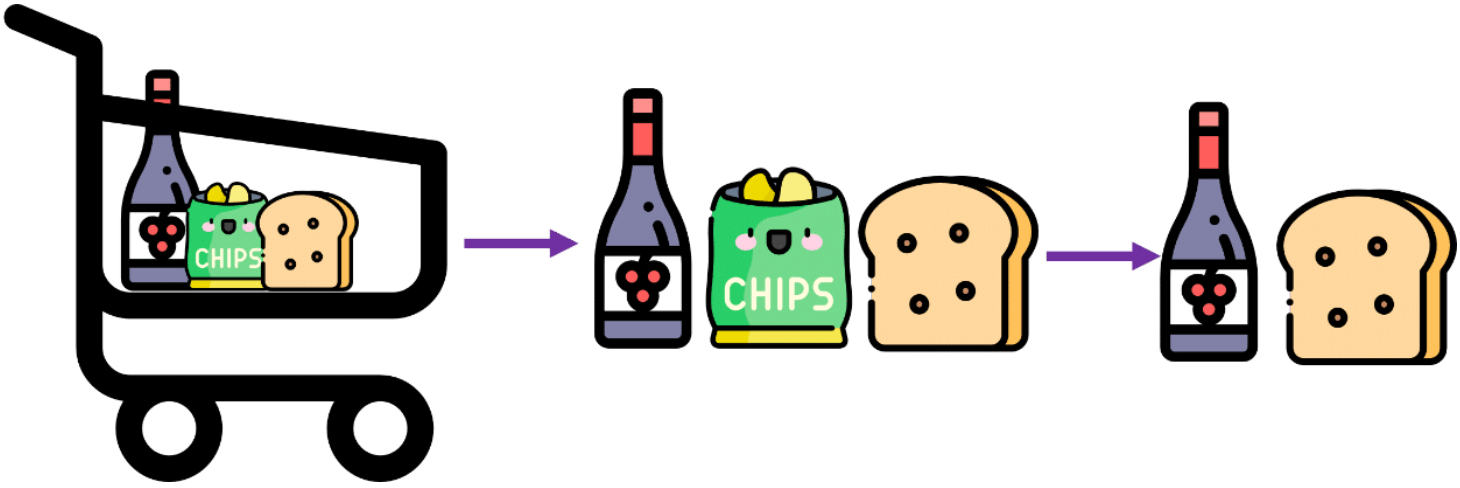
CS 634 - Data Mining

Midterm Project Report

IMPLEMENTATION OF APRIORI ALGORITHM AND BRUTE-FORCE APPROACH



Name - Jaini Bhavsar
UCID - jkb37
Email - jkb37@njit.edu

Contents

Introduction

Part 1 - Apriori Algorithm

With the quick growth in e-commerce applications, there is an accumulation of data in vast quantities, in months not in years. Data mining, also known as Knowledge Discovery in Databases(KDD), to find anomalies, correlations, patterns, and trends to predict outcomes.

Apriori Algorithm is a classical algorithm in data mining. It is used for mining frequent itemsets and relevant association rules. It is devised to operate on a database containing a lot of transactions, for instance, items bought by customers in a store.

It is very important for effective Market Basket Analysis and it helps the customers in purchasing their items with more ease which increases the sales of the market.

It calculates the support and confidence of each item in the transaction.

Support (x) = number of transactions containing x / total number of transactions

confidence (x->y) = number of transactions containing x and y / total number of transactions containing x

In this algorithm, I calculated the support and confidence of each item. Output is FrequentItemSets(whose support is more than min support) and all Qualified items(whose confidence is more than min confidence) and Unqualified items(whose support is less than min confidence). Along with this time interval is calculated.

Requirements

Software Specification

- Language: Python - Version 3.9.7
- Software: PyCharm
- Libraries: Itertools, numpy, pandas, time

Hardware Configuration

- Processor: Apple M1 Pro
- Processor Speed: 3.2 GHz
- RAM: 16 GB
- Hard Disk: 512 GB

Implementation

Overview

Both of the algorithms are implemented using the 'Python' programming language. The implementation covers the following steps:

1. On running the program, it prompts the user to enter minimum support and confidence value.

2. Based on the dataset, it reads the csv file available in the same directory as the py file.

3. The data preprocessing is then applied on the dataset which primarily involves using encoding of transactions.

4. The preprocessed data is then used to generate frequent itemsets.

5. The frequent itemsets are then used to generate association rules.

6. The algorithm is coded and tested using PyCharm as Development Environment.

Flow Chart

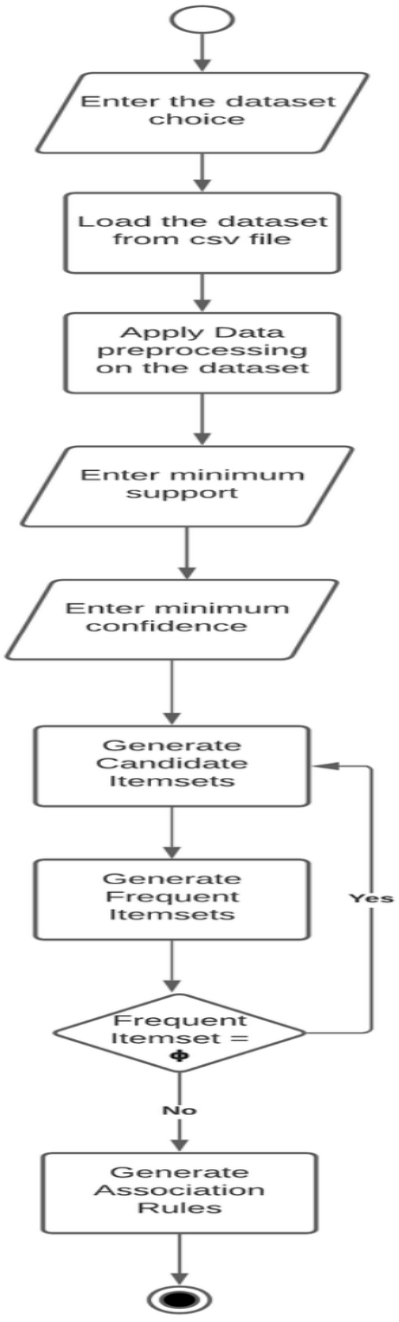


Fig: Algorithm Design

Source Code

```
 import time

start_time = time.time()


print("Choose and enter the number from menu:\n")

print("1.Amazon\n2.Best Buy\n3.K-mart\n4.Nike\n5.Custom Data")

n = int(input().strip())


if n == 1:

    # Amazon Dataset

    print("Amazon Dataset selected!")

    import pandas as pd


    df = pd.read_csv('amazon.csv')

    print(df)

    newTransaction = list([""] * len(df['Transaction']))

    dictTransactions = {}

    # perform run time encoding of transactions of in the for loop

    for i in range(0, len(df['Transaction'])):

        tempStr = ""

        for sstr in df['Transaction'][i].split(","):

            if sstr.strip() == 'A Beginner'sGuide':
```

```
    tempStr = tempStr + 'A'
    dictTransactions['A'] = 'A Beginner'sGuide'

elif sstr.strip() == 'Java: The Complete Reference':
    tempStr = tempStr + 'B'
    dictTransactions['B'] = 'Java: The Complete Reference'

elif sstr.strip() == 'JavaFor Dummies':
    tempStr = tempStr + 'C'
    dictTransactions['C'] = 'JavaFor Dummies'

elif sstr.strip() == 'Android Programming: The Big Nerd Ranch':
    tempStr = tempStr + 'D'
    dictTransactions['D'] = 'Android Programming: The Big Nerd Ranch'

elif sstr.strip() == 'Head First Java 2nd Edition':
    tempStr = tempStr + 'E'
    dictTransactions['E'] = 'Head First Java 2nd Edition'

elif sstr.strip() == 'Beginning Programming with Java':
    tempStr = tempStr + 'F'
    dictTransactions['F'] = 'Beginning Programming with Java'
```

```
            elif sstr.strip() == 'Java 8 Pocket Guide':
                tempStr = tempStr + 'G'
                dictTransactions['G'] = 'Java 8 Pocket Guide'

            elif sstr.strip() == 'C++ Programming in Easy Steps':
                tempStr = tempStr + 'H'
                dictTransactions['H'] = 'C++ Programming in Easy Steps'

            elif sstr.strip() == 'Effective Java (2nd Edition)':
                tempStr = tempStr + 'I'
                dictTransactions['I'] = 'Effective Java (2nd Edition)'

            elif sstr.strip() == 'HTML and CSS: Design and Build Websites':
                tempStr = tempStr + 'J'
                dictTransactions['J'] = 'HTML and CSS: Design and Build Websites'

        newTransaction[i] += tempStr
    df["NewTransaction"] = newTransaction

elif n == 2:
    # Best Buy Dataset
    print("Best Buy Dataset selected!")
    import pandas as pd
```

```
df = pd.read_csv('best_buy.csv')
print(df)
newTransaction = list([""] * len(df['Transaction']))
dictTransactions = {}
# perform run time encoding of transactions of in the for loop
for i in range(0, len(df['Transaction'])):
    tempStr = ""
    for sstr in df['Transaction'][i].split(","):
        if sstr.strip() == 'Digital Camera':
            tempStr = tempStr + 'A'
            dictTransactions['A'] = 'Digital Camera'

        elif sstr.strip() == 'Lab Top':
            tempStr = tempStr + 'B'
            dictTransactions['B'] = 'Lab Top'

        elif sstr.strip() == 'Desk Top':
            tempStr = tempStr + 'C'
            dictTransactions['C'] = 'Desk Top'

        elif sstr.strip() == 'Printer':
            tempStr = tempStr + 'D'
```

```
    dictTransactions['D'] = 'Printer'

elif sstr.strip() == 'Flash Drive':
    tempStr = tempStr + 'E'
    dictTransactions['E'] = 'Flash Drive'

elif sstr.strip() == 'Microsoft Office':
    tempStr = tempStr + 'F'
    dictTransactions['F'] = 'Microsoft Office'

elif sstr.strip() == 'Speakers':
    tempStr = tempStr + 'G'
    dictTransactions['G'] = 'Speakers'

elif sstr.strip() == 'Lab Top Case':
    tempStr = tempStr + 'H'
    dictTransactions['H'] = 'Lab Top Case'

elif sstr.strip() == 'Effective Java (2nd Edition)':
    tempStr = tempStr + 'I'
    dictTransactions['I'] = 'Effective Java (2nd Edition)'

elif sstr.strip() == 'Anti-Virus':
```

```
            tempStr = tempStr + 'J'
            dictTransactions['J'] = 'Anti-Virus'

        elif sstr.strip() == 'External Hard-Drive':
            tempStr = tempStr + 'K'
            dictTransactions['K'] = 'External Hard-Drive'

        newTransaction[i] += tempStr
    df["NewTransaction"] = newTransaction

elif n == 3:
    # KMart Dataset
    print("KMart Dataset selected!")
    import pandas as pd

    df = pd.read_csv('kmart.csv')
    print(df)
    newTransaction = list([""] * len(df['Transaction']))
    dictTransactions = {}
    # perform run time encoding of transactions of in the for loop
    for i in range(0, len(df['Transaction'])):
        tempStr = ""
        for sstr in df['Transaction'][i].split(","):
```

```
if sstr.strip() == 'Quilts':
    tempStr = tempStr + 'A'
    dictTransactions['A'] = 'Quilts'

elif sstr.strip() == 'Bedspreads':
    tempStr = tempStr + 'B'
    dictTransactions['B'] = 'Bedspreads'

elif sstr.strip() == 'Decorative Pillows':
    tempStr = tempStr + 'C'
    dictTransactions['C'] = 'Decorative Pillows'

elif sstr.strip() == 'Bed Skirts':
    tempStr = tempStr + 'D'
    dictTransactions['D'] = 'Bed Skirts'

elif sstr.strip() == 'Sheets':
    tempStr = tempStr + 'E'
    dictTransactions['E'] = 'Sheets'

elif sstr.strip() == 'Shams':
    tempStr = tempStr + 'F'
    dictTransactions['F'] = 'Shams'
```

```
            elif sstr.strip() == 'Bedding Collections':
                tempStr = tempStr + 'G'
                dictTransactions['G'] = 'Bedding Collections'

            elif sstr.strip() == 'Kids Bedding':
                tempStr = tempStr + 'H'
                dictTransactions['H'] = 'Kids Bedding'

            elif sstr.strip() == 'Embroidered Bedspread':
                tempStr = tempStr + 'I'
                dictTransactions['I'] = 'Embroidered Bedspread'

            elif sstr.strip() == 'Towels':
                tempStr = tempStr + 'J'
                dictTransactions['J'] = 'Towels'

        newTransaction[i] += tempStr
    df["NewTransaction"] = newTransaction

elif n == 4:
    # Nike Dataset
    print("Nike Dataset selected!")
```

```
import pandas as pd


df = pd.read_csv('nike.csv')
print(df)
newTransaction = list([""] * len(df['Transaction']))
dictTransactions = {}
# perform run time encoding of transactions of in the for loop
for i in range(0, len(df['Transaction'])):
    tempStr = ""
    for sstr in df['Transaction'][i].split(","):
        if sstr.strip() == 'Running Shoe':
            tempStr = tempStr + 'A'
            dictTransactions['A'] = 'Running Shoe'


        elif sstr.strip() == 'Soccer Shoe':
            tempStr = tempStr + 'B'
            dictTransactions['B'] = 'Soccer Shoe'


        elif sstr.strip() == 'Socks':
            tempStr = tempStr + 'C'
            dictTransactions['C'] = 'Socks'


        elif sstr.strip() == 'Swimming Shirt':
```

```
    tempStr = tempStr + 'D'
    dictTransactions['D'] = 'Swimming Shirt'

elif sstr.strip() == 'Dry Fit V-Nick':
    tempStr = tempStr + 'E'
    dictTransactions['E'] = 'Dry Fit V-Nick'

elif sstr.strip() == 'Rash Guard':
    tempStr = tempStr + 'F'
    dictTransactions['F'] = 'Rash Guard'

elif sstr.strip() == 'Sweatshirts':
    tempStr = tempStr + 'G'
    dictTransactions['G'] = 'Sweatshirts'

elif sstr.strip() == 'Hoodies':
    tempStr = tempStr + 'H'
    dictTransactions['H'] = 'Hoodies'

elif sstr.strip() == 'Tech Pants':
    tempStr = tempStr + 'I'
    dictTransactions['I'] = 'Tech Pants'
```

```
            elif sstr.strip() == 'Modern Pants':
                tempStr = tempStr + 'J'
                dictTransactions['J'] = 'Modern Pants'

        newTransaction[i] += tempStr
    df["NewTransaction"] = newTransaction

elif n == 5:
    # Custom Dataset
    print("Custom Dataset selected!")
    import pandas as pd

    df = pd.read_csv('custom.csv')
    print(df)
    newTransaction = list([""] * len(df['Transaction']))
    dictTransactions = {}
    # perform run time encoding of transactions of in the for loop
    for i in range(0, len(df['Transaction'])):
        tempStr = ""
        for sstr in df['Transaction'][i].split(","):
            if sstr.strip() == 'ink':
                tempStr = tempStr + 'A'
                dictTransactions['A'] = 'ink'
```

```
elif sstr.strip() == 'pen':
    tempStr = tempStr + 'B'
    dictTransactions['B'] = 'pen'

elif sstr.strip() == 'cheese':
    tempStr = tempStr + 'C'
    dictTransactions['C'] = 'cheese'

elif sstr.strip() == 'bag':
    tempStr = tempStr + 'D'
    dictTransactions['D'] = 'bag'

elif sstr.strip() == 'juice':
    tempStr = tempStr + 'E'
    dictTransactions['E'] = 'juice'

elif sstr.strip() == 'milk':
    tempStr = tempStr + 'F'
    dictTransactions['F'] = 'milk'

elif sstr.strip() == 'eggs':
    tempStr = tempStr + 'G'
```

```
            dictTransactions['G'] = 'eggs'


        elif sstr.strip() == 'bread':
            tempStr = tempStr + 'H'
            dictTransactions['H'] = 'bread'


        elif sstr.strip() == 'napkins':
            tempStr = tempStr + 'I'
            dictTransactions['I'] = 'napkins'


        elif sstr.strip() == 'sugar':
            tempStr = tempStr + 'J'
            dictTransactions['J'] = 'sugar'


    newTransaction[i] += tempStr
df["NewTransaction"] = newTransaction


print(dictTransactions)


print("Choose the option for minimum support format:\n1. Percentage\n2. Natural Number")
n = int(input().strip())
if n == 1:
```

```
    minSupPercentage = int(input().strip())
    minSup = (minSupPercentage * len(df['Transaction'])) // 100
elif n == 2:
    minSup = int(input())
print("Minimum Support Count : ", minSup)


print("Choose the option for minimum confidence format:\n1. Percentage\n2. Floating Number")
n = int(input().strip())
if n==1:
    minConfidencePercentage = int(input().strip())
    minConfidence = minConfidencePercentage / 100
elif n==2:
    minConfidence = float(input())
print("Minimum Confidence: ", minConfidence)


myList = list(df['NewTransaction'])
myList


#function for counting the frequency of items in candidate sets by scanning the transactions
def generate_Candidate(c, transactions):
    itemsInC = []
    for key in c.keys():
```

```
        itemsInC.append(key)
    for i in range(0, len(transactions)):
        tid = transactions[i]
        for j in range(0, len(itemsInC)):
            item = itemsInC[j]
            count = 0
            for k in range(0, len(item)):
                atkthpos = item[k] + ""
                if tid.find(atkthpos) >= 0:
                    count = count + 1
            if count == len(item):
                val = c[itemsInC[j]]
                val = val + 1
                c[item] = val
    return c


#function for generating k-item candidate sets by joining the (k-1)frequent itemsets
def join(l):
    c = {}
    itemInL = []
    for key in l.keys():
        itemInL.append(key)
    for i in range(0, len(itemInL)):
```

```
        for j in range(i+1, len(itemInL)):
            forchecki = itemInL[i][0:len(itemInL[i])-1]
            forcheckj = itemInL[j][0:len(itemInL[j])-1]
            if forchecki == forcheckj:
                newPotentialCandidate = itemInL[i] + itemInL[j][len(itemInL[j])-1]
                c[newPotentialCandidate] = 0
    return c


#function for generating frequent itemsets by checking the itemsets frequency with minimum support
def generate_frequent_set(transactions, minSup, c):
    l = {}
    nList = []
    for key in c.keys():
        nList.append(key)
    for i in range(0, len(nList)):
        support = c[nList[i]]
        if support >= minSup:
            l[nList[i]] = support
    return l


# function for pruning the itemsets
def prune(c, l):
```

```
clist = []
for key in c.keys():
    clist.append(key)

llist = []
for key in l.keys():
    llist.append(key)

for i in range(0, len(clist)):
    inclist = clist[i]
    forPruneCheck = inclist[len(inclist) - 2:]
    tobegenerated = inclist[0:len(inclist) - 2]
    for1s = ""
    for j in range(0, len(tobegenerated)):
        for1s = for1s + "1"
    forLeftShift = "1"
    nlist = []
    for j in range(0, len(for1s)):
        res = int(for1s) - int(forLeftShift)
        strres = str(res)
        if len(strres) < len(tobegenerated):
            strres = "0" + strres
        newString = ""
```

```
            for k in range(0, len(strres)):
                if strres[k] == '1':
                    newString = newString + "" + tobegenerated[k]
            newString = newString + forPruneCheck
            nlist.append(newString)
            newvar = int(forLeftShift) << 1
            forLeftShift = "{0:b}".format(newvar)

        for h in range(0, len(nlist)):
            tobechecked = nlist[h]
            flag = 0
            for j in range(0, len(llist)):
                potentialCand = llist[j]
                if tobechecked == potentialCand:
                    flag = 1
                    break
            if flag == 0 and (inclist in c):
                c.pop(inclist)
    return c


transactions = myList
finalDict = {}
```

```
DictForSubsetCounts = {}
print(transactions)
c = {}
for i in range(0, len(transactions)):
    tid = transactions[i]
    for j in range(0, len(tid)):
        ch = tid[j]
        if ch in c.keys():
            val = c[ch]
            val = val + 1
            c[ch] = val
        else:
            c[ch] = 1
print("C1 => ", c)
l = generate_frequent_set(transactions, minSup, c)
finalDict.update(l)
DictForSubsetCounts.update(l)
print("L1 => ", l)
counter = 2
while len(c) > 0:
    c = join(l)
    print("C", counter, " (by joining) L", (counter - 1), " => ", c)
```

```
    c = prune(c, l)
    print("C", counter, " (by pruning) => ", c)

    c = generate_Candidate(c, transactions);
    DictForSubsetCounts.update(c)
    print("C", counter, " (with support count) => ", c);

    l = generate_frequent_set(transactions, minSup, c)
    finalDict.update(l)
    DictForSubsetCounts.update(l)
    print("L", counter, " => ", l);

    counter = counter + 1

tmpDict = DictForSubsetCounts
keysInDictForSubsetCounts = list(DictForSubsetCounts.keys())
from itertools import permutations

for key in keysInDictForSubsetCounts:
    val = tmpDict.get(key)
    perm = [''.join(p) for p in permutations(key)]
    for p in perm:
        DictForSubsetCounts[p] = val
```

```
print("\nDictionary for rule generation =>\n")
print(DictForSubsetCounts)

dictKeys = finalDict.keys()
print(dictKeys)

print("Rules Generated are as follows:\n\n")
for individualString1 in dictKeys:
    for individualString2 in dictKeys:
        flag = 0
        for c in individualString1:
            if c in individualString2:
                flag = 1
                break
        if flag!=1:
            if (individualString1+individualString2) in DictForSubsetCounts.keys():
                confVal =
int(DictForSubsetCounts.get(individualString1+individualString2))/DictForSubsetCounts
.get(individualString1)
                if confVal >= minConfidence:
                    for char in individualString1:
                        print(char+" : "+dictTransactions[char])
                    for char in individualString2:
                        print(char+" : "+dictTransactions[char])
```

```
                print(individualString1+' -> '+individualString2)
                print("Confidence Value : " + str(confVal))
                print()
                print()
    print()
    print()

print("--- The Apriori algorithm now executed takes: %s seconds ---" % (time.time() - start_time))
```

