DATA MINING
MIDTERM PROJECT

Full Name: Francis D'cruz
NJIT UCID: fjd7
Email: fjd7@njit.edu

Programming language: Python

data1.csv

```
bread,cereals,rice,pasta,noodles,wheat
bread,wheat,noodles,cereals,pasta
noodles,wheat
rice,bread,noodles,pasta
rice,pasta,cereals,bread
cereals,bread,rice,noodles,pasta
cereals
rice,pasta,wheat
pasta,wheat
bread,rice,pasta,noodles
bread
rice,pasta,cereals,wheat
pasta,bread,noodles
noodles,bread,rice,pasta,cereals
pasta,wheat,cereals
bread,rice,cereals,noodles
bread,cereals,rice,pasta
rice,bread,rice
cereals,bread,pasta,rice
rice,pasta
```

data2.csv

```
milk,yoghurt,cheese,butter,icecream,cream
milk,icecream
icecream,cream
cheese,milk,icecream,
cheese,butter,yoghurt,milk
yoghurt,milk ,cheese,icecream,butter
yoghurt
cheese,butter
butter,cream
milk,cheese,butter,icecream
milk
cheese,butter
cream,butter,milk
icecream,milk,cheese,butter,yoghurt
butter
cream,milk,cheese,yoghurt
milk,yoghurt
cream,cheese,milk,cheese
yoghurt,milk,butter,cheese
cheese,cream
```

data3.csv

```
coconut,banana,apple,cherry,mango,papaya
banana,cherry,mango,banana
coconut,papaya
coconut,apple,banana,papaya
banana,cherry,mango
mango,papaya,apple
cherry,mango,cherry,banana
cherry,coconut
papaya,mango,banana
apple,coconut,cherry
cherry,apple,papaya,coconut,apple
mango,coconut
apple,bananax`
banana,apple,banana,papaya
banana,mango,coconut,cherry,apple
apple
banana,coconut,banana
papaya,mango,apple,banana
coconut,banana,mango
mango,coconut,papaya
```

data4.csv

```
shirt,pant,coat,socks,tie,belt
coat,belt,socks,socks
socks,belt
socks,pant
coat,pant,shirt,tie,socks
coat,belt,socks
socks,belt,tie
coat,shirt
belt,shirt,pant,tie
socks,tie
coat,belt
shirt,tie,pant
socks,coat,pant,shirt,tie
tie
coat,tie,pant,shirt
pant,belt,coat
pant,tie,shirt
pant,belt
shirt,coat,pant
coat,tie,belt
```

data5.csv

```
coke,pepsi,fanta,redbull,sprite,limca
limca,pepsi,fanta,sprite
fanta,sprite,coke
limca,pepsi,redbull
redbull,redbull,fanta,sprite
fanta,coke
limca,redbull,coke
pepsi,redbull,sprite,coke,fanta
limca,fanta
fanta,coke,sprite
coke
pepsi,coke,limca,redbull,sprite
sprite
fanta,pepsi,redbull
limca,coke,pepsi,sprite
pepsi,sprite,fanta
sprite,limca
redbull
pepsi,coke,fanta,limca
coke,redbull,fanta,sprite
```

all_items.csv(All the items contained in 5 dataset)

```
Bread,cereals,rice,pasta,noodles,milk,yoghurt,cheese,butter,icecream,
Cream,coconut,banana,apple,cherry,mango,shirt,pant,coat,socks,tie,coke,
pepsi,fanta,redbull,sprite,wheat,papaya,belt
```

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

```
import sys
import time

filename=sys.argv[1]                    # First Argument- Filename
min_sup=float(sys.argv[2])              # Second Argument-Minimum Support(in decimal)
min_conf =float(sys.argv[3])     #Third Argument-Minimum Confidence(in decimal)

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

#Reading data from the file name and printing the data to the users
filedata=open(filename,"r")
data=filedata.readlines()
data=[l.replace("\n","").split(",") for l in data]
print("--------------------INPUT DATA-----------------")
for d in data:
    print(d)

#Class for Apriori algorithm
class A_Rule:
    def __init__(self,A_left,A_right,A_all):
```

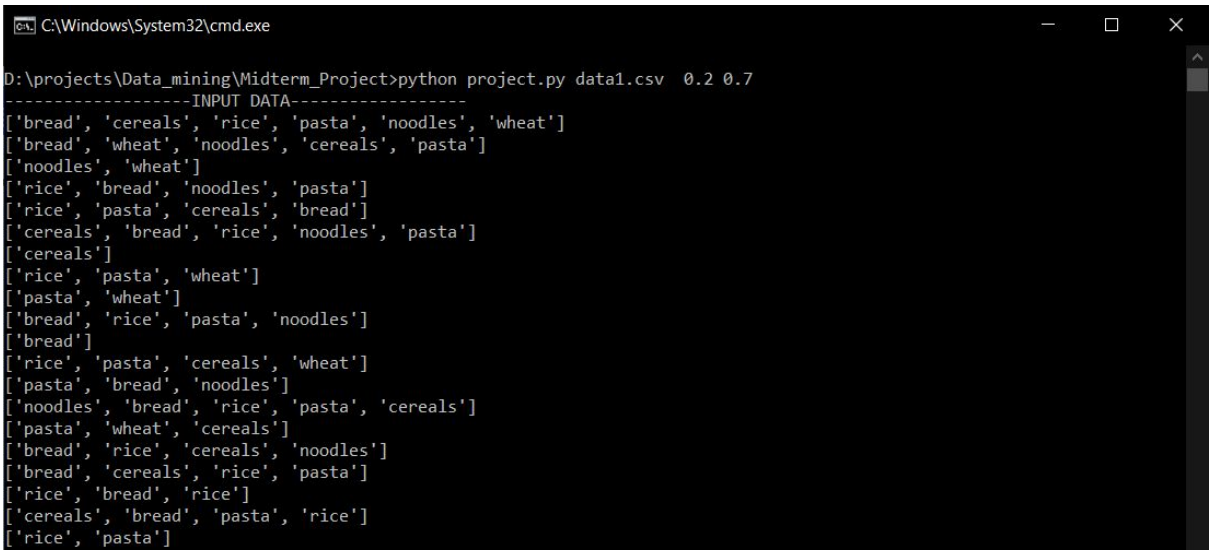
```
        self.A_left=list(A_left)
        self.A_left.sort()
        self.A_right=list(A_right)
        self.A_right.sort()
        self.A_all=A_all

    def __str__(self):
        return ",".join(self.A_left)+" => "+",".join(self.A_right)


#Class for Brute Force algorithm
class B_Rule:

    def __init__(self,B_left,B_right,B_all):
        self.B_left = list(B_left)
        self.B_left.sort()
        self.B_right = list(B_right)
        self.B_right.sort()
        self.B_all = B_all

    def __str__(self):
        return ",".join(self.B_left)+ " => "+",".join(self.B_right)

#generating all possible Sub-combinations for a rule
def A_generating_sub_rule(fs,r,result,support):
    r_size=len(r[0])
    t_size=len(fs)
    if t_size-r_size>0:
        r=B_generate_itemset(r)
        print(r)
        new_r=[]
        for i in r:
            l=fs-i
            if(len(l)==0):
                continue
            conf=support[fs]/support[l]
            if(conf>=min_conf):
                result.append([A_Rule(l,i,fs),support[fs],conf])
                new_r.append(i)

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

#Generating Combinations for Itemset for Apriori
def B_generate_itemset(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


#Generating Combinations for Itemset for Brute Force
def B_generate(items, k):

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

    all_res = []
    for i in range(len(items)-(k-1)):
        for sub in B_generate(items[i+1:], k-1):
            tmp = [items[i]]
```

```
            tmp.extend(sub)
            all_res.append(tmp)
    return all_res


#Function used to scan the database to count frequency for Apriori
def A_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_sup}

#Function used to scan the database to count frequency for Brute Force
def B_scan(db,s):
    count = 0
    for t in db:
        if set(s).issubset(t):
            count += 1
    return count

##Start for Apriori algorithm
print("-------------------------start Apriori----------------------------")
A_start_time = time.time()
support={}
item=[[]]
dk=[[]]
f1=set() #creating a set to hold all the data for scanning the data from the
dictionary
for i in data:
    for items in i:
        f1.add(frozenset([items]))
item.append(f1)
count=A_scan(data,f1)
dk.append(list(count.keys()))
support.update(count)

t=1
while len(dk[t]) > 0:
    item.append(B_generate_itemset(dk[t]))
    count=A_scan(data,item[t+1])
    support.update(count)
    dk.append(list(count.keys()))
    t+=1


#generating the rules for Apriori Algorithm
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
            conf=support[fs]/support[l]
            if conf>=min_conf:
                result.append([A_Rule(l,r,fs),support[fs],conf])
    if(len(frequent_set[0])!=2):
        for fs in frequent_set:
            r=[frozenset([x]) for x in fs]
            A_generating_sub_rule(fs,r,result,support)
```

