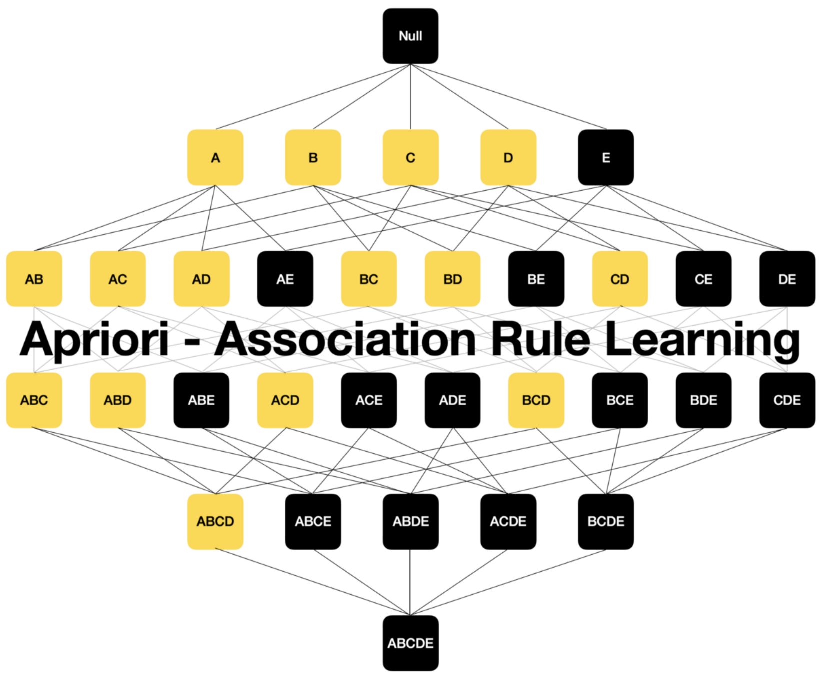
CS 634 -DATA MINING

APRIORI ALGORITHM
IMPLEMENTATION FROM SCRATCH IN PYTHON



GARIMA MATHUR

UCID - gm47

Email id - gm47@njit.edu

GitHub - github.com/Garima-Mathur

Introduction :

Apriori algorithm is used for finding frequent itemsets in a dataset for boolean association rule. Name of the algorithm is Apriori because it uses prior knowledge of frequent itemset properties. We apply an iterative approach or level-wise search where k-frequent itemsets are used to find k+1 itemsets.

Apriori Algorithm Implementation Steps :

Apriori algorithm has a sequence of steps to be followed to find the most frequent itemset in the dataset. The minimum support threshold is set by the user.

1) In the first iteration of the algorithm, each item is taken as a 1-itemsets candidate. The algorithm will count the occurrences of each item.

2) For the set minimum support value,The set of 1 – itemsets whose occurrence is satisfying the min sup are determined. Only those candidates which count more than or equal to min_sup, are taken ahead for the next iteration and the others are pruned or removed.

