DATA MINING
MIDTERM PROJECT

Full Name: Joyce Chettiar
NJIT UCID: jpc68
Email: jpc68@njit.edu

Programming language: Python

How to run the file: python project.py datasetname.csv minimum_support(in decimal) minimum_confidence(in decimal)

Dataset:

transaction1.csv

```
shrimp,eggs,salmon,mango,chicken
chicken,eggs,mango
turkey,shrimp,mango,chicken
turkey,salmon,eggs
eggs,mango,salmon
chicken,salmon
turkey,chicken,mango,eggs
shirmp,salmon, eggs, turkey,mango
eggs,shrimp
shrimp,chicken,turkey
shrimp,mango
turkey,chicken,salmon,mango
salmon,chicken,eggs
chicken,eggs,turkey,shrimp,mango
shrimp,turkey,mango,eggs
eggs,chicken
shrimp,chicken,salmon
turkey,eggs,mango
chicken,salmon,turkey,eggs
chicken,shrimp
```

transaction2.csv

```
spaghetti,meatballs,lemon,fries,cookies
fries,meatballs,lemon
burger,fries,cookies
cookies,lemon,fries, meatballs
cookies,fries,meatballs
spaghetti,fries,burger,cookies,lemon
meatballs,fries, cookies
cookies,fries,spaghetti
spaghetti,lemon,cookies,meatballs
cookies,meatballs
spaghetti,meatballs,fries
spaghetti,burger,lemon
fries,cookies,burger
burger,fries
cookies,meatballs
fries,meatballs,cookies,lemon,
burger,cookies, meatballs,fries
cookies,spaghetti
fries,burger,cookies
cookies,lemon,spaghetti
```

transaction3.csv

```
apple,potato,banana,avocado,strawberry
banana,strawberry,avocado,apple,yams
yams,apple,avocado,
apple,avocado,yams,strawberry
yams,potato,banana
yams,banana,strawberry
avocado,banana,strawberry,apple
strawberry,potato,banana,apple
yams,apple,banana
banana,strawberry,banana,yams
avocado,strawberry,yams,
avocado,yams,strawberry,apple
apple,yams,banana,avocado
strawberry,yams,potato
banana,yams,strawberry,potato
yamss,apple,banana,strawberry
strawberry,potato,banana,potato
avocado,banana,yams,apple
banana,yams,strawberry
yams,banana,apple,avocado
```

transaction4.csv

```
cake,sausages,tea,oil
oil,tea,bread,sausages,cake
wine ,bread,cake,oil
oil,wine,bread,tea,cake,sausages
cake,tea,oil,wine
tea,bread,sausages,oil,cake
wine,bread,oil,sausages,cake
cake,bread,tea,wine
oil,tea,pasta,sausages
bread,cake,tea,wine
tea,cake,bread,wine,sausages
wine,oil,cake,sausages
wine,cake,oil
cake,oil,bread,sausages
oil,bread,sausages
bread,sausages,cake
wine,oil,cake,sausages,
cake,oil,tea,sausages,
tea,oil,sausages,cake,bread
oil,sausages,bread,wine,tea
```

transaction5.csv

```
chocolate,pasta,honey,water,tomato,cheese
cheese,honey,water,chocolate
water,honey,cheese,tomato,chocolate
water,honey,pasta,cheese,tomato
honey,tomato,cheese
chocolate,cheese,pasta,water,tomato
pasta,cheese,water,chocolate,honey
pasta,water,honey,cheese
water,honey,cheese
pasta,chocolate,water
honey,chocolate,tomato
water,cheese,pasta,honey
cheese,tomato,pasta
chocolate,pasta,honey
chocolate,cheese,honey,tomato
cheese,chocolate,honey
pasta,water,cheese,tomato,chocolate,honey
honey,cheese,chocolate,tomato
water,honey,pasta
honey,water,tomato,pasta
```

all_items.csv

```
shrimp,eggs,salmon,turkey,mango,chicken,burger,meatballs,cookies,spaghetti,fries,lemon,apple,banana,yams,strawberry,potato,chocolate,pasta,water,cheese,honey,tomato,bread,wine,oil,tea,cake,sausages
```

This dataset contains all the 30 items.

