_id

GROUP BY c.Customer\_id

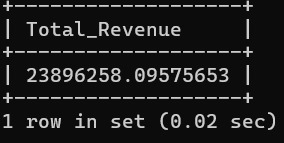
ORDER BY Order\_Count DESC

LIMIT 10;



**Scenario 3:** An employee needs to calculate the total revenue generated by the company within a specific time period

SELECT SUM(Total\_price) As Total\_Revenue FROM order WHERE Date BETWEEN '2019-03-31' AND '2020-09-19';



# 

# Prediction Model:

In our journey to predict sales, we initiated the process by dividing our dataset into an 80-20 train-validation split to ensure a robust framework for model evaluation. Following this, we delved into meticulous feature engineering to refine our predictors and enhance the model's ability to discern patterns in the data. With our features well-prepped, we constructed a Random Forest model, capitalizing on its aptitude for handling complex, non-linear relationships without extensive hyperparameter tuning. After training, the model exhibited a promising Training RMSE (Root Mean Squared Error) of approximately 4.24, signifying a strong fit to the training data. However, the Validation RMSE stood at around 13.09, indicating a disparity in performance on unseen data. This suggests an overfitting scenario where the model's predictions are not as generalizable as desired, pointing to potential avenues for further refinement such as parameter tuning, pruning, or exploring alternative feature engineering strategies.

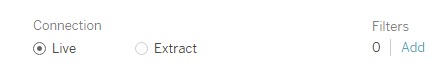
# Visualization Dashboard:

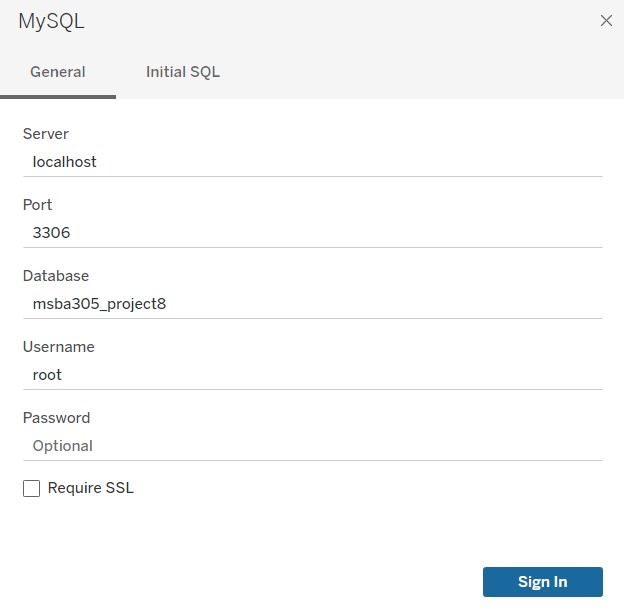
In order to maximize the benefits of using a DataBase for the operations of Linkers, we wanted the company to have the option to view their data in real time. This is why we worked on connecting Tableau to our MySQL DataBase in order to create a visualization dashboard for the company to keep track of data in real-time.

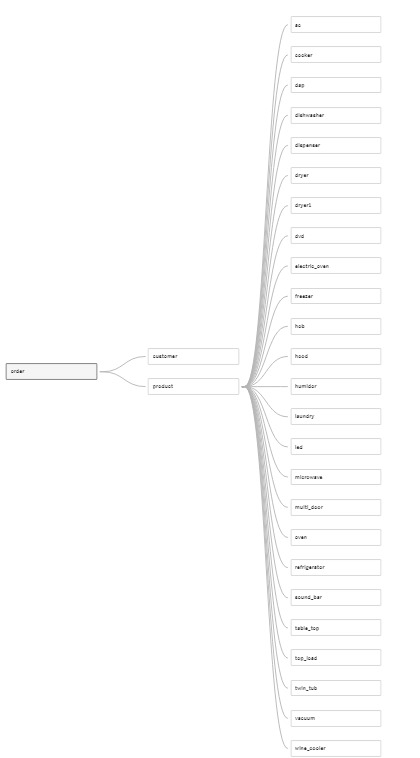
Creating a live connection between Tableau and our database is the first step in the linking process. Tableau can access the data contained in our database directly through this connection, eliminating the need to import or duplicate it. Using the correct connection parameters, including the server address, port, and authentication credentials, we first set up Tableau to connect to our database server.

After connecting, Tableau shows us a selection of our database's accessible tables and views, which we can choose from to create our dashboards. Tableau constantly refreshes these representations with real-time data from our database, giving us the option to either create new ones or use pre-existing ones.

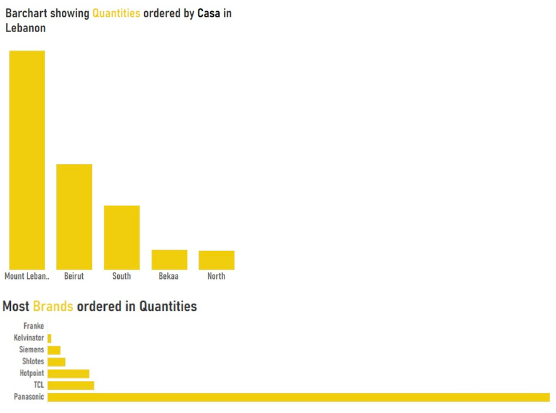
Our dashboards always show the most recent data since Tableau instantly updates itself in response to any modifications or additions made to the data in our database. Our managers can now evaluate and display data in real time without requiring manual data extraction or updates thanks to this live connectivity, which offers insightful information for decision-making.







# 



Here we can see an example of a dashboard that managers can make use of to track the most popular brand among customers as well as the Casa with the most orders.

This is just a small example of some of the many representations and information that management can have access to by setting up their DataBase and connecting it to Tableau in order to track their stats and monitor their success.

# 

# Limitations & Recommendations:

## Recommendations:

* **User Authentication:** Introduce a user authentication system to control access to the database. This ensures that only authorized users can view or modify data.
* **User Interface:** Develop a user interface for the employee to retrieve customer and product data without the need to write the full SQL queries.

## Limitations:

* **1 Item Limit per Order:** If a customer comes and orders multiple items on one day, the employee is required to record multiple records for each item ordered. This can be a limitation, however this is currently how Linkers operates.
* **Employee Training:** Linkers has been operating since 1993, this is the first time their employees will have encountered a database system. Employee training and implement parallel systems (Excel and DBMS) until employees are fully ready for the switch.
* **Primary Key:** Since the primary key item code and customer ID is not a number but rather strings, there is no auto incrementation in MySQL. Thus each employee will have to manually create these ID for new customers.

# Results and Discussions:

Our project ended up designing a successful database structure for Linkers. In short, by following a series of preprocessing steps to ensure data integrity and optimal database structure we were able to extract data from MySQL into Python and build a model to predict sales for each product.

Firstly, implementing a database has several benefits for Linkers. It ensures data integrity by preprocessing the data before storing it. This means that the data is clean and accurate, which is crucial for making informed business decisions. Additionally, storing data in a structured database allows for efficient data management. Linkers can easily retrieve, update, and query data, leading to faster decision-making processes. For example, if Linkers were interested in retrieving what is the least expensive product bought by customers from the Beirut region, they could do so easily.

Moreover, implementing a database has improved the performance of data retrieval and analysis operations for Linkers. Storing data in a database optimized for efficient storage and retrieval has resulted in quicker access to data and improved overall efficiency in data processing workflows. This scalability of MySQL allows Linkers to easily expand their database as their business grows, ensuring that the database can handle increased data volumes without sacrificing performance.

By extracting data from MySQL into Python, Linkers were able to build a predictive model to forecast sales for each product. This provides valuable insights that can help optimize inventory management and marketing strategies. Overall, implementing a database and leveraging data analytics tools has given Linkers a competitive advantage in the market. They can use data-driven insights to make strategic decisions and stay ahead of competitors.

# References

Damini. (2023, August 14). *Effective inventory forecasting: Tools and techniques for manufacturing leaders*. Deskera Blog. https://www.deskera.com/blog/effective-forecasting-inventory-control/

Deng, C., & Liu, Y. (2021, October 11). *A deep learning-based inventory management and demand prediction optimization method for ANOMALY DETECTION*. Wireless Communications and Mobile Computing. https://www.hindawi.com/journals/wcmc/2021/9969357/

Scott Curtiss Head of Field Presales Connect via LinkedIn, Curtiss, S., & Presales, H. of F. (2024, March 25). *Why channel-specific wholesale forecasting is the future*. RELEX Solutions. https://www.relexsolutions.com/resources/wholesale-forecasting/

Tang, Y. M., Chau, K. Y., Lau, Y., & Zheng, Z. (2023, February 27). *Data-intensive inventory forecasting with artificial intelligence models for cross-border e-commerce service automation*. MDPI. https://www.mdpi.com/2076-3417/13/5/3051

# 

# Appendix:

## Detailed Schema:

**-- Create customer table**

CREATE TABLE customer (

Customer\_id varchar(20) NOT NULL,

Casa varchar(20),

Region varchar(20),

Phone\_number bigint,

Customer\_loyalty\_indicator varchar(20),

PRIMARY KEY (Customer\_id),

UNIQUE KEY unique\_phone\_number (Phone\_number)

);

**-- Create product table**

CREATE TABLE product (

Item\_code varchar(20) NOT NULL,

Product\_class varchar(20),

Brand varchar(35),

Unit\_price float,

Description varchar(255),

Tot\_qty int,

Family varchar(30),

PRIMARY KEY (Item\_code),

CONSTRAINT check\_positive\_unit\_price CHECK (Unit\_price > 0),

CONSTRAINT check\_tot\_qty\_product CHECK (Tot\_qty > 0)

);

**CREATE TABLE order** (

Order\_id INT AUTO\_INCREMENT PRIMARY KEY,

Date DATE DEFAULT NULL,

Customer\_id VARCHAR(20) DEFAULT NULL,

Item\_code VARCHAR(20) DEFAULT NULL,

Tot\_qty INT DEFAULT NULL,

Discount FLOAT DEFAULT NULL,

Total\_price FLOAT DEFAULT NULL,

CONSTRAINT check\_discount\_lowerBound\_order CHECK ((Discount >= 0)),

CONSTRAINT check\_discount\_upperBound\_order CHECK ((Discount <= 100)),

CONSTRAINT check\_tot\_qty\_order CHECK ((Tot\_qty >= 0)),

FOREIGN KEY (Customer\_id) REFERENCES customer (Customer\_id),

FOREIGN KEY (Item\_code) REFERENCES product (Item\_code)

);

**-- Create wine\_cooler table**

CREATE TABLE wine\_cooler (

Item\_code varchar(20),

Type varchar(20),

Capacity int,

CONSTRAINT wine\_cooler\_ibfk\_2 FOREIGN KEY (Item\_code) REFERENCES product (Item\_code) ON DELETE CASCADE,

CONSTRAINT check\_capacity\_wine\_cooler CHECK (Capacity > 0)

);

**-- Create ac table**

CREATE TABLE ac (

Item\_code varchar(20),

Inverter varchar(5),

BTU int,

CONSTRAINT ac\_ibfk\_2 FOREIGN KEY (Item\_code) REFERENCES product (Item\_code) ON DELETE CASCADE,

CONSTRAINT check\_BTU\_AC CHECK (BTU > 0)

);

**-- Create dvd table**

CREATE TABLE dvd (

Item\_code varchar(20),

HDMI varchar(5),

CONSTRAINT dvd\_ibfk\_2 FOREIGN KEY (Item\_code) REFERENCES product (Item\_code) ON DELETE CASCADE,

CONSTRAINT check\_yes\_no CHECK (HDMI in ('yes', 'no'))

);

**-- Create sound\_bar table**

CREATE TABLE sound\_bar (

Item\_code varchar(20),

Amplifier\_Output int,

CONSTRAINT sound\_bar\_ibfk\_2 FOREIGN KEY (Item\_code) REFERENCES product (Item\_code) ON DELETE CASCADE,

CONSTRAINT check\_Amplifier\_Output CHECK (Amplifier\_Output >= 0)

);

**-- Create humidor table**

CREATE TABLE humidor (

Item\_code varchar(20),

Capacity int,

CONSTRAINT humidor\_ibfk\_2 FOREIGN KEY (Item\_code) REFERENCES product (Item\_code) ON DELETE CASCADE,

CONSTRAINT check\_capacity CHECK (Capacity >= 0)

);

**-- Create freezer table**

CREATE TABLE freezer (

Item\_code varchar(20),

Number\_of\_Drawers int,

Color varchar(20),

No\_Frost varchar(5),

Freezer\_Type text,

CONSTRAINT freezer\_ibfk\_2 FOREIGN KEY (Item\_code) REFERENCES product (Item\_code) ON DELETE CASCADE,

CONSTRAINT check\_number\_of\_drawers CHECK (Number\_of\_Drawers >= 0),

CONSTRAINT check\_yes\_no\_freezer CHECK (No\_Frost in ('yes', 'no'))

);

**-- Create led table**

CREATE TABLE led (

Item\_code varchar(20),

Size int,

Smart varchar(5),

Resolution varchar(10),

CONSTRAINT led\_ibfk\_2 FOREIGN KEY (Item\_code) REFERENCES product (Item\_code) ON DELETE CASCADE,

CONSTRAINT check\_size\_LED CHECK (Size > 0),

CONSTRAINT check\_yes\_no\_LED CHECK (Smart in ('yes', 'no'))

);

**-- Create cooker table**

CREATE TABLE cooker (

Item\_code varchar(20),

Dimension varchar(10),

Color varchar(30),

Bottle\_Compartment varchar(5),

Full\_Safety varchar(5),

CONSTRAINT cooker\_ibfk\_2 FOREIGN KEY (Item\_code) REFERENCES product (Item\_code) ON DELETE CASCADE,

CONSTRAINT bottle\_compartment\_Cooker CHECK (Bottle\_Compartment in ('yes', 'no')),

CONSTRAINT full\_safety\_Cooker CHECK (Full\_Safety in ('yes', 'no'))

);

**-- Create dishwasher table**

CREATE TABLE dishwasher (

Item\_code varchar(20),

Settings int,

Color varchar(30),

Controls varchar(30),

CONSTRAINT dishwasher\_ibfk\_2 FOREIGN KEY (Item\_code) REFERENCES product (Item\_code) ON DELETE CASCADE,

