**Project No: 7**

**Project Title: Storage Location Allocation**

**Prepared by: Rose Mary Jose**

**Date: 03-08-23**

Prepared for “Thoughtware Training Private Limited”, under Guidance of its CEO Mr. Pattabhi Raman. The project will be subject to further research, modification and exclusive use of “Thoughtware Training Private Limited”

**Objective**

The objective of this study is to enhance storage location strategies using big data technology. Through data collection, cluster analysis, and association analysis an improved storage location strategy is given. The study aims to categorize goods using clustering algorithms based on sales. Later, an association algorithm is employed to understand consumer habits and relationships among goods. The main objective is to propose and validate an improved class-based storage strategy.

This research is essential to leverage big data for optimizing storage strategies. Through data-driven clustering and association analyses, insights into goods categorization and consumer behaviour are gained. The benefits include enhanced picking efficiency, reduced operational costs, and improved customer satisfaction. This approach finds applications in logistics, supply chain management, and e-commerce operations.

**Dataset**

Source: <https://www.kaggle.com/datasets/carrie1/ecommerce-data>

The dataset considered for this analysis is an E-commerce dataset which include the following columns.

InvoiceNo: the unique number given to a customer while he purchased.

stockCode: the code of the item

Description: the product name

Quantity: the quantity of the product purchased

InvoiceDate: the date when the products are purchased

UnitPrice: the unit price of the product

CustomerID: the ID of the customer

Country: countries whose data is given.

import numpy as np  
import pandas as pd  
import io  
import warnings  
import matplotlib.pyplot as plt  
warnings.filterwarnings("ignore")  
from mlxtend.frequent\_patterns import apriori, association\_rules

data = pd.read\_csv('E\_commerce\_Portugal.csv')  
data

InvoiceNo StockCode Description Quantity   
0 536990 21992 VINTAGE PAISLEY STATIONERY SET 6   
1 536990 22383 LUNCH BAG SUKI DESIGN 10   
2 536990 20728 LUNCH BAG CARS BLUE 14   
3 536990 20658 RED RETROSPOT LUGGAGE TAG 12   
4 536990 20669 RED HEART LUGGAGE TAG 12

InvoiceDate UnitPrice CustomerID Country   
0 12-03-2010 15:14 2.95 12793.0 Portugal   
1 12-03-2010 15:14 1.65 12793.0 Portugal   
2 12-03-2010 15:14 1.65 12793.0 Portugal   
3 12-03-2010 15:14 1.25 12793.0 Portugal   
4 12-03-2010 15:14 1.25 12793.0 Portugal   
[1519 rows x 8 columns]

For the simplicity of the problem the country ‘Portugal’ is only considered the rest was filtered out. Here we need to classify into three categories based on their sales, later the products are classified based on their support and confidence, using apriori algorithm and association rules.

**EDA**

The shape of the considered dataset is **1519 rows and 8 columns.**

The dataset's info returns details about the data frame. It contains information on the total number of columns, column labels, datatypes, the number of columns that aren't null, memory utilization, and range index. It is clear from the information that the dataset under consideration has 8 features of float64 dtype, 2, 1 of int64 dtype, and 5 of object dtype. The presence of null values can be detected by non-null values in 1column.

<class 'pandas.core.frame.DataFrame'>  
Int64Index: 1501 entries, 0 to 1518  
Data columns (total 8 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 InvoiceNo 1501 non-null object   
 1 StockCode 1501 non-null object   
 2 Description 1501 non-null object   
 3 Quantity 1501 non-null int64   
 4 InvoiceDate 1501 non-null object   
 5 UnitPrice 1501 non-null float64  
 6 CustomerID 1462 non-null float64  
 7 Country 1501 non-null object   
dtypes: float64(2), int64(1), object(5)  
memory usage: 105.5+ KB

To obtain the dataset's summary statistics, describe function is used. The Describe function of the numerical column provides an overview of the central tendency, dispersion, and distributional form of the dataset. The 25th, 50th, and 75th percentile values are returned, together with the count, mean, standard deviation, minimum value, and maximum value. From the summary statistics of the categorical columns count, number of unique values, the value with highest frequency and its frequency are all given.