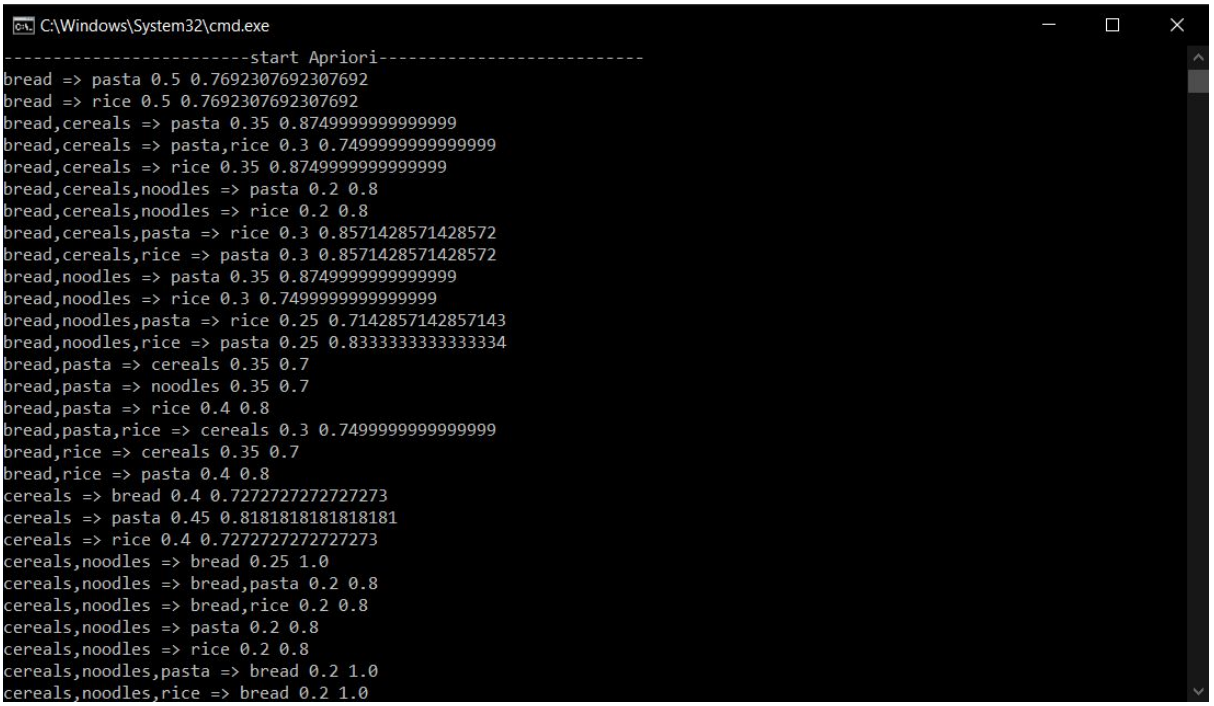
```
result.sort(key=lambda x: str(x[0]))
A_end_time=time.time()
for k in result:
    print(k[0],k[1],k[2])
A_time=A_end_time - A_start_time


print("-------------------------start Brute force--------------------------")
#start for brute force algorithm
B_start_time = time.time()
B_frequent = []
B_support = {}
for k in range(1, len(B_items)+1):
    B_current = []
    for comb in B_generate(B_items, k):
        count = B_scan(data, comb)
        if count/len(data) >= min_sup:
            B_support[frozenset(comb)] = count/len(data)
            B_current.append(comb)
    if len(B_current) == 0:
        break
    B_frequent.append(B_current)


#generating all rules for Brute Force
all_rule = set()
B_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 B_generate(freq, i):
                tmp = freq.copy()
                right = [x for x in tmp if x not in left]
                all_rule.add(B_Rule(left, right, freq))
for rule in all_rule:
    B_confidence = B_support[frozenset(rule.B_all)] /
B_support[frozenset(rule.B_left)]
    if B_confidence >= min_conf:
        B_all_result.append([rule, B_support[frozenset(rule.B_all)],
B_confidence])

B_all_result.sort(key=lambda x: str(x[0]))
B_end_time = time.time()

for r in B_all_result:
    print(r[0], r[1], r[2])
print("\n---------------------------------- RUNNING
TIME:----------------------------------")

#displaying the time calculated
B_time=B_end_time - B_start_time
print("Apriori took ",str(A_end_time - A_start_time) + "s")
print("Brute force took ",str(B_end_time - B_start_time) + "s")
print("Apriori Algorithm is ",str(B_time-A_time)," seconds faster than Brute
Force Algorithm")
```