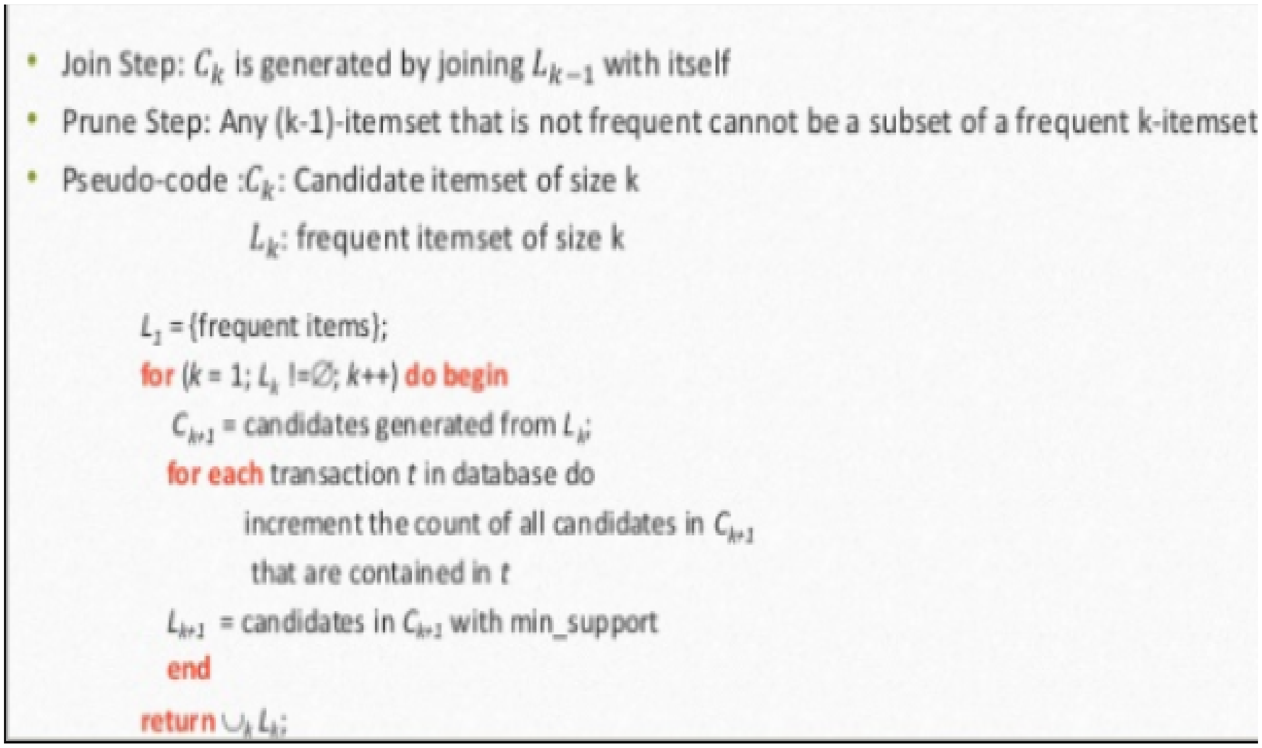
3) Next, 2-itemset frequent items with min_sup are discovered. The 2-itemset is generated by forming a group of 2 by combining items with itself.

4) The 2-itemset candidates are pruned using min-sup threshold value. Now the table will have 2 –itemsets with min-sup only.

5) The next iteration will form 3 –itemsets using the same steps. If all 2-itemset subsets are frequent then the superset will be frequent otherwise it is pruned.

6) Next step will follow making 4-itemset by joining 3-itemset with itself and pruning if its subset does not meet the min_sup criteria. The algorithm is stopped when the most frequent itemset is achieved.

Pseudocode for the Algorithm :



- Join Step: C_k is generated by joining L_{k-1} with itself
- Prune Step: Any (k-1)-itemset that is not frequent cannot be a subset of a frequent k-itemset
- Pseudo-code :C_k: Candidate itemset of size k