InvoiceNo StockCode Description InvoiceDate Country  
count 1519 1519 1519 1519 1519  
unique 71 706 714 71 1  
top 569866 POST POSTAGE 10-06-2011 14:50 Portugal  
freq 182 30 30 182 1519

**Data pre-processing**

**Detecting and handling missing values**: To ensure accuracy, reliability, and robustness of analyses and models, data preparation is essential for detecting and handling missing values and outliers. Due to the fact that missing values result from errors or inadequate recording and outliers represent anomalies or exceptional circumstances, these problems may result in biased conclusions and incorrect predictions. For the sake of maintaining data integrity and arriving with suitable conclusion, these issues must be regularly addressed.

For detecting the missing data is to use Python functions like isnull() and sum(). The isnull().sum() function helps to quickly figure out the amount of data missing from each column. In this step, one can observe the CustomerID column have missing values and it can be handled by dropping the null values using dropna() function in pandas.

data.isnull().sum()

InvoiceNo 0  
StockCode 0  
Description 0  
Quantity 0  
InvoiceDate 0  
UnitPrice 0  
CustomerID 39  
Country 0  
dtype: int64

data = data.drop(['CustomerID'],axis = 1)

data = data.dropna(axis = 0,how = 'any')

**Detecting and handling duplicated and negative values:** Duplicated values are identical or repeated entries in a dataset, resulting from errors, system glitches, or unintentional repetitions. Identifying and handling duplicates is crucial in data preprocessing to maintain integrity and ensure accurate analyses. Functions like duplicated() in Python can detect duplicates and mark subsequent occurrences as duplicates. The value count of the duplicated values is given.

data.duplicated().sum()

9

data.drop\_duplicates(inplace=True)

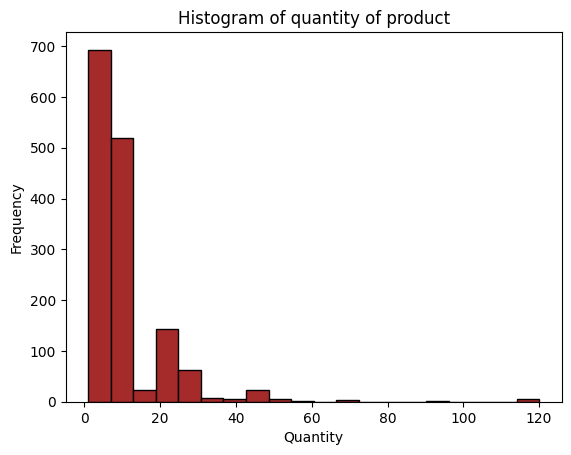
data.duplicated().sum()

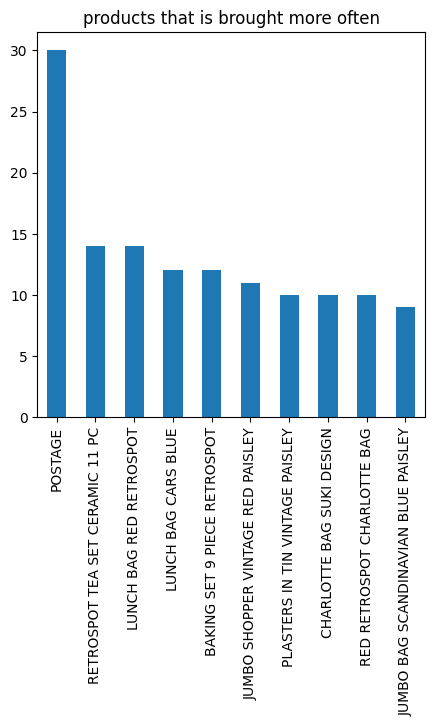
0

From this it can be concluded that there are about 9 dupliacted values and the Quantity column contains negative value. Duplicated values are handled by dropping the rows which has duplicate values and negative values are treated by taking the rows which has Quantity>0.

**Data Visualization**

From the histogram of the quantity column the distribution of data can be visualized. From the histogram it is clear that the column quantity follows a positive skewness. It is also clear that more orders are of quantity between 1 and 20. The highest quantity that has been ordered is for 120.