List of DataSets

The datasets used for testing the Apriori and Brute Force Algorithm implementation are as follows:

1. Amazon

TransactionID	Transaction
Trans1	A Beginner's Guide, Java: The Complete Reference, JavaFor Dummies, Android Programming: The Big Nerd Ranch
Trans2	A Beginner's Guide, Java: The Complete Reference, JavaFor Dummies
Trans3	A Beginner's Guide, Java: The Complete Reference, JavaFor Dummies, Android Programming: The Big Nerd Ranch, Head First Java 2nd Edition
Trans4	Android Programming: The Big Nerd Ranch, Head First Java 2nd Edition , Beginning Programming with Java
Trans5	Android Programming: The Big Nerd Ranch, Beginning Programming with Java, Java 8 Pocket Guide
Trans6	A Beginner's Guide, Android Programming: The Big Nerd Ranch, Head First Java 2nd Edition
Trans7	A Beginner's Guide, Head First Java 2nd Edition , Beginning Programming with Java
Trans8	Java: The Complete Reference, JavaFor Dummies, Android Programming: The Big Nerd Ranch
Trans9	JavaFor Dummies, Android Programming: The Big Nerd Ranch, Head First Java 2nd Edition , Beginning Programming with Java
Trans10	Beginning Programming with Java, Java 8 Pocket Guide, C++ Programming in Easy Steps
Trans11	A Beginner's Guide, Java: The Complete Reference,JavaFor Dummies, Android Programming: The Big Nerd Ranch
Trans12	A Beginner's Guide, Java: The Complete Reference, JavaFor Dummies, HTML and CSS: Design and Build Websites
Trans13	A Beginner's Guide, Java: The Complete Reference, JavaFor Dummies, Java 8 Pocket Guide, HTML and CSS: Design and Build Websites
Trans14	JavaFor Dummies, Android Programming: The Big Nerd Ranch, Head First Java 2nd Edition
Trans15	JavaFor Dummies, Android Programming: The Big Nerd Ranch

Trans16	A Beginner's Guide, Java: The Complete Reference, JavaFor Dummies, Android Programming: The Big Nerd Ranch
Trans17	A Beginner's Guide, Java: The Complete Reference, JavaFor Dummies, Android Programming: The Big Nerd Ranch
Trans18	Head First Java 2nd Edition ,Beginning Programming with Java, Java 8 Pocket Guide
Trans19	Android Programming: The Big Nerd Ranch, Head First Java 2nd Edition
Trans20	A Beginner's Guide, Java: The Complete Reference, JavaFor Dummies

2. Best Buy

TransactionID	Transaction
Trans1	Desk Top, Printer, Flash Drive, Microsoft Office, Speakers, Anti-Virus
Trans2	Lap Top, Flash Drive, Microsoft Office, Lap Top Case, Anti-Virus
Trans3	Lap Top, Printer, Flash Drive, Microsoft Office, Anti-Virus, Lap Top Case, External Hard-Drive
Trans4	Lap Top, Printer, Flash Drive, Anti-Virus, External Hard-Drive, Lap Top Case
Trans5	Lap Top, Flash Drive, Lap Top Case, Anti-Virus
Trans6	Lap Top, Printer, Flash Drive, Microsoft Office
Trans7	Desk Top, Printer, Flash Drive, Microsoft Office
Trans8	Lap Top, External Hard-Drive, Anti-Virus
Trans9	Desk Top, Printer, Flash Drive, Microsoft Office, Lap Top Case, Anti-Virus, Speakers, External Hard-Drive
Trans10	Digital Camera , Lap Top, Desk Top, Printer, Flash Drive, Microsoft Office, Lap Top Case, Anti-Virus, External Hard-Drive, Speakers
Trans11	Lap Top, Desk Top, Lap Top Case, External Hard-Drive, Speakers, Anti-Virus
Trans12	Digital Camera , Lap Top, Lap Top Case, External Hard-Drive, Anti-Virus, Speakers
Trans13	Digital Camera , Speakers
Trans14	Digital Camera , Desk Top, Printer, Flash Drive, Microsoft Office
Trans15	Printer, Flash Drive, Microsoft Office, Anti-Virus, Lap Top Case, Speakers, External Hard-Drive
Trans16	Digital Camera, Flash Drive, Microsoft Office, Anti-Virus, Lap Top Case, External Hard-Drive, Speakers

Trans17	Digital Camera , Lap Top, Lap Top Case
Trans18	Digital Camera , Lap Top Case, Speakers
Trans19	Digital Camera , Lap Top, Printer, Flash Drive, Microsoft Office, Speakers, Lap Top Case, Anti-Virus
Trans20	Digital Camera , Lap Top, Speakers, Anti-Virus, Lap Top Case

3. K Mart

TransactionID	Transaction
Trans1	Decorative Pillows, Quilts, Embroidered Bedspread
Trans2	Embroidered Bedspread, Shams, Kids Bedding, Bedding Collections, Bed Skirts, Bedspreads, Sheets
Trans3	Decorative Pillows, Quilts, Embroidered Bedspread, Shams, Kids Bedding, Bedding Collections
Trans4	Kids Bedding, Bedding Collections, Sheets, Bedspreads, Bed Skirts
Trans5	Decorative Pillows, Kids Bedding, Bedding Collections, Sheets, Bed Skirts, Bedspreads
Trans6	Bedding Collections, Bedspreads, Bed Skirts, Sheets, Shams, Kids Bedding
Trans7	Decorative Pillows, Quilts
Trans8	Decorative Pillows, Quilts, Embroidered Bedspread
Trans9	Bedspreads, Bed Skirts, Shams, Kids Bedding, Sheets
Trans10	Quilts, Embroidered Bedspread, Bedding Collections
Trans11	Bedding Collections, Bedspreads, Bed Skirts, Kids Bedding, Shams, Sheets
Trans12	Decorative Pillows, Quilts
Trans13	Embroidered Bedspread, Shams
Trans14	Sheets, Shams, Bed Skirts, Kids Bedding
Trans15	Decorative Pillows, Quilts
Trans16	Decorative Pillows, Kids Bedding, Bed Skirts, Shams
Trans17	Decorative Pillows, Shams, Bed Skirts
Trans18	Quilts, Sheets, Kids Bedding

Trans19	Shams, Bed Skirts, Kids Bedding, Sheets
Trans20	Decorative Pillows, Bedspreads, Shams, Sheets, Bed Skirts, Kids Bedding

4. Nike

TransactionID	Transaction
Trans1	Running Shoe, Socks, Sweatshirts, Modern Pants
Trans2	Running Shoe, Socks, Sweatshirts
Trans3	Running Shoe, Socks, Sweatshirts, Modern Pants
Trans4	Running Shoe, Sweatshirts, Modern Pants
Trans5	Running Shoe, Socks, Sweatshirts, Modern Pants, Soccer Shoe
Trans6	Running Shoe, Socks, Sweatshirts
Trans7	Running Shoe, Socks, Sweatshirts, Modern Pants, Tech Pants, Rash Guard, Hoodies
Trans8	Swimming Shirt, Socks, Sweatshirts
Trans9	Swimming Shirt, Rash Guard, Dry Fit V-Neck, Hoodies, Tech Pants
Trans10	Swimming Shirt, Rash Guard, Dry Fit V-Neck
Trans11	Swimming Shirt, Rash Guard, Dry Fit V-Neck
Trans12	Running Shoe, Swimming Shirt, Socks, Sweatshirts, Modern Pants, Soccer Shoe, Rash Guard, Hoodies, Tech Pants, Dry Fit V-Neck
Trans13	Running Shoe, Swimming Shirt, Socks, Sweatshirts, Modern Pants, Soccer Shoe, Rash Guard, Tech Pants, Dry Fit V-Neck, Hoodies
Trans14	Running Shoe, Swimming Shirt, Rash Guard, Tech Pants, Hoodies, Dry Fit V-Neck
Trans15	Running Shoe, Swimming Shirt, Socks, Sweatshirts, Modern Pants, Dry Fit V-Neck, Rash Guard, Tech Pants
Trans16	Swimming Shirt, Soccer Shoe, Hoodies, Dry Fit V-Neck, Tech Pants, Rash Guard
Trans17	Running Shoe, Socks
Trans18	Socks, Sweatshirts, Modern Pants, Soccer Shoe, Hoodies, Rash Guard, Tech Pants, Dry Fit V-Neck
Trans19	Running Shoe, Swimming Shirt, Rash Guard

Trans20	Running Shoe, Swimming Shirt, Socks, Sweatshirts, Modern Pants, Soccer Shoe, Hoodies, Tech Pants, Rash Guard, Dry Fit V-Neck

5. Custom

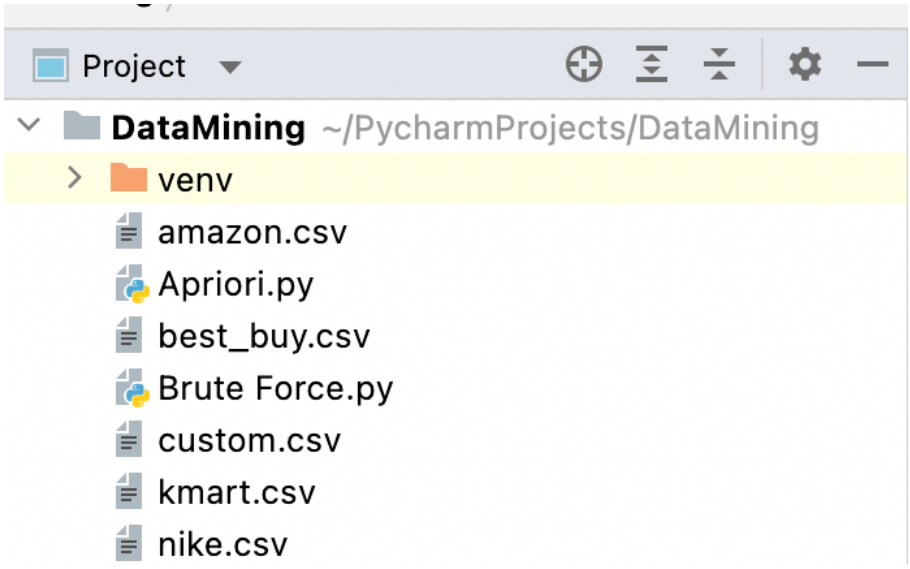
TransactionID	Transaction
Trans1	ink,pen, cheese, bag
Trans2	milk, pen, juice, cheese, eggs
Trans3	milk, juice, eggs
Trans4	juice, milk, cheese, eggs
Trans5	ink, pen, cheese, bag
Trans6	milk, pen, juice, cheese, eggs
Trans7	milk, bread, sugar, napkins
Trans8	milk, eggs, sugar
Trans9	bag, milk, cheese, eggs
Trans10	napkins, milk, sugar, cheese
Trans11	cheese, sugar, eggs, napkins
Trans12	ink, pen, eggs, milk
Trans13	juice, milk, eggs, pen, bag, napkins, cheese
Trans14	bread, milk, eggs, sugar, ink
Trans15	bag, pen
Trans16	sugar, cheese, juice, eggs, milk
Trans17	bread, pen, bag
Trans18	milk, sugar, juice, cheese, napkins
Trans19	bag, milk
Trans20	pen, bread, sugar, napkins

Testing Implementation using Data Sets

How to run the Program ?

To run the program, you need a Python IDE of your choice or Jupyter notebook. The code is developed using PyCharm. It helps break the python code into fragments which makes debugging a lot easier. The following steps test the implementation on PyCharm.

1. Navigate from start/finder to the PyCharm UI and locate the directory where you have the py file for Algorithms.



2. Select the .py file from PyCharm IDE and then press 'Run' from the top right corner to run the program.
3. Once you hit 'Run', you will be prompted to choose the dataset of your choice, minimum support value, and enter the minimum confidence value.
4. Once you are done with inputting minimum confidence and minimum support, wait for some time and you will find all the association rules generated in the terminal window.

Results of Algorithm on Datasets

This section describes association rules generated from datasets as a result, given minimum support and minimum confidence.

1. **Amazon Dataset:**

   Running the Apriori Algorithm on Amazon Dataset for Support 50% and Confidence 60%.