CONSTRAINT settings\_Dishwasher CHECK (Settings >= 1)

);

**-- Create dispenser table**

CREATE TABLE dispenser (

Item\_code varchar(20),

Color varchar(30),

Number\_of\_Faucets int,

Load\_Type varchar(30),

CONSTRAINT dispenser\_ibfk\_2 FOREIGN KEY (Item\_code) REFERENCES product (Item\_code) ON DELETE CASCADE,

CONSTRAINT number\_of\_faucets\_Dispenser CHECK (Number\_of\_Faucets >= 1)

);

**-- Create dryer table**

CREATE TABLE dryer (

Item\_code varchar(20),

Color varchar(30),

Functionality varchar(30),

Size int,

CONSTRAINT dryer\_ibfk\_2 FOREIGN KEY (Item\_code) REFERENCES product (Item\_code) ON DELETE CASCADE,

CONSTRAINT size\_Dryer CHECK (Size > 0)

);

**-- Create electric\_oven table**

CREATE TABLE electric\_oven (

Item\_code varchar(20),

Color varchar(30),

Size int,

CONSTRAINT electric\_oven\_ibfk\_2 FOREIGN KEY (Item\_code) REFERENCES product (Item\_code) ON DELETE CASCADE,

CONSTRAINT size\_Electric\_Oven CHECK (Size > 0)

);

**-- Create hood table**

CREATE TABLE hood (

Item\_code varchar(20),

Size int,

Color varchar(30),

Type varchar(20),

CONSTRAINT hood\_ibfk\_2 FOREIGN KEY (Item\_code) REFERENCES product (Item\_code) ON DELETE CASCADE,

CONSTRAINT size\_Hood CHECK (Size > 0)

);

**-- Create laundry table**

CREATE TABLE laundry (

Item\_code varchar(20),

Size int,

Speed int,

Color varchar(30),

Inverter varchar(20),

Type varchar(20),

CONSTRAINT laundry\_ibfk\_2 FOREIGN KEY (Item\_code) REFERENCES product (Item\_code) ON DELETE CASCADE,

CONSTRAINT check\_yes\_no\_Laundry CHECK (Inverter in ('yes', 'no')),

CONSTRAINT size\_Laundry CHECK (Speed > 0),

CONSTRAINT speed\_Laundry CHECK (Speed > 0)

);

**-- Create microwave table**

CREATE TABLE microwave (

Item\_code varchar(20),

Size int,

Color varchar(30),

Built\_in varchar(5),

With\_Grill varchar(5),

With\_Oven varchar(5),

Control\_Type varchar(20),

CONSTRAINT microwave\_ibfk\_2 FOREIGN KEY (Item\_code) REFERENCES product (Item\_code) ON DELETE CASCADE,

CONSTRAINT check\_yes\_no\_Microwave CHECK (Built\_in in ('yes', 'no')),

CONSTRAINT check\_yes\_no\_Microwave\_Grill CHECK (With\_Grill in ('yes', 'no')),

CONSTRAINT check\_yes\_no\_Microwave\_Oven CHECK (With\_Oven in ('yes', 'no')),

CONSTRAINT size\_Microwave CHECK (Size > 0)

);

**-- Create oven table**

CREATE TABLE oven (

Item\_code varchar(20),

Size int,

Color varchar(30),

Built\_in varchar(5),

With\_Grill varchar(5),

Functionality varchar(20),

CONSTRAINT oven\_ibfk\_2 FOREIGN KEY (Item\_code) REFERENCES product (Item\_code) ON DELETE CASCADE,

CONSTRAINT check\_yes\_no\_Oven\_Built\_in CHECK (Built\_in in ('yes', 'no')),

CONSTRAINT check\_yes\_no\_Oven\_Grill CHECK (With\_Grill in ('yes', 'no')),

CONSTRAINT size\_Oven CHECK (Size > 0)

);

**-- Create top\_load table**

CREATE TABLE top\_load (

Item\_code varchar(20),

Size int,

CONSTRAINT top\_load\_ibfk\_2 FOREIGN KEY (Item\_code) REFERENCES product (Item\_code) ON DELETE CASCADE,

CONSTRAINT size\_Top\_Load CHECK (Size > 0)

);

**-- Create twin\_tub table**

CREATE TABLE twin\_tub (

Item\_code varchar(20),

Size int,

CONSTRAINT twin\_tub\_ibfk\_2 FOREIGN KEY (Item\_code) REFERENCES product (Item\_code) ON DELETE CASCADE,

CONSTRAINT size\_Twin\_Tub CHECK (Size > 0)

);

**-- Create vacuum table**

CREATE TABLE vacuum (

Item\_code varchar(20),

Power int,

Type varchar(30),

Color varchar(30),

Bagless varchar(5),

CONSTRAINT vacuum\_ibfk\_2 FOREIGN KEY (Item\_code) REFERENCES product (Item\_code) ON DELETE CASCADE,

CONSTRAINT check\_yes\_no\_Vacuum CHECK (Bagless in ('yes', 'no')),

CONSTRAINT Power\_Vacuum CHECK (Power > 0)

);

**-- Create multi\_door table**

CREATE TABLE multi\_door (

Item\_code varchar(20),

Number\_of\_Doors int,

Size int,

Color varchar(30),

No\_Frost varchar(5),

Type varchar(30),

CONSTRAINT multi\_door\_ibfk\_2 FOREIGN KEY (Item\_code) REFERENCES product (Item\_code) ON DELETE CASCADE,

CONSTRAINT check\_yes\_no\_Multi\_door\_No\_Frost CHECK (No\_Frost in ('yes', 'no')),

CONSTRAINT number\_of\_doors\_multi\_door CHECK (Number\_of\_Doors > 0),

CONSTRAINT size\_multi\_door CHECK (Size > 0)

);

**-- Create refrigerator table**

CREATE TABLE refrigerator (

Item\_code varchar(20),

Size int,

Color varchar(30),

No\_Frost varchar(5),

Number\_of\_Doors int,

CONSTRAINT refrigerator\_ibfk\_2 FOREIGN KEY (Item\_code) REFERENCES product (Item\_code) ON DELETE CASCADE,

CONSTRAINT check\_yes\_no\_Refrigerator\_No\_Frost CHECK (No\_Frost in ('yes', 'no')),

CONSTRAINT number\_of\_doors\_Refrigerator CHECK (Number\_of\_Doors > 0),

CONSTRAINT size\_Refrigerator CHECK (Size > 0)

);

**-- Create hob table**

CREATE TABLE hob (

Item\_code varchar(20),

Size int,

Color varchar(30),

Number\_of\_Burners int,

Safety varchar(30),

Built\_in varchar(5),

CONSTRAINT hob\_ibfk\_2 FOREIGN KEY (Item\_code) REFERENCES product (Item\_code) ON DELETE CASCADE,

CONSTRAINT check\_yes\_no\_Hob\_Built\_in CHECK (Built\_in in ('yes', 'no')),

CONSTRAINT Number\_of\_Burners\_Hob CHECK (Number\_of\_Burners > 0),

CONSTRAINT size\_Hob CHECK (Size > 0)

);

**-- Create dap table**

CREATE TABLE dap (

Item\_code varchar(20),

Category varchar(30),

CONSTRAINT dap\_ibfk\_2 FOREIGN KEY (Item\_code) REFERENCES product (Item\_code) ON DELETE CASCADE

);

**-- Create table\_top table**

CREATE TABLE table\_top (

Item\_code varchar(20),

Size int,

Color varchar(30),

Type varchar(30),

CONSTRAINT table\_top\_ibfk\_2 FOREIGN KEY (Item\_code) REFERENCES product (Item\_code) ON DELETE CASCADE,

CONSTRAINT size\_Table\_Top CHECK (Size > 0)

);

## Python Code:

# Importing the Packages

import pandas as pd  
 import numpy as np

# Importing the data

data = pd.read\_csv("C:\\Users\\Lenovo\\OneDrive\\01\_Education\\02\_AUB\\02\_Spring2024\\MSBA305-Data Processing Frameworks\\Project\\Data\\Updated\_FinalData305.csv")

data.head()

# Renaming the Dataset Columns

# Assuming `data` is your DataFrame  
 column\_names = list(data.columns)  
  
 print(column\_names)

transformed\_column\_names = []  
 for name in column\_names:  
 # Capitalize the first letter, convert the rest to lowercase, and replace spaces with underscores  
 transformed\_name = name.strip().replace(' ', '\_').lower().capitalize()  
 transformed\_column\_names.append(transformed\_name)  
  
 print(transformed\_column\_names)

# Assuming data is your DataFrame and transformed\_column\_names contains the new column names  
 # Create a dictionary mapping old column names to new column names  
 column\_mapping = {old\_name: new\_name for old\_name, new\_name in zip(data.columns, transformed\_column\_names)}  
  
 # Rename the columns of the DataFrame using the mapping  
 data.rename(columns=column\_mapping, inplace=True)

import pandas as pd  
  
 # Assuming `data` is your DataFrame  
 column\_names = list(data.columns)

# Common Preprocessing for the Entire Dataset

data.info()

## Missing Values

#Missing values  
 # Calculate the percentage of missing values in each column  
 missing\_percentage = (data.isna().sum() / len(data)) \* 100  
  
 # Print the percentage of missing values in each column  
 print("Percentage of missing values in each column:")  
 print(missing\_percentage)

# Identify numerical and categorical columns  
 numerical\_columns = data.select\_dtypes(include=['number']).columns  
 categorical\_columns = data.select\_dtypes(include=['object']).columns  
  
 # Replace missing values in numerical columns with mean, except for "Discount" column  
 for col in numerical\_columns:  
 if col == "Discount":  
 # Replace missing values in "Discount" column with 0  
 data[col].fillna(0, inplace=True)  
 else:  
 # Replace missing values in other numerical columns with mean  
 data[col].fillna(data[col].mean(), inplace=True)  
  
 # Replace missing values in categorical columns with mode  
 for col in categorical\_columns:  
 data[col].fillna(data[col].mode()[0], inplace=True)  
  
 # Confirm that missing values have been filled  
 print("Missing values filled.")

## Column Data Type

# Convert "Date" column to datetime type  
 data['Date'] = pd.to\_datetime(data['Date'])  
 data['Discount'] = data['Discount'].astype(float)  
  
 # Confirm the change  
 print("Data type of 'Date' column after conversion:", data['Date'].dtype)  
 print("Data type of 'Discount' column after conversion:", data['Discount'].dtype)

## String Manipulation - Categorical Columns

# Identify categorical columns  
 categorical\_columns = data.select\_dtypes(include=['object']).columns  
  
 # Remove "Family" column from the list of categorical columns  
 if 'Family' in categorical\_columns:  
 categorical\_columns = categorical\_columns.drop('Family')  
  
 # Loop through each remaining categorical column and apply transformations  
 for col in categorical\_columns:  
 # Convert to lowercase and strip whitespace  
 data[col] = data[col].str.lower().str.strip()  
   
 # Confirm the changes  
 print("Categorical columns transformed.")

## Outliers and illogical values - Numerical Columns

data.head()

### Removing Retuned Orders

# Remove rows where "Tot Qty" is negative  
 data = data[data['Tot\_qty'] > 0]  
 data = data[data['Unit\_price'] > 0]  
 data = data[data['Total\_price'] > 0]

### Normalizing the description Column

import pandas as pd  
 import re  
  
 # Assuming your DataFrame is named 'data'  
  
 # Remove punctuation, trim white spaces, and convert to lowercase  
 data['Description'] = data['Description'].apply(lambda x: re.sub(r'[^\w\s]', '', x).strip().lower())

# Creating the Different Tables

# Process the categories for the DataFrame  
 data['Family'] = data['Family'].str.strip().str.lower()  
 data['Item\_code'] = data['Item\_code'].str.strip()

# Get unique categories from the column  
 categories\_list = data['Family'].unique().tolist()  
 print(categories\_list)  
  
  
 # Process the categories as specified  
 processed\_categories = [category.strip().lower().replace(' ', '\_').replace('/', '') for category in categories\_list]  
  
 print(processed\_categories)

print(len(processed\_categories))  
 print(len(categories\_list))

# Loop through both lists simultaneously  
 for processed\_category, category in zip(processed\_categories, categories\_list):  
 # Filter the DataFrame based on the current category  
 globals()[processed\_category] = data[data['Family']== category.strip().lower()]  
  
 # Now you can access the filtered DataFrames directly using the variable names  
 for processed\_category in processed\_categories:  
 print(f"DataFrame '{processed\_category}':")  
 print(globals()[processed\_category])

num\_dataframes = len(processed\_categories)  
 print("Number of dataframes created:", num\_dataframes)  
  
 # Iterate over the processed\_categories list and print the names  
 for df\_name in processed\_categories:  
 print("DataFrame name:", df\_name)

## Removing Irelevant comlumns across Family tables

# List of columns to remove  
 columns\_to\_remove = ['Date', 'Casa', 'Region', 'Family', 'Tot\_qty',  
 'Payment\_terms', 'Discount', 'Customer\_id', 'Unit\_price', 'Total\_price' ,  
 'Order\_id', 'Phone\_number', 'Customer\_loyalty\_indicator', 'Product\_class', 'Brand', 'Return\_rate']  
  
 # Iterate through each dataframe name and drop the specified columns  
 for df\_name in processed\_categories:  
 # Access the dataframe object using its name  
 df = globals()[df\_name] # or locals()[df\_name] if the dataframes are defined locally  
 df.drop(columns=columns\_to\_remove, inplace=True)

# Iterate through each dataframe name in the list  
 for df\_name in processed\_categories:  
 # Access the dataframe object using its name  
 df = globals()[df\_name] # or locals()[df\_name] if the dataframes are defined locally  
   
 # Remove duplicates of 'Item Code' column, keeping the first occurrence  
 df.drop\_duplicates(subset=['Item\_code'], keep='first', inplace=True)  
 df.reset\_index(drop=True, inplace=True)

## Family

### 'wine\_cooler'

import pandas as pd  
  
 # Function to extract type and capacity from description  
 def extract\_type\_and\_capacity(description):  
 description = description.lower() # Convert description to lowercase for case-insensitive matching  
   
 if 'cooler' in description:  
 # Extract type as 'cooler' and capacity (number of bottles)  
 type\_str = 'cooler'  
 capacity\_str = description.split('cooler')[1].strip() # Extract the part after 'cooler'  
 elif 'cellar' in description:  
 # Extract type as 'cellar' and capacity (number of bottles)  
 type\_str = 'cellar'  
 capacity\_str = description.split('cellar')[1].strip() # Extract the part after 'cellar'  
 else:  
 type\_str = 'cooler'  
 capacity\_str = None  
   