****

****

From the bar chart of the description column from the dataset, the 10 most often brought products can be visually represented. From the bar chart it is evident that the product with Descrpition ‘postage’ is brought more frequently, and its frequency is about 30. The rest 9 products that brought more often are having a frequnecy between 10 to 15.

**Data Transformation and Aggregation**

First, the 'data' data frame is grouped by the 'Description' column and applies aggregation functions to calculate the mean unit price and sum of quantity for each group of descriptions. The 'agg' function is used to perform these aggregations. The resulting grouped data frame is reset, creating a new data frame with the grouped data. Then merges this grouped data frame with a subset of the original data frame, containing the 'Description' and 'InvoiceDate' columns, associating each description's aggregated data with its respective invoice dates. The head of the grouped\_df is given below.

grouped\_df = data.groupby('Description').agg({  
 'UnitPrice': 'mean',  
 'Quantity': 'sum'  
}).reset\_index()  
grouped\_df = grouped\_df.merge(data[['Description', 'InvoiceDate']], on='Description')

grouped\_df

Description UnitPrice Quantity \  
0 10 COLOUR SPACEBOY PEN 0.85 24   
1 12 PENCIL SMALL TUBE WOODLAND 0.65 5   
2 12 PENCIL SMALL TUBE WOODLAND 0.65 5   
3 12 PENCILS SMALL TUBE RED RETROSPOT 0.65 7   
4 12 PENCILS SMALL TUBE RED RETROSPOT 0.65 7

InvoiceDate   
0 01-09-2011 15:56   
1 05-12-2011 19:01   
2 10-06-2011 14:50   
3 05-12-2011 19:01   
4 10-06-2011 14:50

Here, high-impact products are identified and their collective influence on revenue distribution by calculating the 'TotalValue' for each product. 'TotalValue for each product was calculated by multiplying its 'Quantity' with its 'UnitPrice'. The data frame is sorted in descending order based on the 'TotalValue', identifying products with the highest financial impact. The code calculates the 'CumulativeValue' by computing the cumulative sum of the 'TotalValue' column, providing a comprehensive perspective on the gradual accumulation of revenue. The 'PercentageContribution' is calculated to determine the relative importance of each product's contribution to the total revenue, creating a metric that indicates how much each product influences the overall revenue composition. These lines of code add an additional layer of analysis to the data, providing insights into the proportional significance of each product within the larger financial context.

The code defines a function called 'categorize' that categorizes products based on their 'PercentageContribution' values. It takes a numerical value as input and assigns a category label 'A', 'B', or 'C' to it. The categorization logic is as follows:

* If the input value is greater than or equal to 99, the function assigns the category 'A' to the product, indicating a high contribution level.
* If the input value is greater than or equal to 90 and less than 99, the function assigns the category 'B' to the product, representing a moderate contribution level.
* For all other cases, when the input value is below 90, the function assigns the category 'C' to the product, implying a lower contribution level.

The resulting category labels are stored in a new 'Category' column within the data frame. The head of the data frame after doing ABC analysis based on sales is given below.

grouped\_df['TotalValue'] = grouped\_df['Quantity'] \* grouped\_df['UnitPrice']  
grouped\_df = grouped\_df.sort\_values(by='TotalValue', ascending=False)  
grouped\_df['CumulativeValue'] = grouped\_df['TotalValue'].cumsum()  
grouped\_df['PercentageContribution'] = (grouped\_df['CumulativeValue'] / grouped\_df['TotalValue'].sum()) \* 100

def categorize(value):  
 if value >= 99:  
 return 'A'  
 elif value >= 90 <=99:  
 return 'B'  
 else:  
 return 'C'  
grouped\_df['Category'] = grouped\_df['PercentageContribution'].apply(categorize)  
grouped\_df

Description UnitPrice Quantity InvoiceDate   
717 Manual 603.241429 8 10/17/2011 11:08   
718 Manual 603.241429 8 10/17/2011 11:11   
715 Manual 603.241429 8 5/23/2011 14:58   
714 Manual 603.241429 8 5/23/2011 14:46