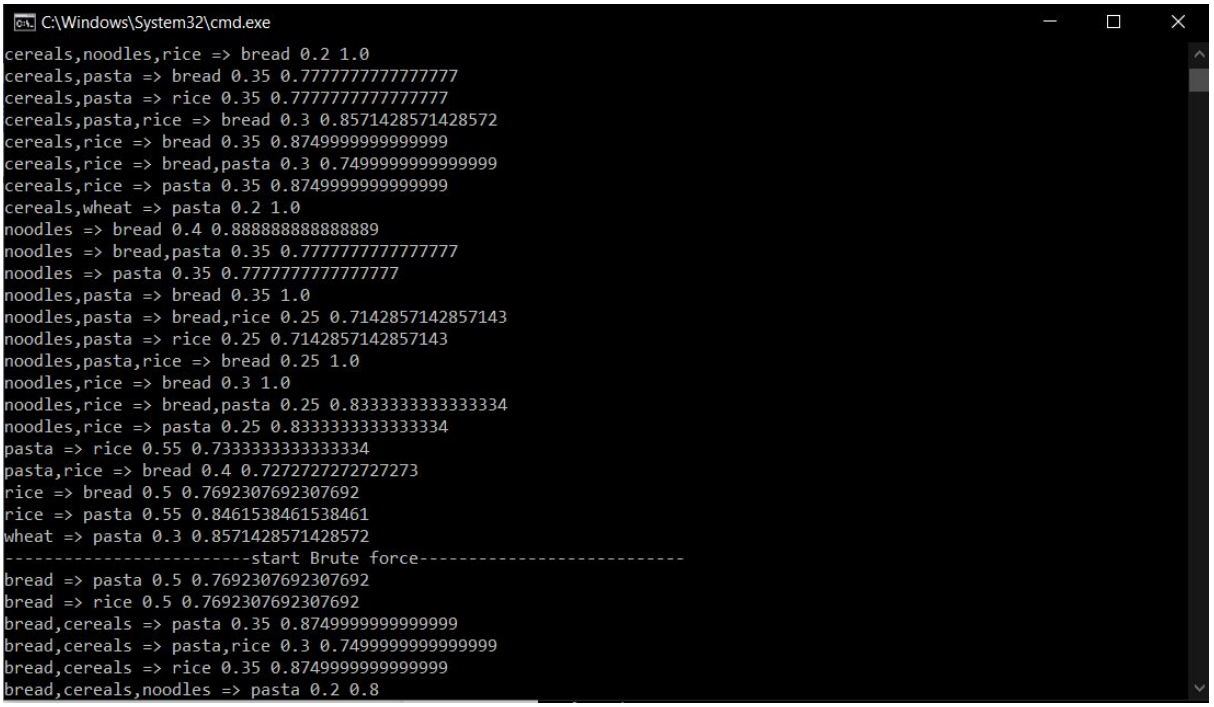
For Dataset 1

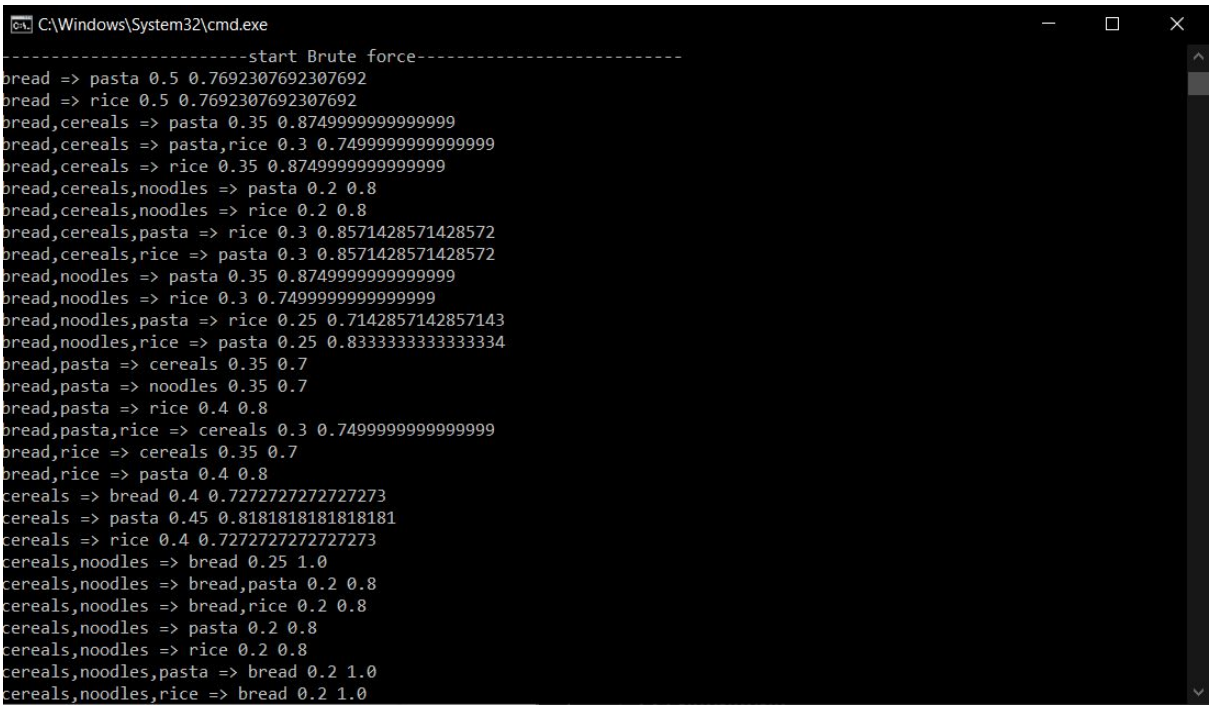
INPUT:

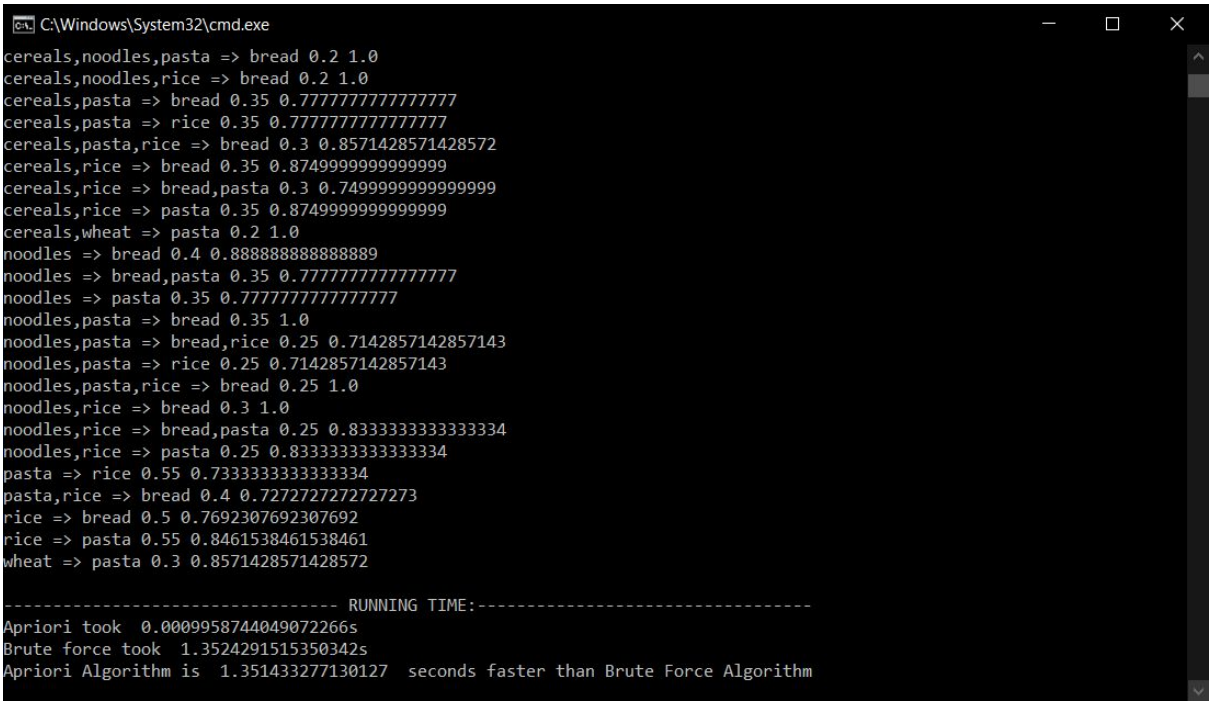


OUTPUT:



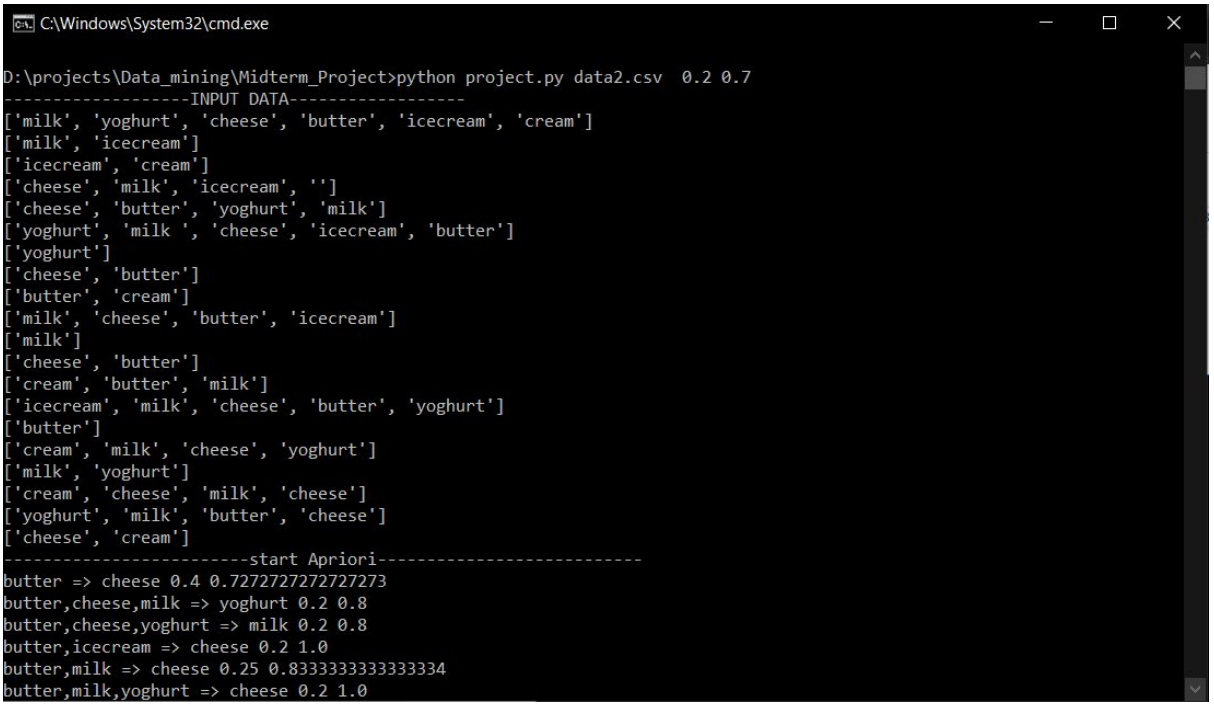

C:\Windows\System32\cmd.exe
cereals,noodles,rice => bread 0.2 1.0
cereals,pasta => bread 0.35 0.7777777777777777
cereals,pasta => rice 0.35 0.7777777777777777
cereals,pasta,rice => bread 0.3 0.8571428571428572
cereals,rice => bread 0.35 0.8749999999999999
cereals,rice => bread,pasta 0.3 0.7499999999999999
cereals,rice => pasta 0.35 0.8749999999999999
cereals,wheat => pasta 0.2 1.0
noodles => bread 0.4 0.888888888888889
noodles => bread,pasta 0.35 0.7777777777777777
noodles => pasta 0.35 0.7777777777777777
noodles,pasta => bread 0.35 1.0
noodles,pasta => bread,rice 0.25 0.7142857142857143
noodles,pasta => rice 0.25 0.7142857142857143
noodles,pasta,rice => bread 0.25 1.0
noodles,rice => bread 0.3 1.0
noodles,rice => bread,pasta 0.25 0.8333333333333334
noodles,rice => pasta 0.25 0.8333333333333334
pasta => rice 0.55 0.7333333333333334
pasta,rice => bread 0.4 0.7272727272727273
rice => bread 0.5 0.7692307692307692
rice => pasta 0.55 0.8461538461538461
wheat => pasta 0.3 0.8571428571428572
-------------------------start Brute force--------------------------
bread => pasta 0.5 0.7692307692307692
bread => rice 0.5 0.7692307692307692
bread,cereals => pasta 0.35 0.8749999999999999
bread,cereals => pasta,rice 0.3 0.7499999999999999
bread,cereals => rice 0.35 0.8749999999999999
bread,cereals,noodles => pasta 0.2 0.8


C:\Windows\System32\cmd.exe
-------------------------start Brute force--------------------------
bread => pasta 0.5 0.7692307692307692
bread => rice 0.5 0.7692307692307692
bread,cereals => pasta 0.35 0.8749999999999999
bread,cereals => pasta,rice 0.3 0.7499999999999999
bread,cereals => rice 0.35 0.8749999999999999
bread,cereals,noodles => pasta 0.2 0.8
bread,cereals,noodles => rice 0.2 0.8
bread,cereals,pasta => rice 0.3 0.8571428571428572
bread,cereals,rice => pasta 0.3 0.8571428571428572
bread,noodles => pasta 0.35 0.8749999999999999
bread,noodles => rice 0.3 0.7499999999999999
bread,noodles,pasta => rice 0.25 0.7142857142857143
bread,noodles,rice => pasta 0.25 0.8333333333333334
bread,pasta => cereals 0.35 0.7
bread,pasta => noodles 0.35 0.7
bread,pasta => rice 0.4 0.8
bread,pasta,rice => cereals 0.3 0.7499999999999999
bread,rice => cereals 0.35 0.7
bread,rice => pasta 0.4 0.8
cereals => bread 0.4 0.7272727272727273
cereals => pasta 0.45 0.8181818181818181
cereals => rice 0.4 0.7272727272727273
cereals,noodles => bread 0.25 1.0
cereals,noodles => bread,pasta 0.2 0.8
cereals,noodles => bread,rice 0.2 0.8
cereals,noodles => pasta 0.2 0.8
cereals,noodles => rice 0.2 0.8
cereals,noodles,pasta => bread 0.2 1.0
cereals,noodles,rice => bread 0.2 1.0

