**MACHINE LEARNING ASSIGNMENT-2**

the transactions occurring between 01/12/2010 and 09/12/2011 for a UK-based and registered non-store online retail.

NAME:- CHANCHALA KUMARI.

ENROLL:- 24/10/JC/033

**Output:-**

Dataset shape: (541909, 6)

Description Quantity I nvoiceDate UnitPrice \

0 WHITE HANGING HEART T-LIGHT HOLDER 6 12/1/2010 8:26 2.55

1 WHITE METAL LANTERN 6 12/1/2010 8:26 3.39

2 CREAM CUPID HEARTS COAT HANGER 8 12/1/2010 8:26 2.75

3 KNITTED UNION FLAG HOT WATER BOTTLE 6 12/1/2010 8:26 3.39

4 RED WOOLLY HOTTIE WHITE HEART. 6 12/1/2010 8:26 3.39

CustomerID Country

0 17850.0 United Kingdom

1 17850.0 United Kingdom

2 17850.0 United Kingdom

3 17850.0 United Kingdom

4 17850.0 United Kingdom

Number of transactions: 18444

Number of frequent itemsets: 243

Top Rules:

Antecedent Consequent \

0 {PAPER CHAIN KIT VINTAGE CHRISTMAS} (PAPER CHAIN KIT 50'S CHRISTMAS )

1 {PINK REGENCY TEACUP AND SAUCER} (ROSES REGENCY TEACUP AND SAUCER )

2 {ROSES REGENCY TEACUP AND SAUCER } (PINK REGENCY TEACUP AND SAUCER)

3 {GARDENERS KNEELING PAD CUP OF TEA } (GARDENERS KNEELING PAD KEEP CALM )

4 {GARDENERS KNEELING PAD KEEP CALM } (GARDENERS KNEELING PAD CUP OF TEA )

Support Confidence Lift

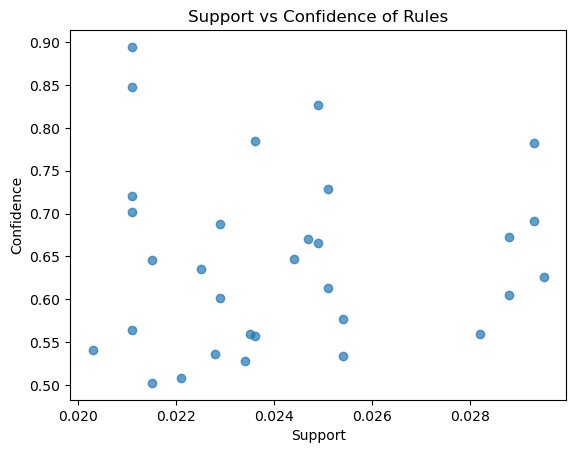
...

1 0.0236 0.7842 18.4716

2 0.0236 0.5568 18.4716

3 0.0251 0.7291 17.8121

4 0.0251 0.6132 17.8121

****

**\*code:-**

import pandas as pd

import matplotlib.pyplot as plt

from itertools import combinations

# Loading the  Dataset, importing as python

from ucimlrepo import fetch\_ucirepo

online\_retail = fetch\_ucirepo(id=352)

df = online\_retail.data.features

print("Dataset shape:", df.shape)

print(df.head())

# Preprocessing

# Remove missing CustomerID, keep positive quantities

df = df.dropna(subset=['CustomerID'])

df = df[df['Quantity'] > 0]

# Create transaction ID (CustomerID + InvoiceDate)

df['TransactionID'] = df['CustomerID'].astype(str) + "\_" + df['InvoiceDate'].astype(str)

# Keep only TransactionID and Description

transactions = df.groupby('TransactionID')['Description'].apply(set)

print("Number of transactions:", len(transactions))

#  Apriori Implementation

def get\_support(itemset, transactions):

    """Support = fraction of transactions containing itemset"""

    count = sum(1 for t in transactions if itemset.issubset(t))

    return count / len(transactions)

def apriori(transactions, min\_support=0.01):

    """Generate frequent itemsets using Apriori"""

    # Step 1: single items

    items = set(i for t in transactions for i in t)

    freq\_itemsets = []

    current\_itemsets = [{i} for i in items]

    k = 1

    while current\_itemsets:

        itemset\_support = []

        for item in current\_itemsets:

            sup = get\_support(item, transactions)

            if sup >= min\_support:

                itemset\_support.append((item, sup))

                freq\_itemsets.append((item, sup))

        # Generate next candidate itemsets

        next\_itemsets = []

        all\_items = [list(x[0]) for x in itemset\_support]

        for a in range(len(all\_items)):

            for b in range(a+1, len(all\_items)):

                union = set(all\_items[a]).union(set(all\_items[b]))

                if len(union) == k+1:

                    next\_itemsets.append(union)

        # Deduplicate

        current\_itemsets = list(map(frozenset, set(map(tuple, next\_itemsets))))

        k += 1

    return freq\_itemsets

# Run Apriori

frequent\_itemsets = apriori(transactions, min\_support=0.02)

print("Number of frequent itemsets:", len(frequent\_itemsets))

# Generate Association Rules

def generate\_rules(frequent\_itemsets, transactions, min\_conf=0.4):

    rules = []

    for itemset, support in frequent\_itemsets:

        if len(itemset) > 1:

            for i in range(1, len(itemset)):

                for antecedent in combinations(itemset, i):

                    antecedent = set(antecedent)

                    consequent = itemset - antecedent

                    sup\_itemset = get\_support(itemset, transactions)

                    sup\_ante = get\_support(antecedent, transactions)

                    sup\_cons = get\_support(consequent, transactions)

                    confidence = sup\_itemset / sup\_ante if sup\_ante > 0 else 0

                    lift = confidence / sup\_cons if sup\_cons > 0 else 0

                    if confidence >= min\_conf:

                        rules.append({

                            'Antecedent': antecedent,

                            'Consequent': consequent,

                            'Support': round(sup\_itemset, 4),

                            'Confidence': round(confidence, 4),

                            'Lift': round(lift, 4)

                        })

    return pd.DataFrame(rules)

rules\_df = generate\_rules(frequent\_itemsets, transactions, min\_conf=0.5)

print("Top Rules:")

print(rules\_df.head())

# Visualization

plt.scatter(rules\_df['Support'], rules\_df['Confidence'], alpha=0.7)

plt.xlabel('Support')

plt.ylabel('Confidence')

plt.title('Support vs Confidence of Rules')

plt.show()