TotalValue CumulativeValue PercentageContribution Category   
717 4825.931429 4825.931429 2.175452 C   
718 4825.931429 9651.862857 4.350904 C   
715 4825.931429 14477.794286 6.526356 C   
714 4825.931429 19303.725714 8.701808 C   
[1492 rows x 8 columns]

The value count of the column category gives the number of products in each category. It is evident from this value count that about 20% of products is categorized as A, while 39 % as B and 42 % as C.

grouped\_df['Category'].value\_counts()

C 614  
B 588  
A 290  
Name: Category, dtype: int64

Data is transformed for analysis by creating a new data frame named 'df' and converting the 'InvoiceDate' column into a datetime format. It then identifies unique products in the 'Description' column and generates corresponding product codes, aiming to assign distinct codes for improved tracking and analysis. A mapping dictionary ('product\_code\_mapping') is established to connect product descriptions to their respective codes, resulting in a new 'ProductCode' column appended to the 'df' data frame. This effectively structures and enriches the data by introducing product codes and mapping descriptions to their corresponding codes, enhancing the dataset's organization and setting the stage for more efficient analysis. By taking the number of unique values of the column ProductCode using the nunique function from pandas, it is clear that there are 713 unique products in the considered dataset.

df = grouped\_df[['InvoiceDate','Description','Category','Quantity']]  
df['InvoiceDate'] = pd.to\_datetime(df['InvoiceDate'])

unique\_products = df['Description'].unique()  
product\_codes = ['G' + str(i + 1) for i in range(len(unique\_products))]  
product\_code\_mapping = dict(zip(unique\_products, product\_codes))  
df['ProductCode'] = df['Description'].map(product\_code\_mapping)  
df.ProductCode.nunique()

713

The 'basket\_df' data frame is grouped by 'InvoiceDate' and 'ProductCode', and the sum of 'Quantity' is calculated for each group. The 'unstack' function pivots the data to a tabular format, representing the quantities of each product bought in various transactions. The 'unstacked' DataFrame is then transformed back into a time-series format, with 'InvoiceDate' as the index, and any missing values are filled with zeros. This 'basket\_df' forms the basis for analyzing patterns of product purchases over time. A function called 'hot\_encode' is defined to process individual cell values in 'basket\_df', encoding them as '0' indicating no purchase and '1' indicating a purchase of the corresponding product in that transaction. The resulting data frame, 'basket', contains binary values, which are crucial for analysis, such as association rule mining, which requires binary data to identify item co-occurrences.

basket\_df = (df.groupby(['InvoiceDate','ProductCode'])  
 ['Quantity'].sum()  
 .unstack().reset\_index().fillna(0)  
 .set\_index('InvoiceDate'))

def hot\_encode(x):  
 if(x<= 0):  
 return 0  
 if(x>= 1):  
 return 1  
# Encoding the datasets  
basket = basket\_df.applymap(hot\_encode)  
basket\_df = basket

basket\_df

ProductCode G1 G10 G100 G101 G102 G103 G104 G105 G106 G107……   
\

InvoiceDate   
2010-12-03 15:14:00 0 0 0 0 0 0 0 0 0 0 …  
2010-12-06 10:27:00 0 0 0 0 0 0 0 0 0 0 …  
2010-12-08 13:53:00 0 0 1 0 0 0 0 1 0 0 …  
2010-12-09 10:55:00 0 0 1 0 0 0 0 0 0 0 …

[58 rows x 713 columns]

**Implementation of Apriori Algorithm and Association Rules**

The Apriori algorithm is a widely used data mining technique for discovering frequent itemsets and association rules in large transactional datasets. It is particularly employed in market basket analysis, where the goal is to find relationships between items frequently co-purchased by customers. The algorithm works on the principle of the "apriori property," which states that if an itemset is frequent, then all of its subsets must also be frequent. This property is leveraged to reduce the search space and computational complexity.

The Apriori algorithm is a widely used method in data mining for discovering frequent item sets and association rules in large datasets. The algorithm generates meaningful output attributes that offer insights into patterns and relationships within the data. The key output attributes of the Apriori algorithm:

>Frequent Item Sets: The primary goal of the Apriori algorithm is to identify frequent item sets, which are combinations of items that appear together frequently in the dataset. These item sets are crucial for uncovering common patterns and relationships among items.