Run: Apriori

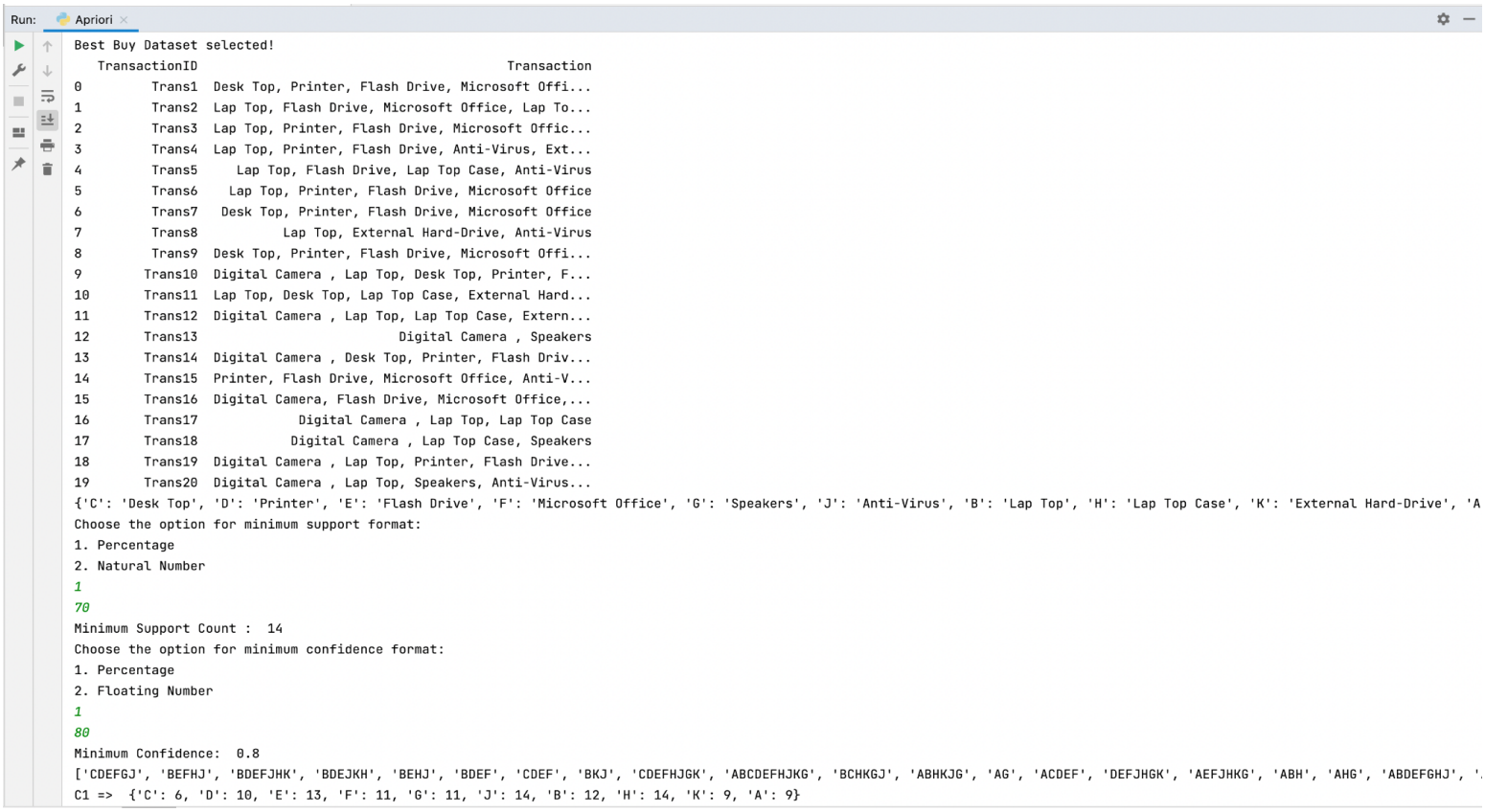
```
Minimum Support Count :  10
Choose the option for minimum confidence format:
1. Percentage
2. Floating Number
1
60
Minimum Confidence:  0.6
['ABCD', 'ABC', 'ABCDE', 'DEF', 'DFG', 'ADE', 'AEF', 'BCD', 'CDEF', 'FGH', 'ABCD', 'ABCJ', 'ABCGJ', 'CDE', 'CD', 'ABCD', 'ABCD', 'EFG', 'DE', 'ABC']
C1 =>  {'A': 11, 'B': 10, 'C': 13, 'D': 13, 'E': 8, 'F': 6, 'G': 4, 'H': 1, 'J': 2}
L1 =>  {'A': 11, 'B': 10, 'C': 13, 'D': 13}
C 2  (by joining) L 1  =>  {'AB': 0, 'AC': 0, 'AD': 0, 'BC': 0, 'BD': 0, 'CD': 0}
C 2  (by pruning) =>  {'AB': 0, 'AC': 0, 'AD': 0, 'BC': 0, 'BD': 0, 'CD': 0}
C 2  (with support count) =>  {'AB': 9, 'AC': 9, 'AD': 6, 'BC': 10, 'BD': 6, 'CD': 9}
L 2  =>  {'BC': 10}
C 3  (by joining) L 2  =>  {}
C 3  (by pruning) =>  {}
C 3  (with support count) =>  {}
L 3  =>  {}

Dictionary for rule generation =>

{'A': 11, 'B': 10, 'C': 13, 'D': 13, 'AB': 9, 'AC': 9, 'AD': 6, 'BC': 10, 'BD': 6, 'CD': 9, 'BA': 9, 'CA': 9, 'DA': 6, 'CB': 10, 'DB': 6, 'DC': 9}
dict_keys(['A', 'B', 'C', 'D', 'BC'])
Rules Generated are as follows:


A : A Beginner'sGuide
B : Java: The Complete Reference
A -> B
Confidence Value : 0.8181818181818182


A : A Beginner'sGuide
C : JavaFor Dummies
A -> C
Confidence Value : 0.8181818181818182
```

```
B : Java: The Complete Reference
A : A Beginner'sGuide
B -> A
Confidence Value : 0.9


B : Java: The Complete Reference
C : JavaFor Dummies
B -> C
Confidence Value : 1.0


B : Java: The Complete Reference
D : Android Programming: The Big Nerd Ranch
B -> D
Confidence Value : 0.6



C : JavaFor Dummies
A : A Beginner'sGuide
C -> A
Confidence Value : 0.6923076923076923


C : JavaFor Dummies
B : Java: The Complete Reference
C -> B
Confidence Value : 0.7692307692307693


C : JavaFor Dummies
D : Android Programming: The Big Nerd Ranch
C -> D
Confidence Value : 0.6923076923076923
```

```
D : Android Programming: The Big Nerd Ranch
C : JavaFor Dummies
D -> C
Confidence Value : 0.6923076923076923




--- The Apriori algorithm now executed takes: 7.565478801727295 seconds ---

Process finished with exit code 0
```

2. Best Buy Dataset:

Running the Apriori Algorithm on Best Buy Dataset for Support 70% and Confidence 80%.

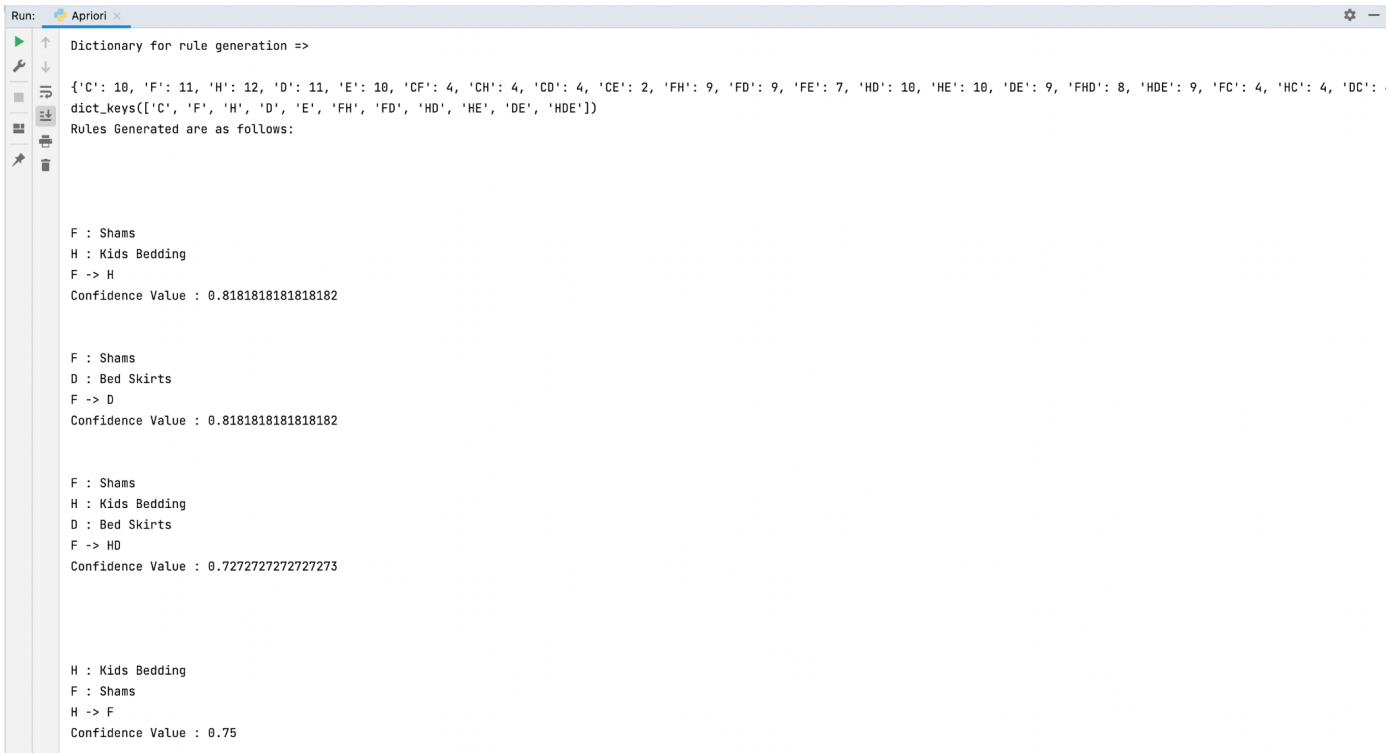


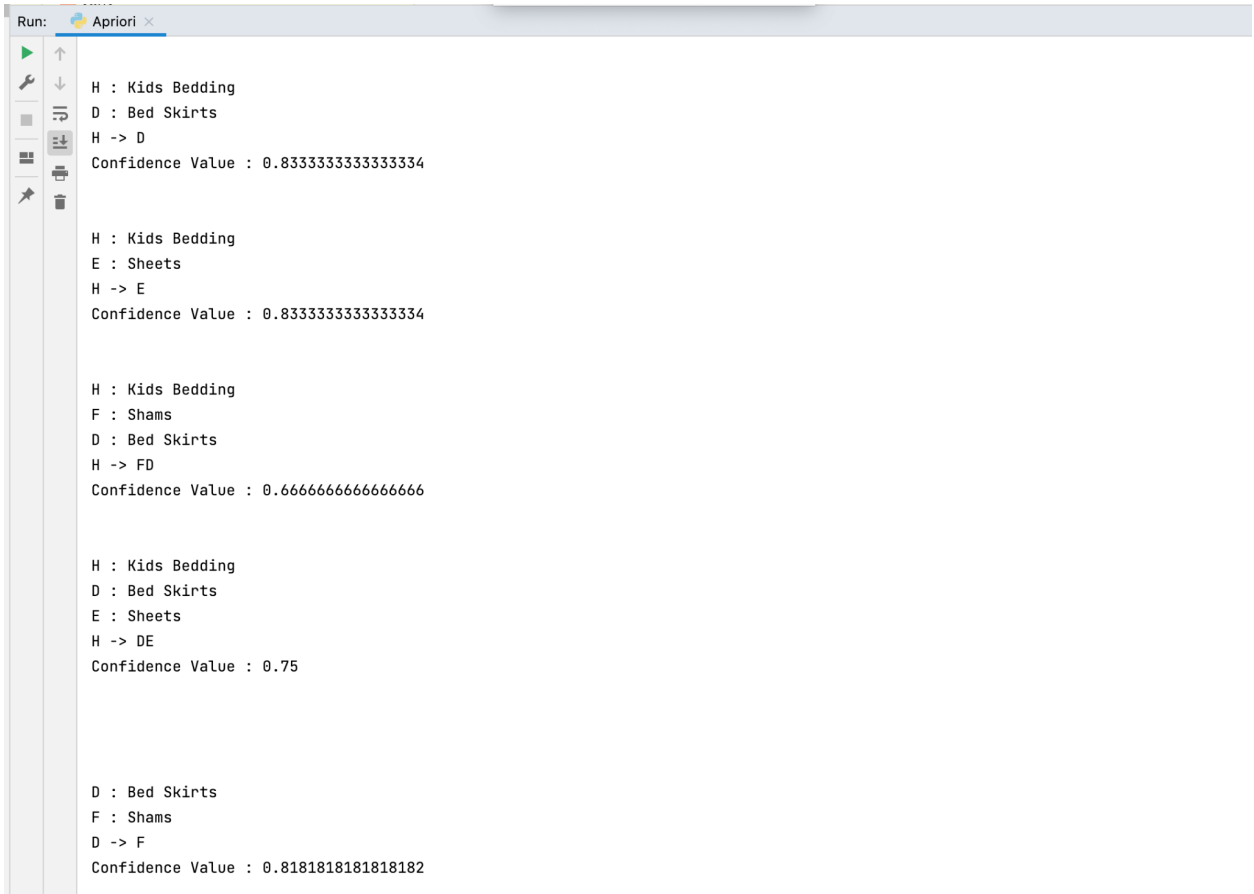
```
Run:  Apriori
Best Buy Dataset selected!
   TransactionID                                    Transaction
0         Trans1  Desk Top, Printer, Flash Drive, Microsoft Offi...
1         Trans2  Lap Top, Flash Drive, Microsoft Office, Lap To...
2         Trans3  Lap Top, Printer, Flash Drive, Microsoft Offic...
3         Trans4  Lap Top, Printer, Flash Drive, Anti-Virus, Ext...
4         Trans5     Lap Top, Flash Drive, Lap Top Case, Anti-Virus
5         Trans6    Lap Top, Printer, Flash Drive, Microsoft Office
6         Trans7   Desk Top, Printer, Flash Drive, Microsoft Office
7         Trans8           Lap Top, External Hard-Drive, Anti-Virus
8         Trans9  Desk Top, Printer, Flash Drive, Microsoft Offi...
9        Trans10  Digital Camera , Lap Top, Desk Top, Printer, F...
10       Trans11  Lap Top, Desk Top, Lap Top Case, External Hard...
11       Trans12  Digital Camera , Lap Top, Lap Top Case, Extern...
12       Trans13                        Digital Camera , Speakers
13       Trans14  Digital Camera , Desk Top, Printer, Flash Driv...
14       Trans15  Printer, Flash Drive, Microsoft Office, Anti-V...
15       Trans16  Digital Camera, Flash Drive, Microsoft Office,...
16       Trans17             Digital Camera , Lap Top, Lap Top Case
17       Trans18            Digital Camera , Lap Top Case, Speakers
18       Trans19  Digital Camera , Lap Top, Printer, Flash Drive...
19       Trans20  Digital Camera , Lap Top, Speakers, Anti-Virus...
{'C': 'Desk Top', 'D': 'Printer', 'E': 'Flash Drive', 'F': 'Microsoft Office', 'G': 'Speakers', 'J': 'Anti-Virus', 'B': 'Lap Top', 'H': 'Lap Top Case', 'K': 'External Hard-Drive', 'A
Choose the option for minimum support format:
1. Percentage
2. Natural Number
1
70
Minimum Support Count :  14
Choose the option for minimum confidence format:
1. Percentage
2. Floating Number
1
80
Minimum Confidence:  0.8
['CDEFGJ', 'BEFHJ', 'BDEFJHK', 'BDEJKH', 'BEHJ', 'BDEF', 'CDEF', 'BKJ', 'CDEFHJGK', 'ABCDEFHJKG', 'BCHKGJ', 'ABHKJG', 'AG', 'ACDEF', 'DEFJHGK', 'AEFJHKG', 'ABH', 'AHG', 'ABDEFGHJ', '
C1 =>  {'C': 6, 'D': 10, 'E': 13, 'F': 11, 'G': 11, 'J': 14, 'B': 12, 'H': 14, 'K': 9, 'A': 9}
```

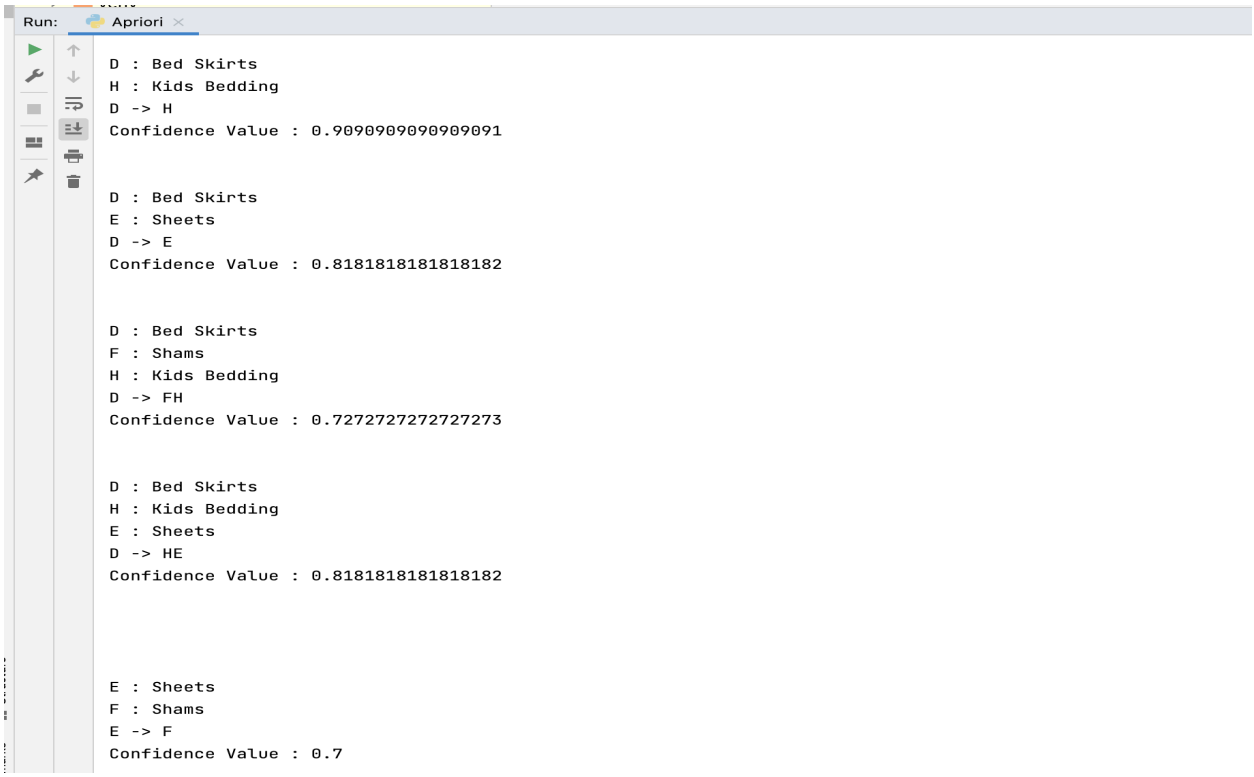
```
C1 =>  {'C': 6, 'D': 10, 'E': 13, 'F': 11, 'G': 11, 'J': 14, 'B': 12, 'H': 14, 'K': 9, 'A': 9}
L1 =>  {'J': 14, 'H': 14}
C 2  (by joining) L 1  =>  {'JH': 0}
C 2  (by pruning) =>  {'JH': 0}
C 2  (with support count) =>  {'JH': 12}
L 2  =>  {}
C 3  (by joining) L 2  =>  {}
C 3  (by pruning) =>  {}
C 3  (with support count) =>  {}
L 3  =>  {}

Dictionary for rule generation =>

{'J': 14, 'H': 14, 'JH': 12, 'HJ': 12}
dict_keys(['J', 'H'])
Rules Generated are as follows:


J : Anti-Virus
H : Lap Top Case
J -> H
Confidence Value : 0.8571428571428571




H : Lap Top Case
J : Anti-Virus
H -> J
Confidence Value : 0.8571428571428571



--- The Apriori algorithm now executed takes: 7.191391944885254 seconds ---
```