 # Extract numeric capacity value (without 'bottles')  
 if capacity\_str:  
 capacity\_value = ''.join(filter(str.isdigit, capacity\_str)) # Keep only digits  
 else:  
 capacity\_value = 70  
   
 return type\_str, capacity\_value  
  
 # Apply the function to extract type and capacity from each description  
 wine\_cooler['Type'], wine\_cooler['Capacity'] = zip(\*wine\_cooler['Description'].apply(extract\_type\_and\_capacity))  
  
 # Display the updated DataFrame with extracted type and capacity columns  
 print("Updated DataFrame with Type and Capacity columns:")

wine\_cooler.head()

# Replace None values in 'Capacity' column with 45  
 wine\_cooler['Capacity'].fillna(45, inplace=True)

# Convert 'Capacity' column to integers  
 wine\_cooler['Capacity'] = pd.to\_numeric(wine\_cooler['Capacity'], errors='coerce').astype('Int64')

wine\_cooler.info()

# Drop the 'Description' column  
 wine\_cooler.drop(columns=['Description'], inplace=True)

wine\_cooler.head()

### AC

# Function to extract inverter status and BTU information  
 def extract\_inverter\_and\_btu(description):  
 description = description.lower() # Convert description to lowercase for case-insensitive matching  
   
 # Extract inverter status ('yes' if 'inverter' is in description, otherwise 'no')  
 inverter\_status = 'yes' if 'inverter' in description else 'no'  
   
 # Extract BTU information (numeric part from the description)  
 btu\_value = ''.join(filter(str.isdigit, description)) # Keep only digits  
   
 return inverter\_status, btu\_value  
  
 # Apply the function to extract inverter status and BTU information from each description  
 ac['Inverter'], ac['BTU'] = zip(\*ac['Description'].apply(extract\_inverter\_and\_btu))  
  
 # Remove 'BTU' from the 'BTU' column and convert to numeric  
 ac['BTU'] = ac['BTU'].str.replace('BTU', '').astype(int)  
  
 # Display the updated DataFrame with extracted 'Inverter' and 'BTU' columns  
 print("Updated DataFrame with Inverter and BTU columns:")

ac.head()

ac.info()

# Drop the 'Description' column  
 ac.drop(columns=['Description'], inplace=True)

### DVD

# Function to extract HDMI presence information  
 def extract\_hdmi\_presence(description):  
 description = description.lower() # Convert description to lowercase for case-insensitive matching  
   
 # Check if 'with hdmi' is in the description  
 hdmi\_presence = 'yes' if 'with hdmi' in description else 'no'  
   
 return hdmi\_presence  
  
 # Apply the function to extract HDMI presence information from each description  
 dvd['HDMI'] = dvd['Description'].apply(extract\_hdmi\_presence)  
  
 # Display the updated DataFrame with the 'HDMI' column  
 print("Updated DataFrame with HDMI column:")

dvd.head()

# Drop the 'Description' column  
 dvd.drop(columns=['Description'], inplace=True)

### SoundBar

sound\_bar.head()

# Function to extract amplifier output in Watts  
 def extract\_amplifier\_output(description):  
 # Use regular expression to find numeric values in the description  
 # We use \d+ to match one or more digits  
 # The result will be a list of all numeric substrings found in the description  
 output\_values = re.findall(r'\d+', description)  
   
 if output\_values:  
 # Take the first value as the amplifier output (assumes the first numeric value is the desired one)  
 amplifier\_output\_value = int(output\_values[0]) # Convert to integer  
 else:  
 amplifier\_output\_value = None  
   
 return amplifier\_output\_value  
  
 # Apply the function to extract amplifier output from each description  
 sound\_bar['Amplifier\_Output'] = sound\_bar['Description'].apply(extract\_amplifier\_output)

sound\_bar.head()

sound\_bar.info()

# Drop the 'Description' column  
 sound\_bar.drop(columns=['Description'], inplace=True)

### Humidor

# Function to extract capacity (number of cigars)  
 def extract\_capacity(description):  
 # Use regular expression to find numeric values in the description  
 # We use \d+ to match one or more digits  
 # The result will be a list of all numeric substrings found in the description  
 capacity\_values = re.findall(r'\d+', description)  
   
 if capacity\_values:  
 # Take the first value as the capacity (assumes the first numeric value is the desired one)  
 capacity\_value = int(capacity\_values[0]) # Convert to integer  
 else:  
 capacity\_value = None  
   
 return capacity\_value  
  
 # Apply the function to extract capacity from each description  
 humidor['Capacity'] = humidor['Description'].apply(extract\_capacity)

humidor.info()

# Drop the 'Description' column  
 humidor.drop(columns=['Description'], inplace=True)

### Freezer

freezer.head()

# Function to extract attributes from description  
 def extract\_freezer\_attributes(description):  
 description = description.lower() # Convert description to lowercase for case-insensitive matching  
   
 # Initialize attributes to None  
 num\_drawers = None  
 color = None  
 no\_frost = None  
 freezer\_type = None  
   
 # Extract number of drawers using regex  
 num\_drawers\_match = re.search(r'(\d+)\s\*drawers?', description)  
 if num\_drawers\_match:  
 num\_drawers = int(num\_drawers\_match.group(1)) # Extract the numeric part  
   
 # Extract color (white or silver) using regex  
 if 'white' in description:  
 color = 'White'  
 elif 'silver' in description:  
 color = 'Silver'  
   
 # Check for 'no frost' indication  
 if 'no frost' in description:  
 no\_frost = 'Yes'  
 else:  
 no\_frost = 'No'  
   
 # Determine freezer type (vaccine or regular)  
 if 'vaccine' in description:  
 freezer\_type = 'Vaccine'  
 else:  
 freezer\_type = 'Regular'  
   
 return num\_drawers, color, no\_frost, freezer\_type  
  
 # Apply the function to extract attributes from each description  
 freezer['Number\_of\_Drawers'], freezer['Color'], freezer['No\_Frost'], freezer['Freezer\_Type'] = \  
 zip(\*freezer['Description'].apply(extract\_freezer\_attributes))

freezer.head()

freezer.info()

# Drop the 'Description' column  
 freezer.drop(columns=['Description'], inplace=True)

### LED

# Assuming you have a DataFrame named 'freezer' with a column 'Description'  
  
 # Get the unique categories from the 'Description' column  
 description\_categories = led['Description'].unique().tolist()  
  
 for i in description\_categories:  
 print(i)

# Function to extract attributes from description  
 def extract\_tv\_attributes(description):  
 description = description.lower().strip() # Normalize description: convert to lowercase and strip whitespace  
   
 # Initialize attributes to None  
 size = None  
 smart = None  
 resolution = None  
   
 # Extract size (in inches) using regex  
 size\_match = re.search(r'(\d+)', description)  
 if size\_match:  
 size = int(size\_match.group(1)) # Extract the numeric part  
   
 # Determine if TV is smart  
 if 'smart' in description:  
 smart = 'Yes'  
 else:  
 smart = 'No'  
   
 # Determine resolution (FHD, UHD, HD)  
 if 'uhd' in description:  
 resolution = 'UHD'  
 elif 'fhd' in description:  
 resolution = 'FHD'  
 else:  
 resolution = 'HD'  
   
 return size, smart, resolution  
  
 # Apply the function to extract attributes from each description  
 led['Size'], led['Smart'], led['Resolution'] = \  
 zip(\*led['Description'].apply(extract\_tv\_attributes))

led.head()

led.info()

# Drop the 'Description' column  
 led.drop(columns=['Description'], inplace=True)

### Cooker

# Assuming you have a DataFrame named 'freezer' with a column 'Description'  
  
 # Get the unique categories from the 'Description' column  
 description\_categories = cooker['Description'].unique().tolist()  
  
 for i in description\_categories:  
 print(i)

# Function to extract attributes from description  
 def extract\_cooker\_attributes(description):  
 description = description.lower().strip() # Normalize description: convert to lowercase and strip whitespace  
   
 # Initialize attributes  
 dimension = None  
 color = None  
 bottle\_compartment = None  
 full\_safety = None  
   
 # Extract dimension (e.g., 90x60, 60x60, 5555)  
 dimension\_match = re.search(r'(\d{2,4})x(\d{2,4})', description)  
 if dimension\_match:  
 dimension = f"{dimension\_match.group(1)}x{dimension\_match.group(2)}" # Format as '90x60', '60x60', '55x55'  
   
 # Determine color (white, black, stainless steel, inox, red)  
 if 'white' in description:  
 color = 'white'  
 elif 'black' in description:  
 color = 'black'  
 elif 'ss' in description or 'stainless steel' in description:  
 color = 'stainless steel'  
 elif 'inox' in description:  
 color = 'stainless steel' # Assuming 'inox' means stainless steel  
 elif 'red' in description:  
 color = 'red'  
   
 # Determine if bottle compartment is present  
 if 'bottle compartment' in description:  
 bottle\_compartment = 'Yes'  
 else:  
 bottle\_compartment = 'No'  
   
 # Determine if full safety is present  
 if 'full safety' in description:  
 full\_safety = 'Yes'  
 else:  
 full\_safety = 'No'  
   
 return dimension, color, bottle\_compartment, full\_safety  
  
 # Apply the function to extract attributes from each description  
 cooker['Dimension'], cooker['Color'], cooker['Bottle\_Compartment'], cooker['Full\_Safety'] = \  
 zip(\*cooker['Description'].apply(extract\_cooker\_attributes))

cooker.head()

# Drop the 'Description' column  
 cooker.drop(columns=['Description'], inplace=True)

### Dishwasher

# Assuming you have a DataFrame named 'freezer' with a column 'Description'  
  
 # Get the unique categories from the 'Description' column  
 description\_categories = dishwasher['Description'].unique().tolist()  
  
 for i in description\_categories:  
 print(i)

# Function to add space between number and 'settings'  
 def add\_space\_to\_settings(description):  
 # Split the description at 'settings' and join with space added  
 parts = description.split('settings')  
 if len(parts) > 1 and parts[0].isdigit():  
 return f"{parts[0]} settings{parts[1]}"  
 else:  
 return description  
  
 # Apply the function to transform the description  
 dishwasher['Description'] = dishwasher['Description'].apply(add\_space\_to\_settings)

# Function to extract attributes from description  
 def extract\_dishwasher\_attributes(description):  
 description = description.lower().strip() # Normalize description: convert to lowercase and strip whitespace  
   
 # Initialize attributes  
 settings = None  
 color = None  
 controls = None  
   
 # Extract settings (e.g., 13, 5, 16, 12 for integrated)  
 settings\_match = re.search(r'(\d+)', description)  
 if settings\_match:  
 settings = int(settings\_match.group(1)) # Convert settings to integer  
 else:  
 settings = 12 # Default settings to 12 if no number is found  
   
 # Determine color (white, inox, silver)  
 if 'white' in description:  
 color = 'white'  
 elif 'inox' in description:  
 color = 'inox'  
 elif 'silver' in description:  
 color = 'silver'  
   
 # Determine controls (fully integrated or semi integrated)  
 if 'fully integrated' in description:  
 controls = 'fully integrated'  
 elif 'semi integrated' in description or 'semi' in description:  
 controls = 'semi integrated'  
 else:  
 controls = 'semi integrated' # Default to semi integrated if controls not specified  
   
 return settings, color, controls  
  
 # Apply the function to extract attributes from each description  
 dishwasher['Settings'], dishwasher['Color'], dishwasher['Controls'] = \  
 zip(\*dishwasher['Description'].apply(extract\_dishwasher\_attributes))

# Convert the 'Settings' column to integer  
 dishwasher['Settings'] = dishwasher['Settings'].astype(int)

dishwasher.info()

# Drop the 'Description' column  
 dishwasher.drop(columns=['Description'], inplace=True)

### Dispenser

# Assuming you have a DataFrame named 'freezer' with a column 'Description'  
  
 # Get the unique categories from the 'Description' column  
 description\_categories = dispenser['Description'].unique().tolist()  
  
 for i in description\_categories:  
 print(i)

# Function to extract attributes from description  
 def extract\_dispenser\_attributes(description):  
 description = description.lower().strip() # Normalize description: convert to lowercase and strip whitespace  
   
 # Initialize attributes  
 color = None  
 num\_faucets = None  
 load\_type = None  
   
 # Determine color (white, black, silver, stainless steel)  
 if 'white' in description:  
 color = 'white'  
 elif 'black' in description:  
 color = 'black'  
 elif 'silver' in description:  
 color = 'silver'  
 elif 'ss' in description or 'stainless steel' in description:  
 color = 'stainless steel'  
 else:  
 color = 'silver' # Default color to silver if not specified  
   
 # Determine number of faucets (2 or 3)  
 if '3 faucet' in description:  
 num\_faucets = 3  
 else:  
 num\_faucets = 2 # Default number of faucets to 2 if not specified  
   
 # Determine load type (bottom load or upper load)  
 if 'bottom load' in description:  
 load\_type = 'Bottom Load'  
 else:  
 load\_type = 'Upper Load' # Default load type to Upper Load if not specified  
   
 return color, num\_faucets, load\_type  
  
 # Apply the function to extract attributes from each description  
 dispenser['Color'], dispenser['Number\_of\_Faucets'], dispenser['Load\_Type'] = \  
 zip(\*dispenser['Description'].apply(extract\_dispenser\_attributes))

dispenser.head()

dispenser.info()

# Drop the 'Description' column  
 dispenser.drop(columns=['Description'], inplace=True)

### Dryer

# Assuming you have a DataFrame named 'freezer' with a column 'Description'  
  
 # Get the unique categories from the 'Description' column  
 description\_categories = dryer['Description'].unique().tolist()  
  
 for i in description\_categories:  
 print(i)

# Function to extract attributes from description  
 def extract\_dryer\_attributes(description):  
 description = description.lower().strip() # Normalize description: convert to lowercase and strip whitespace  
   
 # Filter out rows with specific descriptions  
 if description == 'dvd player' or description == 'washer 15kgs white':  
 return None, None, None # Return None to indicate dropping this row  
   