>Support: Support measures the frequency of occurrence of a specific item set in the dataset. It helps in determining how often the item set appears, indicating its significance. Higher support values indicate stronger associations.

The Apriori algorithm's importance lies in its efficiency in handling large datasets and its ability to uncover meaningful associations between items. It has applications in various domains, including market basket analysis, recommendation systems, and customer behavior analysis. The algorithm's findings can guide businesses in making informed decisions about product placements, cross-selling strategies, and marketing campaigns to enhance customer satisfaction and business profitability.

This algorithm creates a frequent itemset mining model. The model is applied to the encoded dataset of transactions, basket\_df, with the min\_support parameter set to 0.06. The resulting data frame frq\_items are then expanded to include a column called 'num\_items', which calculates the number of items in each itemset using a lambda function. The code then displays the frequent itemsets, their support values, and the number of items in each itemset, providing insights into the most frequently co-occurring items and their occurrences in the dataset.

# Building the model  
frq\_items = apriori(basket\_df, min\_support = 0.06, use\_colnames = True)  
frq\_items['num\_items'] = frq\_items['itemsets'].apply(lambda x: len(x))  
# Display the frequent itemsets with the number of items  
frq\_items[['support','itemsets', 'num\_items']]

support itemsets num\_items  
0 0.120690 (G1) 1  
1 0.155172 (G10) 1  
2 0.068966 (G100) 1  
3 0.086207 (G101) 1  
4 0.120690 (G105) 1  
... ... ... ...  
1128 0.068966 (G5, G8, G10, G9, G44, G4, G20) 7  
1129 0.068966 (G11, G40, G64, G8, G9, G4, G20) 7  
1130 0.068966 (G11, G5, G8, G9, G44, G4, G20) 7  
1131 0.068966 (G11, G40, G64, G8, G10, G9, G4, G20) 8  
1132 0.068966 (G11, G5, G8, G10, G9, G44, G4, G20) 8  
  
[1133 rows x 3 columns]

The code counts the occurrences of different itemset lengths in the "num\_items" column of the frequent item sets data frame obtained from the Apriori algorithm.

Output attributes of association rules in data mining represent the results of analyzing data sets using association rule mining techniques. Association rules aim to discover interesting relationships or patterns within large datasets by identifying items or events that frequently co-occur together. The output attributes provide valuable insights into these relationships, enabling businesses to make informed decisions and strategies. Here's an explanation of some common output attributes:

>Antecedent and Consequent Items: An association rule consists of an antecedent (left-hand side) and a consequent (right-hand side) item set. These represent the items or events that are related to each other. For example, if {A, B} -> {C} is an association rule, then A and B are the antecedent items, while C is the consequent item.

>Support: Support is the frequency or proportion of transactions that contain both the antecedent and consequent items. It indicates how often the rule is applicable in the dataset. High support suggests a strong relationship between the items.

>Confidence: Confidence measures the strength of the relationship between the antecedent and consequent items. It is the proportion of transactions containing the antecedent that also contain the consequent. High confidence indicates a high likelihood that the consequent will occur given the antecedent.

>Lift: Lift quantifies the degree of dependency between the antecedent and consequent items compared to their independent occurrence. A lift greater than 1 indicates a positive correlation, suggesting that the presence of the antecedent increases the likelihood of the consequent.

>Leverage: Leverage calculates the difference between the observed frequency of both the antecedent and consequent items co-occurring and the frequency expected if they were independent. A positive leverage suggests a positive correlation.

>Conviction: Conviction assesses the impact of the antecedent on the consequent, considering how unlikely the consequent would be without the antecedent. A high conviction value indicates a strong relationship between the items.

These output attributes collectively provide insights into the relationships, significance, and impact of item sets on each other. Businesses can use these attributes to make informed decisions about product placement, cross-selling, recommendation systems, and other strategic initiatives based on patterns discovered within their data.