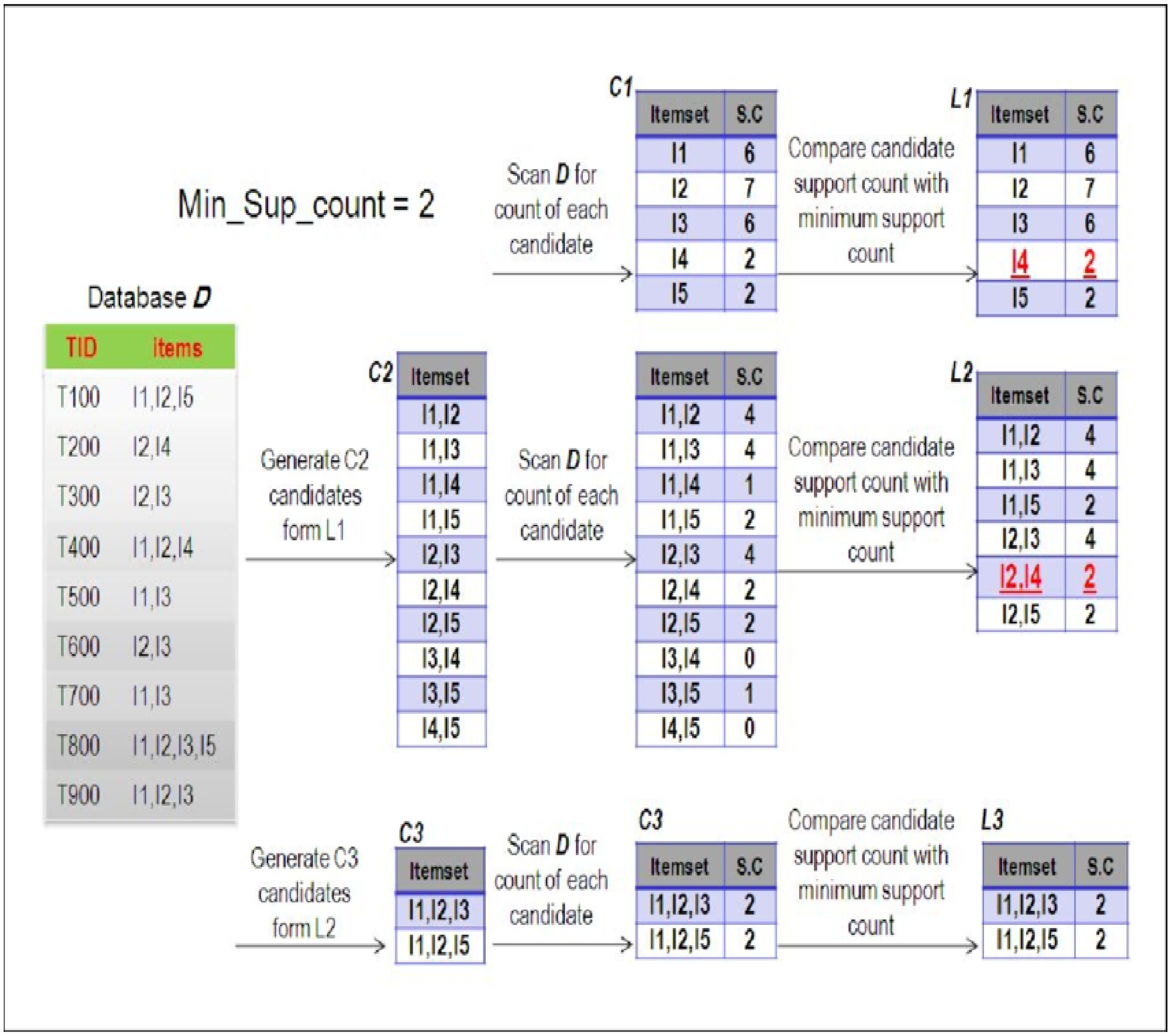
  L_k: frequent itemset of size k

```
$L_1$ = {frequent items};
for ($k$ = 1; $L_k$ !=∅; $k$++) do begin
    $C_{k+1}$ = candidates generated from $L_k$;
    for each transaction $t$ in database do
        increment the count of all candidates in $C_{k+1}$
        that are contained in $t$
    $L_{k+1}$ = candidates in $C_{k+1}$ with min_support
    end
return $\cup_k L_k$;
```

Join Step: This step generates (K+1) itemset from K-itemsets by joining each item with itself.

Prune Step: This step scans the count of each item in the database. If the candidate item does not meet minimum support, then it is regarded as infrequent and thus it is removed. This step is performed to reduce the size of the candidate itemsets.

An example of solving the Apriori algorithm is as follows :


Min_Sup_count = 2
Scan *D* for count of each candidate
Compare candidate support count with minimum support count
Generate C2 candidates form L1
Generate C3 candidates form L2
C1
L1
C2
L2
C3
L3

Database *D*

TID	items
T100	I1,I2,I5
T200	I2,I4
T300	I2,I3
T400	I1,I2,I4
T500	I1,I3
T600	I2,I3
T700	I1,I3
T800	I1,I2,I3,I5
T900	I1,I2,I3

C1

Itemset	S.C
I1	6
I2	7
I3	6
I4	2
I5	2

L1

Itemset	S.C
I1	6
I2	7
I3	6
I4	2
I5	2

C2

Itemset
I1,I2
I1,I3
I1,I4
I1,I5
I2,I3
I2,I4
I2,I5
I3,I4
I3,I5
I4,I5

Itemset	S.C
I1,I2	4
I1,I3	4
I1,I4	1
I1,I5	2
I2,I3	4
I2,I4	2
I2,I5	2
I3,I4	0
I3,I5	1
I4,I5	0

L2

Itemset	S.C
I1,I2	4
I1,I3	4
I1,I5	2
I2,I3	4
I2,I4	2
I2,I5	2

C3

Itemset
I1,I2,I3
I1,I2,I5

C3

Itemset	S.C
I1,I2,I3	2
I1,I2,I5	2

L3

Itemset	S.C
I1,I2,I3	2
I1,I2,I5	2

Implementation Overview :

The implementation is done from scratch. It uses Python 3 as the main programming language.