 # Separate number followed by "kg" into "<number> kg" format  
 if 'kg' in description:  
 description = description.replace('kg', ' kg') # Add space before 'kg' if not present  
   
 # Determine color (white, black, silver, anthracite)  
 if 'white' in description:  
 color = 'white'  
 elif 'black' in description:  
 color = 'black'  
 elif 'silver' in description:  
 color = 'silver'  
 elif 'anthracite' in description:  
 color = 'anthracite'  
 else:  
 color = 'white' # Default color to white if not specified  
   
 # Determine functionality (vented, condenser, heat pump)  
 if 'vented' in description:  
 functionality = 'vented'  
 elif 'condenser' in description:  
 functionality = 'condenser'  
 elif 'heat pump' in description:  
 functionality = 'heat pump'  
 else:  
 functionality = 'vented' # Default functionality to vented if not specified  
   
 # Extract size (7, 8, 9, 10)  
 size = None  
 if '7' in description:  
 size = 7  
 elif '8' in description:  
 size = 8  
 elif '9' in description:  
 size = 9  
 elif '10' in description:  
 size = 10  
   
 return color, functionality, size  
  
 # Apply the function to extract attributes from each description and filter rows  
 dryer['Color'], dryer['Functionality'], dryer['Size'] = \  
 zip(\*dryer['Description'].apply(extract\_dryer\_attributes))  
  
 # Drop rows where 'Description' is 'dvd player' or 'washer 15kgs white'  
 dryer = dryer.dropna()

# Convert the 'Settings' column to integer  
 dryer['Size'] = dryer['Size'].astype(int)

dryer.head()

dryer.info()

# Drop the 'Description' column  
 dryer.drop(columns=['Description'], inplace=True)

### Electric Oven

# Assuming you have a DataFrame named 'freezer' with a column 'Description'  
  
 # Get the unique categories from the 'Description' column  
 description\_categories = electric\_oven['Description'].unique().tolist()  
  
 for i in description\_categories:  
 print(i)

# Function to extract attributes from description  
 def extract\_oven\_attributes(description):  
 description = description.lower().strip() # Normalize description: convert to lowercase and strip whitespace  
   
 # Separate number followed by "ltrs" into "<number> ltrs" format  
 if 'ltrs' in description:  
 description = description.replace('ltrs', ' ltrs') # Add space before 'ltrs' if not present  
   
 # Determine color (white, black)  
 if 'white' in description:  
 color = 'white'  
 elif 'black' in description:  
 color = 'black'  
 else:  
 color = 'white' # Default color to white if not specified  
   
 # Extract size (48, 65)  
 size = None  
 if '48' in description:  
 size = 48  
 elif '65' in description:  
 size = 65  
   
 return color, size  
  
 # Apply the function to extract attributes from each description  
 electric\_oven['Color'], electric\_oven['Size'] = zip(\*electric\_oven['Description'].apply(extract\_oven\_attributes))

electric\_oven.head()

electric\_oven.info()

# Drop the 'Description' column  
 electric\_oven.drop(columns=['Description'], inplace=True)

### Hood

# Assuming you have a DataFrame named 'freezer' with a column 'Description'  
  
 # Get the unique categories from the 'Description' column  
 description\_categories = hood['Description'].unique().tolist()  
  
 for i in description\_categories:  
 print(i)

# Function to extract attributes from description  
 def extract\_hood\_attributes(description):  
 description = description.lower().strip() # Normalize description: convert to lowercase and strip whitespace  
   
 # Separate number followed by "cm" into "<number> cm" format  
 if 'cm' in description:  
 description = description.replace('cm', ' cm') # Add space before 'cm' if not present  
   
 # Extract size (60, 90)  
 size = None  
 if '60' in description:  
 size = 60  
 elif '90' in description:  
 size = 90  
   
 # Determine color (black, ss/stainless steel)  
 color = None  
 if 'black' in description:  
 color = 'black'  
 elif 'ss' in description or 'stainless steel' in description:  
 color = 'stainless steel'  
 else:  
 color = 'black' # Default color to black if not specified  
   
 # Determine type (pop, glass back, t shape, swing, quadra)  
 if 'pop' in description:  
 type\_str = 'pop'  
 elif 'glass back' in description:  
 type\_str = 'glass back'  
 elif 't shape' in description:  
 type\_str = 't shape'  
 elif 'swing' in description:  
 type\_str = 'swing'  
 elif 'quadra' in description:  
 type\_str = 'quadra'  
 else:  
 type\_str = None  
   
 return size, color, type\_str  
  
 # Apply the function to extract attributes from each description  
 hood['Size'], hood['Color'], hood['Type'] = zip(\*hood['Description'].apply(extract\_hood\_attributes))

hood.head()

hood.info()

# Drop the 'Description' column  
 hood.drop(columns=['Description'], inplace=True)

### Laundry

# Assuming you have a DataFrame named 'freezer' with a column 'Description'  
  
 # Get the unique categories from the 'Description' column  
 description\_categories = laundry['Description'].unique().tolist()  
  
 for i in description\_categories:  
 print(i)

# Function to extract attributes from description  
 def extract\_washer\_attributes(description):  
 description = description.lower().strip() # Normalize description: convert to lowercase and strip whitespace  
   
 # Separate number followed by "kgs" or "kg" into "<number> kg" format  
 if 'kgs' in description:  
 description = description.replace('kgs', ' kg') # Replace 'kgs' with ' kg'  
 elif 'kg' in description:  
 description = description.replace('kg', ' kg') # Replace 'kg' with ' kg'  
   
 # Separate number followed by "rpm" into "<number> rpm" format  
 if 'rpm' in description:  
 description = description.replace('rpm', ' rpm') # Replace 'rpm' with ' rpm'  
   
 # Separate number followed by "rpminverter" into "<number> rpm inverter" format  
 if 'rpminverter' in description:  
 description = description.replace('rpminverter', ' rpm inverter') # Replace 'rpminverter' with ' rpm inverter'  
   
 # Extract size (7, 10, 8, 9, 13, 15)  
 size = None  
 if '7 kg' in description:  
 size = 7  
 elif '10 kg' in description:  
 size = 10  
 elif '8 kg' in description:  
 size = 8  
 elif '9 kg' in description:  
 size = 9  
 elif '13 kg' in description:  
 size = 13  
 elif '15 kg' in description:  
 size = 15  
   
 # Extract speed (1000, 1200, 1300, 1500, 1400)  
 speed = None  
 if '1000 rpm' in description:  
 speed = 1000  
 elif '1200 rpm' in description:  
 speed = 1200  
 elif '1300 rpm' in description:  
 speed = 1300  
 elif '1500 rpm' in description:  
 speed = 1500  
 elif '1400 rpm' in description:  
 speed = 1400  
 else:  
 speed = 1000  
   
 # Determine color (stainless steel, white, silver, anthracite)  
 color = None  
 if 'stainless steel' in description:  
 color = 'stainless steel'  
 elif 'white' in description:  
 color = 'white'  
 elif 'silver' in description:  
 color = 'silver'  
 elif 'anthracite' in description:  
 color = 'anthracite'  
 elif 'black' in description:  
 color = 'black'  
 else:  
 color = 'silver'  
   
 # Determine inverter (yes, no)  
 inverter = 'yes' if 'inverter' in description else 'no'  
   
 # Determine type (front load, regular)  
 type\_str = 'front load' if 'front load' in description else 'regular'  
   
 return size, speed, color, inverter, type\_str  
  
 # Apply the function to extract attributes from each description  
 laundry['Size'], laundry['Speed'], laundry['Color'], laundry['Inverter'], laundry['Type'] = zip(\*laundry['Description'].apply(extract\_washer\_attributes))  
  
 # Filter out rows with invalid descriptions (e.g., 'upright full fridge stainless steel', 'washer dryer 128kg 1400rpm white')  
 laundry = laundry[~laundry['Description'].str.contains('upright full fridge|washer dryer 128kg', case=False)]

# Convert the 'Settings' column to integer  
 laundry['Size'] = laundry['Size'].astype(int)

laundry.head()

laundry.info()

# Drop the 'Description' column  
 laundry.drop(columns=['Description'], inplace=True)

### Microwave

# Assuming you have a DataFrame named 'freezer' with a column 'Description'  
  
 # Get the unique categories from the 'Description' column  
 description\_categories = microwave['Description'].unique().tolist()  
  
 for i in description\_categories:  
 print(i)

# Function to extract attributes from description  
 def extract\_microwave\_attributes(description):  
 description = description.lower().strip() # Normalize description: convert to lowercase and strip whitespace  
   
 # Separate number followed by "liters" into "<number>" format  
 if 'liters' in description:  
 description = description.replace('liters', '').strip() # Remove 'liters' and strip whitespace  
   
 # Extract size (25, 44, 22, 30, 28)  
 size = None  
 if ' 25' in description:  
 size = 25  
 elif ' 44' in description:  
 size = 44  
 elif ' 22' in description:  
 size = 22  
 elif ' 30' in description:  
 size = 30  
 elif ' 28' in description:  
 size = 28  
   
 # Determine color (stainless steel, white, silver, black)  
 color = None  
 if 'stainless steel' in description:  
 color = 'stainless steel'  
 elif 'white' in description:  
 color = 'white'  
 elif 'silver' in description:  
 color = 'silver'  
 elif 'black' in description:  
 color = 'black'  
   
 # Determine if built-in (yes, no)  
 built\_in = 'yes' if 'builtin' in description or 'built in' in description else 'no'  
   
 # Determine if with grill (yes, no)  
 with\_grill = 'yes' if 'with grill' in description else 'no'  
   
 # Determine if with oven (yes, no)  
 with\_oven = 'yes' if 'oven' in description else 'no'  
   
 # Determine control type (mechanical, digital)  
 control\_type = 'mechanical' if 'mechanical' in description else 'digital'  
   
 return size, color, built\_in, with\_grill, with\_oven, control\_type  
  
 # Apply the function to extract attributes from each description  
 microwave['Size'], microwave['Color'], microwave['Built\_in'], microwave['With\_Grill'], microwave['With\_Oven'], microwave['Control\_Type'] = zip(\*microwave['Description'].apply(extract\_microwave\_attributes))

microwave.head()

microwave.info()

# Drop the 'Description' column  
 microwave.drop(columns=['Description'], inplace=True)

### Oven

# Assuming you have a DataFrame named 'freezer' with a column 'Description'  
  
 # Get the unique categories from the 'Description' column  
 description\_categories = oven['Description'].unique().tolist()  
  
 for i in description\_categories:  
 print(i)

# Function to extract attributes from description  
 def extract\_dispenser\_attributes(description):  
 description = description.lower().strip() # Normalize description: convert to lowercase and strip whitespace  
   
 # Extract size (60cm, 90cm)  
 size = None  
 if '60cm' in description:  
 size = 60  
 elif '90cm' in description:  
 size = 90  
 else:  
 size = 60  
   
 # Determine color (stainless steel/ss, white, black)  
 color = None  
 if 'ss' in description or 'stainless steel' in description:  
 color = 'stainless steel'  
 elif 'white' in description:  
 color = 'white'  
 elif 'black' in description:  
 color = 'black'  
   
 # Determine if built-in (yes, no)  
 built\_in = 'yes' if 'built in' in description or 'built-in' in description else 'no'  
   
 # Determine if with grill (yes, no)  
 with\_grill = 'yes' if 'grill' in description else 'no'  
   
 # Determine functionality (Full Gas, Full Electric, Gas/Electric, Full Gas)  
 functionality = None  
 if 'full gas' in description or 'gas' in description:  
 functionality = 'Full Gas'  
 elif 'electric' in description:  
 functionality = 'Full Electric'  
 elif 'gas/electric' in description:  
 functionality = 'Gas/Electric'  
 else:  
 functionality = 'Full Gas'  
   
 return size, color, built\_in, with\_grill, functionality  
  
 # Apply the function to extract attributes from each description  
 oven['Size'], oven['Color'], oven['Built\_in'], oven['With\_Grill'], oven['Functionality'] = zip(\*oven['Description'].apply(extract\_dispenser\_attributes))

oven.head(10)

oven.info()

# Drop the 'Description' column  
 oven.drop(columns=['Description'], inplace=True)

### top load

# Assuming you have a DataFrame named 'freezer' with a column 'Description'  
  
 # Get the unique categories from the 'Description' column  
 description\_categories = top\_load['Description'].unique().tolist()  
  
 for i in description\_categories:  
 print(i)

# Function to extract attributes from description  
 def extract\_washer\_attributes(description):  
 description = description.lower().strip() # Normalize description: convert to lowercase and strip whitespace  
   
 # Separate number followed by "kg" into "<number>" format  
 if 'kg' in description:  
 description = description.replace('kg', '').strip() # Remove 'kg' and strip whitespace  
   
 # Extract size (10, 13, 16)  
 size = None  
 if ' 10' in description:  
 size = 10  
 elif ' 13' in description:  
 size = 13  
 elif ' 16' in description:  
 size = 16  
   
 return size  
  
 # Apply the function to extract size from each description  
 top\_load['Size'] = top\_load['Description'].apply(extract\_washer\_attributes)

top\_load.head()

top\_load.info()

# Drop the 'Description' column  
 top\_load.drop(columns=['Description'], inplace=True)

### Table top

# Assuming you have a DataFrame named 'freezer' with a column 'Description'  
  
 # Get the unique categories from the 'Description' column  
 description\_categories = table\_top['Description'].unique().tolist()  
  
 for i in description\_categories:  
 print(i)

# Function to extract information from description  
 def extract\_info(description):  
 # Normalize description: convert to lowercase and remove extra spaces  
 description = description.lower().strip()  
   
 # Extract size using regex to find digits followed by 'cf'  
 size\_match = re.search(r'(\d+)\s\*cf', description)  
 if size\_match:  
 size = int(size\_match.group(1))  
 else:  
 size = 3  
   
 # Extract color: check for 'white' or 'inox' in description  
 if 'white' in description:  
 color = 'white'  
 elif 'inox' in description:  
 color = 'inox'  
 else:  
 color = 'white' # Default to white if neither 'white' nor 'inox' found  
   
 # Determine type based on 'refrigerator' or 'freezer' in description  
 if 'refrigerator' in description:  
 product\_type = 'Refrigerator'  
 elif 'freezer' in description:  
 product\_type = 'Freezer'  
 else:  
 product\_type = None  
   
 return size, color, product\_type  
  
 # Apply the function to extract information from each description  
 table\_top['Size'], table\_top['Color'], table\_top['Type'] = zip(\*table\_top['Description'].apply(extract\_info))

table\_top.head()