CODE:

```
import sys
import time


filename=sys.argv[1]                    #The first argument is the filename
min_support=float(sys.argv[2])  #The second argument is the minimum support
min_confidence =float(sys.argv[3])#The third argument is the minimum
confidence



with open("all_items.csv") as f:
    Brute_items = f.read().replace("\n", "").split(",")
    Brute_items.sort()


filedata=open(filename,"r")
```

```
data=filedata.readlines()
data=[l.replace("\n","").split(",") for l in data]
print("-------------------------------------------------------")
print("              INPUT DATA")
print("-------------------------------------------------------")
for t in data:
    print(t)


class Brute:

    def __init__(self,Brute_left,Brute_right,Brute_all):
        self.Brute_left = list(Brute_left)
        self.Brute_left.sort()
        self.Brute_right = list(Brute_right)
        self.Brute_right.sort()
        self.Brute_all =Brute_all

    def __str__(self):
        return ",".join(self.Brute_left)+" => "+",".join(self.Brute_right)


def Brute_generate_k(items, k):

    if k == 1:
        return [[x] for x in items]

    all_res = []
    for i in range(len(items)-(k-1)):
        for sub in Brute_generate_k(items[i+1:], k-1):
            tmp = [items[i]]
            tmp.extend(sub)
            all_res.append(tmp)
    return all_res


def Brute_scan(db, s):
    count = 0
    for t in db:
        if set(s).issubset(t):
            count += 1
    return count


print("-------------------------------------------------------")
print("              Start of Brute force")
print("-------------------------------------------------------")
B_start_time = time.time()
B_frequent = []
B_support = {}

for k in range(1, len(Brute_items)+1):
    B_current = []
    for comb in Brute_generate_k(Brute_items, k):
        count = Brute_scan(data, comb)
        if count/len(data) >= min_support:
```

```
            B_support[frozenset(comb)] = count/len(data)
            B_current.append(comb)
    if len(B_current) == 0:
        break
    B_frequent.append(B_current)

all_rule = set()
Brute_all_result = []
for k_freq in B_frequent:
    if len(k_freq) == 0:
        continue
    if len(k_freq[0]) < 2:
        continue
    for freq in k_freq:
        for i in range(1, len(freq)):
            for left in Brute_generate_k(freq, i):
                tmp = freq.copy()
                right = [x for x in tmp if x not in left]
                all_rule.add(Brute(left, right, freq))
for rule in all_rule:
    Brute_confidence = B_support[frozenset(rule.Brute_all)] /
B_support[frozenset(rule.Brute_left)]
    if Brute_confidence >= min_confidence:
        Brute_all_result.append([rule, B_support[frozenset(rule.Brute_all)],
Brute_confidence])

Brute_all_result.sort(key=lambda x: str(x[0]))
B_end_time = time.time()
print("-----------------------------------------------------")
print("                RULES SUPPORT CONFIDENCE:")
print("-----------------------------------------------------")
for r in Brute_all_result:
    print(r[0], r[1], r[2])
print("-----------------------------------------------------")
print("                RUNNING TIME FOR BRUTE FORCE")
print("-----------------------------------------------------")
print(str(B_end_time - B_start_time) + "s")



class Apriori:
    def __init__(self,Apriori_left,apriori_right,apriori_all):
        self.Apriori_left=list(Apriori_left)
        self.Apriori_left.sort()
        self.apriori_right=list(apriori_right)
        self.apriori_right.sort()
        self.apriori_all=apriori_all

    def __str__(self):
        return ",".join(self.Apriori_left)+" =>
"+",".join(self.apriori_right)


def Apriori_generating_sub_rule(fs,r,result,support):
    r_size=len(r[0])
    t_size=len(fs)
```

```
    if t_size-r_size>0:
        r=Apriori_generate_items(r)
        new_r=[]
        for i in r:
            l=fs-i
            if(len(l)==0):
                continue
            confidence=support[fs]/support[l]
            if(confidence>=min_confidence):
                result.append([Apriori(l,i,fs),support[fs],confidence])
                new_r.append(i)

        if(len(new_r)>1):
            Apriori_generating_sub_rule(fs,new_r,result,support)


def Apriori_generate_items(dk):
    res=[]
    for i in range(len(dk)):
        for j in range(i+1,len(dk)):
            l,r=dk[i],dk[j]
            ll,rr=list(l),list(r)
            ll.sort()
            rr.sort()
            if ll[:len(l)-1] == rr[:len(r)-1]:
                res.append(l | r)
    return res


def Apriori_scan(data,f1):
    count = {s:0 for s in f1}
    for i in data:
        for freqset in f1:
            if(freqset.issubset(i)):
                count[freqset]+=1
    n=len(data)
    return{freqset: support/n for freqset, support in count.items() if
support/n>=min_support}



#Start for Apriori alogorithm
print("-------------------------------------------------------")
print("                Start of Apriori")
print("-------------------------------------------------------")
start_time = time.time()
support={}
item=[[]]
dk=[[]]
f1=set()
for i in data:
    for items in i:
        f1.add(frozenset([items]))
item.append(f1)
count=Apriori_scan(data,f1)
dk.append(list(count.keys()))
```

```
support.update(count)

t=1
while len(dk[t]) > 0:
    item.append(Apriori_generate_items(dk[t]))
    count=Apriori_scan(data,item[t+1])
    support.update(count)
    dk.append(list(count.keys()))
    t+=1
#generation of the rules

result=[]

for i in range(2,len(dk)):
    if(len(dk[i])==0):
        break
    frequent_set=dk[i]

    for fs in frequent_set:
        for r in [frozenset([x]) for x in fs]:
            l=fs-r
            confidence=support[fs]/support[l]
            if confidence>=min_confidence:
                result.append([Apriori(l,r,fs),support[fs],confidence])
    if(len(frequent_set[0])!=2):
        for fs in frequent_set:
            r=[frozenset([x]) for x in fs]
            Apriori_generating_sub_rule(fs,r,result,support)

result.sort(key=lambda x: str(x[0]))
end_time=time.time()

for k in result:
    print(k[0],k[1],k[2])
print("-----------------------------------------------------")
print("                  RUNNING TIME FOR APRIORI")
print("-----------------------------------------------------")
print(str(end_time - start_time) + "s")
```

For Dataset 1
INPUT:

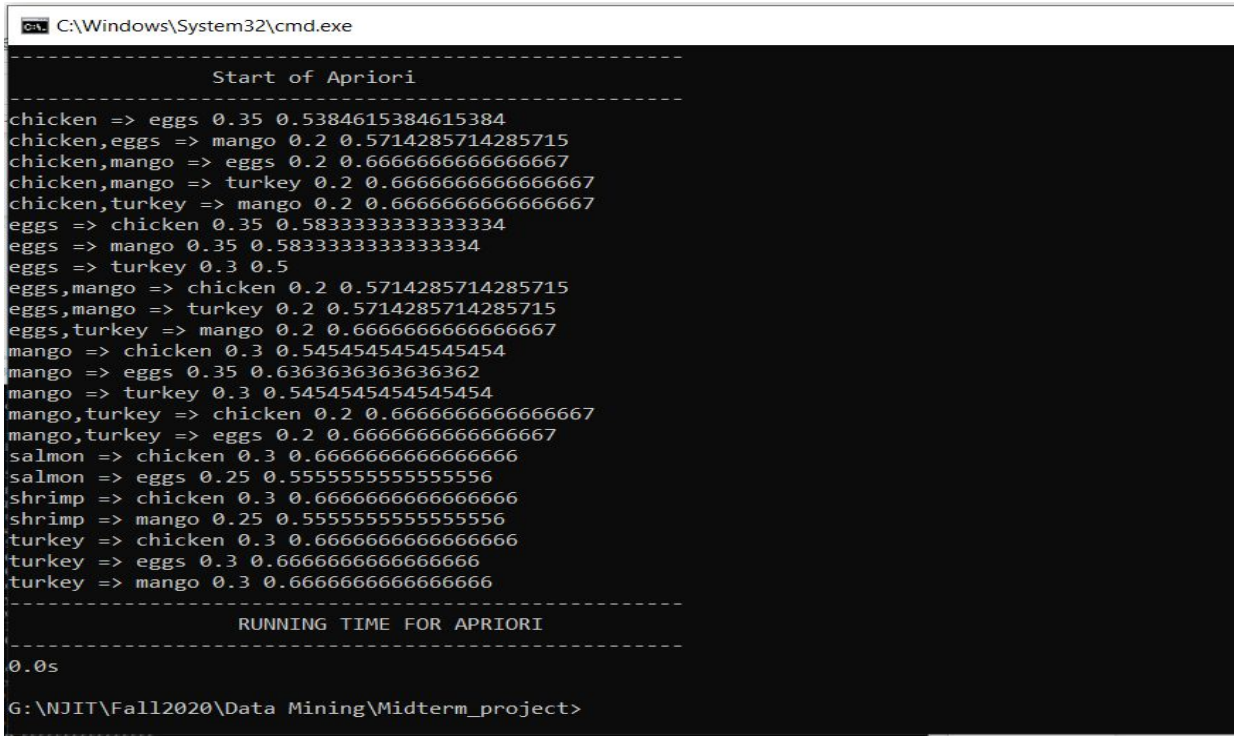
```
C:\Windows\System32\cmd.exe

G:\NJIT\Fall2020\Data Mining\Midterm_project>python project.py transaction1.csv 0.2 0.5
----------------------------------------------------
            INPUT DATA
----------------------------------------------------
['shrimp', 'eggs', 'salmon', 'mango', 'chicken']
['chicken', 'eggs', 'mango']
['turkey', 'shrimp', 'mango', 'chicken']
['turkey', 'salmon', 'eggs']
['eggs', 'mango', 'salmon']
['chicken', 'salmon']
['turkey', 'chicken', 'mango', 'eggs']
['shirmp', 'salmon', ' eggs', ' turkey', 'mango']
['eggs', 'shrimp']
['shrimp', 'chicken', 'turkey']
['shrimp', 'mango']
['turkey', 'chicken', 'salmon', 'mango']
['salmon', 'chicken', 'eggs']
['chicken', 'eggs', 'turkey', 'shrimp', 'mango']
['shrimp', 'turkey', 'mango', 'eggs']
['eggs', 'chicken']
['shrimp', 'chicken', 'salmon']
['turkey', 'eggs', 'mango']
['chicken', 'salmon', 'turkey', 'eggs']
['chicken', 'shrimp']
----------------------------------------------------
```

OUTPUT:
Brute Force Method:

```
C:\Windows\System32\cmd.exe
----------------------------------------------------
            Start of Brute force
----------------------------------------------------
----------------------------------------------------
               RULES SUPPORT CONFIDENCE:
----------------------------------------------------
chicken => eggs 0.35 0.5384615384615384
chicken,eggs => mango 0.2 0.5714285714285715
chicken,mango => eggs 0.2 0.6666666666666667
chicken,mango => turkey 0.2 0.6666666666666667
chicken,turkey => mango 0.2 0.6666666666666667
eggs => chicken 0.35 0.5833333333333334
eggs => mango 0.35 0.5833333333333334
eggs => turkey 0.3 0.5
eggs,mango => chicken 0.2 0.5714285714285715
eggs,mango => turkey 0.2 0.5714285714285715
eggs,turkey => mango 0.2 0.6666666666666667
mango => chicken 0.3 0.5454545454545454
mango => eggs 0.35 0.6363636363636362
mango => turkey 0.3 0.5454545454545454
mango,turkey => chicken 0.2 0.6666666666666667
mango,turkey => eggs 0.2 0.6666666666666667
salmon => chicken 0.3 0.6666666666666666
salmon => eggs 0.25 0.5555555555555556
turkey => chicken 0.3 0.6666666666666666
turkey => eggs 0.3 0.6666666666666666
turkey => mango 0.3 0.6666666666666666
----------------------------------------------------
               RUNNING TIME FOR BRUTE FORCE
----------------------------------------------------
0.21046876907348633s
----------------------------------------------------
```