The implementation uses 5 different Datasets to hold the itemsets and transactions.

When you run the program, it will prompt you with a number of choices to choose from like the Dataset you want to run your code on, the number of Transactions,Maximum number of Items per Transaction,Minimum Support Value and the Minimum confidence value to input .

On the basis of the input, the program will read the flat file database, parse the items names and transactions, and then load them to the database to parse the transaction for support and confidence to find the association rules using Apriori algorithm.

Assumptions :

- Number of Transactions range from 1 to a max possible value of 20.
- Maximum items per transactions range from 1 to a max possible value of 5
- Minimum value of support input value should not exceed 100.
- Minimum confidence value input value should not exceed 100.

Requirements :

Software Configuration :

Jupyter Notebook 6.4.3 Anaconda version

Python 3.8

NumPy, pandas and python csv libraries used so far.

Hardware Configuration :

Operating System: Windows 10

Processor: Intel(R) Core (TM) i5-7200U CPU @ 2.50GHz   2.70 GHz

RAM: 8GB

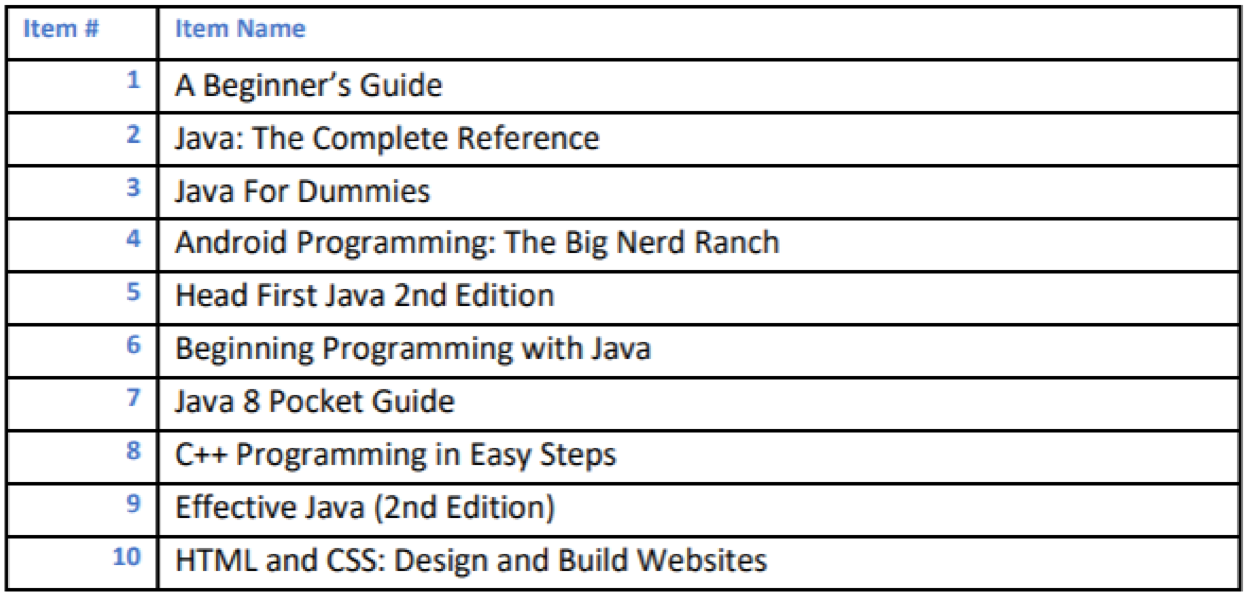
How to Run the Application :

Prerequisites :

- Python 3 and Jupyter Notebook 6.4.3 installed in the system.
- Take care of all Assumptions listed in the Assumptions section.
- Enter the required path to the Dataset for which we want to run the code.