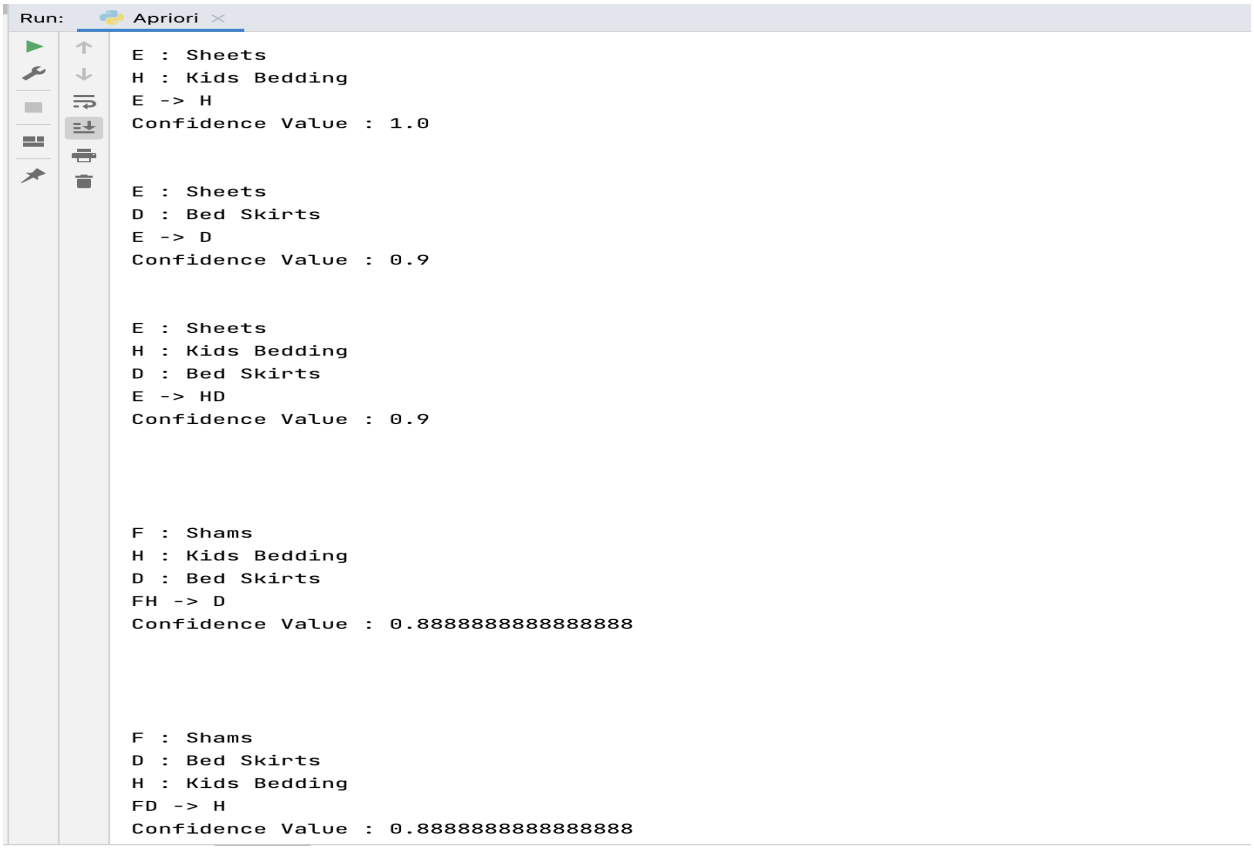
3. K Mart Dataset:

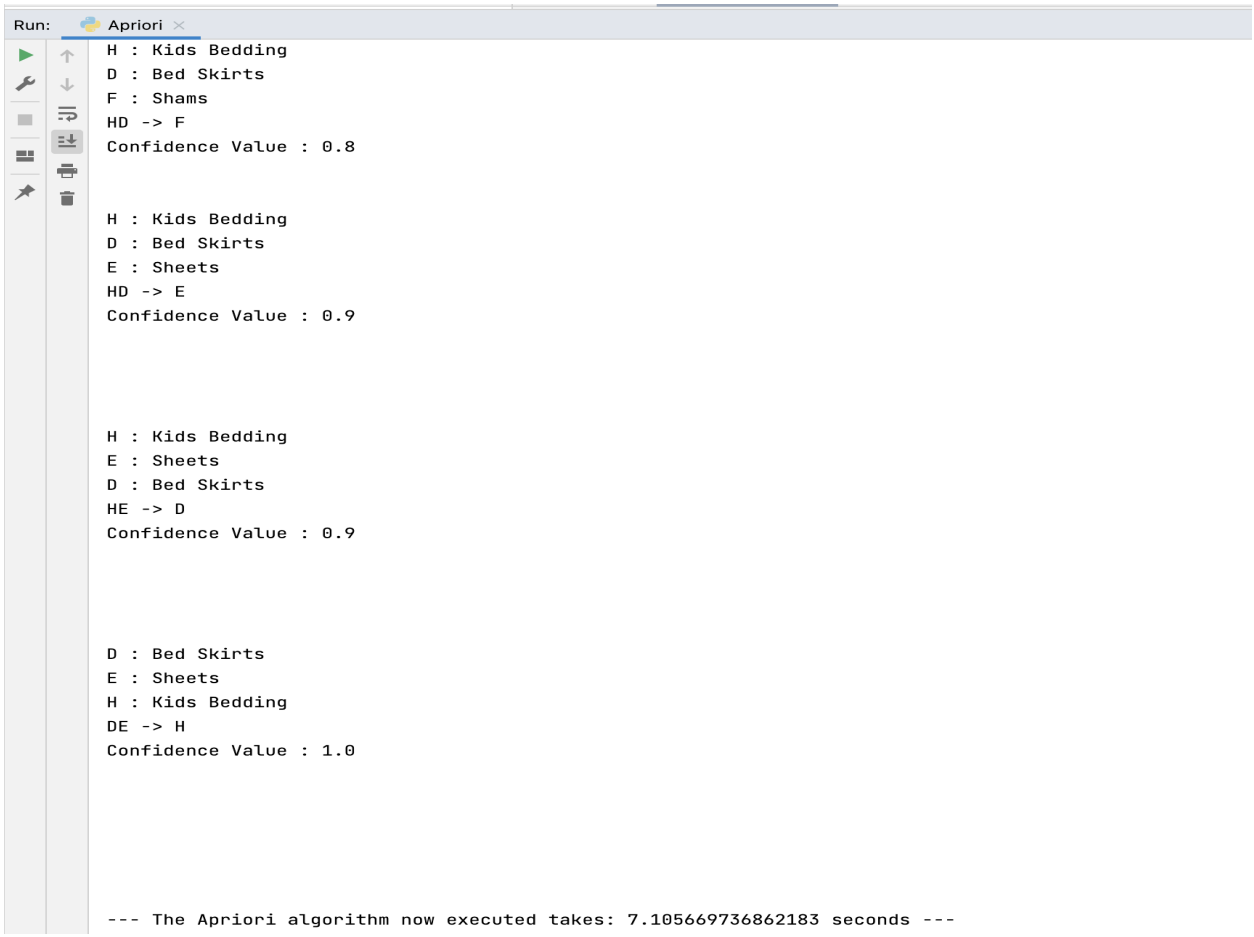
Running the Apriori Algorithm on K Mart Dataset for Support 45% and Confidence 65%.


Run:
Apriori
Dictionary for rule generation =>
{'C': 10, 'F': 11, 'H': 12, 'D': 11, 'E': 10, 'CF': 4, 'CH': 4, 'CD': 4, 'CE': 2, 'FH': 9, 'FD': 9, 'FE': 7, 'HD': 10, 'HE': 10, 'DE': 9, 'FHD': 8, 'HDE': 9, 'FC': 4, 'HC': 4, 'DC':
dict_keys(['C', 'F', 'H', 'D', 'E', 'FH', 'FD', 'HD', 'HE', 'DE', 'HDE'])
Rules Generated are as follows:
F : Shams
H : Kids Bedding
F -> H
Confidence Value : 0.8181818181818182
F : Shams
D : Bed Skirts
F -> D
Confidence Value : 0.8181818181818182
F : Shams
H : Kids Bedding
D : Bed Skirts
F -> HD
Confidence Value : 0.7272727272727273
H : Kids Bedding
F : Shams
H -> F
Confidence Value : 0.75


Run:
Apriori
H : Kids Bedding
D : Bed Skirts
H -> D
Confidence Value : 0.8333333333333334
H : Kids Bedding
E : Sheets
H -> E
Confidence Value : 0.8333333333333334
H : Kids Bedding
F : Shams
D : Bed Skirts
H -> FD
Confidence Value : 0.6666666666666666
H : Kids Bedding
D : Bed Skirts
E : Sheets
H -> DE
Confidence Value : 0.75
D : Bed Skirts
F : Shams
D -> F
Confidence Value : 0.8181818181818182

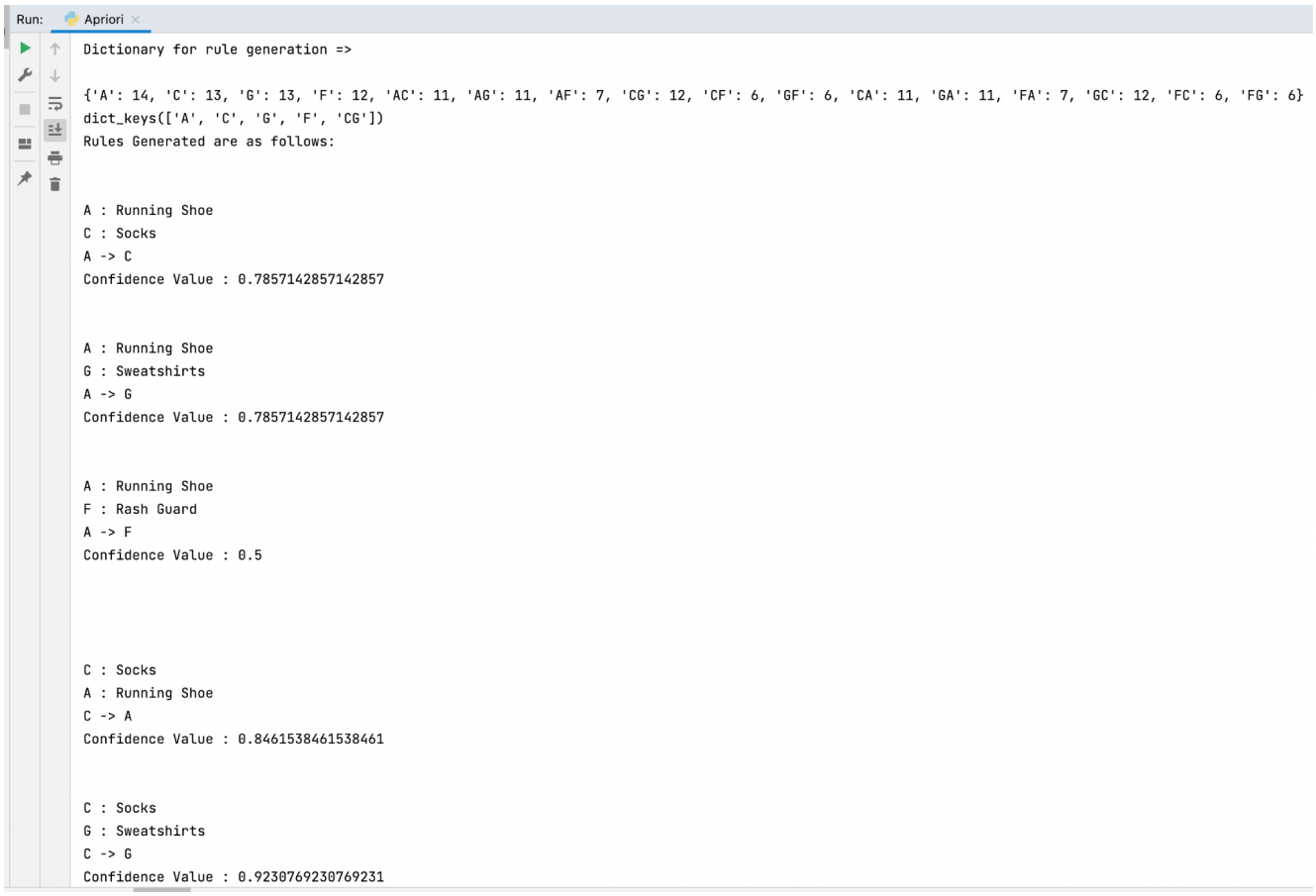

Run:
Apriori
D : Bed Skirts
H : Kids Bedding
D -> H
Confidence Value : 0.9090909090909091
D : Bed Skirts
E : Sheets
D -> E
Confidence Value : 0.8181818181818182
D : Bed Skirts
F : Shams
H : Kids Bedding
D -> FH
Confidence Value : 0.7272727272727273
D : Bed Skirts
H : Kids Bedding
E : Sheets
D -> HE
Confidence Value : 0.8181818181818182
E : Sheets
F : Shams
E -> F
Confidence Value : 0.7

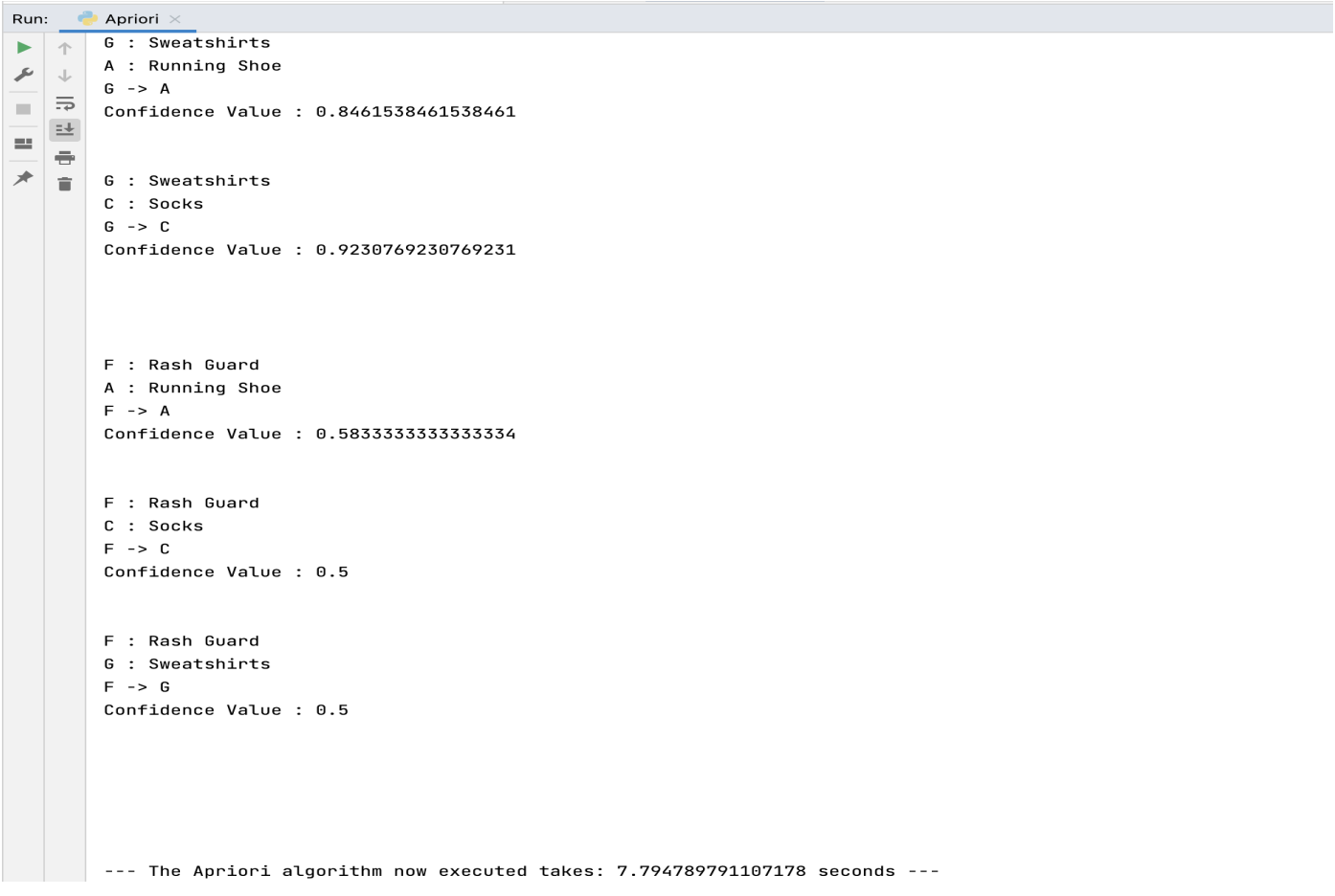

Run:
Apriori
E : Sheets
H : Kids Bedding
E -> H
Confidence Value : 1.0
E : Sheets
D : Bed Skirts
E -> D
Confidence Value : 0.9
E : Sheets
H : Kids Bedding
D : Bed Skirts
E -> HD
Confidence Value : 0.9
F : Shams
H : Kids Bedding
D : Bed Skirts
FH -> D
Confidence Value : 0.8888888888888888
F : Shams
D : Bed Skirts
H : Kids Bedding
FD -> H
Confidence Value : 0.8888888888888888



4. Nike DataSet:

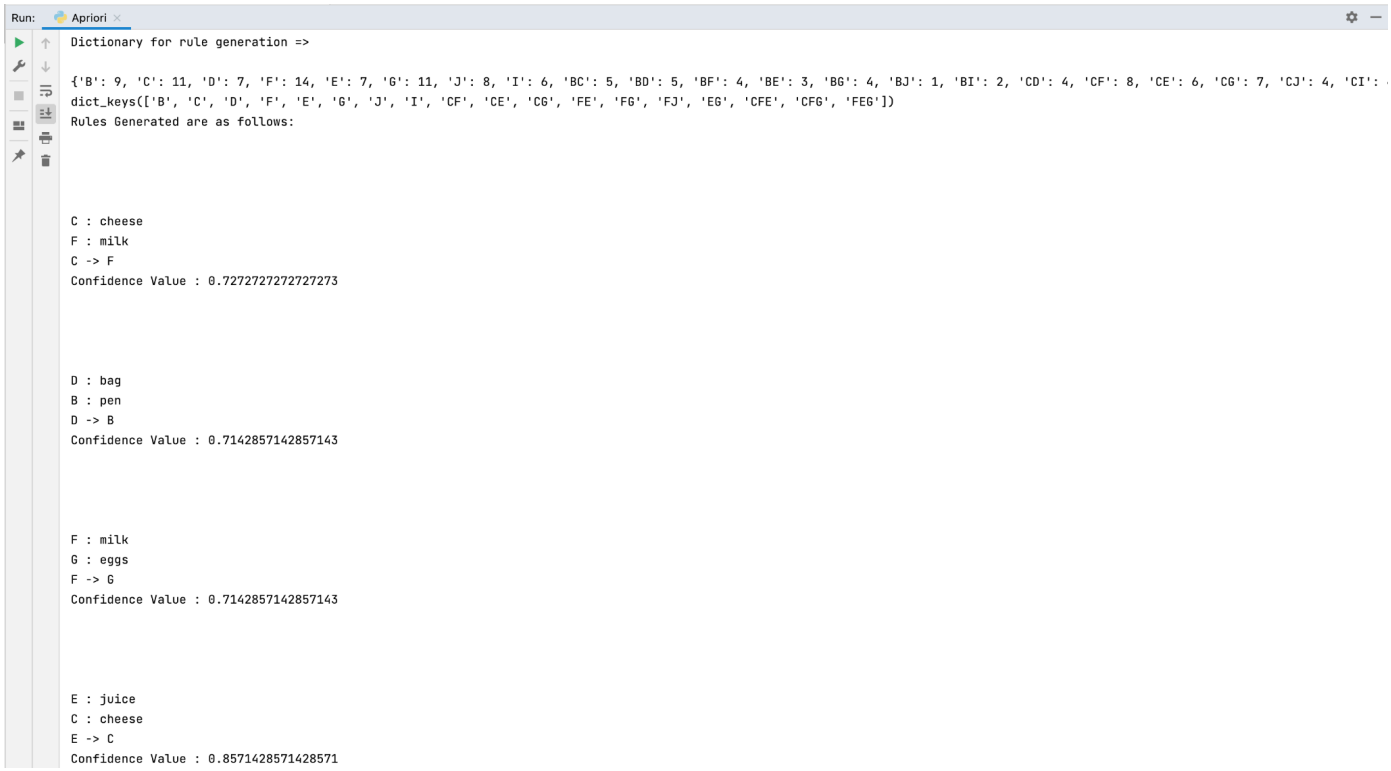
Running the Apriori Algorithm on Nike Dataset for Support 60% and Confidence 50%.