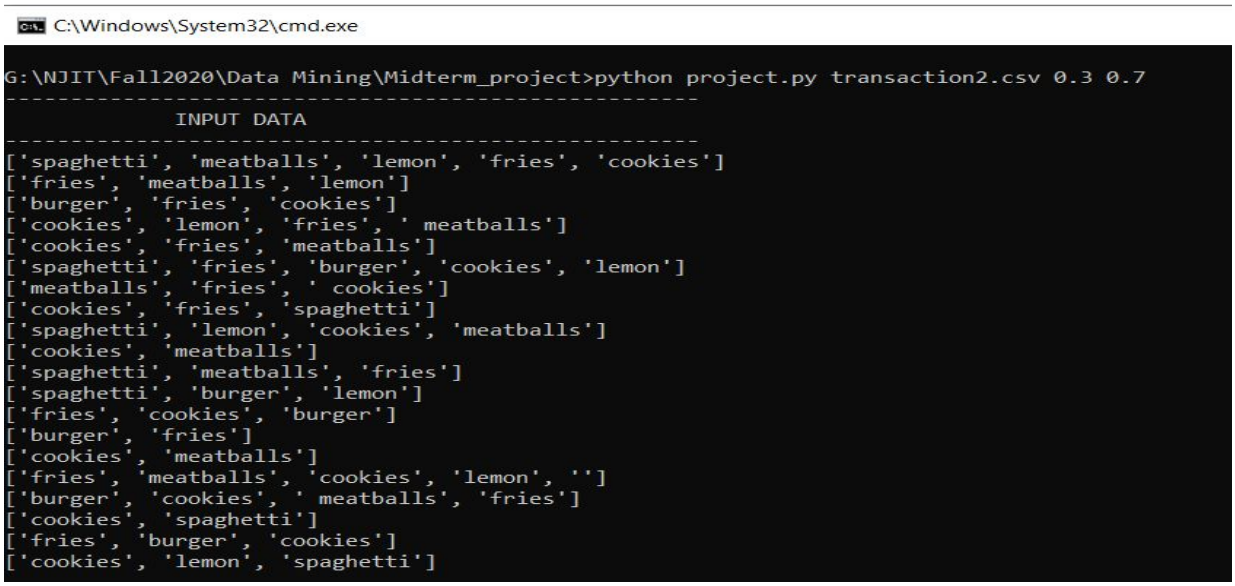
Apriori Method:



Brute Force takes approximately 0.2 seconds longer than Apriori to execute.

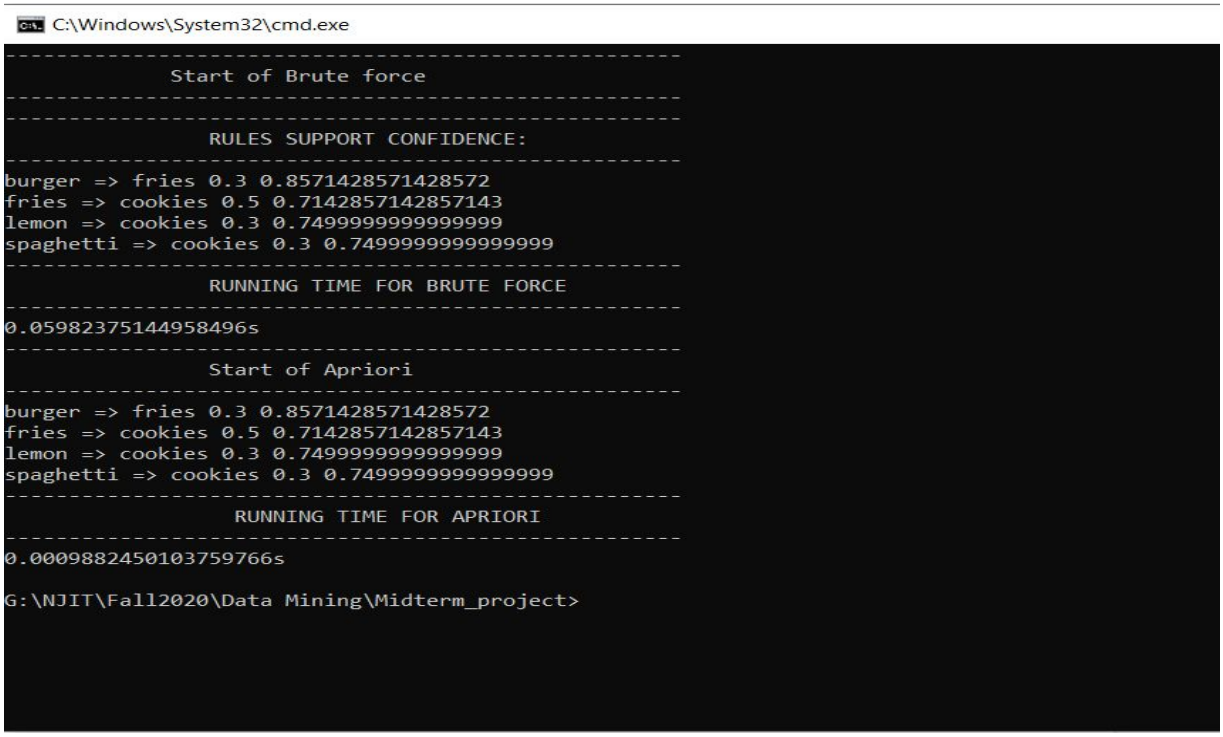
For Dataset 2

INPUT:



OUTPUT:

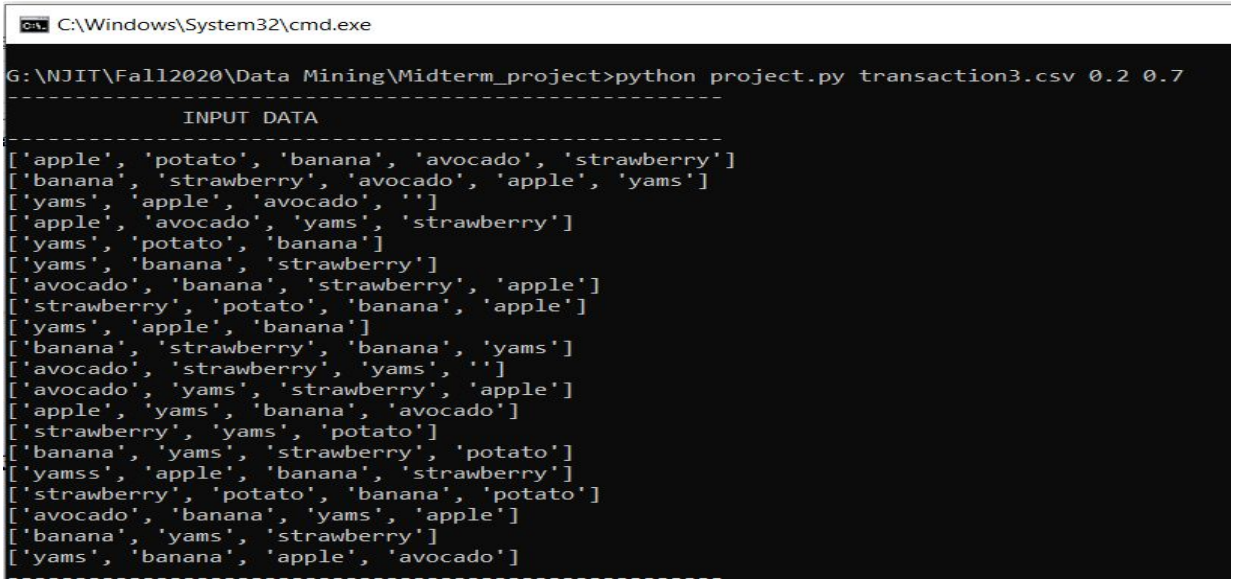
Brute Force Method and Apriori Method:



Brute Force takes approximately 0.05 seconds longer than Apriori to execute.

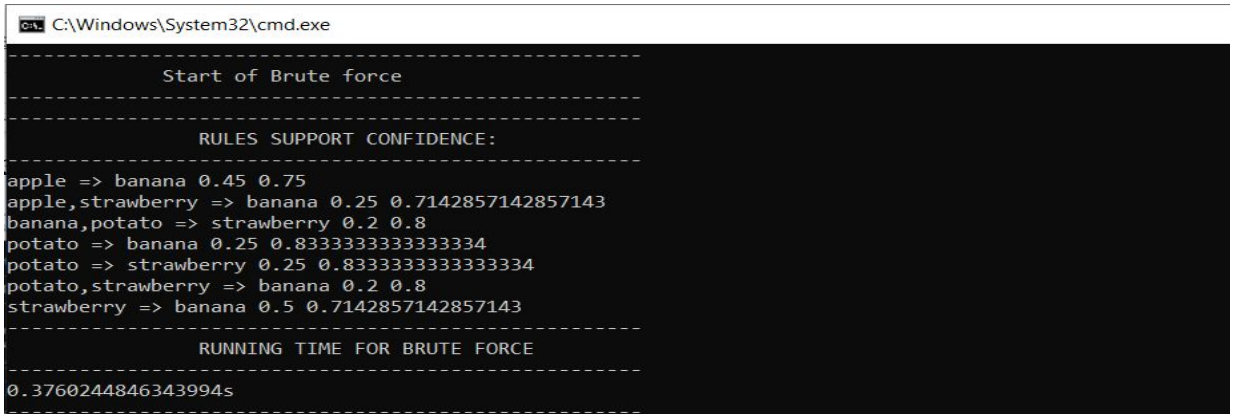
For Dataset 3

INPUT:

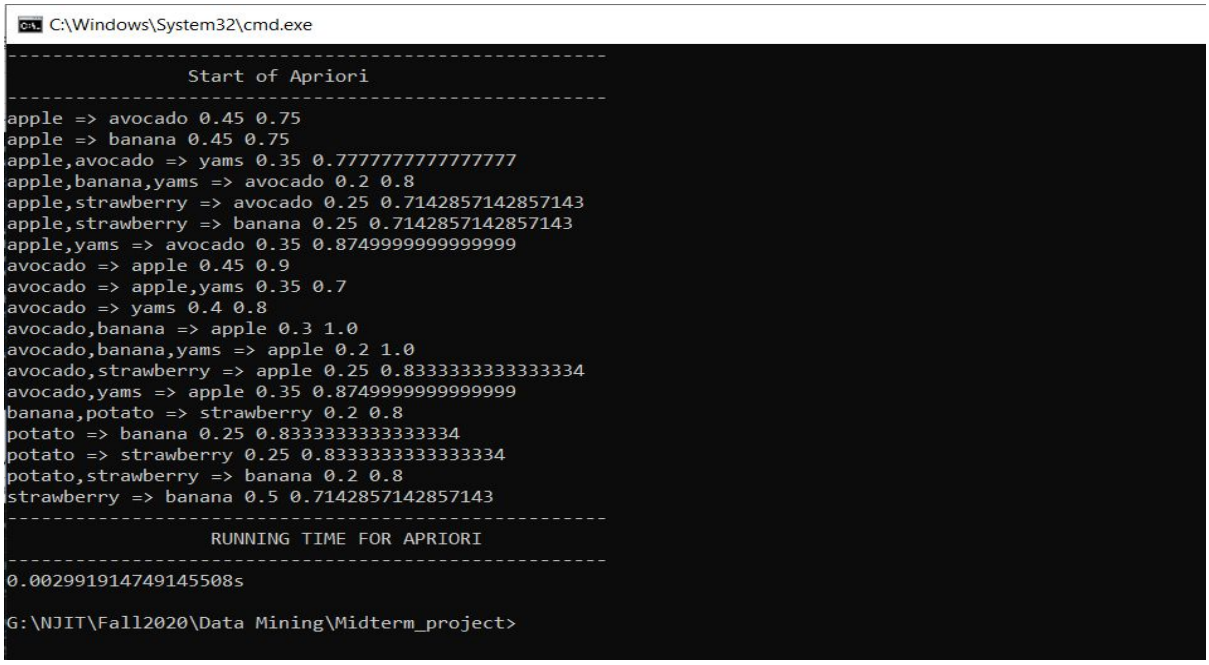


OUTPUT:

Brute Force Method:



Apriori Method:



Brute Force takes approximately 0.3 seconds longer than Apriori to execute.