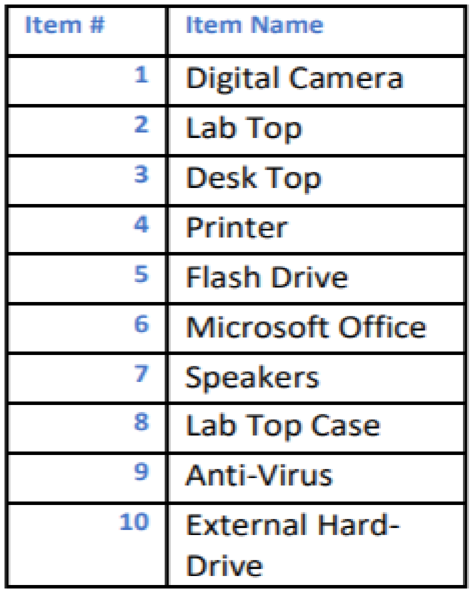
Datasets Used :

1) Amazon :



Item #	Item Name
1	A Beginner's Guide
2	Java: The Complete Reference
3	Java For Dummies
4	Android Programming: The Big Nerd Ranch
5	Head First Java 2nd Edition
6	Beginning Programming with Java
7	Java 8 Pocket Guide
8	C++ Programming in Easy Steps
9	Effective Java (2nd Edition)
10	HTML and CSS: Design and Build Websites

2) Best Buy :



Item #	Item Name
1	Digital Camera
2	Lab Top
3	Desk Top
4	Printer
5	Flash Drive
6	Microsoft Office
7	Speakers
8	Lab Top Case
9	Anti-Virus
10	External Hard-Drive

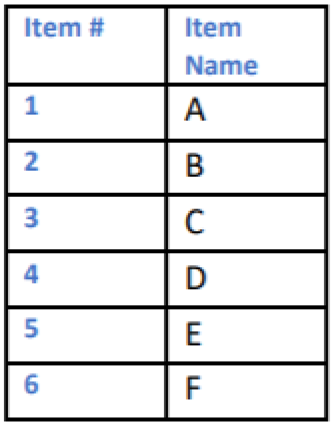
3) K-Mart :

Item #	Item Name
1	Quilts
2	Bedspreads
3	Decorative Pillows
4	Bed Skirts
5	Sheets
6	Shams
7	Bedding Collections
8	Kids Bedding
9	Embroidered Bedspread
10	Towels

4) Nike :

Item #	Item Name
1	Running Shoe
2	Soccer Shoe
3	Socks
4	Swimming Shirt
5	Dry Fit V-Nick
6	Rash Guard
7	Sweatshirts
8	Hoodies
9	Tech Pants
10	Modern Pants

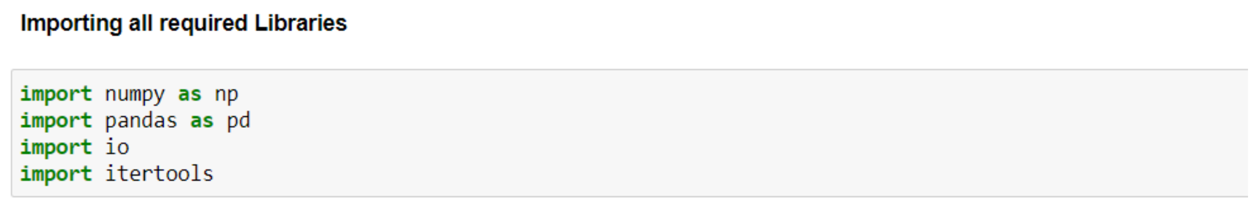
5) Generic :



Item #	Item Name
1	A
2	B
3	C
4	D
5	E
6	F

Code Implementation :

Importing all required Libraries for the program .

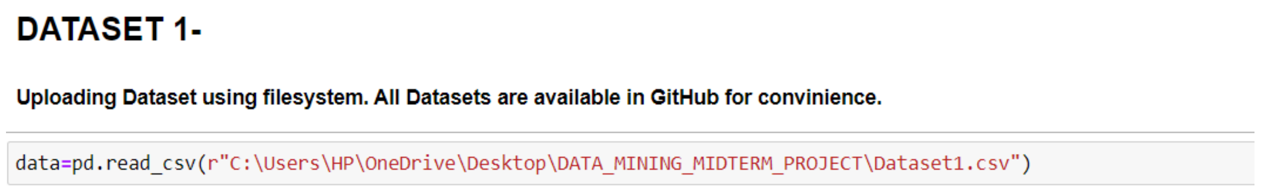


Importing all required Libraries

```
import numpy as np
import pandas as pd
import io
import itertools
```

Dataset 1 (Amazon) :

Representation of how the Datasets once uploaded to drive look like :