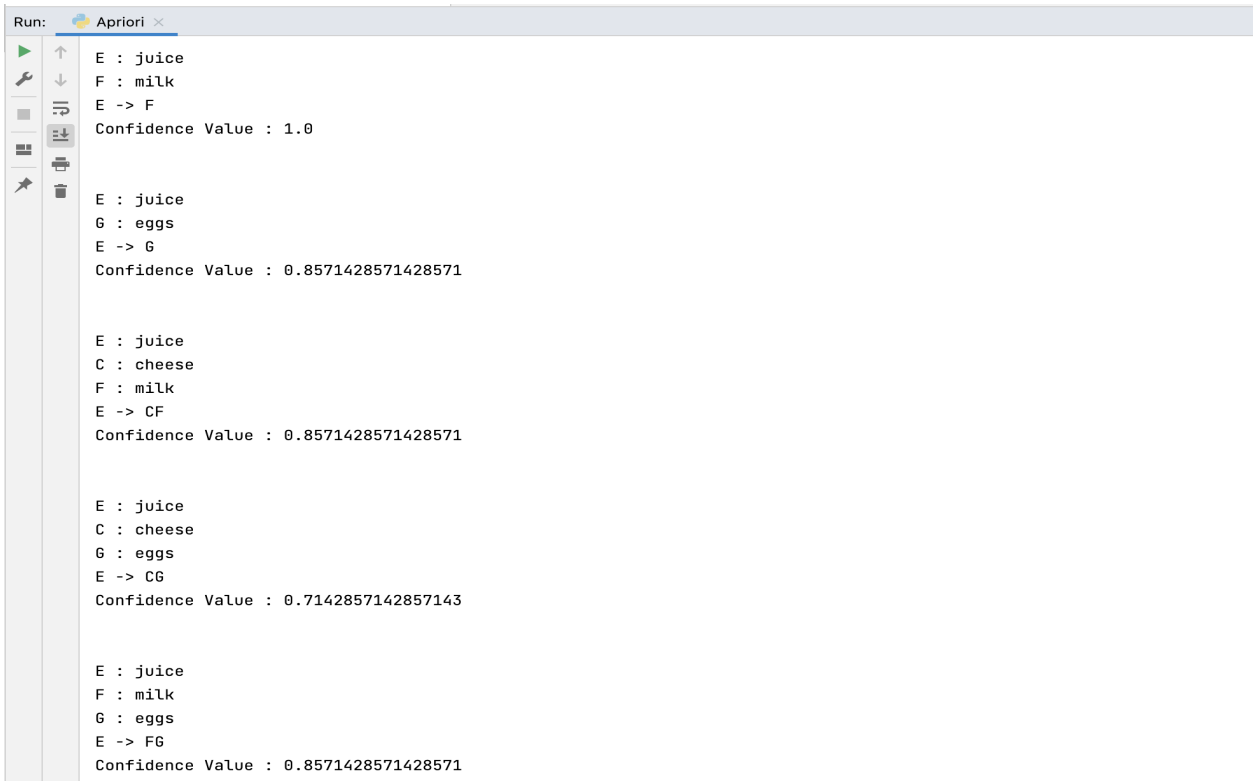

Run:
Apriori
Dictionary for rule generation =>
{'A': 14, 'C': 13, 'G': 13, 'F': 12, 'AC': 11, 'AG': 11, 'AF': 7, 'CG': 12, 'CF': 6, 'GF': 6, 'CA': 11, 'GA': 11, 'FA': 7, 'GC': 12, 'FC': 6, 'FG': 6}
dict_keys(['A', 'C', 'G', 'F', 'CG'])
Rules Generated are as follows:
A : Running Shoe
C : Socks
A -> C
Confidence Value : 0.7857142857142857
A : Running Shoe
G : Sweatshirts
A -> G
Confidence Value : 0.7857142857142857
A : Running Shoe
F : Rash Guard
A -> F
Confidence Value : 0.5
C : Socks
A : Running Shoe
C -> A
Confidence Value : 0.8461538461538461
C : Socks
G : Sweatshirts
C -> G
Confidence Value : 0.9230769230769231

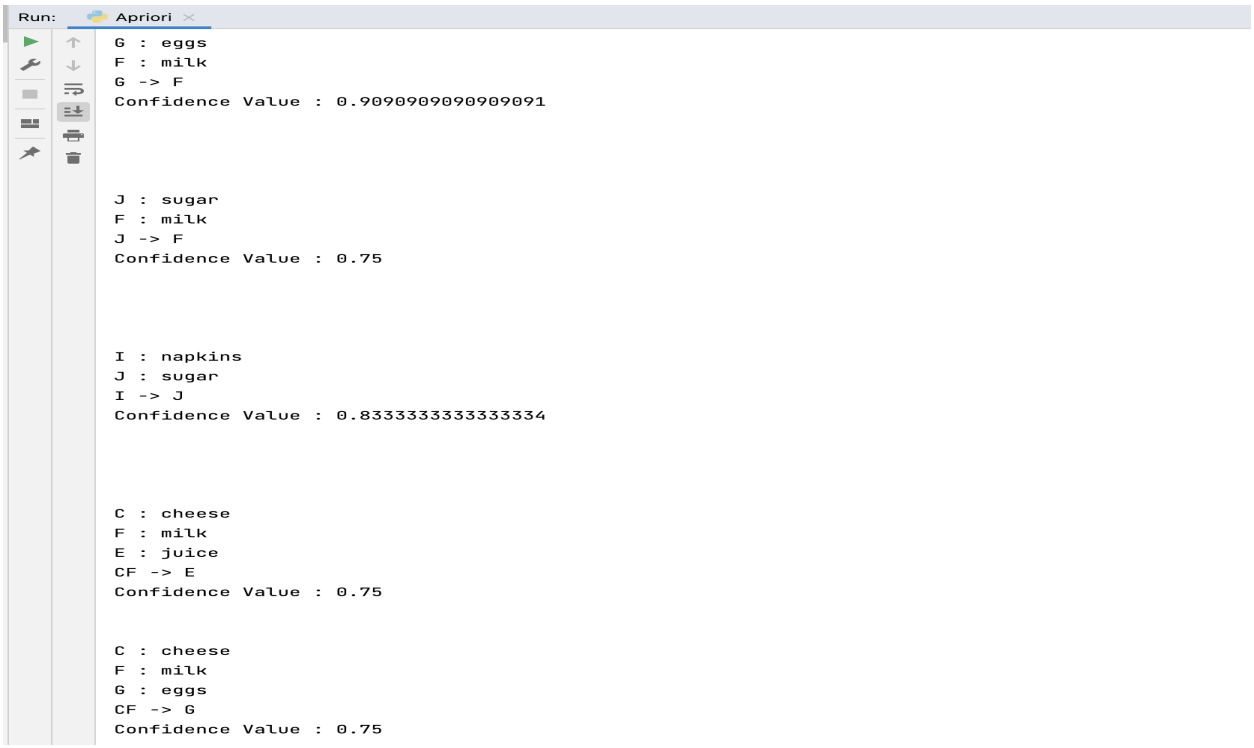

Run:
Apriori
G : Sweatshirts
A : Running Shoe
G -> A
Confidence Value : 0.8461538461538461
G : Sweatshirts
C : Socks
G -> C
Confidence Value : 0.9230769230769231
F : Rash Guard
A : Running Shoe
F -> A
Confidence Value : 0.5833333333333334
F : Rash Guard
C : Socks
F -> C
Confidence Value : 0.5
F : Rash Guard
G : Sweatshirts
F -> G
Confidence Value : 0.5
--- The Apriori algorithm now executed takes: 7.794789791107178 seconds ---

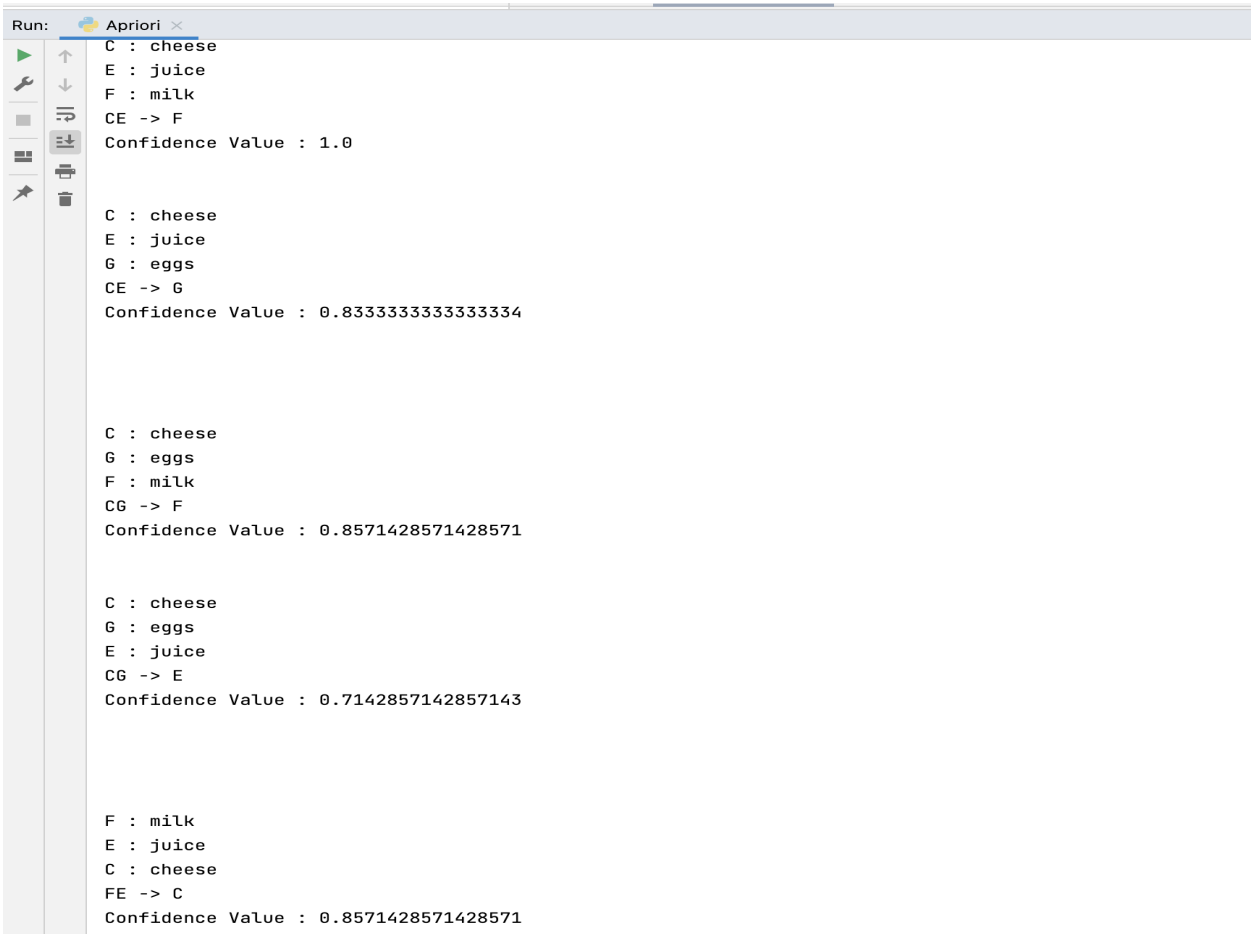
5. Custom DataSet:

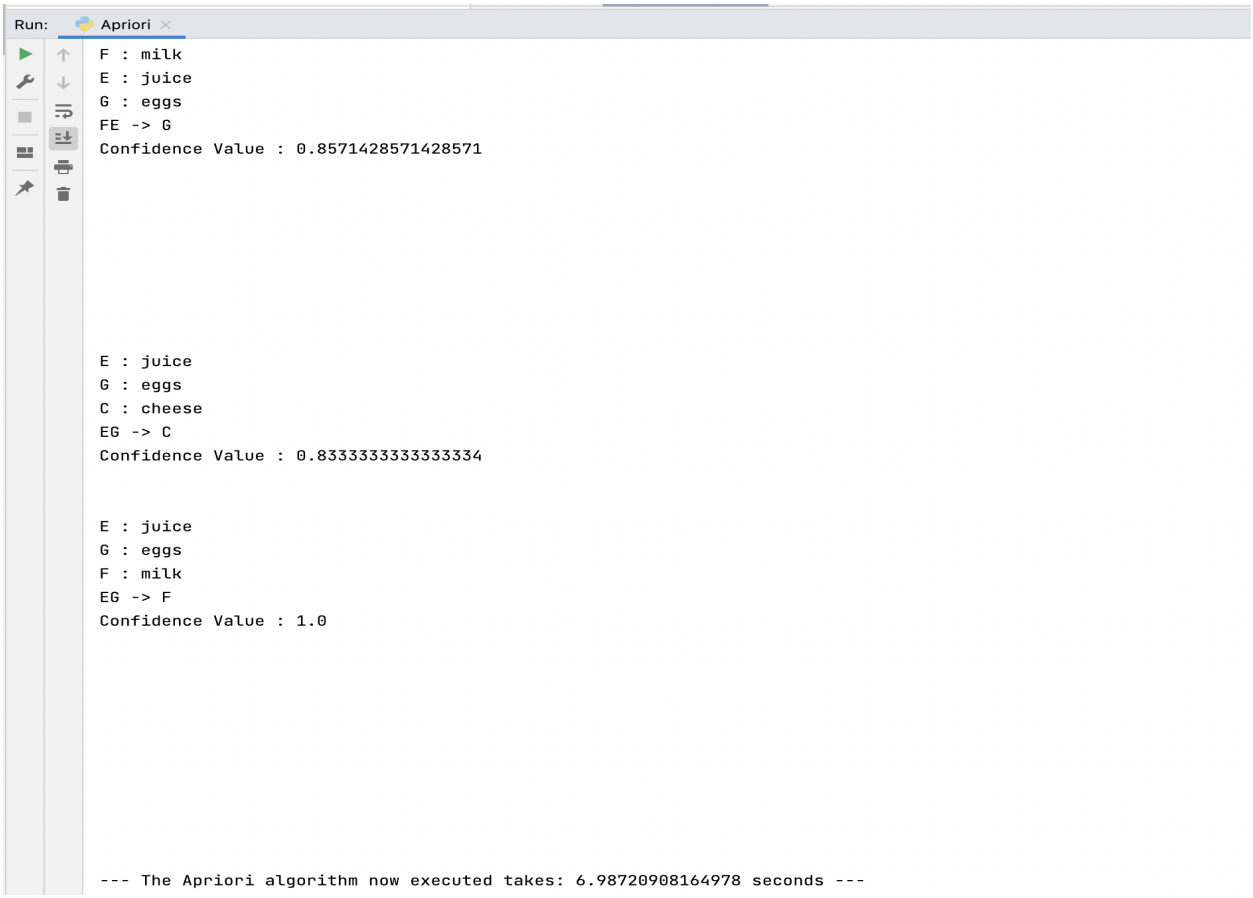
Running the Apriori Algorithm on Custom Dataset for Support 30% and Confidence 70%.






Run:
Apriori
G : eggs
F : milk
G -> F
Confidence Value : 0.9090909090909091
J : sugar
F : milk
J -> F
Confidence Value : 0.75
I : napkins
J : sugar
I -> J
Confidence Value : 0.8333333333333334
C : cheese
F : milk
E : juice
CF -> E
Confidence Value : 0.75
C : cheese
F : milk
G : eggs
CF -> G
Confidence Value : 0.75


Run:
Apriori
C : cheese
E : juice
F : milk
CE -> F
Confidence Value : 1.0
C : cheese
E : juice
G : eggs
CE -> G
Confidence Value : 0.8333333333333334
C : cheese
G : eggs
F : milk
CG -> F
Confidence Value : 0.8571428571428571
C : cheese
G : eggs
E : juice
CG -> E
Confidence Value : 0.7142857142857143
F : milk
E : juice
C : cheese
FE -> C
Confidence Value : 0.8571428571428571


Run:
Apriori
F : milk
E : juice
G : eggs
FE -> G
Confidence Value : 0.8571428571428571
E : juice
G : eggs
C : cheese
EG -> C
Confidence Value : 0.8333333333333334
E : juice
G : eggs
F : milk
EG -> F
Confidence Value : 1.0
--- The Apriori algorithm now executed takes: 6.98720908164978 seconds ---

Part 2 - Brute Force Approach

A brute force approach is an approach that finds every possible solution to find a satisfactory solution to a given problem. The brute force algorithm calculation evaluates every one of the potential outcomes till a satisfactory solution is not found.

A classic and simple example in computer science is the traveling salesman problem (TSP). Suppose a salesman needs to travel 20 cities across the state. How does one decide the order in which those cities should be visited such that the total distance traveled is minimized?

The brute force algorithm solution is simply to calculate the total distance for every possible route and then select the shortest one. This is not particularly efficient because it is possible to eliminate many possible routes through other clever algorithms.

Advantage of this type of algorithm is that it is applicable to a wide range of domains. It is mainly utilized for solving simpler and small problems.

It is an inefficient algorithm as it requires solving each and every state.