The rules are filtered based on a minimum confidence threshold of 0.5 and then sorted in descending order of confidence and lift. Confidence quantifies the strength of a rule, while lift measures how much more likely the consequent item occurs when the antecedent item is present. This process enables the identification of significant patterns and relationships within the dataset, aiding decision-making and insights for business applications.

# Collecting the inferred rules in a dataframe  
rules = association\_rules(frq\_items, metric ="confidence", min\_threshold = 0.5)  
rules = rules.sort\_values(['confidence', 'lift'], ascending =[False, False])  
rules

antecedents consequents antecedent support consequent support \  
98 (G155) (G47) 0.068966 0.068966   
99 (G47) (G155) 0.068966 0.068966   
232 (G40) (G64) 0.068966 0.068966   
233 (G64) (G40) 0.068966 0.068966   
509 (G10, G40) (G64) 0.068966 0.068966

support confidence lift leverage conviction zhangs\_metric   
98 0.068966 1.0 14.500000 0.064209 inf 1.000000   
99 0.068966 1.0 14.500000 0.064209 inf 1.000000   
232 0.068966 1.0 14.500000 0.064209 inf 1.000000   
233 0.068966 1.0 14.500000 0.064209 inf 1.000000   
509 0.068966 1.0 14.500000 0.064209 inf 1.000000

[17654 rows x 10 columns]

The association rules generated using the Apriori algorithm and frequent item sets is analysed here. It merges frequent item sets with association rules based on shared antecedents, creating a new data frame with columns like antecedents, consequents, support, confidence, and support for both antecedents and consequents. Additional calculations are performed to enhance insights, such as determining the frequency of occurrence for both antecedents and consequents, calculating the frequency of Y occurrence as a percentage, and expressing the higher degree of confidence as a percentage. The final data frame is sorted by support in descending order to prioritize the most significant rules.

merged\_df = frq\_items.merge(rules, left\_on='itemsets', right\_on='antecedents')

final\_df = merged\_df[['num\_items','antecedents','antecedent support',  
 'consequents', 'consequent support',  
 'support\_y', 'confidence', 'lift', 'leverage', 'conviction']]  
final\_df.rename(columns={'support\_y': 'rule\_support'},  
 inplace=True)

result\_df = final\_df[final\_df['num\_items']==1]  
total\_transactions = basket\_df.shape[0]

# Calculate the support as whole numbers  
result\_df['support'] = (result\_df['rule\_support'] \* total\_transactions).round(0).astype(int)  
  
# Create the final DataFrame with the required columns  
final\_dff = result\_df[['antecedents', 'consequents', 'support', 'confidence',  
 'antecedent support', 'consequent support']]  
  
# Calculate the frequency of antecedents and consequents occurrence  
antecedent\_count = result\_df['antecedents'].value\_counts()  
consequent\_count = result\_df['consequents'].value\_counts()

# Add the frequency of X and Y occurrence columns to the final DataFrame  
final\_dff['Frequnecy of X occurrence'] = final\_dff['antecedents'].map(antecedent\_count)  
final\_dff['Frequnecy of Y occurrence'] = final\_dff['consequents'].map(consequent\_count)  
  
# Calculate the frequency of Y occurrence as a percentage  
final\_dff['frequencty of Y occurance(%)'] = (final\_dff['Frequnecy of Y occurrence'] / total\_transactions) \* 100  
  
# Calculate the higher degree of confidence as a percentage  
final\_dff['higher degree of confidenec(%)'] = final\_dff['confidence'] \* 100  
  
# Rename the columns for better readability  
final\_dff.rename(columns={'antecedent support': 'support P(X)',  
 'consequent support': 'support P(Y)',  
 'confidence': 'confidence P(X|Y)%'},  
 inplace=True)  
  
# Sort the final DataFrame by support in descending order  
final\_dff = final\_dff.sort\_values(by='support', ascending=False)  
  