# Convert the 'Settings' column to integer  
 table\_top['Size'] = table\_top['Size'].astype(int)

table\_top.info()

# Drop the 'Description' column  
 table\_top.drop(columns=['Description'], inplace=True)

### Twin Tub

# Assuming you have a DataFrame named 'freezer' with a column 'Description'  
  
 # Get the unique categories from the 'Description' column  
 description\_categories = twin\_tub['Description'].unique().tolist()  
  
 for i in description\_categories:  
 print(i)

# Function to extract size (weight in kg) from description  
 def extract\_size(description):  
 # Normalize description: convert to lowercase and remove extra spaces  
 description = description.lower().strip()  
   
 # Extract size using regex to find digits followed by 'kg'  
 size\_match = re.search(r'(\d+)\s\*kg', description)  
 if size\_match:  
 size = int(size\_match.group(1))  
 else:  
 size = None  
   
 return size  
  
 # Apply the function to extract size from each description  
 twin\_tub['Size'] = twin\_tub['Description'].apply(extract\_size)

twin\_tub.head()

twin\_tub.info()

# Drop the 'Description' column  
 twin\_tub.drop(columns=['Description'], inplace=True)

### Vacuum

# Assuming you have a DataFrame named 'freezer' with a column 'Description'  
  
 # Get the unique categories from the 'Description' column  
 description\_categories = vacuum['Description'].unique().tolist()  
  
 for i in description\_categories:  
 print(i)

# Function to extract information from description  
 def extract\_info(description):  
 # Normalize description: convert to lowercase and strip leading/trailing spaces  
 description = description.lower().strip()  
   
 # Extract power (in watts) using regex to find digits followed by 'watts'  
 import re  
 power\_match = re.search(r'(\d+)\s\*watts', description)  
 if power\_match:  
 power = int(power\_match.group(1))  
 else:  
 power = 500 # Default power if not specified  
   
 # Determine type (handy or regular)  
 if 'handy' in description:  
 type\_str = 'Handy'  
 elif 'regular' in description:  
 type\_str = 'Regular'  
 else:  
 type\_str = 'Regular' # Default type if not specified  
   
 # Determine color (black or white)  
 if 'black' in description:  
 color = 'Black'  
 elif 'white' in description:  
 color = 'White'  
 else:  
 color = 'White' # Default color if not specified  
   
 # Determine bagless status  
 if 'bagless' in description:  
 bagless = 'Yes'  
 else:  
 bagless = 'No'  
   
 return power, type\_str, color, bagless  
  
 # Apply the function to extract information from each description  
 vacuum['Power'], vacuum['Type'], vacuum['Color'], vacuum['Bagless'] = zip(\*vacuum['Description'].apply(extract\_info))

vacuum.head()

vacuum.info()

# Drop the 'Description' column  
 vacuum.drop(columns=['Description'], inplace=True)

### Multi door

# Assuming you have a DataFrame named 'freezer' with a column 'Description'  
  
 # Get the unique categories from the 'Description' column  
 description\_categories = multi\_door['Description'].unique().tolist()  
  
 for i in description\_categories:  
 print(i)

# Function to extract information from description  
 def extract\_info(description):  
 # Normalize description: convert to lowercase and strip leading/trailing spaces  
 description = description.lower().strip()  
   
 # Extract number of doors  
 if '3 doors' in description:  
 doors = 3  
 elif '4 doors' in description:  
 doors = 4  
 else:  
 doors = 4 # Default to 4 doors if not specified  
   
 # Extract size (in cuft) using regex to find digits followed by 'cuft'  
 import re  
 size\_match = re.search(r'(\d+)\s\*cuft', description)  
 if size\_match:  
 size = int(size\_match.group(1))  
 else:  
 size = 30 # Default size if not specified  
   
 # Determine color (black, stainless steel, silver)  
 if 'black' in description:  
 color = 'Black'  
 elif 'stainless steel' in description:  
 color = 'Stainless Steel'  
 elif 'silver' in description:  
 color = 'Silver'  
 else:  
 color = 'Silver' # Default color if not specified  
   
 # Determine no frost status  
 if 'no frost' in description:  
 no\_frost = 'Yes'  
 else:  
 no\_frost = 'No'  
   
 # Determine type (bottom mount or regular)  
 if 'bottom mount' in description:  
 fridge\_type = 'Bottom Mount'  
 else:  
 fridge\_type = 'Regular'  
   
 return doors, size, color, no\_frost, fridge\_type  
  
 # Apply the function to extract information from each description  
 multi\_door['Number\_of\_Doors'], multi\_door['Size'], multi\_door['Color'], multi\_door['No\_Frost'], multi\_door['Type'] = zip(\*multi\_door['Description'].apply(extract\_info))

multi\_door.head()

multi\_door.info()

# Drop the 'Description' column  
 multi\_door.drop(columns=['Description'], inplace=True)

### Refrigerator

# Assuming you have a DataFrame named 'freezer' with a column 'Description'  
  
 # Get the unique categories from the 'Description' column  
 description\_categories = refrigerator['Description'].unique().tolist()  
  
 for i in description\_categories:  
 print(i)

# Function to extract information from description  
 def extract\_info(description):  
 # Normalize description: convert to lowercase and strip leading/trailing spaces  
 description = description.lower().strip()  
   
 # Extract Size (in cft) using regex to find digits followed by 'cft'  
 size\_match = re.search(r'(\d+)\s\*cft', description)  
 if size\_match:  
 size = int(size\_match.group(1))  
 else:  
 size = 20 # Default size if not specified  
   
 # Determine Color (white, stainless steel, silver, inox, black)  
 if 'stainless steel' in description or 'ss' in description:  
 color = 'Stainless Steel'  
 elif 'silver' in description:  
 color = 'Silver'  
 elif 'inox' in description:  
 color = 'Inox'  
 elif 'black' in description:  
 color = 'Black'  
 else:  
 color = 'White' # Default color if not specified  
   
 # Determine No Frost status (yes or no)  
 if 'nofrost' in description or 'no frost' in description:  
 no\_frost = 'Yes'  
 else:  
 no\_frost = 'No'  
   
 # Determine Number of Doors (1 for upright, 2 otherwise)  
 if 'upright' in description or 'vaccine icepack freeze' in description:  
 doors = 1  
 else:  
 doors = 2  
   
 return size, color, no\_frost, doors  
  
 # Apply the function to extract information from each description  
 refrigerator['Size'], refrigerator['Color'], refrigerator['No\_Frost'], refrigerator['Number\_of\_Doors'] = zip(\*refrigerator['Description'].apply(extract\_info))

refrigerator.head()

refrigerator.info()

# Drop the 'Description' column  
 refrigerator.drop(columns=['Description'], inplace=True)

### hob

# Assuming you have a DataFrame named 'freezer' with a column 'Description'  
  
 # Get the unique categories from the 'Description' column  
 description\_categories = hob['Description'].unique().tolist()  
  
 for i in description\_categories:  
 print(i)

# Function to extract information from description  
 def extract\_info(description):  
 # Normalize description: convert to lowercase and strip leading/trailing spaces  
 description = description.lower().strip()  
   
 # Extract Size (in cm) using regex to find digits followed by 'cm'  
 size\_match = re.search(r'(\d+)\s\*cm', description)  
 if size\_match:  
 size = int(size\_match.group(1))  
 else:  
 size = 60 # Default size if not specified  
   
 # Extract Number of Burners using regex to find digits followed by 'gas' or 'triple'  
 burners\_match = re.search(r'(\d+)\s\*(gas|triple|induction)', description)  
 if burners\_match:  
 num\_burners = int(burners\_match.group(1))  
 else:  
 num\_burners = 4 # Default number of burners if not specified  
   
 # Determine Color (stainless steel, black, white)  
 if 'stainless steel' in description or 'ss' in description:  
 color = 'Stainless Steel'  
 elif 'black' in description:  
 color = 'Black'  
 elif 'white' in description:  
 color = 'White'  
 else:  
 color = 'Black' # Default color if not specified  
   
 # Determine Safety status (Full Safety or Burner Safety)  
 if 'safety' in description:  
 safety = 'Full Safety'  
 else:  
 safety = 'Burner Safety'  
   
 # Determine Built In status (yes or no)  
 if 'built in' in description:  
 built\_in = 'Yes'  
 else:  
 built\_in = 'No'  
   
 return size, color, num\_burners, safety, built\_in  
  
 # Apply the function to extract information from each description  
 hob['Size'], hob['Color'], hob['Number\_of\_Burners'], hob['Safety'], hob['Built\_In'] = zip(\*hob['Description'].apply(extract\_info))

hob.head()

hob.info()

# Drop the 'Description' column  
 hob.drop(columns=['Description'], inplace=True)

### dap

# Assuming you have a DataFrame named 'freezer' with a column 'Description'  
  
 # Get the unique categories from the 'Description' column  
 description\_categories = dap['Description'].unique().tolist()  
  
 for i in description\_categories:  
 print(i)

# Function to extract category from description  
 def extract\_category(description):  
 # Normalize description: convert to lowercase  
 description = description.lower()  
   
 # Define category keywords mapping  
 category\_keywords = {  
 'food processor': ['food processor'],  
 'hand mixer': ['hand mixer'],  
 'citrus juicer': ['citrus juicer'],  
 'juice extractor': ['juice extractor'],  
 'table blender': ['table blender'],  
 'steam iron': ['steam iron'],  
 'hair dryer': ['hair dryer'],  
 'chopper': ['chopper'],  
 'contact grill': ['contact grill'],  
 'kettle': ['kettle'],  
 'fan heater': ['fan heater'],  
 'hand blender': ['hand blender'],  
 'mixer with bowl': ['mixer with bowl'],  
 'heater': ['heater', 'quartz elements'],  
 'iron': ['iron']  
 }  
   
 # Determine the category based on keywords  
 for category, keywords in category\_keywords.items():  
 for keyword in keywords:  
 if keyword in description:  
 return category  
   
 # Default category if no match is found  
 return None  
  
 # Apply the function to extract category from each description  
 dap['Category'] = dap['Description'].apply(extract\_category)

dap.head()

# Drop the 'Description' column  
 dap.drop(columns=['Description'], inplace=True)

dap.head()

## Customer table

### Creating the customer table

data.head()

import pandas as pd  
  
 # Assuming 'data' is your original dataframe  
 # Create a new dataframe with unique Customer IDs  
 customers = data.drop\_duplicates(subset=['Customer\_id'])  
  
 # Print the new dataframe  
 print(customers)

customers.shape

### Removing Irelevant Columns from the customer table

# List of columns to remove  
 columns\_to\_remove = ['Date', 'Family', 'Brand', 'Item\_code', 'Description',  
 'Tot\_qty', 'Payment\_terms', 'Discount', 'Return\_rate',  
 'Product\_class', 'Discount', 'Order\_id', 'Unit\_price', 'Total\_price']  
  
 # Remove the specified columns from the 'customer' dataframe  
 customers.drop(columns=columns\_to\_remove, inplace=True)

customers.head()

## Order Table

data.head()

# Specify the columns you want to keep  
 order\_columns = ['Order\_id', 'Date', 'Discount', 'Customer\_id', 'Item\_code', 'Tot\_qty', 'Return\_rate', 'Total\_price']  
  
 # Create the order dataframe  
 orders = data.loc[:, order\_columns].copy()

orders.head()

# Product Table

# Specify the columns you want to keep  
 product\_columns = ['Item\_code', 'Product\_class', 'Brand', 'Unit\_price', 'Description', 'Tot\_qty', 'Family']  
  
 # Create the order dataframe  
 product= data.loc[:, product\_columns].copy()

product.head()

# Remove duplicates of 'Item Code' column, keeping the first occurrence  
 product.drop\_duplicates(subset=['Item\_code'], keep='first', inplace=True)  
 product.reset\_index(drop=True, inplace=True)

product.head()

# SQL

## General Preliminary Code

from sqlalchemy import create\_engine  
 from sqlalchemy import create\_engine, Column, Integer, String, ForeignKey  
 from sqlalchemy.ext.declarative import declarative\_base  
 from sqlalchemy.orm import relationship  
 from sqlalchemy import Integer, VARCHAR  
 from sqlalchemy import Integer, VARCHAR, Column  
 from sqlalchemy.ext.declarative import declarative\_base

# Replace 'username', 'password', 'hostname', 'database\_name' with your MySQL credentials  
 engine = create\_engine('mysql+pymysql://root:2452002Az)@localhost/MSBA305\_Project8')

## Customer Table

### Initial Table

customers.head()

# Write the dataframe to the database  
 customers.to\_sql('Customer', con=engine, if\_exists='replace', index=False)

from sqlalchemy import text  
  
 # Define the SQL statement to change the data type  
 sql\_statement = """  
 ALTER TABLE Customer  
 MODIFY COLUMN Customer\_id VARCHAR(20),  
 MODIFY COLUMN Region VARCHAR(20),  
 MODIFY COLUMN Casa VARCHAR(20),  
 MODIFY COLUMN Customer\_loyalty\_indicator VARCHAR(20);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

# Define the SQL statement to add the primary key with a key length  
 sql\_statement = """  
 ALTER TABLE Customer  
 ADD PRIMARY KEY (Customer\_id);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 1

from sqlalchemy import text  
  
 # # Define the SQL statement to change the data type  
 sql\_statement = """  
 ALTER TABLE Customer  
 ADD CONSTRAINT unique\_email UNIQUE (Phone\_number);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

## Product table

product.head()

### Initial Table

# Write the dataframe to the database  
 product.to\_sql('Product', con=engine, if\_exists='replace', index=False)

from sqlalchemy import text  
  
 # Define the SQL statement to change the data type  
 sql\_statement = """  
 ALTER TABLE Product  
 MODIFY COLUMN Item\_code VARCHAR(20),  
 MODIFY COLUMN Product\_class VARCHAR(20),  
 MODIFY COLUMN Brand VARCHAR(35),  
 MODIFY COLUMN Unit\_price FLOAT,  
 MODIFY COLUMN Tot\_qty INTEGER,  
 MODIFY COLUMN Family VARCHAR(30),  
 MODIFY COLUMN Description VARCHAR(255);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