However, It is a very slow algorithm to find the correct solution as it solves each state without considering whether the solution is feasible or not.

Source Code

```
import time

start_time = time.time()


print("Choose and enter the number from menu:\n")

print("1.Amazon\n2.Best Buy\n3.K-mart\n4.Nike\n5.Custom Data")

n = int(input().strip())


if n == 1:

    # Amazon Dataset

    print("Amazon Dataset selected!")

    import pandas as pd


    df = pd.read_csv('amazon.csv')

    print(df)

    newTransaction = list([""] * len(df['Transaction']))

    dictTransactions = {}

    # perform run time encoding of transactions of in the for loop

    for i in range(0, len(df['Transaction'])):

        tempStr = ""

        for sstr in df['Transaction'][i].split(","):

            if sstr.strip() == 'A Beginner'sGuide':
```

```
    tempStr = tempStr + 'A'
    dictTransactions['A'] = 'A Beginner'sGuide'

elif sstr.strip() == 'Java: The Complete Reference':
    tempStr = tempStr + 'B'
    dictTransactions['B'] = 'Java: The Complete Reference'

elif sstr.strip() == 'JavaFor Dummies':
    tempStr = tempStr + 'C'
    dictTransactions['C'] = 'JavaFor Dummies'

elif sstr.strip() == 'Android Programming: The Big Nerd Ranch':
    tempStr = tempStr + 'D'
    dictTransactions['D'] = 'Android Programming: The Big Nerd Ranch'

elif sstr.strip() == 'Head First Java 2nd Edition':
    tempStr = tempStr + 'E'
    dictTransactions['E'] = 'Head First Java 2nd Edition'

elif sstr.strip() == 'Beginning Programming with Java':
    tempStr = tempStr + 'F'
    dictTransactions['F'] = 'Beginning Programming with Java'
```

```
            elif sstr.strip() == 'Java 8 Pocket Guide':
                tempStr = tempStr + 'G'
                dictTransactions['G'] = 'Java 8 Pocket Guide'

            elif sstr.strip() == 'C++ Programming in Easy Steps':
                tempStr = tempStr + 'H'
                dictTransactions['H'] = 'C++ Programming in Easy Steps'

            elif sstr.strip() == 'Effective Java (2nd Edition)':
                tempStr = tempStr + 'I'
                dictTransactions['I'] = 'Effective Java (2nd Edition)'

            elif sstr.strip() == 'HTML and CSS: Design and Build Websites':
                tempStr = tempStr + 'J'
                dictTransactions['J'] = 'HTML and CSS: Design and Build Websites'

        newTransaction[i] += tempStr
    df["NewTransaction"] = newTransaction

elif n == 2:
    # Best Buy Dataset
    print("Best Buy Dataset selected!")
    import pandas as pd
```

```
df = pd.read_csv('best_buy.csv')
print(df)
newTransaction = list([""] * len(df['Transaction']))
dictTransactions = {}
# perform run time encoding of transactions of in the for loop
for i in range(0, len(df['Transaction'])):
    tempStr = ""
    for sstr in df['Transaction'][i].split(","):
        if sstr.strip() == 'Digital Camera':
            tempStr = tempStr + 'A'
            dictTransactions['A'] = 'Digital Camera'

        elif sstr.strip() == 'Lap Top':
            tempStr = tempStr + 'B'
            dictTransactions['B'] = 'Lap Top'

        elif sstr.strip() == 'Desk Top':
            tempStr = tempStr + 'C'
            dictTransactions['C'] = 'Desk Top'

        elif sstr.strip() == 'Printer':
            tempStr = tempStr + 'D'
```

```
    dictTransactions['D'] = 'Printer'

elif sstr.strip() == 'Flash Drive':
    tempStr = tempStr + 'E'
    dictTransactions['E'] = 'Flash Drive'

elif sstr.strip() == 'Microsoft Office':
    tempStr = tempStr + 'F'
    dictTransactions['F'] = 'Microsoft Office'

elif sstr.strip() == 'Speakers':
    tempStr = tempStr + 'G'
    dictTransactions['G'] = 'Speakers'

elif sstr.strip() == 'Lap Top Case':
    tempStr = tempStr + 'H'
    dictTransactions['H'] = 'Lap Top Case'

elif sstr.strip() == 'Effective Java (2nd Edition)':
    tempStr = tempStr + 'I'
    dictTransactions['I'] = 'Effective Java (2nd Edition)'

elif sstr.strip() == 'Anti-Virus':
```

```
            tempStr = tempStr + 'J'
            dictTransactions['J'] = 'Anti-Virus'

        elif sstr.strip() == 'External Hard-Drive':
            tempStr = tempStr + 'K'
            dictTransactions['K'] = 'External Hard-Drive'

      newTransaction[i] += tempStr
   df["NewTransaction"] = newTransaction

elif n == 3:
   # KMart Dataset
   print("KMart Dataset selected!")
   import pandas as pd

   df = pd.read_csv('kmart.csv')
   print(df)
   newTransaction = list([""] * len(df['Transaction']))
   dictTransactions = {}
   # perform run time encoding of transactions of in the for loop
   for i in range(0, len(df['Transaction'])):
      tempStr = ""
      for sstr in df['Transaction'][i].split(","):
```

```
if sstr.strip() == 'Quilts':
    tempStr = tempStr + 'A'
    dictTransactions['A'] = 'Quilts'

elif sstr.strip() == 'Bedspreads':
    tempStr = tempStr + 'B'
    dictTransactions['B'] = 'Bedspreads'

elif sstr.strip() == 'Decorative Pillows':
    tempStr = tempStr + 'C'
    dictTransactions['C'] = 'Decorative Pillows'

elif sstr.strip() == 'Bed Skirts':
    tempStr = tempStr + 'D'
    dictTransactions['D'] = 'Bed Skirts'

elif sstr.strip() == 'Sheets':
    tempStr = tempStr + 'E'
    dictTransactions['E'] = 'Sheets'

elif sstr.strip() == 'Shams':
    tempStr = tempStr + 'F'
    dictTransactions['F'] = 'Shams'
```

```
            elif sstr.strip() == 'Bedding Collections':
                tempStr = tempStr + 'G'
                dictTransactions['G'] = 'Bedding Collections'

            elif sstr.strip() == 'Kids Bedding':
                tempStr = tempStr + 'H'
                dictTransactions['H'] = 'Kids Bedding'

            elif sstr.strip() == 'Embroidered Bedspread':
                tempStr = tempStr + 'I'
                dictTransactions['I'] = 'Embroidered Bedspread'

            elif sstr.strip() == 'Towels':
                tempStr = tempStr + 'J'
                dictTransactions['J'] = 'Towels'

        newTransaction[i] += tempStr
    df["NewTransaction"] = newTransaction

elif n == 4:
    # Nike Dataset
    print("Nike Dataset selected!")
```

```
import pandas as pd


df = pd.read_csv('nike.csv')
print(df)
newTransaction = list([""] * len(df['Transaction']))
dictTransactions = {}
# perform run time encoding of transactions of in the for loop
for i in range(0, len(df['Transaction'])):
    tempStr = ""
    for sstr in df['Transaction'][i].split(","):
        if sstr.strip() == 'Running Shoe':
            tempStr = tempStr + 'A'
            dictTransactions['A'] = 'Running Shoe'


        elif sstr.strip() == 'Soccer Shoe':
            tempStr = tempStr + 'B'
            dictTransactions['B'] = 'Soccer Shoe'


        elif sstr.strip() == 'Socks':
            tempStr = tempStr + 'C'
            dictTransactions['C'] = 'Socks'


        elif sstr.strip() == 'Swimming Shirt':
```

```
    tempStr = tempStr + 'D'
    dictTransactions['D'] = 'Swimming Shirt'

elif sstr.strip() == 'Dry Fit V-Neck':
    tempStr = tempStr + 'E'
    dictTransactions['E'] = 'Dry Fit V-Neck'

elif sstr.strip() == 'Rash Guard':
    tempStr = tempStr + 'F'
    dictTransactions['F'] = 'Rash Guard'

elif sstr.strip() == 'Sweatshirts':
    tempStr = tempStr + 'G'
    dictTransactions['G'] = 'Sweatshirts'

elif sstr.strip() == 'Hoodies':
    tempStr = tempStr + 'H'
    dictTransactions['H'] = 'Hoodies'

elif sstr.strip() == 'Tech Pants':
    tempStr = tempStr + 'I'
    dictTransactions['I'] = 'Tech Pants'
```

```
            elif sstr.strip() == 'Modern Pants':
                tempStr = tempStr + 'J'
                dictTransactions['J'] = 'Modern Pants'

        newTransaction[i] += tempStr
    df["NewTransaction"] = newTransaction

elif n == 5:
    # Custom Dataset
    print("Custom Dataset selected!")
    import pandas as pd

    df = pd.read_csv('custom.csv')
    print(df)
    newTransaction = list([""] * len(df['Transaction']))
    dictTransactions = {}
    # perform run time encoding of transactions of in the for loop
    for i in range(0, len(df['Transaction'])):
        tempStr = ""
        for sstr in df['Transaction'][i].split(","):
            if sstr.strip() == 'ink':
                tempStr = tempStr + 'A'
                dictTransactions['A'] = 'ink'
```

```
elif sstr.strip() == 'pen':
    tempStr = tempStr + 'B'
    dictTransactions['B'] = 'pen'

elif sstr.strip() == 'cheese':
    tempStr = tempStr + 'C'
    dictTransactions['C'] = 'cheese'

elif sstr.strip() == 'bag':
    tempStr = tempStr + 'D'
    dictTransactions['D'] = 'bag'

elif sstr.strip() == 'juice':
    tempStr = tempStr + 'E'
    dictTransactions['E'] = 'juice'

elif sstr.strip() == 'milk':
    tempStr = tempStr + 'F'
    dictTransactions['F'] = 'milk'

elif sstr.strip() == 'eggs':
    tempStr = tempStr + 'G'
```

```
            dictTransactions['G'] = 'eggs'


        elif sstr.strip() == 'bread':
            tempStr = tempStr + 'H'
            dictTransactions['H'] = 'bread'


        elif sstr.strip() == 'napkins':
            tempStr = tempStr + 'I'
            dictTransactions['I'] = 'napkins'


        elif sstr.strip() == 'sugar':
            tempStr = tempStr + 'J'
            dictTransactions['J'] = 'sugar'


    newTransaction[i] += tempStr
df["NewTransaction"] = newTransaction


print(dictTransactions)


print("Choose the option for minimum support format:\n1. Percentage\n2. Natural Number")
n = int(input().strip())
if n == 1:
```

```
    minSupPercentage = int(input().strip())
    minSup = (minSupPercentage * len(df['Transaction'])) // 100
elif n == 2:
    minSup = int(input())
print("Minimum Support Count : ", minSup)


print("Choose the option for minimum confidence format:\n1. Percentage\n2. Floating Number")
n = int(input().strip())
if n==1:
    minConfidencePercentage = int(input().strip())
    minConfidence = minConfidencePercentage / 100
elif n==2:
    minConfidence = float(input())
print("Minimum Confidence: ", minConfidence)


myList = list(df['NewTransaction'])
myList


#function for counting the frequency of items in candidate sets by scanning the transactions
def generate_Candidate(c, transactions):
    itemsInC = []
    for key in c.keys():
```

```
        itemsInC.append(key)
    for i in range(0, len(transactions)):
        tid = transactions[i]
        for j in range(0, len(itemsInC)):
            item = itemsInC[j]
            count = 0
            for k in range(0, len(item)):
                atkthpos = item[k] + ""
                if tid.find(atkthpos) >= 0:
                    count = count + 1
            if count == len(item):
                val = c[itemsInC[j]]
                val = val + 1
                c[item] = val
    return c


#function for generating k-item candidate sets by joining the (k-1)frequent itemsets
def join(l):
    c = {}
    itemInL = []
    for key in l.keys():
        itemInL.append(key)
    for i in range(0, len(itemInL)):
```

```
        for j in range(i+1, len(itemInL)):
            forchecki = itemInL[i][0:len(itemInL[i])-1]
            forcheckj = itemInL[j][0:len(itemInL[j])-1]
            if forchecki == forcheckj:
                newPotentialCandidate = itemInL[i] + itemInL[j][len(itemInL[j])-1]
                c[newPotentialCandidate] = 0
    return c


#function for generating frequent itemsets by checking the itemsets frequency with minimum support
def generate_frequent_set(transactions, minSup, c):
    l = {}
    nList = []
    for key in c.keys():
        nList.append(key)
    for i in range(0, len(nList)):
        support = c[nList[i]]
        if support >= minSup:
            l[nList[i]] = support
    return l


transactions = myList
finalDict = {}
```

```
DictForSubsetCounts = {}
print(transactions)
c = {}
for i in range(0, len(transactions)):
    tid = transactions[i]
    for j in range(0, len(tid)):
        ch = tid[j]
        if ch in c.keys():
            val = c[ch]
            val = val + 1
            c[ch] = val
        else:
            c[ch] = 1
print("C1 => ", c)
l = generate_frequent_set(transactions, minSup, c)
finalDict.update(l)
DictForSubsetCounts.update(l)
print("L1 => ", l)
counter = 2
while len(c) > 0:
    c = join(l)
    print("C", counter, " (by joining) L", (counter - 1), " => ", c)
```

```
    # c = prune(c, l)
    # print("C", counter, " (by pruning) => ", c)

    c = generate_Candidate(c, transactions);
    DictForSubsetCounts.update(c)
    print("C", counter, " (with support count) => ", c);

    l = generate_frequent_set(transactions, minSup, c)
    finalDict.update(l)
    DictForSubsetCounts.update(l)
    print("L", counter, " => ", l);

    counter = counter + 1

tmpDict = DictForSubsetCounts
keysInDictForSubsetCounts = list(DictForSubsetCounts.keys())
from itertools import permutations

for key in keysInDictForSubsetCounts:
    val = tmpDict.get(key)
    perm = [''.join(p) for p in permutations(key)]
    for p in perm:
        DictForSubsetCounts[p] = val
```

```
print("\nDictionary for rule generation =>\n")

print(DictForSubsetCounts)


dictKeys = finalDict.keys()

print(dictKeys)


print("Rules Generated are as follows:\n\n")

for individualString1 in dictKeys:

    for individualString2 in dictKeys:

        flag = 0

        for c in individualString1:

            if c in individualString2:

                flag = 1

                break

        if flag!=1:

            if (individualString1+individualString2) in DictForSubsetCounts.keys():

                confVal =
int(DictForSubsetCounts.get(individualString1+individualString2))/DictForSubsetCounts
.get(individualString1)

                if confVal >= minConfidence:

                    for char in individualString1:

                        print(char+" : "+dictTransactions[char])

                    for char in individualString2:

                        print(char+" : "+dictTransactions[char])
```

```
                print(individualString1+' -> '+individualString2)
                print("Confidence Value : " + str(confVal))
                print()
                print()
    print()
    print()

print("--- The Brute Force algorithm now executed takes: %s seconds ---" % (time.time() - start_time))
```

Results of Algorithm on Datasets

This section describes association rules generated from datasets as a result, given minimum support and minimum confidence.

1. **Amazon Dataset:**

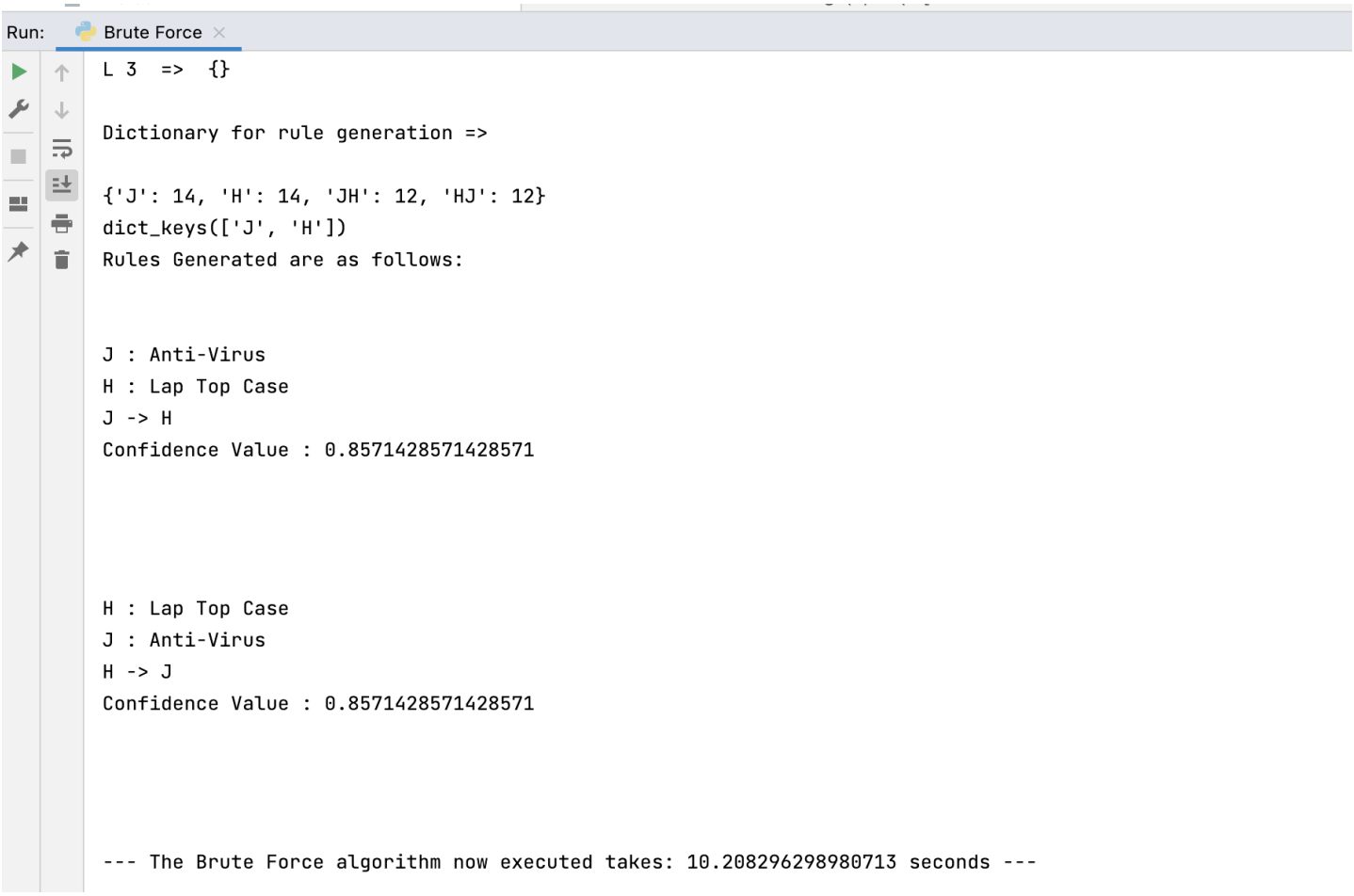
   Running the Brute Force Algorithm on Amazon Dataset for Support 50% and Confidence 60%.

```
Dictionary for rule generation =>

{'A': 11, 'B': 10, 'C': 13, 'D': 13, 'AB': 9, 'AC': 9, 'AD': 6, 'BC': 10, 'BD': 6, 'CD': 9, 'BA': 9, 'CA': 9, 'DA': 6, 'CB': 10, 'DB': 6, 'DC': 9}
dict_keys(['A', 'B', 'C', 'D', 'BC'])
Rules Generated are as follows:


A : A Beginner'sGuide
B : Java: The Complete Reference
A -> B
Confidence Value : 0.8181818181818182


A : A Beginner'sGuide
C : JavaFor Dummies
A -> C
Confidence Value : 0.8181818181818182




B : Java: The Complete Reference
A : A Beginner'sGuide
B -> A
Confidence Value : 0.9


B : Java: The Complete Reference
C : JavaFor Dummies
B -> C
Confidence Value : 1.0


B : Java: The Complete Reference
D : Android Programming: The Big Nerd Ranch
B -> D
Confidence Value : 0.6
```

```
C : JavaFor Dummies
A : A Beginner’sGuide
C -> A
Confidence Value : 0.6923076923076923


C : JavaFor Dummies
B : Java: The Complete Reference
C -> B
Confidence Value : 0.7692307692307693


C : JavaFor Dummies
D : Android Programming: The Big Nerd Ranch
C -> D
Confidence Value : 0.6923076923076923




D : Android Programming: The Big Nerd Ranch
C : JavaFor Dummies
D -> C
Confidence Value : 0.6923076923076923




--- The Brute Force algorithm now executed takes: 9.801676034927368 seconds ---
```