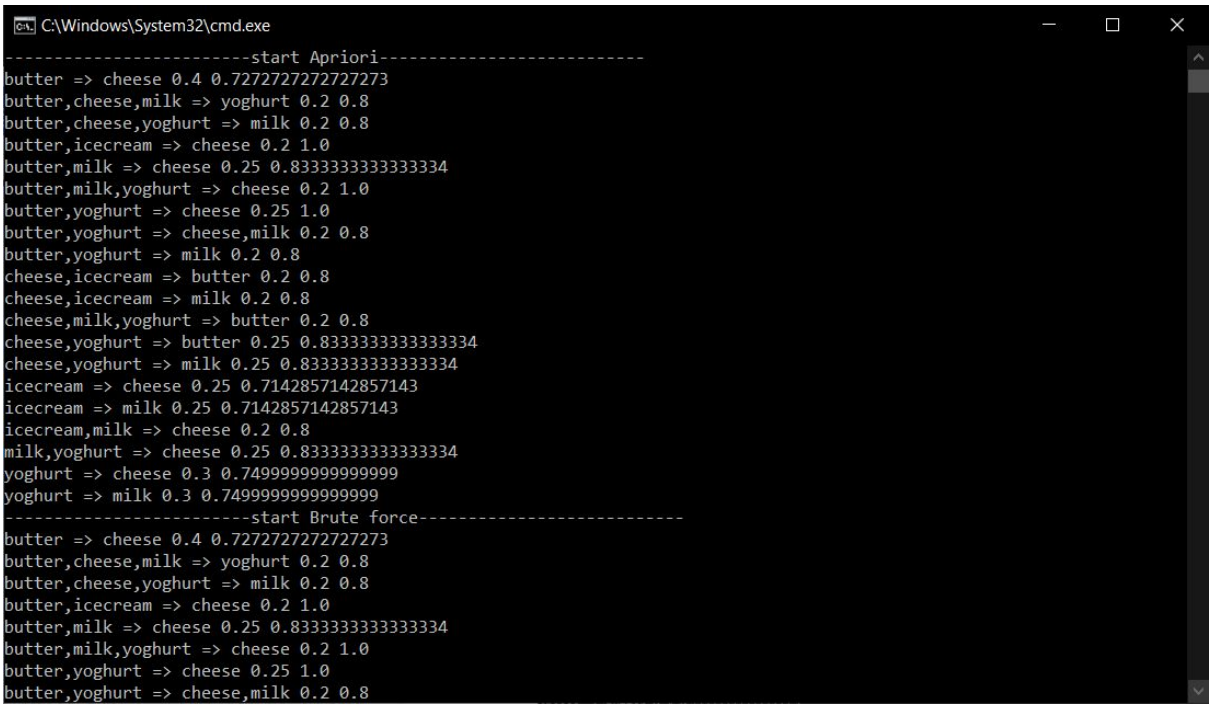


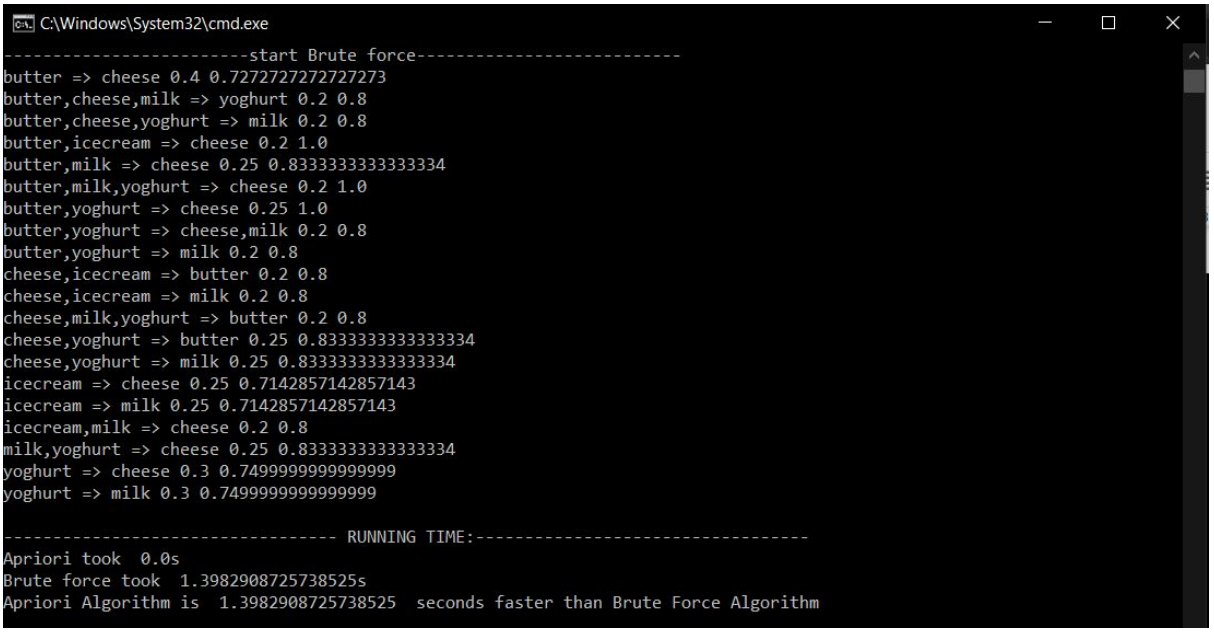
For Dataset 2

INPUT:



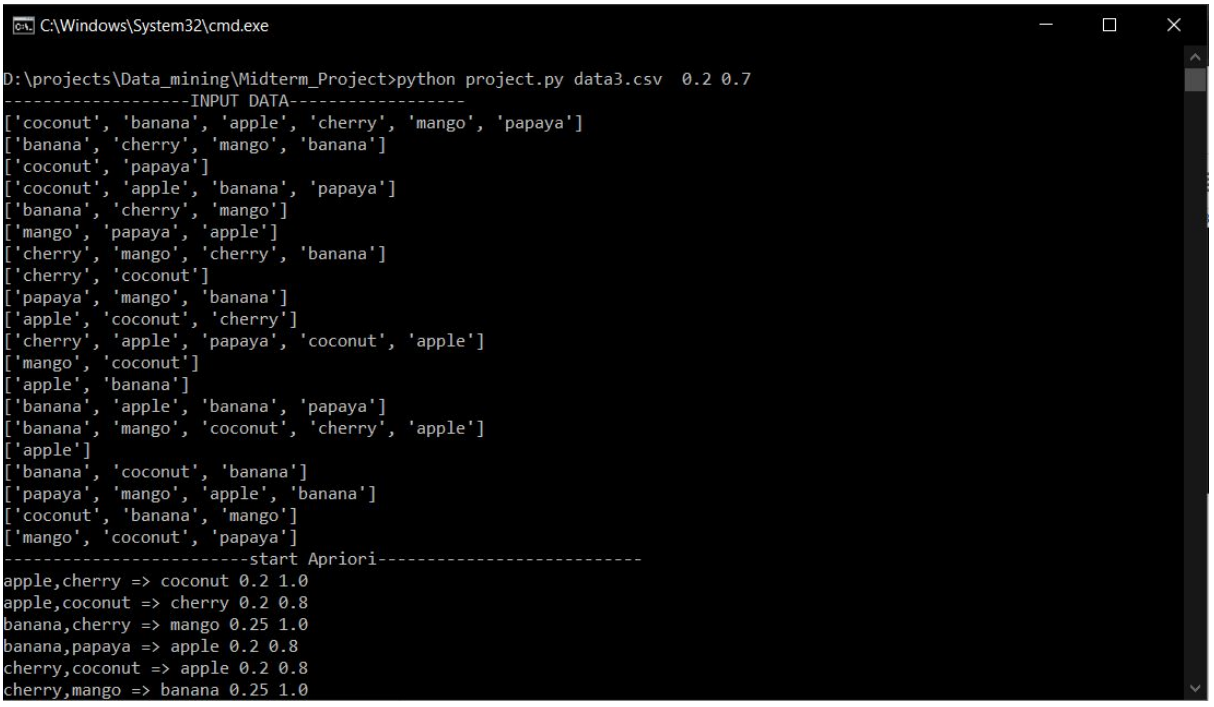
OUTPUT:


C:\Windows\System32\cmd.exe
----------start Apriori----------
butter => cheese 0.4 0.7272727272727273
butter,cheese,milk => yoghurt 0.2 0.8
butter,cheese,yoghurt => milk 0.2 0.8
butter,icecream => cheese 0.2 1.0
butter,milk => cheese 0.25 0.8333333333333334
butter,milk,yoghurt => cheese 0.2 1.0
butter,yoghurt => cheese 0.25 1.0
butter,yoghurt => cheese,milk 0.2 0.8
butter,yoghurt => milk 0.2 0.8
cheese,icecream => butter 0.2 0.8
cheese,icecream => milk 0.2 0.8
cheese,milk,yoghurt => butter 0.2 0.8
cheese,yoghurt => butter 0.25 0.8333333333333334
cheese,yoghurt => milk 0.25 0.8333333333333334
icecream => cheese 0.25 0.7142857142857143
icecream => milk 0.25 0.7142857142857143
icecream,milk => cheese 0.2 0.8
milk,yoghurt => cheese 0.25 0.8333333333333334
yoghurt => cheese 0.3 0.7499999999999999
yoghurt => milk 0.3 0.7499999999999999
----------start Brute force----------
butter => cheese 0.4 0.7272727272727273
butter,cheese,milk => yoghurt 0.2 0.8
butter,cheese,yoghurt => milk 0.2 0.8
butter,icecream => cheese 0.2 1.0
butter,milk => cheese 0.25 0.8333333333333334
butter,milk,yoghurt => cheese 0.2 1.0
butter,yoghurt => cheese 0.25 1.0
butter,yoghurt => cheese,milk 0.2 0.8

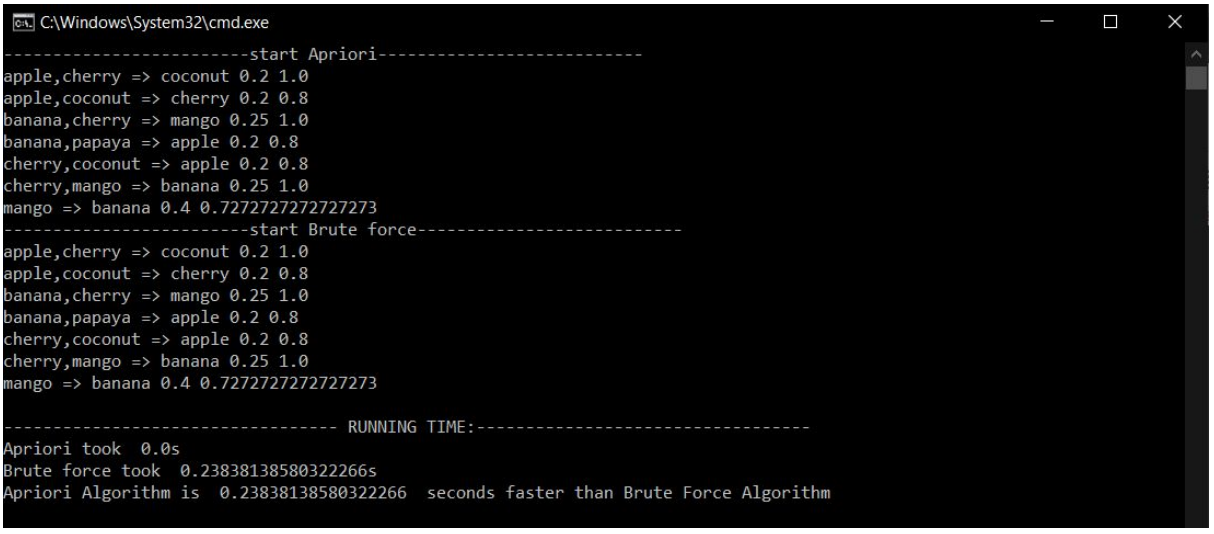

C:\Windows\System32\cmd.exe
----------start Brute force----------
butter => cheese 0.4 0.7272727272727273
butter,cheese,milk => yoghurt 0.2 0.8
butter,cheese,yoghurt => milk 0.2 0.8
butter,icecream => cheese 0.2 1.0
butter,milk => cheese 0.25 0.8333333333333334
butter,milk,yoghurt => cheese 0.2 1.0
butter,yoghurt => cheese 0.25 1.0
butter,yoghurt => cheese,milk 0.2 0.8
butter,yoghurt => milk 0.2 0.8
cheese,icecream => butter 0.2 0.8
cheese,icecream => milk 0.2 0.8
cheese,milk,yoghurt => butter 0.2 0.8
cheese,yoghurt => butter 0.25 0.8333333333333334
cheese,yoghurt => milk 0.25 0.8333333333333334
icecream => cheese 0.25 0.7142857142857143
icecream => milk 0.25 0.7142857142857143
icecream,milk => cheese 0.2 0.8
milk,yoghurt => cheese 0.25 0.8333333333333334
yoghurt => cheese 0.3 0.7499999999999999
yoghurt => milk 0.3 0.7499999999999999
---------- RUNNING TIME:----------
Apriori took 0.0s
Brute force took 1.3982908725738525s
Apriori Algorithm is 1.3982908725738525 seconds faster than Brute Force Algorithm

For Dataset 3

INPUT:

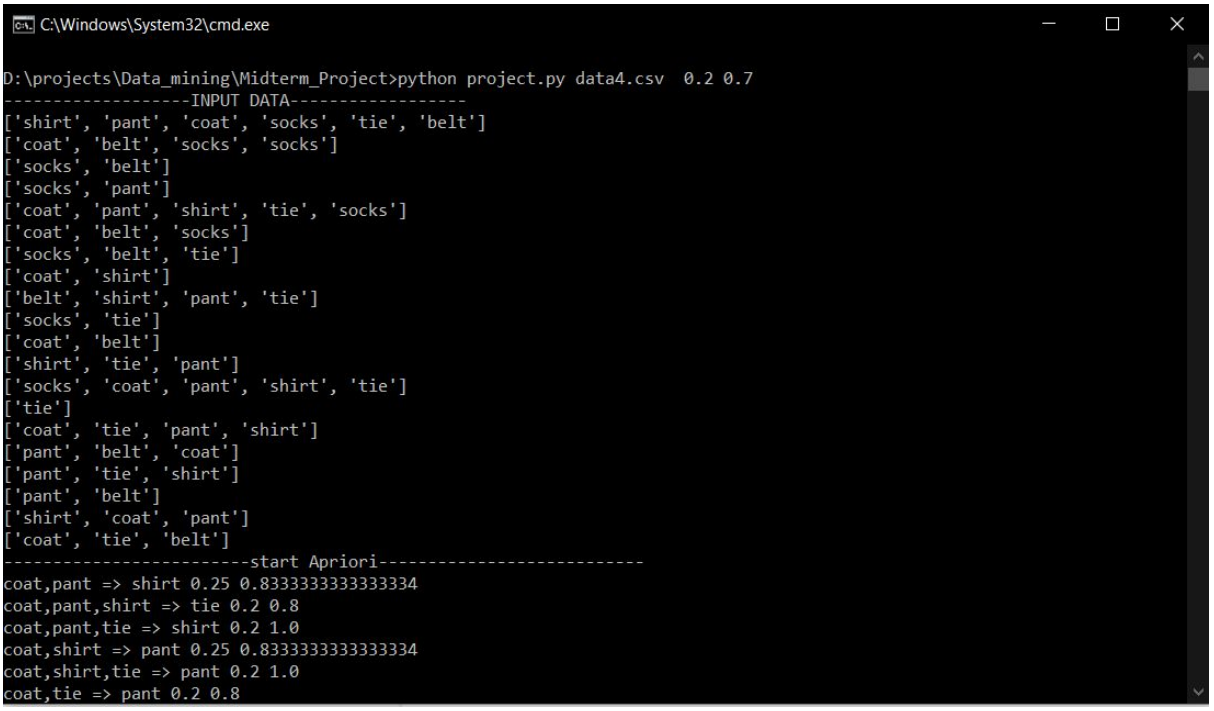


OUTPUT:

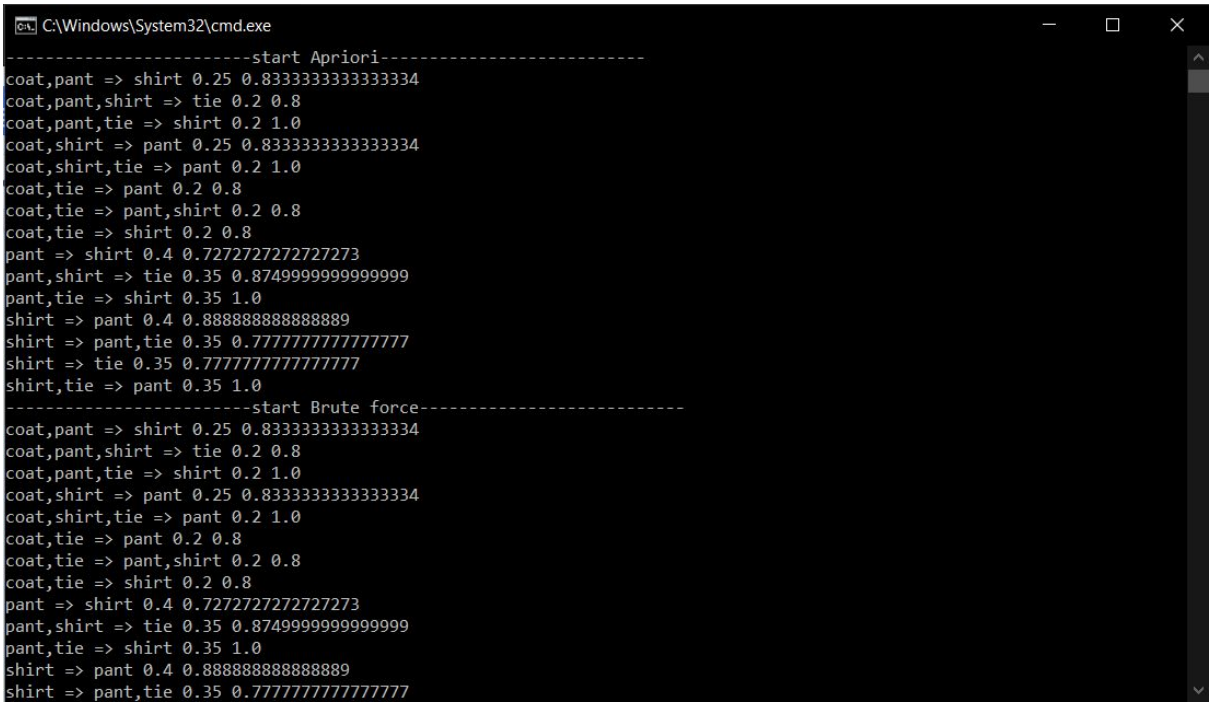


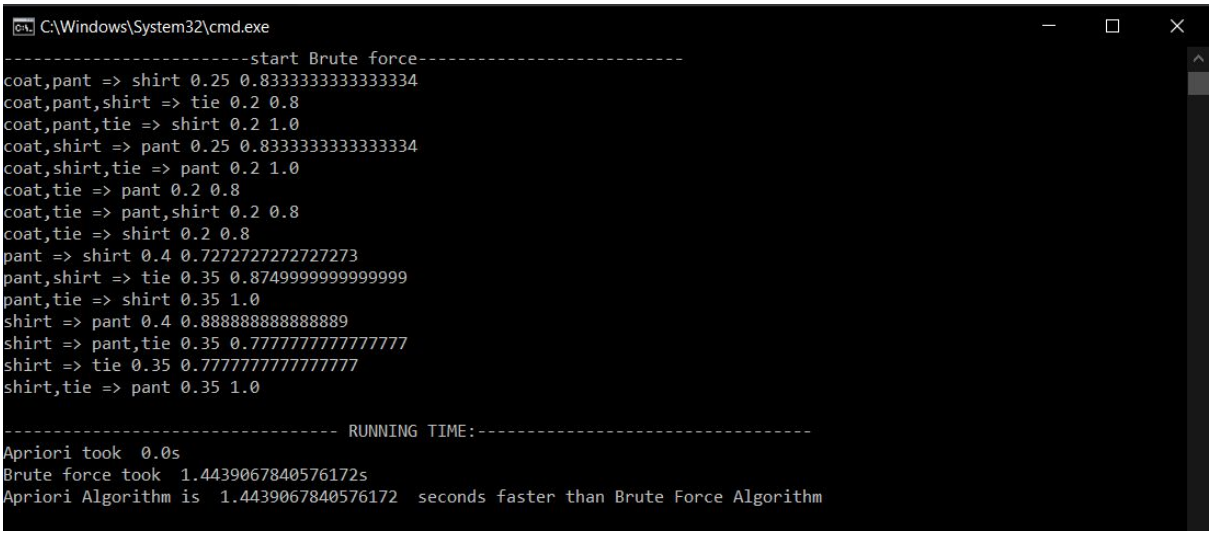
For Dataset 4

INPUT:



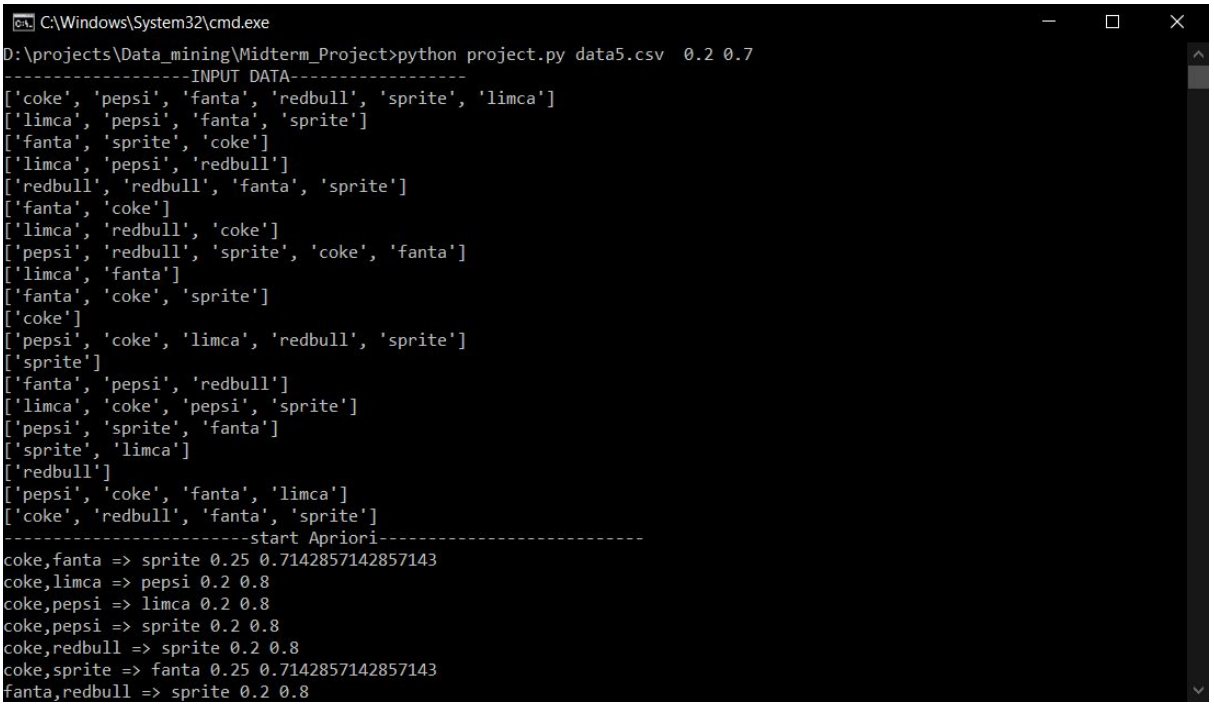
OUTPUT:



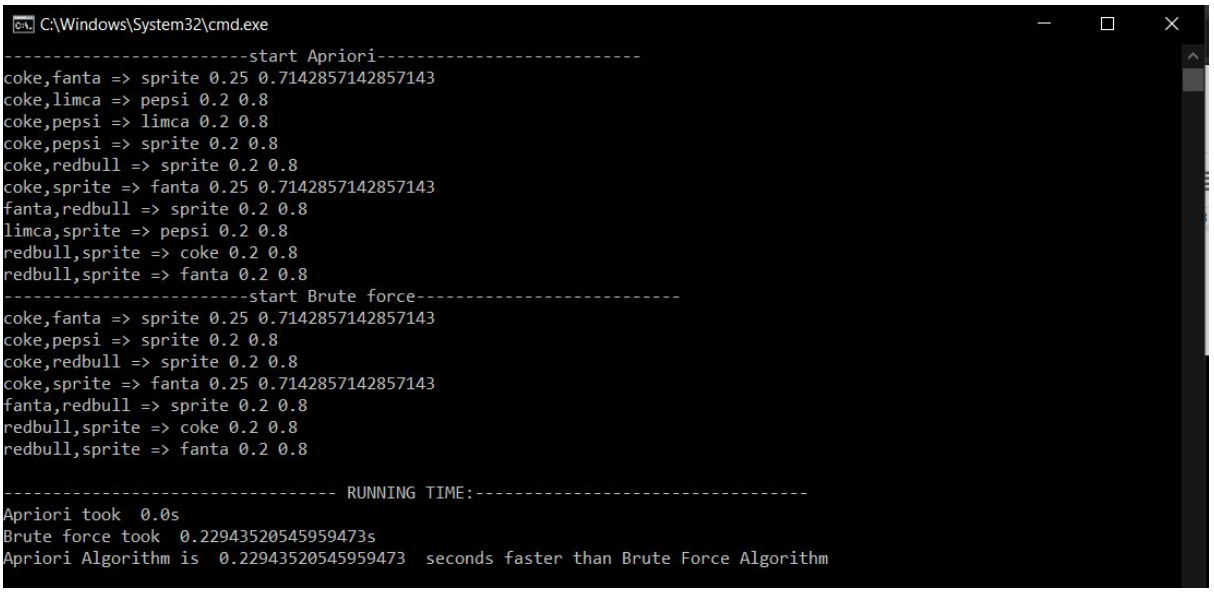


For Dataset 5:

INPUT:



OUTPUT:


C:\Windows\System32\cmd.exe
----------start Apriori----------
coke,fanta => sprite 0.25 0.7142857142857143
coke,limca => pepsi 0.2 0.8
coke,pepsi => limca 0.2 0.8
coke,pepsi => sprite 0.2 0.8
coke,redbull => sprite 0.2 0.8
coke,sprite => fanta 0.25 0.7142857142857143
fanta,redbull => sprite 0.2 0.8
limca,sprite => pepsi 0.2 0.8
redbull,sprite => coke 0.2 0.8
redbull,sprite => fanta 0.2 0.8
----------start Brute force----------
coke,fanta => sprite 0.25 0.7142857142857143
coke,pepsi => sprite 0.2 0.8
coke,redbull => sprite 0.2 0.8
coke,sprite => fanta 0.25 0.7142857142857143
fanta,redbull => sprite 0.2 0.8
redbull,sprite => coke 0.2 0.8
redbull,sprite => fanta 0.2 0.8
---------- RUNNING TIME:----------
Apriori took 0.0s
Brute force took 0.22943520545959473s
Apriori Algorithm is 0.22943520545959473 seconds faster than Brute Force Algorithm