# Display the final DataFrame  
final\_dff

antecedents consequents support confidence P(X|Y)% \  
0 (G10) (G4) 9 1.000000   
1548 (G6) (G3) 9 0.750000   
1094 (G3) (G6) 9 0.642857   
1208 (G4) (G10) 9 0.818182   
1211 (G4) (G8) 8 0.727273   
 support P(X) support P(Y) Frequnecy of X occurrence \  
0 0.155172 0.189655 73   
1548 0.206897 0.241379 2   
1094 0.241379 0.206897 2   
1208 0.189655 0.155172 25   
1211 0.189655 0.155172 25   
 Frequnecy of Y occurrence frequencty of Y occurance(%) \  
0 17 29.310345   
1548 18 31.034483   
1094 5 8.620690   
1208 16 27.586207   
1211 16 27.586207

higher degree of confidenec(%)   
0 100.000000   
1548 75.000000   
1094 64.285714   
1208 81.818182   
1211 72.727273

From the summary statistics of the final data frame, it can be concluded that the value range of support ranges from 4 to 9 and that of confidence percentage ranges between 0.50 to 1.

support confidence P(X|Y)% support P(X) support P(Y) \  
count 1810.000000 1810.000000 1810.000000 1810.000000   
mean 4.364641 0.648550 0.123833 0.101448   
std 0.803302 0.186892 0.033358 0.053105   
min 4.000000 0.500000 0.068966 0.068966   
25% 4.000000 0.500000 0.086207 0.068966   
50% 4.000000 0.555556 0.137931 0.086207   
75% 4.000000 0.800000 0.137931 0.120690   
max 9.000000 1.000000 0.241379 0.517241   
  
 Frequnecy of X occurrence Frequnecy of Y occurrence \  
count 1810.000000 1810.000000   
mean 162.466298 5.469613   
std 118.411572 4.737138   
min 1.000000 1.000000   
25% 71.000000 2.000000   
50% 127.000000 4.000000   
75% 311.000000 8.000000   
max 319.000000 20.000000   
  
 frequencty of Y occurance(%) higher degree of confidenec(%)   
count 1810.000000 1810.000000   
mean 9.430368 64.854956   
std 8.167480 18.689181   
min 1.724138 50.000000   
25% 3.448276 50.000000   
50% 6.896552 55.555556   
75% 13.793103 80.000000   
max 34.482759 100.000000

The range of values of support and confidence from the summary statistics can be used for further categorizing the data frame into three categories – category\_a, category\_b, and category\_c, based on predefined percentage ranges. The 'category\_ranges' dictionary specifies the boundaries for each category, and then calculates the number of rows within each category based on the total number of rules and the defined percentage ranges. The association rules data frame is sorted by support in descending order, and the sorted data frame is divided into separate data frames for each category. This process ensures that rules are distributed among categories a, b, and c according to their percentage ranges, aiding in decision-making and strategy formulation for different scenarios.

The number of rows in the data frame after association rule was 1810, which is categorized into three and each category consist of rows according to the percentage range mentioned. Thus, the category\_a\_df consist of 271 rows i.e., it accounts for 15% of the total data, category\_b\_df consist of 633 rows i.e., it accounts for 35% of the total data and category\_c\_df consist of 905 rows i.e., it accounts for 50% of the total data.

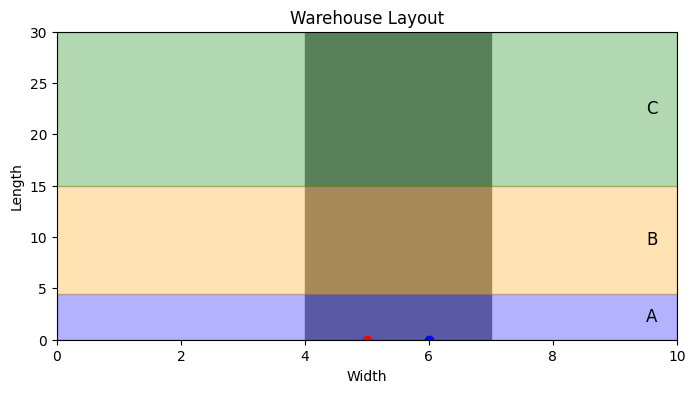
# Define the percentage ranges for each category  
category\_ranges = {  
 'A': (0, 0.15),  
 'B': (0.15, 0.15 + 0.35),  
 'C': (0.15 + 0.35, 1)  
}  
# Calculate the number of rows for each category based on the percentage ranges  
total\_rows = len(final\_dff)  
category\_counts = {  
 category: int(total\_rows \* (upper - lower))  
 for category, (lower, upper) in category\_ranges.items()  
}  
# Sort the dataframe by support in descending order  
final\_dff = final\_dff.sort\_values(by='support', ascending=False)  
# Divide the dataframe into categories  
categories = ['A', 'B', 'C']  
category\_dfs = {}  
start\_idx = 0  
# Now the three dataframes have an additional 'category' column  
for category in categories:  
 end\_idx = start\_idx + category\_counts[category]  
 category\_dfs[category] = final\_dff.iloc[start\_idx:end\_idx]  
 start\_idx = end\_idx  
# Now you have separate dataframes for categories A, B, and C  
category\_a\_df = category\_dfs['A']  
category\_b\_df = category\_dfs['B']  
category\_c\_df = category\_dfs['C']