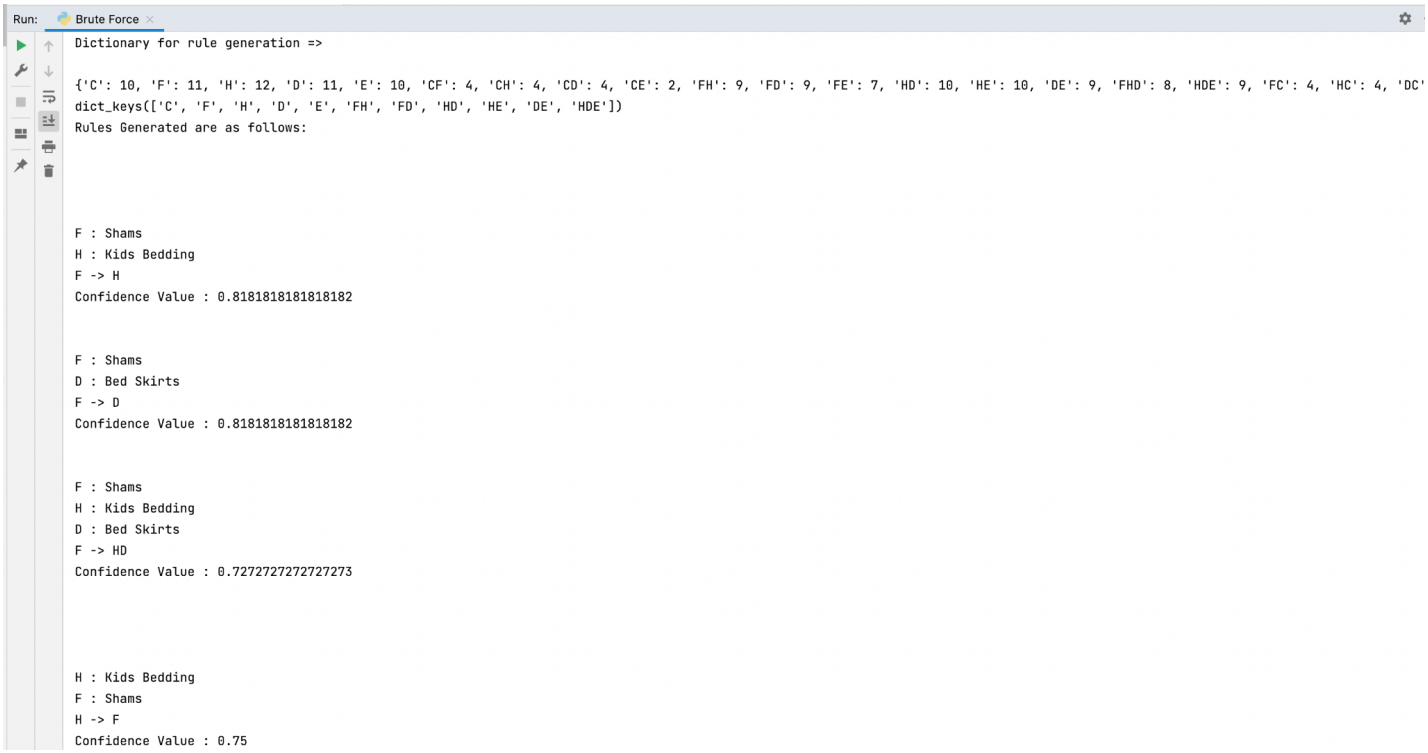
2. Best Buy Dataset:

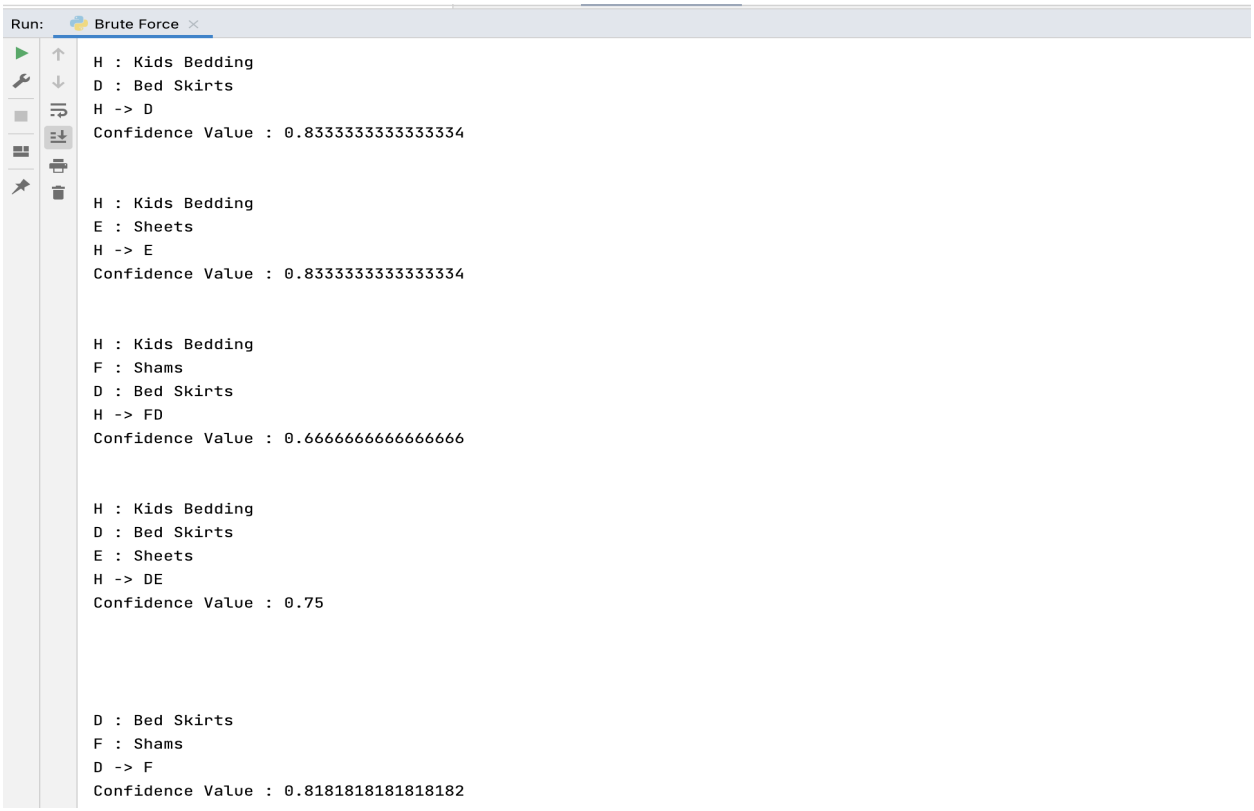
Running the Brute Force Algorithm on Best Buy Dataset for Support 70% and Confidence 80%.

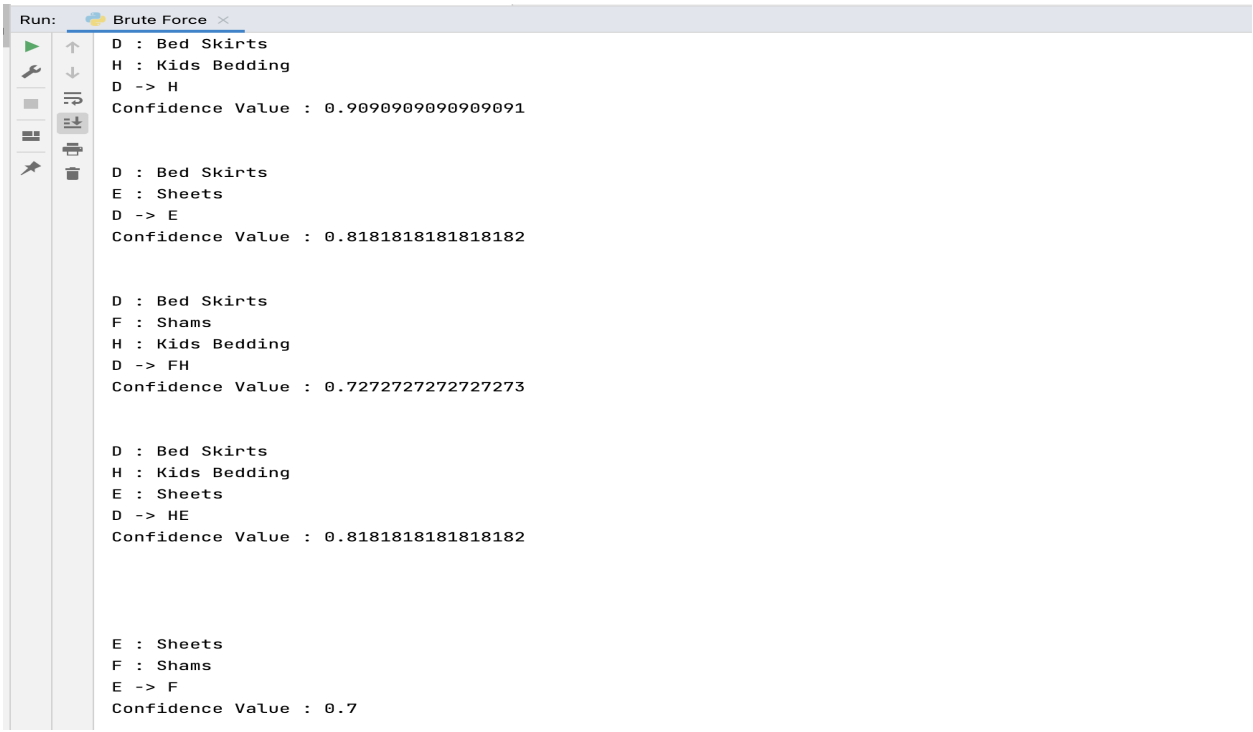


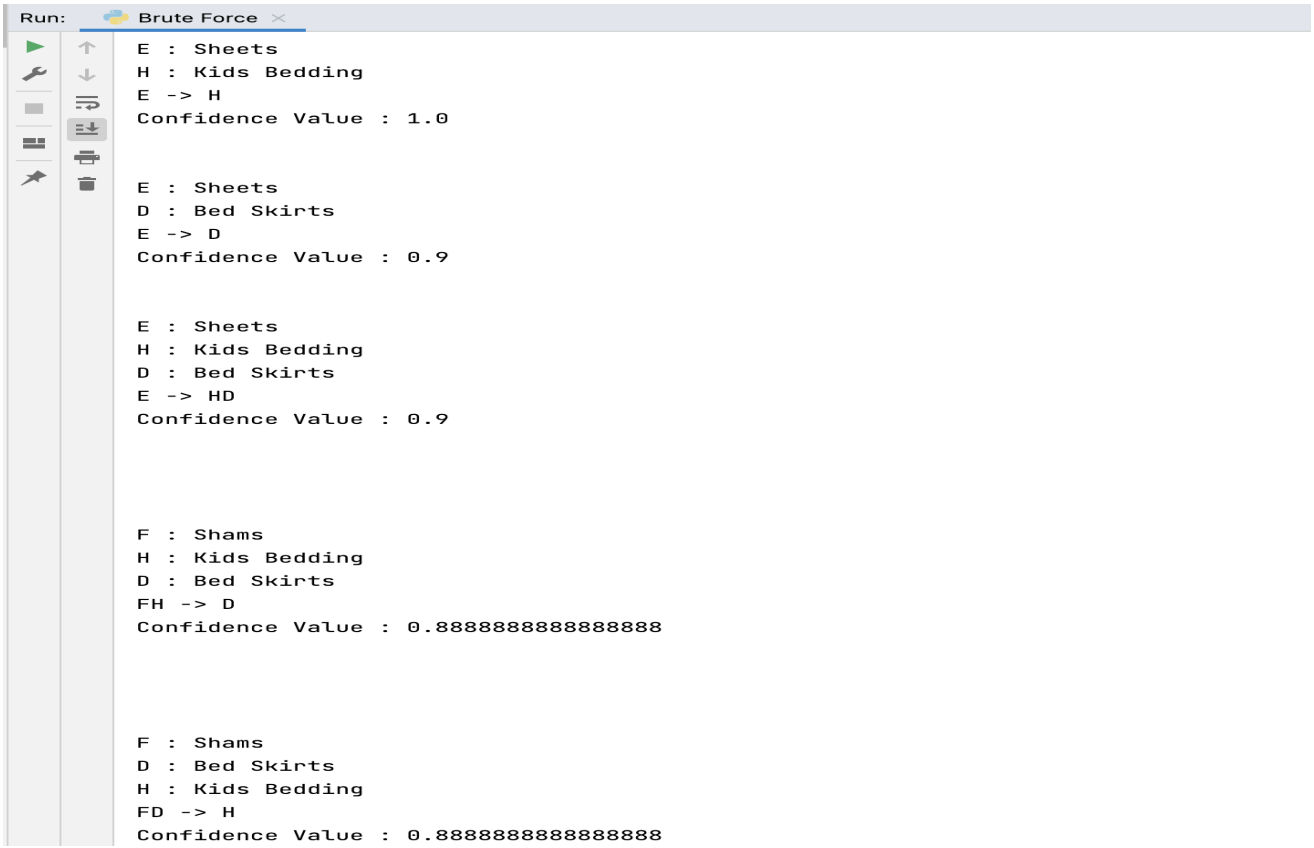
3. K Mart Dataset:

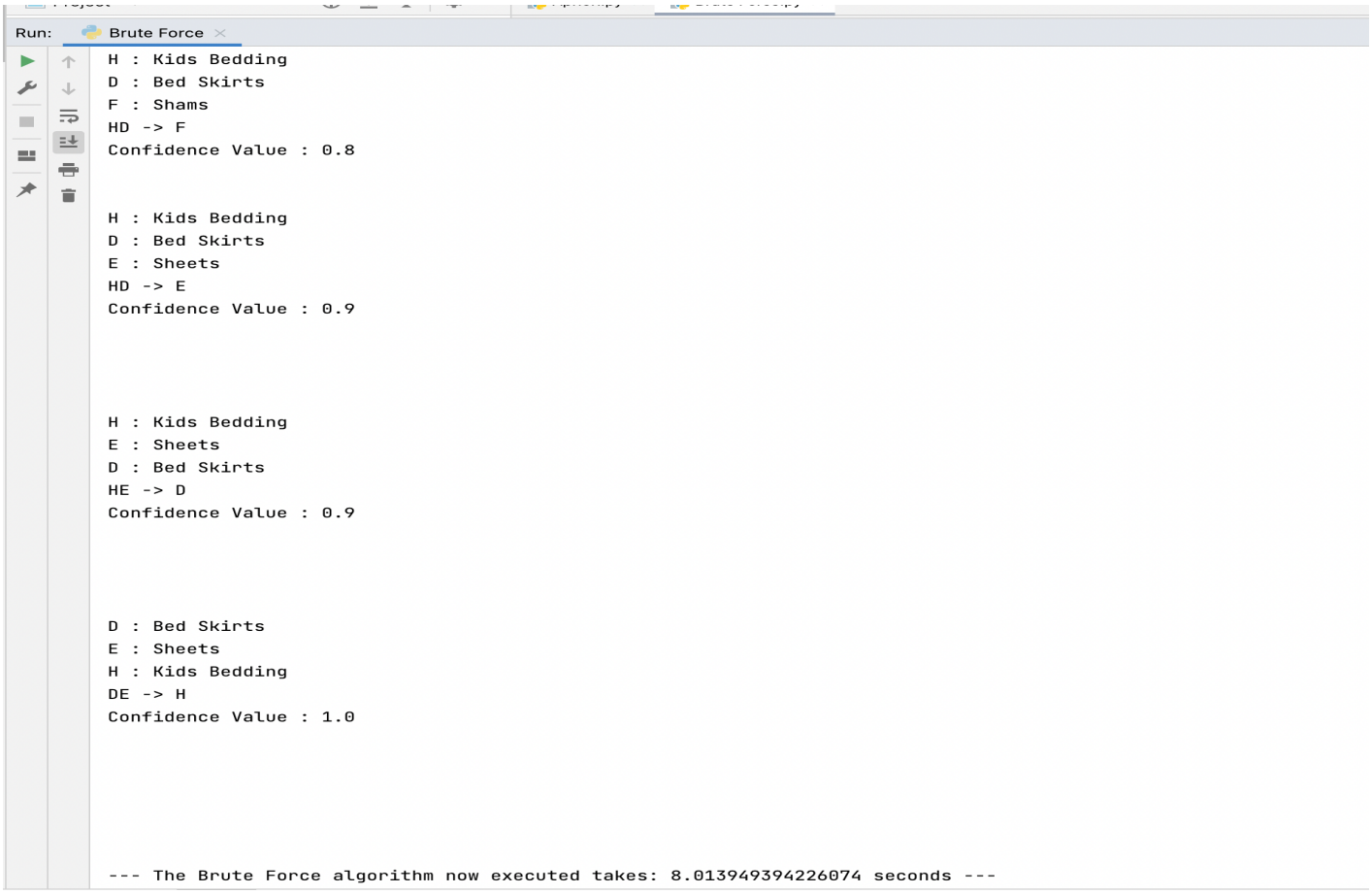
Running the Brute Force Algorithm on K Mart Dataset for Support 45% and Confidence 65%.