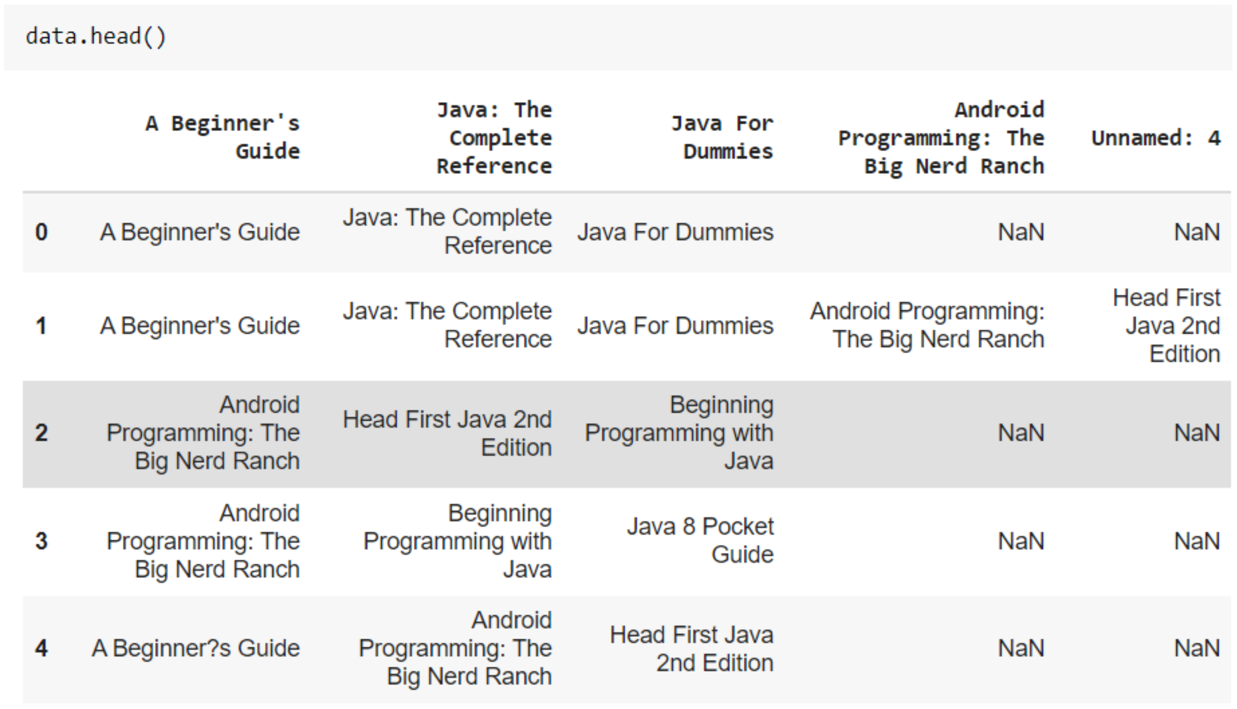


DATASET 1-

Uploading Dataset using filesystem. All Datasets are available in GitHub for convinience.

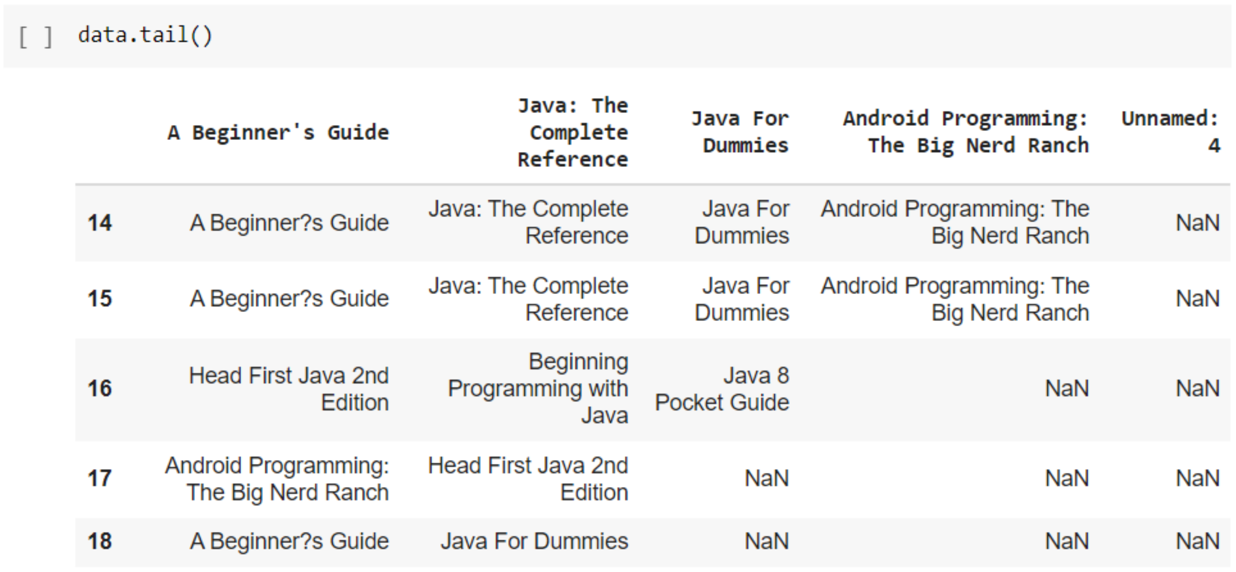
```
data=pd.read_csv(r"C:\Users\HP\OneDrive\Desktop\DATA_MINING_MIDTERM_PROJECT\Dataset1.csv")
```