# Define the SQL statement to add the primary key with a key length  
 sql\_statement = """  
 ALTER TABLE Product  
 ADD PRIMARY KEY (Item\_code);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 1

from sqlalchemy import text  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Product  
 ADD CONSTRAINT check\_tot\_qty\_product CHECK (Tot\_qty > 0);  
 """  
  
 # # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 2

# Define the SQL statement to modify the column datatype  
 sql\_statement = """  
 ALTER TABLE Product  
 ADD CONSTRAINT check\_positive\_unit\_price CHECK (Unit\_price > 0);  
 """  
  
 # # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

## Order

orders.head()

### Initial Table

orders.info()

from sqlalchemy import Column, Integer, VARCHAR, Float, ForeignKey

# Write the dataframe to the database  
 orders.to\_sql('Order', con=engine, if\_exists='replace', index=False)

from sqlalchemy import text  
  
 # Define the SQL statement to change the data type  
 sql\_statement = """  
 ALTER TABLE `Order`  
 MODIFY COLUMN Order\_id INTEGER,  
 MODIFY COLUMN Item\_code VARCHAR(20),  
 MODIFY COLUMN Customer\_id VARCHAR(20),  
 MODIFY COLUMN Date DATE,  
 MODIFY COLUMN Discount FLOAT,  
 MODIFY COLUMN Tot\_qty FLOAT,  
 MODIFY COLUMN Return\_rate VARCHAR(10),  
 MODIFY COLUMN Total\_price FLOAT;  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

# Define the SQL statement to add the primary key with a key length  
 sql\_statement = """  
 ALTER TABLE `Order`  
 ADD PRIMARY KEY (Order\_id),  
 ADD FOREIGN KEY (Customer\_id) REFERENCES Customer(Customer\_id),  
 ADD FOREIGN KEY (Item\_code) REFERENCES Product(Item\_code);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 1

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE `Order`  
 ADD CONSTRAINT check\_tot\_qty\_order CHECK (Tot\_qty >= 0);  
 """  
  
 # # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 2

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE `Order`  
 ADD CONSTRAINT check\_discount\_lowerBound\_order CHECK (Discount >= 0),  
 ADD CONSTRAINT check\_discount\_upperBound\_order CHECK (Discount <= 100);  
 """  
  
 # # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 3

from sqlalchemy import text  
  
 # Define the SQL statement  
 sql\_statement = """  
 ALTER TABLE `order`  
 MODIFY COLUMN `Order\_id` INT AUTO\_INCREMENT;  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

## Wine Cooler

wine\_cooler.head()

### Initial Table

# Write the dataframe to the database  
 wine\_cooler.to\_sql('Wine\_cooler', con=engine, if\_exists='replace', index=False)

from sqlalchemy import text  
  
 # Define the SQL statement to change the data type  
 sql\_statement = """  
 ALTER TABLE Wine\_cooler  
 MODIFY COLUMN Item\_code VARCHAR(20),  
 MODIFY COLUMN Type VARCHAR(20),  
 MODIFY COLUMN Capacity INTEGER;  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

# Define the SQL statement to add the primary key with a key length  
 sql\_statement = """  
 ALTER TABLE Wine\_cooler  
 ADD FOREIGN KEY (Item\_code) REFERENCES Product(Item\_code);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 1

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Wine\_cooler  
 ADD CONSTRAINT check\_capacity\_wine\_cooler CHECK (Capacity > 0);  
 """  
  
 # # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 2

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Wine\_cooler  
 DROP FOREIGN KEY wine\_cooler\_ibfk\_1,  
 ADD FOREIGN KEY (Item\_code)  
 REFERENCES Product(Item\_code)  
 ON DELETE CASCADE;  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

## AC

ac.head()

### Initial Table

# Write the dataframe to the database  
 ac.to\_sql('AC', con=engine, if\_exists='replace', index=False)

from sqlalchemy import text  
  
 # Define the SQL statement to change the data type  
 sql\_statement = """  
 ALTER TABLE AC  
 MODIFY COLUMN Item\_code VARCHAR(20),  
 MODIFY COLUMN Inverter VARCHAR(5),  
 MODIFY COLUMN BTU INTEGER;  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

# Define the SQL statement to add the primary key with a key length  
 sql\_statement = """  
 ALTER TABLE AC  
 ADD FOREIGN KEY (Item\_code) REFERENCES Product(Item\_code);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 1

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE AC  
 ADD CONSTRAINT check\_BTU\_AC CHECK (BTU > 0);  
 """  
  
 # # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 2

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE AC  
 DROP FOREIGN KEY AC\_ibfk\_1,  
 ADD FOREIGN KEY (Item\_code)  
 REFERENCES Product(Item\_code)  
 ON DELETE CASCADE;  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

## DVD

dvd.head()

### Initial Table

# Write the dataframe to the database  
 dvd.to\_sql('DVD', con=engine, if\_exists='replace', index=False)

from sqlalchemy import text  
  
 # Define the SQL statement to change the data type  
 sql\_statement = """  
 ALTER TABLE DVD  
 MODIFY COLUMN Item\_code VARCHAR(20),  
 MODIFY COLUMN HDMI VARCHAR(5);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

# Define the SQL statement to add the primary key with a key length  
 sql\_statement = """  
 ALTER TABLE DVD  
 ADD FOREIGN KEY (Item\_code) REFERENCES Product(Item\_code);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 1

from sqlalchemy import text  
  
 # Define the SQL statement  
 sql\_statement = """  
 ALTER TABLE DVD  
 ADD CONSTRAINT check\_yes\_no CHECK (HDMI IN ('yes', 'no'));  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 2

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE DVD  
 DROP FOREIGN KEY DVD\_ibfk\_1,  
 ADD FOREIGN KEY (Item\_code)  
 REFERENCES Product(Item\_code)  
 ON DELETE CASCADE;  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

## Sound Bar

sound\_bar.head()

### Initial Table

# Write the dataframe to the database  
 sound\_bar.to\_sql('Sound\_Bar', con=engine, if\_exists='replace', index=False)

from sqlalchemy import text  
  
 # Define the SQL statement to change the data type  
 sql\_statement = """  
 ALTER TABLE Sound\_Bar  
 MODIFY COLUMN Item\_code VARCHAR(20),  
 MODIFY COLUMN Amplifier\_Output INTEGER;  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

# Define the SQL statement to add the primary key with a key length  
 sql\_statement = """  
 ALTER TABLE Sound\_Bar  
 ADD FOREIGN KEY (Item\_code) REFERENCES Product(Item\_code);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 1

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Sound\_Bar  
 ADD CONSTRAINT check\_Amplifier\_Output CHECK (Amplifier\_Output >= 0);  
 """  
  
 # # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 2

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Sound\_Bar  
 DROP FOREIGN KEY Sound\_Bar\_ibfk\_1,  
 ADD FOREIGN KEY (Item\_code)  
 REFERENCES Product(Item\_code)  
 ON DELETE CASCADE;  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

## Humidor

humidor.head()

### Initial Table

# Write the dataframe to the database  
 humidor.to\_sql('Humidor', con=engine, if\_exists='replace', index=False)

from sqlalchemy import text  
  
 # Define the SQL statement to change the data type  
 sql\_statement = """  
 ALTER TABLE Humidor  
 MODIFY COLUMN Item\_code VARCHAR(20),  
 MODIFY COLUMN Capacity INTEGER;  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

# Define the SQL statement to add the primary key with a key length  
 sql\_statement = """  
 ALTER TABLE Humidor  
 ADD FOREIGN KEY (Item\_code) REFERENCES Product(Item\_code);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 1

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Humidor  
 ADD CONSTRAINT check\_capacity CHECK (Capacity >= 0);  
 """  
  
 # # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 2

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Humidor  
 DROP FOREIGN KEY Humidor\_ibfk\_1,  
 ADD FOREIGN KEY (Item\_code)  
 REFERENCES Product(Item\_code)  
 ON DELETE CASCADE;  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

## Freezer

freezer.head()

# Write the dataframe to the database  
 freezer.to\_sql('Freezer', con=engine, if\_exists='replace', index=False)

from sqlalchemy import text  
  
 # Define the SQL statement to change the data type  
 sql\_statement = """  
 ALTER TABLE Freezer  
 MODIFY COLUMN Item\_code VARCHAR(20),  
 MODIFY COLUMN Number\_of\_Drawers INTEGER,  
 MODIFY COLUMN Color VARCHAR(20),  
 MODIFY COLUMN No\_Frost VARCHAR(5);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

# Define the SQL statement to add the primary key with a key length  
 sql\_statement = """  
 ALTER TABLE Freezer  
 ADD FOREIGN KEY (Item\_code) REFERENCES Product(Item\_code);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 1

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Freezer  
 ADD CONSTRAINT check\_number\_of\_drawers CHECK (Number\_of\_Drawers >= 0);  
 """  
  
 # # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 2

from sqlalchemy import text  
  
 # Define the SQL statement  
 sql\_statement = """  
 ALTER TABLE Freezer  
 ADD CONSTRAINT check\_yes\_no\_freezer CHECK (No\_Frost IN ('yes', 'no'));  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 3

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Freezer  
 DROP FOREIGN KEY Freezer\_ibfk\_1,  
 ADD FOREIGN KEY (Item\_code)  
 REFERENCES Product(Item\_code)  
 ON DELETE CASCADE;  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

## LED

led.head()

### Initial Table

# Write the dataframe to the database  
 led.to\_sql('LED', con=engine, if\_exists='replace', index=False)

from sqlalchemy import text  
  
 # Define the SQL statement to change the data type  
 sql\_statement = """  
 ALTER TABLE LED  
 MODIFY COLUMN Item\_code VARCHAR(20),  
 MODIFY COLUMN Size INTEGER,  
 MODIFY COLUMN Smart VARCHAR(5),  
 MODIFY COLUMN Resolution VARCHAR(10);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

# Define the SQL statement to add the primary key with a key length  
 sql\_statement = """  
 ALTER TABLE LED  
 ADD FOREIGN KEY (Item\_code) REFERENCES Product(Item\_code);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 1

from sqlalchemy import text  
  
 # Define the SQL statement  
 sql\_statement = """  
 ALTER TABLE LED  
 ADD CONSTRAINT check\_yes\_no\_LED CHECK (Smart IN ('yes', 'no'));  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 2

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE LED  
 ADD CONSTRAINT check\_size\_LED CHECK (Size > 0);  
 """  
  
 # # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 3

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE LED  
 DROP FOREIGN KEY LED\_ibfk\_1,  
 ADD FOREIGN KEY (Item\_code)  
 REFERENCES Product(Item\_code)  
 ON DELETE CASCADE;  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

## Cooker

cooker.head()

### Initial Table

# Write the dataframe to the database  
 cooker.to\_sql('Cooker', con=engine, if\_exists='replace', index=False)

from sqlalchemy import text  
  
 # Define the SQL statement to change the data type  
 sql\_statement = """  
 ALTER TABLE Cooker  
 MODIFY COLUMN Item\_code VARCHAR(20),  
 MODIFY COLUMN Dimension VARCHAR(10),  
 MODIFY COLUMN Color VARCHAR(30),  
 MODIFY COLUMN Bottle\_Compartment VARCHAR(5),  
 MODIFY COLUMN Full\_Safety VARCHAR(5);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

# Define the SQL statement to add the primary key with a key length  
 sql\_statement = """  
 ALTER TABLE Cooker  
 ADD FOREIGN KEY (Item\_code) REFERENCES Product(Item\_code);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 1

from sqlalchemy import text  
  
 # Define the SQL statement  
 sql\_statement = """  
 ALTER TABLE Cooker  
 ADD CONSTRAINT bottle\_compartment\_Cooker CHECK (Bottle\_Compartment IN ('yes', 'no'));  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 2

from sqlalchemy import text  
  
 # Define the SQL statement  
 sql\_statement = """  
 ALTER TABLE Cooker  
 ADD CONSTRAINT full\_safety\_Cooker CHECK (Full\_Safety IN ('yes', 'no'));  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 3

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Cooker  
 DROP FOREIGN KEY Cooker\_ibfk\_1,  
 ADD FOREIGN KEY (Item\_code)  
 REFERENCES Product(Item\_code)  
 ON DELETE CASCADE;  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

## Dishwasher

dishwasher.head()

### Initial Table

# Write the dataframe to the database  
 dishwasher.to\_sql('Dishwasher', con=engine, if\_exists='replace', index=False)

from sqlalchemy import text  
  
 # Define the SQL statement to change the data type  
 sql\_statement = """  
 ALTER TABLE Dishwasher  
 MODIFY COLUMN Item\_code VARCHAR(20),  
 MODIFY COLUMN Settings INTEGER,  
 MODIFY COLUMN Color VARCHAR(30),  
 MODIFY COLUMN Controls VARCHAR(30);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

# Define the SQL statement to add the primary key with a key length  
 sql\_statement = """  
 ALTER TABLE Dishwasher  
 ADD FOREIGN KEY (Item\_code) REFERENCES Product(Item\_code);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 1

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Dishwasher  
 ADD CONSTRAINT settings\_Dishwasher CHECK (Settings >= 1);  
 """  
  
 # # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 2

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Dishwasher  
 DROP FOREIGN KEY Dishwasher\_ibfk\_1,  
 ADD FOREIGN KEY (Item\_code)  
 REFERENCES Product(Item\_code)  
 ON DELETE CASCADE;  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

## Dispenser

dispenser.head()

### Initial Table

# Write the dataframe to the database  
 dispenser.to\_sql('Dispenser', con=engine, if\_exists='replace', index=False)

from sqlalchemy import text  
  
 # Define the SQL statement to change the data type  
 sql\_statement = """  
 ALTER TABLE Dispenser  
 MODIFY COLUMN Item\_code VARCHAR(20),  
 MODIFY COLUMN Number\_of\_Faucets INTEGER,  
 MODIFY COLUMN Color VARCHAR(30),  
 MODIFY COLUMN Load\_Type VARCHAR(30);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

# Define the SQL statement to add the primary key with a key length  
 sql\_statement = """  
 ALTER TABLE Dispenser  
 ADD FOREIGN KEY (Item\_code) REFERENCES Product(Item\_code);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 1

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Dispenser  
 ADD CONSTRAINT number\_of\_faucets\_Dispenser CHECK (Number\_of\_Faucets >= 1);  
 """  
  
 # # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 2

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Dispenser  
 DROP FOREIGN KEY Dispenser\_ibfk\_1,  
 ADD FOREIGN KEY (Item\_code)  
 REFERENCES Product(Item\_code)  
 ON DELETE CASCADE;  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

## Dryer

dryer.head()

### Initial Table

# Write the dataframe to the database  
 dryer.to\_sql('Dryer', con=engine, if\_exists='replace', index=False)

from sqlalchemy import text  
  
 # Define the SQL statement to change the data type  
 sql\_statement = """  
 ALTER TABLE Dryer  
 MODIFY COLUMN Item\_code VARCHAR(20),  
 MODIFY COLUMN Size INTEGER,  
 MODIFY COLUMN Color VARCHAR(30),  
 MODIFY COLUMN Functionality VARCHAR(30);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

# Define the SQL statement to add the primary key with a key length  
 sql\_statement = """  
 ALTER TABLE Dryer  
 ADD FOREIGN KEY (Item\_code) REFERENCES Product(Item\_code);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 1

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Dryer  
 ADD CONSTRAINT size\_Dryer CHECK (size > 0);  
 """  
  
 # # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 2

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Dryer  
 DROP FOREIGN KEY Dryer\_ibfk\_1,  
 ADD FOREIGN KEY (Item\_code)  
 REFERENCES Product(Item\_code)  
 ON DELETE CASCADE;  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

## Electric Oven

electric\_oven.head()

### Initial Table

# Write the dataframe to the database  
 electric\_oven.to\_sql('Electric\_Oven', con=engine, if\_exists='replace', index=False)

from sqlalchemy import text  
  
 # Define the SQL statement to change the data type  
 sql\_statement = """  
 ALTER TABLE Electric\_Oven  
 MODIFY COLUMN Item\_code VARCHAR(20),  
 MODIFY COLUMN Size INTEGER,  
 MODIFY COLUMN Color VARCHAR(30);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

# Define the SQL statement to add the primary key with a key length  
 sql\_statement = """  
 ALTER TABLE Electric\_Oven  
 ADD FOREIGN KEY (Item\_code) REFERENCES Product(Item\_code);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 1

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Electric\_Oven  
 ADD CONSTRAINT size\_Electric\_Oven CHECK (Size > 0);  
 """  
  
 # # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 2

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Electric\_Oven  
 DROP FOREIGN KEY Electric\_Oven\_ibfk\_1,  
 ADD FOREIGN KEY (Item\_code)  
 REFERENCES Product(Item\_code)  
 ON DELETE CASCADE;  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

## Hood

hood.head()

### Initial Table

# Write the dataframe to the database  
 hood.to\_sql('Hood', con=engine, if\_exists='replace', index=False)

from sqlalchemy import text  
  
 # Define the SQL statement to change the data type  
 sql\_statement = """  
 ALTER TABLE Hood  
 MODIFY COLUMN Item\_code VARCHAR(20),  
 MODIFY COLUMN Size INTEGER,  
 MODIFY COLUMN Color VARCHAR(30),  
 MODIFY COLUMN Type VARCHAR(20);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

# Define the SQL statement to add the primary key with a key length  
 sql\_statement = """  
 ALTER TABLE Hood  
 ADD FOREIGN KEY (Item\_code) REFERENCES Product(Item\_code);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 1

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Hood  
 ADD CONSTRAINT size\_Hood CHECK (Size > 0);  
 """  
  
 # # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 2

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Hood  
 DROP FOREIGN KEY Hood\_ibfk\_1,  
 ADD FOREIGN KEY (Item\_code)  
 REFERENCES Product(Item\_code)  
 ON DELETE CASCADE;  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

## Laundry

laundry.head()

### Initial Table

# Write the dataframe to the database  
 laundry.to\_sql('Laundry', con=engine, if\_exists='replace', index=False)

from sqlalchemy import text  
  
 # Define the SQL statement to change the data type  
 sql\_statement = """  
 ALTER TABLE Laundry  
 MODIFY COLUMN Item\_code VARCHAR(20),  
 MODIFY COLUMN Size INTEGER,  
 MODIFY COLUMN Speed INTEGER,  
 MODIFY COLUMN Inverter VARCHAR(20),  
 MODIFY COLUMN Color VARCHAR(30),  
 MODIFY COLUMN Type VARCHAR(20);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

# Define the SQL statement to add the primary key with a key length  
 sql\_statement = """  
 ALTER TABLE Laundry  
 ADD FOREIGN KEY (Item\_code) REFERENCES Product(Item\_code);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 1

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Laundry  
 ADD CONSTRAINT speed\_Laundry CHECK (Speed > 0);  
 """  
  
 # # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 2

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Laundry  
 ADD CONSTRAINT size\_Laundry CHECK (Speed > 0);  
 """  
  
 # # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 3

from sqlalchemy import text  
  
 # Define the SQL statement  
 sql\_statement = """  
 ALTER TABLE Laundry  
 ADD CONSTRAINT check\_yes\_no\_Laundry CHECK (Inverter IN ('yes', 'no'));  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 4

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Laundry  
 DROP FOREIGN KEY Laundry\_ibfk\_1,  
 ADD FOREIGN KEY (Item\_code)  
 REFERENCES Product(Item\_code)  
 ON DELETE CASCADE;  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

## Microwave

microwave.head()

### Initial Table

# Write the dataframe to the database  
 microwave.to\_sql('Microwave', con=engine, if\_exists='replace', index=False)

from sqlalchemy import text  
  
 # Define the SQL statement to change the data type  
 sql\_statement = """  
 ALTER TABLE Microwave  
 MODIFY COLUMN Item\_code VARCHAR(20),  
 MODIFY COLUMN Size INTEGER,  
 MODIFY COLUMN Color VARCHAR(30),  
 MODIFY COLUMN Built\_in VARCHAR(5),  
 MODIFY COLUMN With\_Grill VARCHAR(5),  
 MODIFY COLUMN With\_Oven VARCHAR(5),  
 MODIFY COLUMN Control\_Type VARCHAR(20);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

# Define the SQL statement to add the primary key with a key length  
 sql\_statement = """  
 ALTER TABLE Microwave  
 ADD FOREIGN KEY (Item\_code) REFERENCES Product(Item\_code);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 1

from sqlalchemy import text  
  
 # Define the SQL statement  
 sql\_statement = """  
 ALTER TABLE Microwave  
 ADD CONSTRAINT check\_yes\_no\_Microwave CHECK (Built\_in IN ('yes', 'no'));  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 2

from sqlalchemy import text  
  
 # Define the SQL statement  
 sql\_statement = """  
 ALTER TABLE Microwave  
 ADD CONSTRAINT check\_yes\_no\_Microwave\_Grill CHECK (With\_Grill IN ('yes', 'no'));  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 3

from sqlalchemy import text  
  
 # Define the SQL statement  
 sql\_statement = """  
 ALTER TABLE Microwave  
 ADD CONSTRAINT check\_yes\_no\_Microwave\_Oven CHECK (With\_Oven IN ('yes', 'no'));  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 4

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Microwave  
 ADD CONSTRAINT size\_Microwave CHECK (Size > 0);  
 """  
  
 # # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 5

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Microwave  
 DROP FOREIGN KEY Microwave\_ibfk\_1,  
 ADD FOREIGN KEY (Item\_code)  
 REFERENCES Product(Item\_code)  
 ON DELETE CASCADE;  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

## Oven

oven.head()

### Initial Table

# Write the dataframe to the database  
 oven.to\_sql('Oven', con=engine, if\_exists='replace', index=False)

from sqlalchemy import text  
  
 # Define the SQL statement to change the data type  
 sql\_statement = """  
 ALTER TABLE Oven  
 MODIFY COLUMN Item\_code VARCHAR(20),  
 MODIFY COLUMN Size INTEGER,  
 MODIFY COLUMN Color VARCHAR(30),  
 MODIFY COLUMN Built\_in VARCHAR(5),  
 MODIFY COLUMN With\_Grill VARCHAR(5),  
 MODIFY COLUMN Functionality VARCHAR(20);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

# Define the SQL statement to add the primary key with a key length  
 sql\_statement = """  
 ALTER TABLE Oven  
 ADD FOREIGN KEY (Item\_code) REFERENCES Product(Item\_code);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 1

from sqlalchemy import text  
  
 # Define the SQL statement  
 sql\_statement = """  
 ALTER TABLE Oven  
 ADD CONSTRAINT check\_yes\_no\_Oven\_Grill CHECK (With\_Grill IN ('yes', 'no'));  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 2

from sqlalchemy import text  
  
 # Define the SQL statement  
 sql\_statement = """  
 ALTER TABLE Oven  
 ADD CONSTRAINT check\_yes\_no\_Oven\_Built\_in CHECK (Built\_in IN ('yes', 'no'));  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 3

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Oven  
 ADD CONSTRAINT size\_Oven CHECK (Size > 0);  
 """  
  
 # # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 4

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Oven  
 DROP FOREIGN KEY Oven\_ibfk\_1,  
 ADD FOREIGN KEY (Item\_code)  
 REFERENCES Product(Item\_code)  
 ON DELETE CASCADE;  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

## Top load

top\_load.head()

### Initial Table

# Write the dataframe to the database  
 top\_load.to\_sql('Top\_Load', con=engine, if\_exists='replace', index=False)

from sqlalchemy import text  
  
 # Define the SQL statement to change the data type  
 sql\_statement = """  
 ALTER TABLE Top\_Load  
 MODIFY COLUMN Item\_code VARCHAR(20),  
 MODIFY COLUMN Size INTEGER;  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

# Define the SQL statement to add the primary key with a key length  
 sql\_statement = """  
 ALTER TABLE Top\_Load  
 ADD FOREIGN KEY (Item\_code) REFERENCES Product(Item\_code);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 1

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Top\_Load  
 ADD CONSTRAINT size\_Top\_Load CHECK (Size > 0);  
 """  
  
 # # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 2

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Top\_Load  
 DROP FOREIGN KEY Top\_Load\_ibfk\_1,  
 ADD FOREIGN KEY (Item\_code)  
 REFERENCES Product(Item\_code)  
 ON DELETE CASCADE;  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

## Table Top

table\_top.head()

### Initial Table

#Write the dataframe to the database  
 table\_top.to\_sql('Table\_Top', con=engine, if\_exists='replace', index=False)

from sqlalchemy import text  
  
 # Define the SQL statement to change the data type  
 sql\_statement = """  
 ALTER TABLE Table\_Top  
 MODIFY COLUMN Item\_code VARCHAR(20),  
 MODIFY COLUMN Size INTEGER,  
 MODIFY COLUMN Color VARCHAR(30),  
 MODIFY COLUMN Type VARCHAR(30);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

# Define the SQL statement to add the primary key with a key length  
 sql\_statement = """  
 ALTER TABLE Table\_Top  
 ADD FOREIGN KEY (Item\_code) REFERENCES Product(Item\_code);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 1

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Table\_Top  
 ADD CONSTRAINT size\_Table\_Top CHECK (Size > 0);  
 """  
  
 # # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 2

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Table\_Top  
 DROP FOREIGN KEY Table\_Top\_ibfk\_1,  
 ADD FOREIGN KEY (Item\_code)  
 REFERENCES Product(Item\_code)  
 ON DELETE CASCADE;  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

## Twin Tub

twin\_tub.head()

### Initial Table

#Write the dataframe to the database  
 twin\_tub.to\_sql('Twin\_Tub', con=engine, if\_exists='replace', index=False)

from sqlalchemy import text  
  
 # Define the SQL statement to change the data type  
 sql\_statement = """  
 ALTER TABLE Twin\_Tub  
 MODIFY COLUMN Item\_code VARCHAR(20),  
 MODIFY COLUMN Size INTEGER;  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

# Define the SQL statement to add the primary key with a key length  
 sql\_statement = """  
 ALTER TABLE Twin\_Tub  
 ADD FOREIGN KEY (Item\_code) REFERENCES Product(Item\_code);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 1

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Twin\_Tub  
 ADD CONSTRAINT size\_Twin\_Tub CHECK (Size > 0);  
 """  
  
 # # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 2

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Twin\_Tub  
 DROP FOREIGN KEY TWin\_Tub\_ibfk\_1,  
 ADD FOREIGN KEY (Item\_code)  
 REFERENCES Product(Item\_code)  
 ON DELETE CASCADE;  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

## Vacuum

vacuum.head()

### Initial Table

# Write the dataframe to the database  
 vacuum.to\_sql('Vacuum', con=engine, if\_exists='replace', index=False)

from sqlalchemy import text  
  
 # Define the SQL statement to change the data type  
 sql\_statement = """  
 ALTER TABLE Vacuum  
 MODIFY COLUMN Item\_code VARCHAR(20),  
 MODIFY COLUMN Power INTEGER,  
 MODIFY COLUMN Type VARCHAR(30),  
 MODIFY COLUMN Color VARCHAR(30),  
 MODIFY COLUMN Bagless VARCHAR(5);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

# Define the SQL statement to add the primary key with a key length  
 sql\_statement = """  
 ALTER TABLE Vacuum  
 ADD FOREIGN KEY (Item\_code) REFERENCES Product(Item\_code);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 1

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Vacuum  
 ADD CONSTRAINT Power\_Vacuum CHECK (Power > 0);  
 """  
  
 # # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 2

from sqlalchemy import text  
  
 # Define the SQL statement  
 sql\_statement = """  
 ALTER TABLE Vacuum  
 ADD CONSTRAINT check\_yes\_no\_Vacuum CHECK (Bagless IN ('yes', 'no'));  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 3

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Vacuum  
 DROP FOREIGN KEY Vacuum\_ibfk\_1,  
 ADD FOREIGN KEY (Item\_code)  
 REFERENCES Product(Item\_code)  
 ON DELETE CASCADE;  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

## Multi Door

multi\_door.head()

### Initial Table

# Write the dataframe to the database  
 multi\_door.to\_sql('Multi\_Door', con=engine, if\_exists='replace', index=False)

from sqlalchemy import text  
  
 # Define the SQL statement to change the data type  
 sql\_statement = """  
 ALTER TABLE Multi\_Door  
 MODIFY COLUMN Item\_code VARCHAR(20),  
 MODIFY COLUMN Number\_of\_Doors INTEGER,  
 MODIFY COLUMN Size INTEGER,  
 MODIFY COLUMN Color VARCHAR(30),  
 MODIFY COLUMN No\_Frost VARCHAR(5),  
 MODIFY COLUMN Type VARCHAR(30);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