The head of the category\_a\_df is given below. The final three categorized data frames only contain 3 columns the antecedents, consequents, and support.

antecedents consequents support  
0 (G10) (G4) 9  
1094 (G3) (G6) 9  
1208 (G4) (G10) 9  
1548 (G6) (G3) 9  
1792 (G9) (G5) 8

**Warehouse Layout using Python**

For visually representing the model of the warehouse layout and how products can be allocated in a warehouse here employs Matplotlib, a popular data visualization library in Python. The visualization aims to illustrate various elements crucial to warehouse organization and logistics. The warehouse layout is sketched within a predefined area using rectangles representing aisles and pathways, along with designated input and output points. Different categories (A, B, and C) of items are visualized as distinct colored and labeled sections, each occupying vertical portions within the warehouse. The heights of these sections are proportional to the number of items within each category, ensuring a visual representation of item distribution. The central pathway is designated for the movement of goods, facilitating efficient access to various categories. This visual representation provides valuable insights into the spatial arrangement of the warehouse, aiding decision-makers in optimizing storage, aisle placement, and inventory flow for enhanced operational efficiency in supply chain management.



From the visual representation of the warehouse layout, it is clear that the warehouse is categorized into A, B, and C based on the association rule. Here, A category consist of 15 % of the total products, while B consists of 35% and C of 50%. From the above plot it can be visually observed the number of products in each category is varied and A contains the smaller number of products compared to all. It is also evident that the input and output point of the warehouse is near A category since that has the products which are brought frequently together. The blue region represents the products from A category. Yellow represents products from B and green represent products from C category. Also, the gray region is the central pathway through which vehicles can pass through all the categories of the warehouse.

This is a model representing the warehouse layout. This can be modified with respect to the conditions and parameters we have while constructing a real one. In the above model the products from category\_a\_df can be placed in the area of A, while category\_b\_df products can be arranged at area B, similarly, category\_c\_df products at area C.

**Conclusion**

In this study, we undertook a comprehensive exploration of storage location allocation with the integration of the Apriori algorithm and association rules. This innovative approach allowed us to efficiently categorize products into three distinct categories based on turnover, value, sales volume, and customer preferences. By leveraging big data technology, we harnessed the power of cluster analysis and association analysis to gain deeper insights into consumer behavior and relationships among goods. These insights formed the basis for enhancing the class-based storage strategy, an integral aspect of efficient warehouse management.

The practical implications of our approach were vividly demonstrated through the construction of a sample warehouse layout. The visualization showcased the strategic arrangement of aisles, pathways, input-output points, and category-specific storage sections. Each section's height accurately represented the proportion of products within the respective category, providing a tangible depiction of how different product categories are allocated and accessed within the warehouse environment.

This study highlights the potential of integrating advanced data analysis techniques in supply chain operations, improving order picking efficiency and streamlining inventory management. Warehouse layout visualization aids decision-makers in layout design, aisle placement, and product distribution, enhancing resource utilization and supply chain performance.

**Reference**

<https://link.springer.com/article/10.1007/s00500-018-03702-9>

<https://www.kaggle.com/datasets/carrie1/ecommerce-data>