Data representation :



```
data.head()
```

	A Beginner's Guide	Java: The Complete Reference	Java For Dummies	Android Programming: The Big Nerd Ranch	Unnamed: 4
0	A Beginner's Guide	Java: The Complete Reference	Java For Dummies	NaN	NaN
1	A Beginner's Guide	Java: The Complete Reference	Java For Dummies	Android Programming: The Big Nerd Ranch	Head First Java 2nd Edition
2	Android Programming: The Big Nerd Ranch	Head First Java 2nd Edition	Beginning Programming with Java	NaN	NaN
3	Android Programming: The Big Nerd Ranch	Beginning Programming with Java	Java 8 Pocket Guide	NaN	NaN
4	A Beginner?s Guide	Android Programming: The Big Nerd Ranch	Head First Java 2nd Edition	NaN	NaN



```
[ ]  data.tail()
```

	A Beginner's Guide	Java: The Complete Reference	Java For Dummies	Android Programming: The Big Nerd Ranch	Unnamed: 4
14	A Beginner?s Guide	Java: The Complete Reference	Java For Dummies	Android Programming: The Big Nerd Ranch	NaN
15	A Beginner?s Guide	Java: The Complete Reference	Java For Dummies	Android Programming: The Big Nerd Ranch	NaN
16	Head First Java 2nd Edition	Beginning Programming with Java	Java 8 Pocket Guide	NaN	NaN
17	Android Programming: The Big Nerd Ranch	Head First Java 2nd Edition	NaN	NaN	NaN
18	A Beginner?s Guide	Java For Dummies	NaN	NaN	NaN

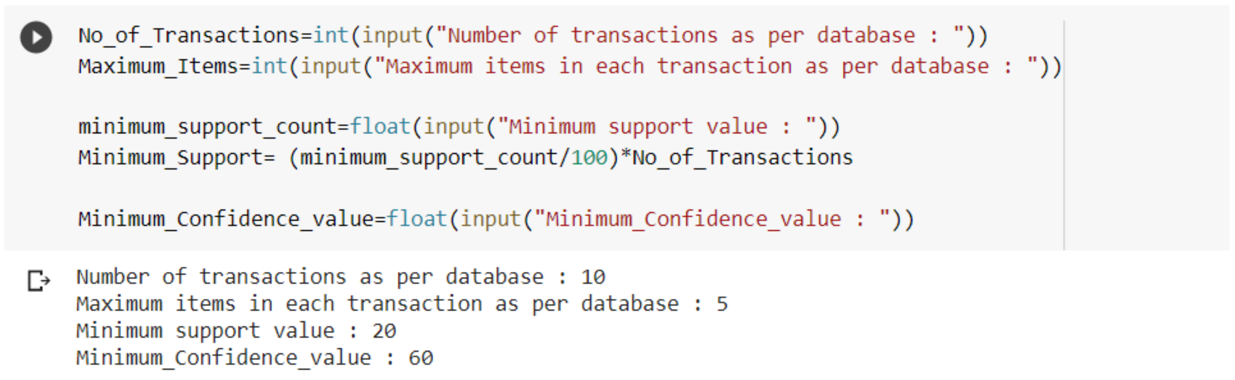
Getting input from the user for the following :