# Define the SQL statement to add the primary key with a key length  
 sql\_statement = """  
 ALTER TABLE Multi\_Door  
 ADD FOREIGN KEY (Item\_code) REFERENCES Product(Item\_code);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 1

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Multi\_Door  
 ADD CONSTRAINT number\_of\_doors\_multi\_door CHECK (Number\_of\_Doors > 0);  
 """  
  
 # # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 2

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Multi\_Door  
 ADD CONSTRAINT size\_multi\_door CHECK (Size > 0);  
 """  
  
 # # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 3

from sqlalchemy import text  
  
 # Define the SQL statement  
 sql\_statement = """  
 ALTER TABLE Multi\_door  
 ADD CONSTRAINT check\_yes\_no\_Multi\_door\_No\_Frost CHECK (No\_Frost IN ('yes', 'no'));  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 4

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Multi\_Door  
 DROP FOREIGN KEY Multi\_Door\_ibfk\_1,  
 ADD FOREIGN KEY (Item\_code)  
 REFERENCES Product(Item\_code)  
 ON DELETE CASCADE;  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

## Refrigerator

refrigerator.head()

### Initial Table

# Write the dataframe to the database  
 refrigerator.to\_sql('Refrigerator', con=engine, if\_exists='replace', index=False)

from sqlalchemy import text  
  
 # Define the SQL statement to change the data type  
 sql\_statement = """  
 ALTER TABLE Refrigerator  
 MODIFY COLUMN Item\_code VARCHAR(20),  
 MODIFY COLUMN Size INTEGER,  
 MODIFY COLUMN Color VARCHAR(30),  
 MODIFY COLUMN No\_Frost VARCHAR(5),  
 MODIFY COLUMN Number\_of\_Doors INTEGER;  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

# Define the SQL statement to add the primary key with a key length  
 sql\_statement = """  
 ALTER TABLE Refrigerator  
 ADD FOREIGN KEY (Item\_code) REFERENCES Product(Item\_code);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 1

from sqlalchemy import text  
  
 # Define the SQL statement  
 sql\_statement = """  
 ALTER TABLE Refrigerator  
 ADD CONSTRAINT check\_yes\_no\_Refrigerator\_No\_Frost CHECK (No\_Frost IN ('yes', 'no'));  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 2

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Refrigerator  
 ADD CONSTRAINT number\_of\_doors\_Refrigerator CHECK (Number\_of\_Doors > 0);  
 """  
  
 # # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 3

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Refrigerator  
 ADD CONSTRAINT size\_Refrigerator CHECK (Size > 0);  
 """  
  
 # # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 4

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Refrigerator  
 DROP FOREIGN KEY Refrigerator\_ibfk\_1,  
 ADD FOREIGN KEY (Item\_code)  
 REFERENCES Product(Item\_code)  
 ON DELETE CASCADE;  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

## Hob

hob.head()

### Initial Table

# Write the dataframe to the database  
 hob.to\_sql('Hob', con=engine, if\_exists='replace', index=False)

from sqlalchemy import text  
  
 # Define the SQL statement to change the data type  
 sql\_statement = """  
 ALTER TABLE Hob  
 MODIFY COLUMN Item\_code VARCHAR(20),  
 MODIFY COLUMN Size INTEGER,  
 MODIFY COLUMN Color VARCHAR(30),  
 MODIFY COLUMN Number\_of\_Burners INTEGER,  
 MODIFY COLUMN Safety VARCHAR(30),  
 MODIFY COLUMN Built\_in VARCHAR(5);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

# Define the SQL statement to add the primary key with a key length  
 sql\_statement = """  
 ALTER TABLE Hob  
 ADD FOREIGN KEY (Item\_code) REFERENCES Product(Item\_code);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 1

from sqlalchemy import text  
  
 # Define the SQL statement  
 sql\_statement = """  
 ALTER TABLE Hob  
 ADD CONSTRAINT check\_yes\_no\_Hob\_Built\_in CHECK (Built\_in IN ('yes', 'no'));  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 2

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Hob  
 ADD CONSTRAINT size\_Hob CHECK (Size > 0);  
 """  
  
 # # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 3

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Hob  
 ADD CONSTRAINT Number\_of\_Burners\_Hob CHECK (Number\_of\_Burners > 0);  
 """  
  
 # # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 4

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Hob  
 DROP FOREIGN KEY Hob\_ibfk\_1,  
 ADD FOREIGN KEY (Item\_code)  
 REFERENCES Product(Item\_code)  
 ON DELETE CASCADE;  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

## Dap

dap.head()

### Initial Table

# # # Write the dataframe to the database  
 dap.to\_sql('Dap', con=engine, if\_exists='replace', index=False)

from sqlalchemy import text  
  
 # Define the SQL statement to change the data type  
 sql\_statement = """  
 ALTER TABLE Dap  
 MODIFY COLUMN Item\_code VARCHAR(20),  
 MODIFY COLUMN Category VARCHAR(30);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

# Define the SQL statement to add the primary key with a key length  
 sql\_statement = """  
 ALTER TABLE Dap  
 ADD FOREIGN KEY (Item\_code) REFERENCES Product(Item\_code);  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

### Constraint 1

from sqlalchemy import text  
  
 # Define the SQL statements  
 sql\_statement = """  
 ALTER TABLE Dap  
 DROP FOREIGN KEY Dap\_ibfk\_1,  
 ADD FOREIGN KEY (Item\_code)  
 REFERENCES Product(Item\_code)  
 ON DELETE CASCADE;  
 """  
  
 # Execute the SQL statement  
 with engine.connect() as connection:  
 connection.execute(text(sql\_statement))

Code:

import pandas as pd  
 # Load the dataset  
 file\_path = r"C:\Users\user\OneDrive\Desktop\Updated\_FinalData305.csv"  
 data = pd.read\_csv(file\_path)  
  
 # Display the first few rows of the dataset  
 print(data.head())  
  
 # Check for the data types and missing values  
 print(data.info())

Output:

Date Casa Region Brand Family Item Code \  
 0 9/19/20 0:00 Mount Lebanon North Metn Westpoint Wine cooler W045   
 1 7/12/18 0:00 Bekaa Baalabeck Kelvinator Multi Door KT23   
 2 7/4/18 0:00 Bekaa Zahle Kelvinator Multi Door KT23   
 3 3/23/19 0:00 Bekaa Zahle Kelvinator Multi Door KT23   
 4 1/17/18 0:00 Mount Lebanon Chouf Kelvinator Multi Door KT23   
  
 Description Tot Qty Unit Price \  
 0 Cooler 1.0 712.500000   
 1 3 doors Refrigerator no Frost Stainless steel 1.0 2158.772981   
 2 3 doors Refrigerator no Frost Stainless steel 1.0 1818.000000   
 3 3 doors Refrigerator no Frost Stainless steel 1.0 1818.000000   
 4 3 doors Refrigerator no Frost Stainless steel 1.0 1544.550000   
  
 Total Price Payment Terms Return Rate Product Class Bins Discount \  
 0 712.500000 Net 60 no Appliances NaN NaN   
 1 2158.772981 Net 30 no Appliances NaN NaN   
 2 1818.000000 Net 30 no Appliances NaN NaN   
 3 1818.000000 Net 30 no Appliances NaN NaN   
 4 1544.550000 Cash on Delivery no Appliances NaN NaN   
  
 Customer ID Order ID Phone Number Customer Loyalty Indicator   
 0 Cust007 1 70406054 Returning   
 1 Cust039 2 80778885 New   
 2 Cust033 3 3202242 New   
 3 Cust058 4 80271760 New   
 4 Cust022 5 81509199 Frequent   
 <class 'pandas.core.frame.DataFrame'>  
 RangeIndex: 31278 entries, 0 to 31277  
 Data columns (total 19 columns):  
 # Column Non-Null Count Dtype   
 --- ------ -------------- -----   
 0 Date 31278 non-null object  
 1 Casa 31278 non-null object  
 2 Region 31278 non-null object  
 3 Brand 31278 non-null object  
 4 Family 31278 non-null object  
 5 Item Code 31278 non-null object  
 6 Description 31278 non-null object  
 7 Tot Qty 31276 non-null float64  
 8 Unit Price 31278 non-null float64  
 9 Total Price 31278 non-null float64  
 10 Payment Terms 29715 non-null object  
 11 Return Rate 31278 non-null object  
 12 Product Class 31278 non-null object  
 13 Bins 10616 non-null object  
 14 Discount 10616 non-null float64  
 15 Customer ID 31278 non-null object  
 16 Order ID 31278 non-null int64   
 17 Phone Number 31278 non-null int64   
 18 Customer Loyalty Indicator 31278 non-null object  
 dtypes: float64(4), int64(2), object(13)  
 memory usage: 4.5+ MB  
 None

Code:

from sklearn.model\_selection import train\_test\_split  
 # Splitting the data into training and validation sets with an 80-20 ratio  
 train\_data, val\_data = train\_test\_split(data, test\_size=0.2, random\_state=42)  
  
 # Print the shape of the datasets to confirm the split  
 print(f"Training Data Shape: {train\_data.shape}")  
 print(f"Validation Data Shape: {val\_data.shape}")

Output:

Training Data Shape: (25022, 19)  
 Validation Data Shape: (6256, 19)

Code:

# Filling numerical missing values in 'Tot Qty' with the median  
 train\_data['Tot Qty'] = train\_data['Tot Qty'].fillna(train\_data['Tot Qty'].median())  
 val\_data['Tot Qty'] = val\_data['Tot Qty'].fillna(train\_data['Tot Qty'].median())  
  
 # Filling categorical missing values in 'Payment Terms' with the mode  
 train\_data['Payment Terms'] = train\_data['Payment Terms'].fillna(train\_data['Payment Terms'].mode()[0])  
 val\_data['Payment Terms'] = val\_data['Payment Terms'].fillna(train\_data['Payment Terms'].mode()[0])

Output:

Code:

# Convert 'Date' to datetime  
 train\_data['Date'] = pd.to\_datetime(train\_data['Date'])  
 val\_data['Date'] = pd.to\_datetime(val\_data['Date'])  
  
 # Extract year, month, and day from 'Date'  
 train\_data['Year'] = train\_data['Date'].dt.year  
 train\_data['Month'] = train\_data['Date'].dt.month  
 train\_data['Day'] = train\_data['Date'].dt.day  
  
 val\_data['Year'] = val\_data['Date'].dt.year  
 val\_data['Month'] = val\_data['Date'].dt.month  
 val\_data['Day'] = val\_data['Date'].dt.day

Output:

C:\Users\user\AppData\Local\Temp\ipykernel\_16596\3439131288.py:2: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `dateutil`. To ensure parsing is consistent and as-expected, please specify a format.  
 train\_data['Date'] = pd.to\_datetime(train\_data['Date'])  
 C:\Users\user\AppData\Local\Temp\ipykernel\_16596\3439131288.py:3: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `dateutil`. To ensure parsing is consistent and as-expected, please specify a format.  
 val\_data['Date'] = pd.to\_datetime(val\_data['Date'])

Code:

from sklearn.preprocessing import StandardScaler  
  
 # Initialize the StandardScaler  
 scaler = StandardScaler()  
  
 # Scale numerical features  
 for column in ['Tot Qty', 'Unit Price', 'Total Price']:  
 train\_data[column] = scaler.fit\_transform(train\_data[[column]])  
 val\_data[column] = scaler.transform(val\_data[[column]])

Output:

Code:

from sklearn.ensemble import RandomForestRegressor  
  
 # Initialize the Random Forest regressor  
 model = RandomForestRegressor(random\_state=42)

Output:

Code:

# Drop 'Tot Qty' from the features set and set it as the target  
 X\_train = train\_data.drop('Tot Qty', axis=1)  
 y\_train = train\_data['Tot Qty']  
  
 # Similarly for validation set  
 X\_val = val\_data.drop('Tot Qty', axis=1)  
 y\_val = val\_data['Tot Qty']

Output:

Code:

# Check for missing values in the training data  
 missing\_values = X\_train.isnull().sum()  
 print(missing\_values[missing\_values > 0]) # This will print columns with missing values and their count

Output:

Discount 16555  
 dtype: int64

Code:

# For numerical columns, fill missing values with the median  
 for column in X\_train.select\_dtypes(include=['number']).columns:  
 median\_value = X\_train[column].median()  
 X\_train[column].fillna(median\_value, inplace=True)  
 X\_val[column].fillna(median\_value, inplace=True)  
  
 # For categorical columns, fill missing values with the mode  
 for column in X\_train.select\_dtypes(include=['object', 'category']).columns:  
 mode\_value = X\_train[column].mode()[0]  
 X\_train[column].fillna(mode\_value, inplace=True)  
 X\_val[column].fillna(mode\_value, inplace=True)

Code:

X\_train, X\_val = X\_train.align(X\_val, join='inner', axis=1) # This aligns both dataframes by columns

Code:

# Check for missing values in the target variable  
 print(y\_train.isnull().sum())

Code:

# Drop rows where the target variable is NaN  
 original\_size = len(X\_train)  
 X\_train = X\_train[y\_train.notna()]  
 y\_train = y\_train[y\_train.notna()]  
 new\_size = len(X\_train)  
 print(f"Dropped {original\_size - new\_size} rows with NaN in the target variable.")

Dropped 2 rows with NaN in the target variable.

Code:

# Initialize the RandomForestRegressor  
 model = RandomForestRegressor(random\_state=42)  
 # Retrain the model  
 model.fit(X\_train, y\_train)  
  
 # Predicting on training data  
 y\_train\_pred = model.predict(X\_train)  
  
 # Predicting on validation data - Ensure y\_val has no NaNs as well  
 y\_val\_pred = model.predict(X\_val[y\_val.notna()])  
 y\_val = y\_val[y\_val.notna()]  
  
 # Calculating RMSE for both datasets  
 rmse\_train = np.sqrt(mean\_squared\_error(y\_train, y\_train\_pred))  
 rmse\_val = np.sqrt(mean\_squared\_error(y\_val, y\_val\_pred))  
  
 print(f"Training RMSE: {rmse\_train}")  
 print(f"Validation RMSE: {rmse\_val}")

Output:

Training RMSE: 4.242415049431006

Validation RMSE: 13.094606117901842