Run:
Brute Force
D : Bed Skirts
H : Kids Bedding
D -> H
Confidence Value : 0.9090909090909091
D : Bed Skirts
E : Sheets
D -> E
Confidence Value : 0.8181818181818182
D : Bed Skirts
F : Shams
H : Kids Bedding
D -> FH
Confidence Value : 0.7272727272727273
D : Bed Skirts
H : Kids Bedding
E : Sheets
D -> HE
Confidence Value : 0.8181818181818182
E : Sheets
F : Shams
E -> F
Confidence Value : 0.7


Run:
Brute Force
E : Sheets
H : Kids Bedding
E -> H
Confidence Value : 1.0
E : Sheets
D : Bed Skirts
E -> D
Confidence Value : 0.9
E : Sheets
H : Kids Bedding
D : Bed Skirts
E -> HD
Confidence Value : 0.9
F : Shams
H : Kids Bedding
D : Bed Skirts
FH -> D
Confidence Value : 0.8888888888888888
F : Shams
D : Bed Skirts
H : Kids Bedding
FD -> H
Confidence Value : 0.8888888888888888



4. Nike Dataset:

Running the Brute Force Algorithm on Nike Dataset for Support 60% and Confidence 50%.

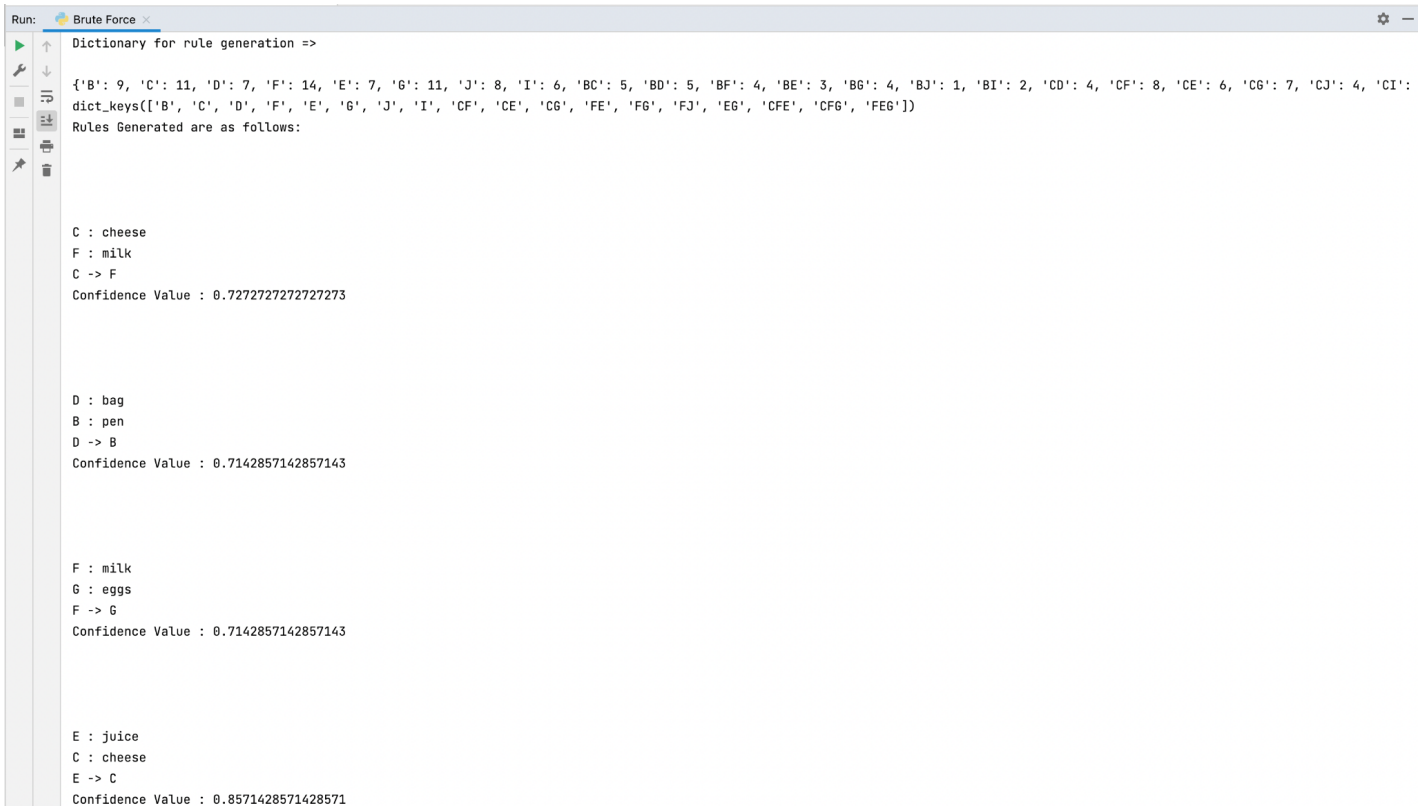
```
Dictionary for rule generation =>

{'A': 14, 'C': 13, 'G': 13, 'F': 12, 'AC': 11, 'AG': 11, 'AF': 7, 'CG': 12, 'CF': 6, 'GF': 6, 'CA': 11, 'GA': 11, 'FA': 7, 'GC': 12, 'FC': 6, 'FG': 6}
dict_keys(['A', 'C', 'G', 'F', 'CG'])
Rules Generated are as follows:


A : Running Shoe
C : Socks
A -> C
Confidence Value : 0.7857142857142857


A : Running Shoe
G : Sweatshirts
A -> G
Confidence Value : 0.7857142857142857


A : Running Shoe
F : Rash Guard
A -> F
Confidence Value : 0.5



C : Socks
A : Running Shoe
C -> A
Confidence Value : 0.8461538461538461


C : Socks
G : Sweatshirts
C -> G
Confidence Value : 0.9230769230769231
```

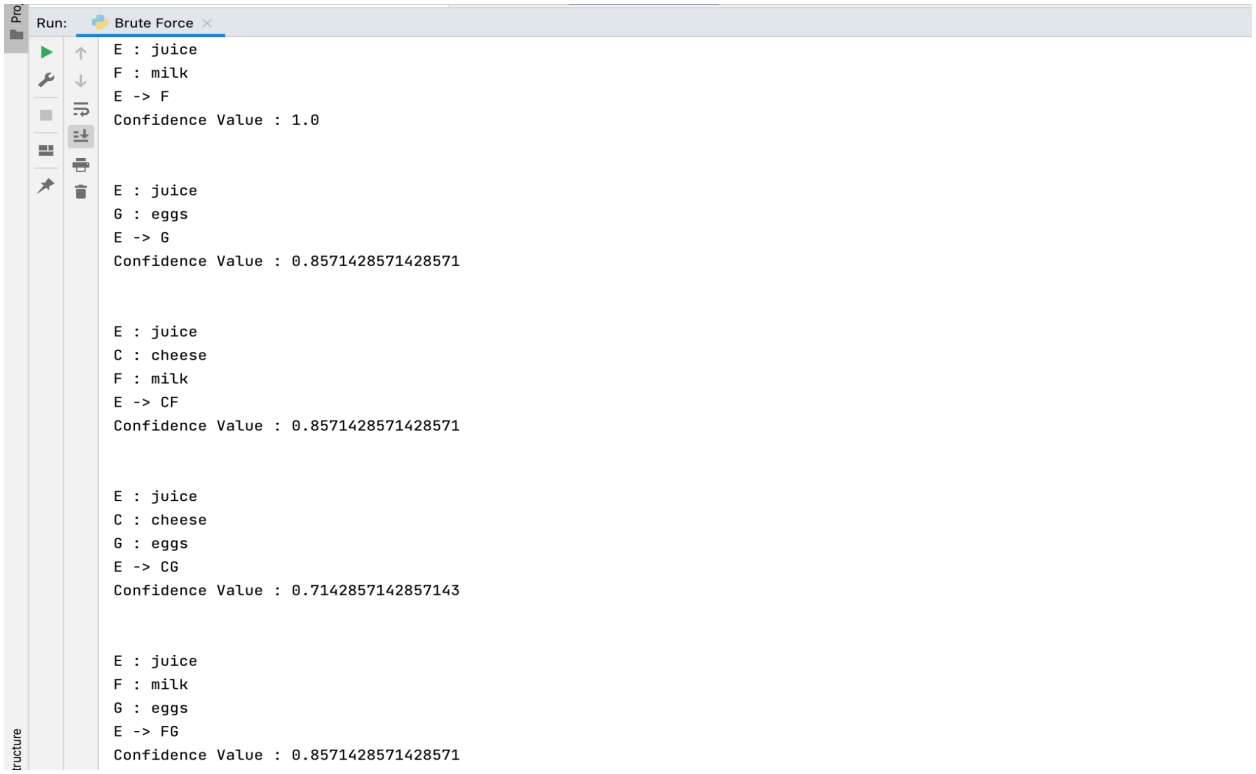
Run: Brute Force

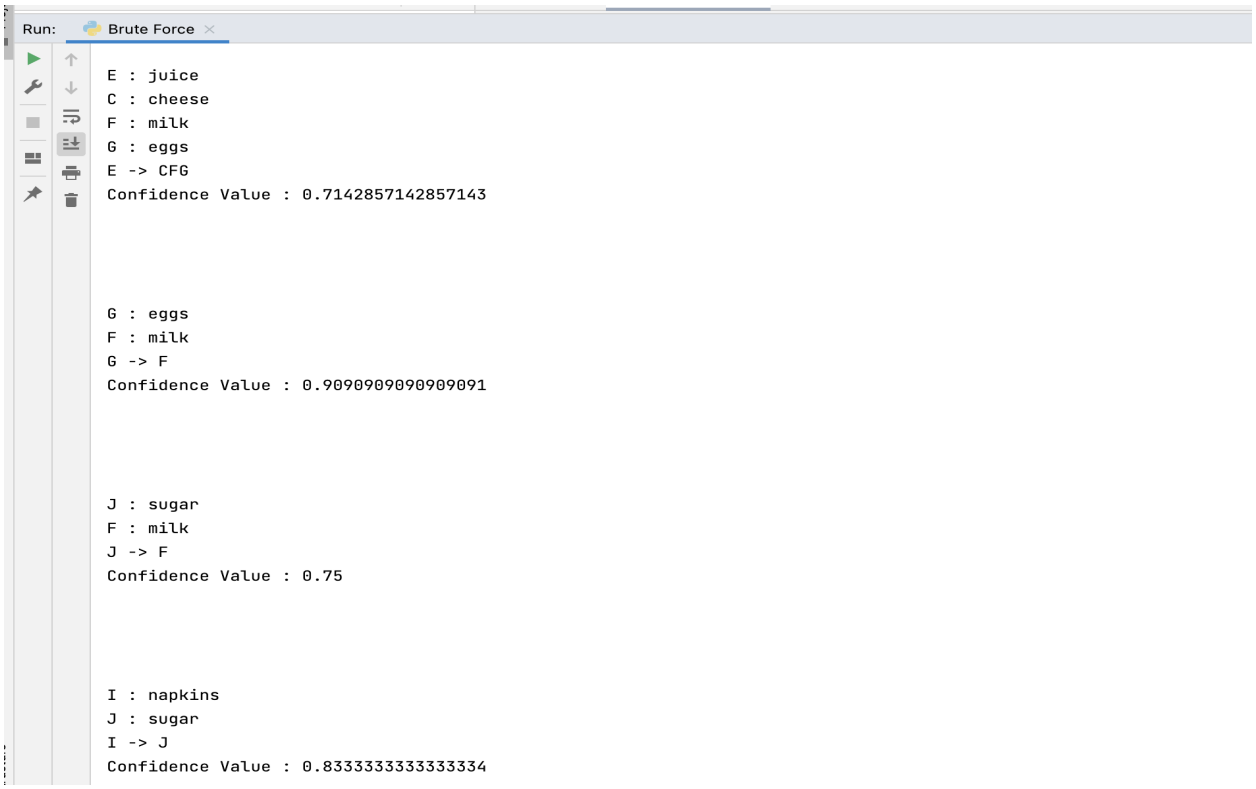
```
G : Sweatshirts
A : Running Shoe
G -> A
Confidence Value : 0.8461538461538461


G : Sweatshirts
C : Socks
G -> C
Confidence Value : 0.9230769230769231



F : Rash Guard
A : Running Shoe
F -> A
Confidence Value : 0.5833333333333334


F : Rash Guard
C : Socks
F -> C
Confidence Value : 0.5


F : Rash Guard
G : Sweatshirts
F -> G
Confidence Value : 0.5



--- The Brute Force algorithm now executed takes: 9.223587989807129 seconds ---
```

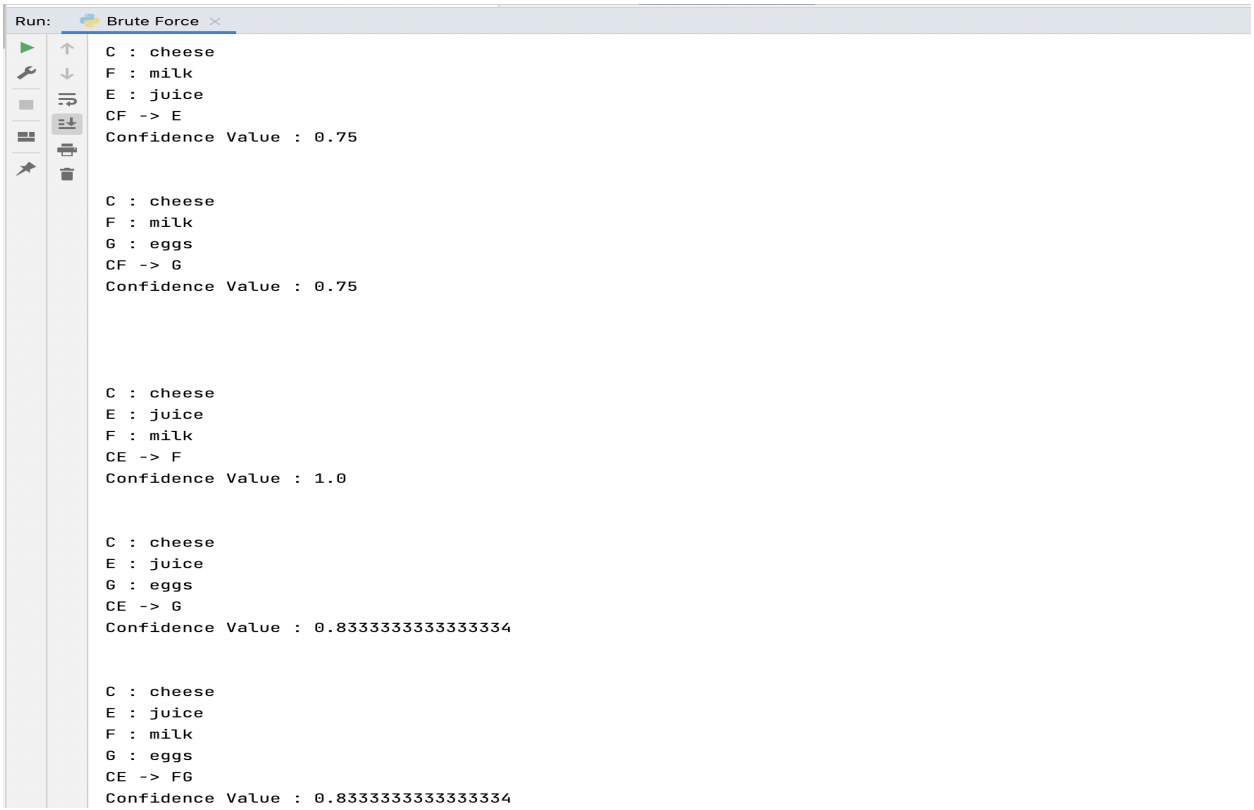
5. Custom Dataset:

Running the Brute Force Algorithm on Custom Dataset for Support 30% and Confidence 70%.






Run:
Brute Force
E : juice
C : cheese
F : milk
G : eggs
E -> CFG
Confidence Value : 0.7142857142857143
G : eggs
F : milk
G -> F
Confidence Value : 0.9090909090909091
J : sugar
F : milk
J -> F
Confidence Value : 0.75
I : napkins
J : sugar
I -> J
Confidence Value : 0.8333333333333334


Run:
Brute Force
C : cheese
F : milk
E : juice
CF -> E
Confidence Value : 0.75
C : cheese
F : milk
G : eggs
CF -> G
Confidence Value : 0.75
C : cheese
E : juice
F : milk
CE -> F
Confidence Value : 1.0
C : cheese
E : juice
G : eggs
CE -> G
Confidence Value : 0.8333333333333334
C : cheese
E : juice
F : milk
G : eggs
CE -> FG
Confidence Value : 0.8333333333333334

```
C : cheese
F : milk
E : juice
G : eggs
CFE -> G
Confidence Value : 0.8333333333333334



C : cheese
F : milk
G : eggs
E : juice
CFG -> E
Confidence Value : 0.8333333333333334



F : milk
E : juice
G : eggs
C : cheese
FEG -> C
Confidence Value : 0.8333333333333334



--- The Brute Force algorithm now executed takes: 7.550689935684204 seconds ---
```

Comparison of both Algorithms

The Time difference between Apriori and Brute-Force is calculated below:

1. **Amazon**

   Apriori :- 7.6 seconds

   Brute-Force :- 9.8 seconds

   The difference is = 2.2 seconds

2. **Best Buy**

   Apriori :- 7.2 seconds

   Brute-Force :- 10.2 seconds

   The difference is = 3.0 seconds

3. **K Mart**

   Apriori :- 7.1 seconds

   Brute-Force :- 8.0 seconds

   The difference is = 0.9 seconds

4. **Nike**

   Apriori :- 7.8 seconds

   Brute-Force :- 9.2 seconds

   The difference is = 1.4 seconds

5. **Custom**

   Apriori :- 6.9 seconds

   Brute-Force :- 7.5 seconds

   The difference is = 0.6 seconds

From all these cases we can conclude that Apriori algorithm is faster than brute force approach.