For Dataset 4
INPUT:

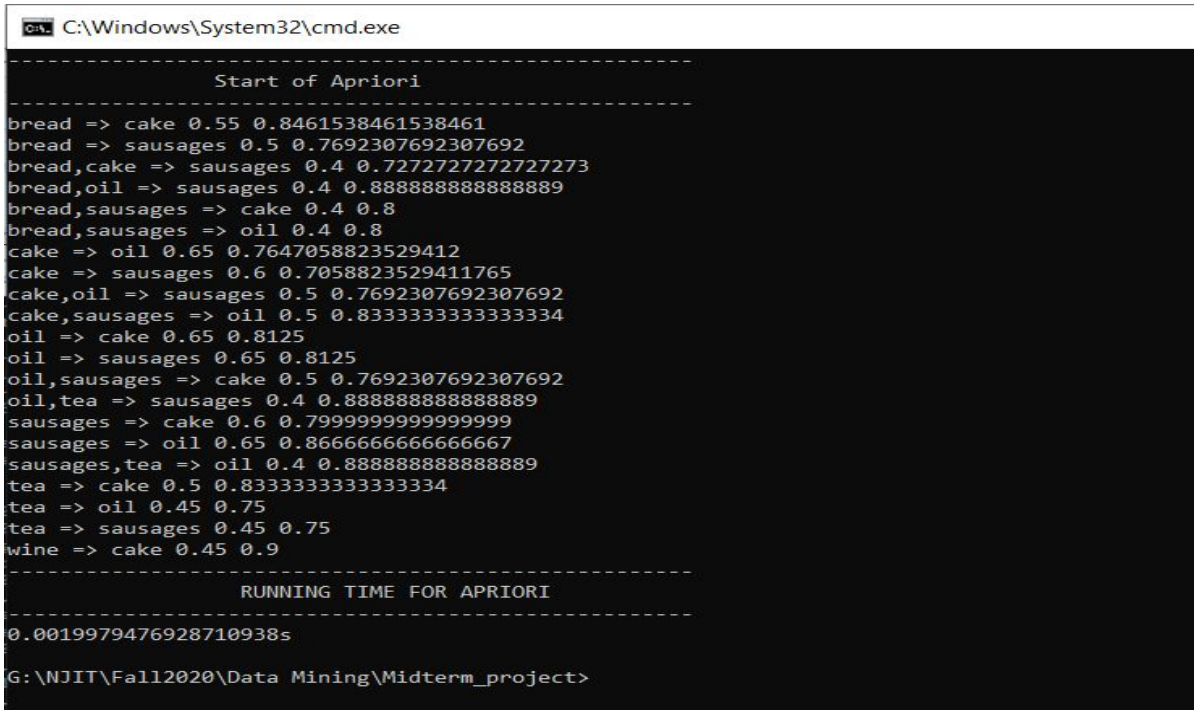
```
C:\Windows\System32\cmd.exe

G:\NJIT\Fall2020\Data Mining\Midterm_project>python project.py transaction4.csv 0.4 0.7
-----------------------------------------------------
             INPUT DATA
-----------------------------------------------------
['cake', 'sausages', 'tea', 'oil']
['oil', 'tea', 'bread', 'sausages', 'cake']
['wine ', 'bread', 'cake', 'oil']
['oil', 'wine', 'bread', 'tea', 'cake', 'sausages']
['cake', 'tea', 'oil', 'wine']
['tea', 'bread', 'sausages', 'oil', 'cake']
['wine', 'bread', 'oil', 'sausages', 'cake']
['cake', 'bread', 'tea', 'wine']
['oil', 'tea', 'pasta', 'sausages']
['bread', 'cake', 'tea', 'wine']
['tea', 'cake', 'bread', 'wine', 'sausages']
['wine', 'oil', 'cake', 'sausages']
['wine', 'cake', 'oil']
['cake', 'oil', 'bread', 'sausages']
['oil', 'bread', 'sausages']
['bread', 'sausages', 'cake']
['wine', 'oil', 'cake', 'sausages', '']
['cake', 'oil', 'tea', 'sausages', '']
['tea', 'oil', 'sausages', 'cake', 'bread']
['oil', 'sausages', 'bread', 'wine', 'tea']
```

OUTPUT:
Brute Force Method:

```
C:\Windows\System32\cmd.exe
-----------------------------------------------------
             Start of Brute force
-----------------------------------------------------
-----------------------------------------------------
               RULES SUPPORT CONFIDENCE:
-----------------------------------------------------
bread => cake 0.55 0.8461538461538461
bread => sausages 0.5 0.7692307692307692
bread,cake => sausages 0.4 0.7272727272727273
bread,oil => sausages 0.4 0.888888888888889
bread,sausages => cake 0.4 0.8
bread,sausages => oil 0.4 0.8
cake => oil 0.65 0.7647058823529412
cake => sausages 0.6 0.7058823529411765
cake,oil => sausages 0.5 0.7692307692307692
cake,sausages => oil 0.5 0.8333333333333334
oil => cake 0.65 0.8125
oil => sausages 0.65 0.8125
oil,sausages => cake 0.5 0.7692307692307692
oil,tea => sausages 0.4 0.888888888888889
sausages => cake 0.6 0.7999999999999999
sausages => oil 0.65 0.8666666666666667
sausages,tea => oil 0.4 0.888888888888889
tea => cake 0.5 0.8333333333333334
tea => oil 0.45 0.75
tea => sausages 0.45 0.75
wine => cake 0.45 0.9
-----------------------------------------------------
               RUNNING TIME FOR BRUTE FORCE
-----------------------------------------------------
0.610377311706543s
-----------------------------------------------------
```

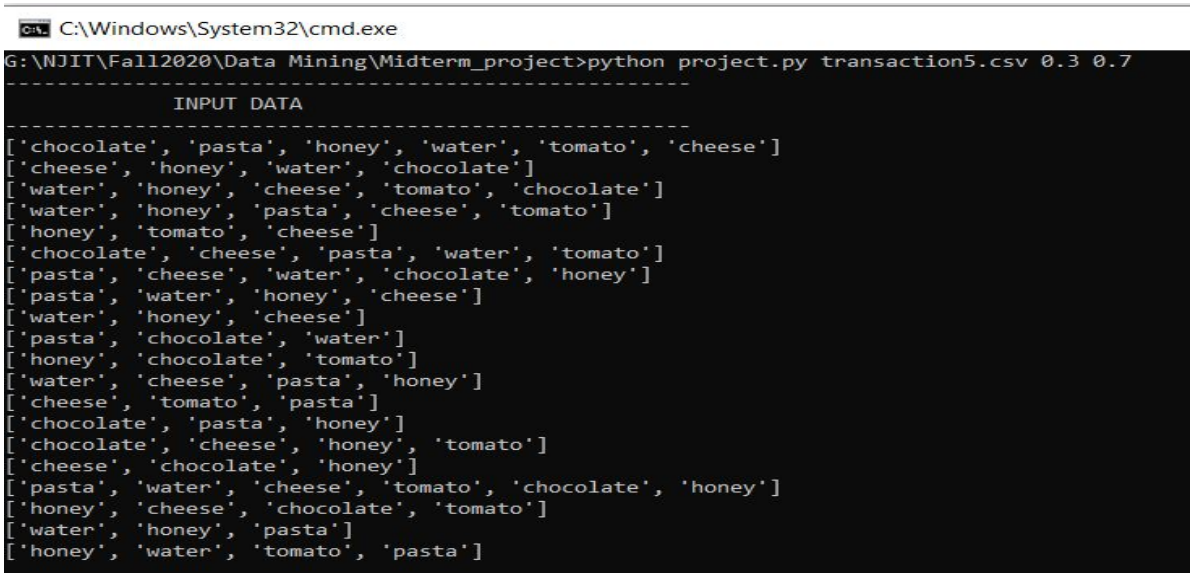
Apriori Method:



Brute Force takes approximately 0.5 seconds longer than Apriori to execute.

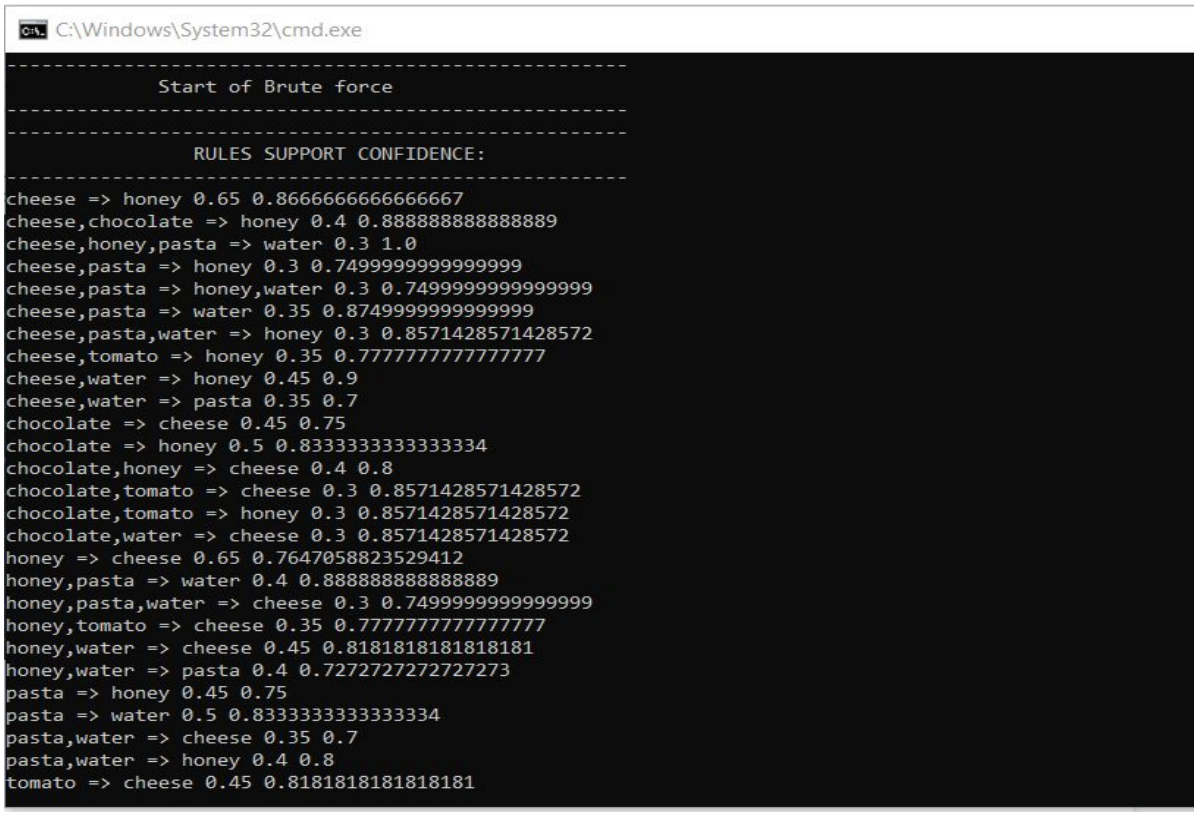
For Dataset 5

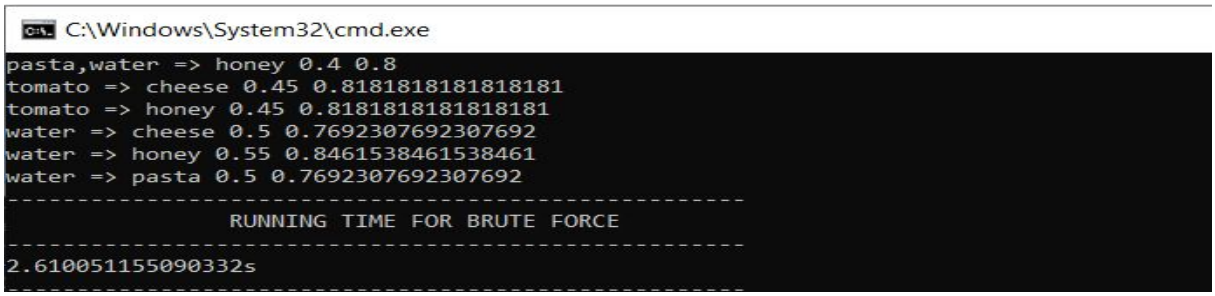
INPUT:



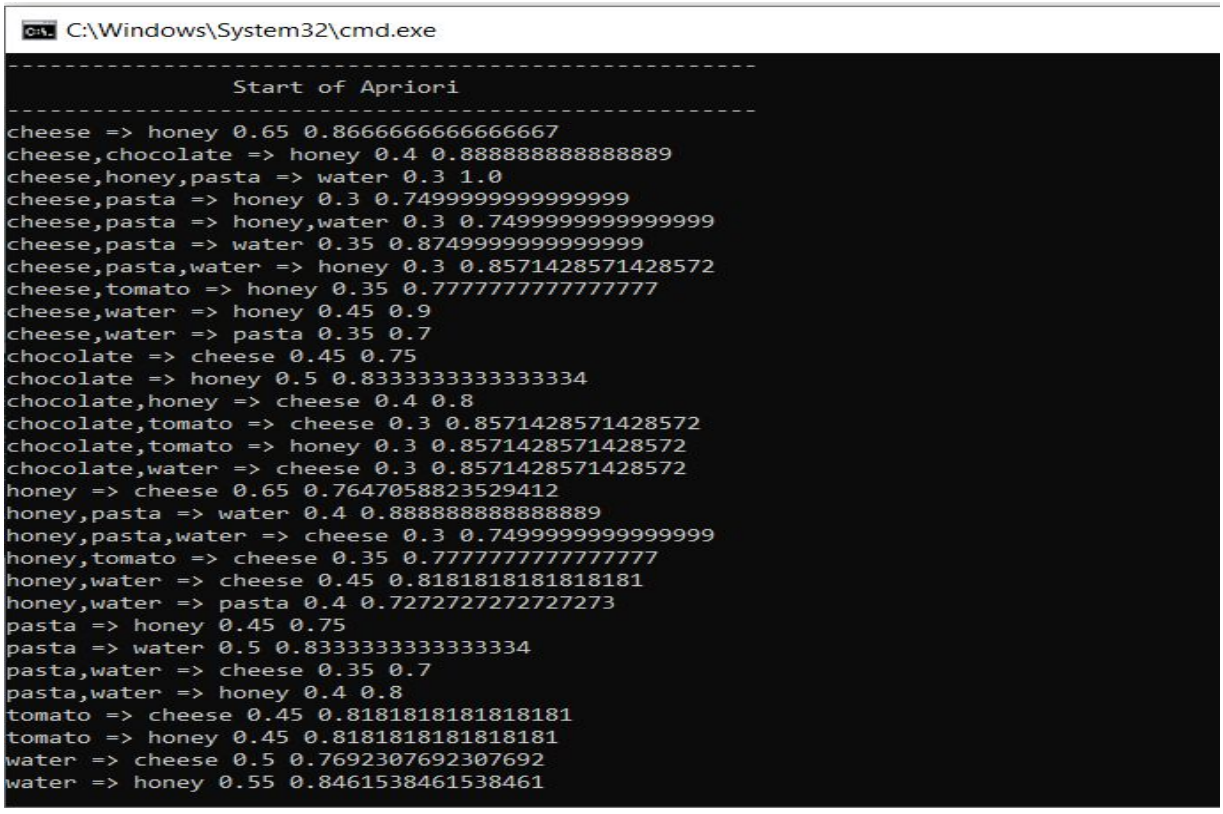
OUTPUT:

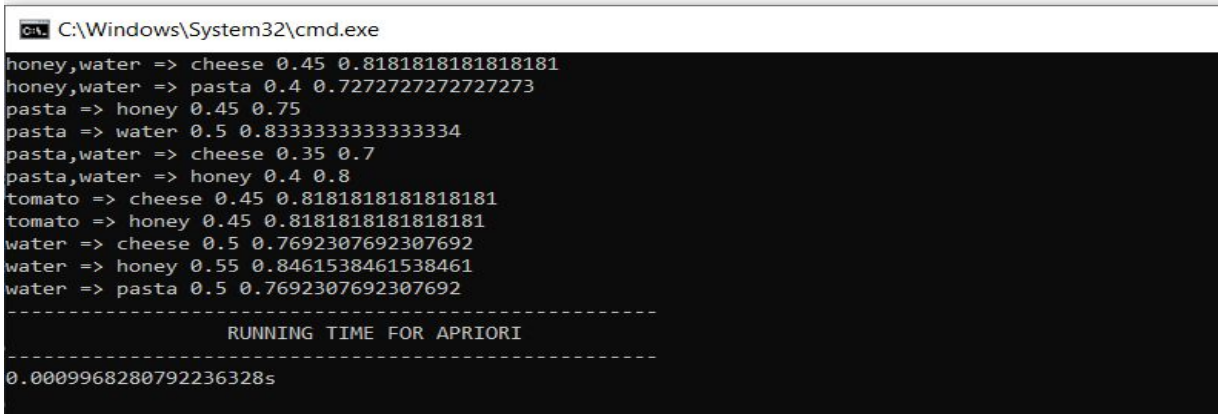
Brute Force Method





Apriori Method





Brute Force takes approximately 2 seconds longer than Apriori to execute.