1.Number of Transactions (Max possible value = 20)

2.Maximum items per transactions (Max possible value = 5)

3.Minimum value of support (Min possible value = 20)

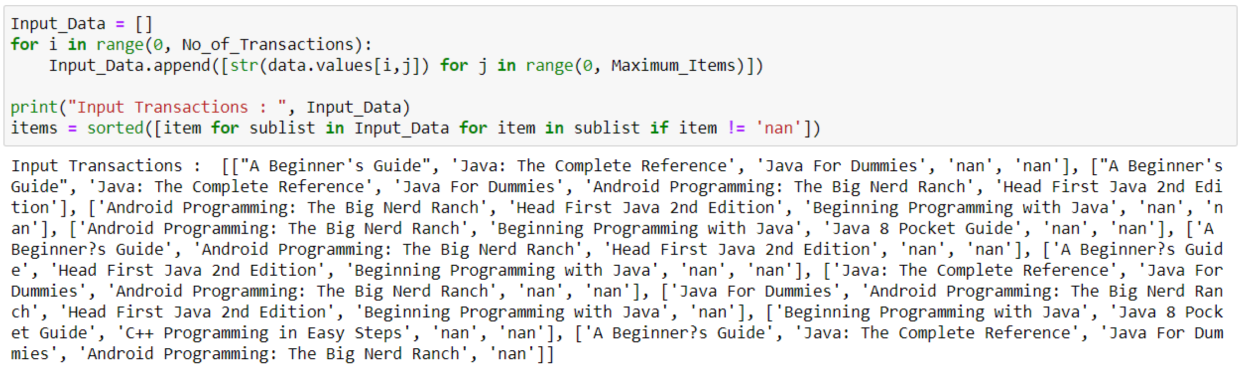
4.Minimum confidence value (Maximum possible value =100)



```
No_of_Transactions=int(input("Number of transactions as per database : "))
Maximum_Items=int(input("Maximum items in each transaction as per database : "))

minimum_support_count=float(input("Minimum support value : "))
Minimum_Support= (minimum_support_count/100)*No_of_Transactions

Minimum_Confidence_value=float(input("Minimum_Confidence_value : "))
```

```
Number of transactions as per database : 10
Maximum items in each transaction as per database : 5
Minimum support value : 20
Minimum_Confidence_value : 60
```

Data processing to create the required records and item-list for further processing.

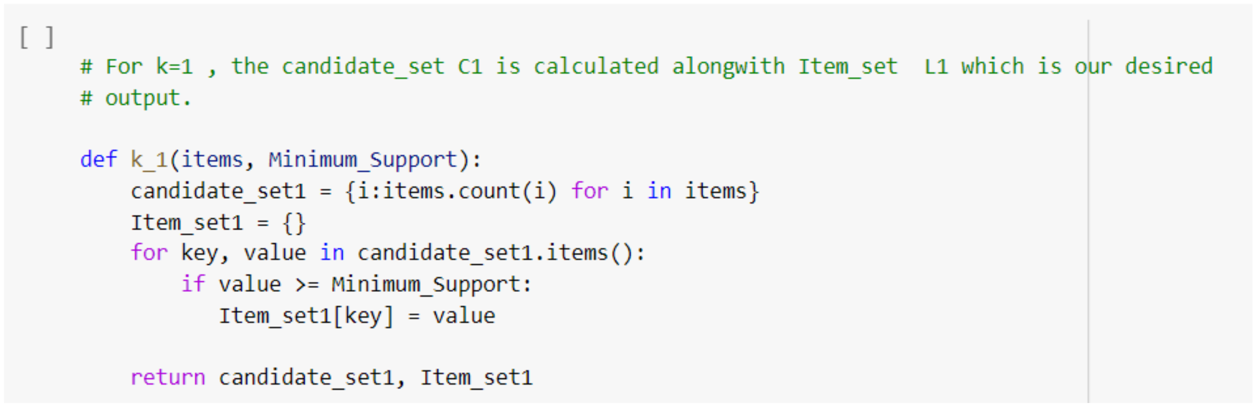


```
Input_Data = []
for i in range(0, No_of_Transactions):
    Input_Data.append([str(data.values[i,j]) for j in range(0, Maximum_Items)])

print("Input Transactions : ", Input_Data)
items = sorted([item for sublist in Input_Data for item in sublist if item != 'nan'])
```

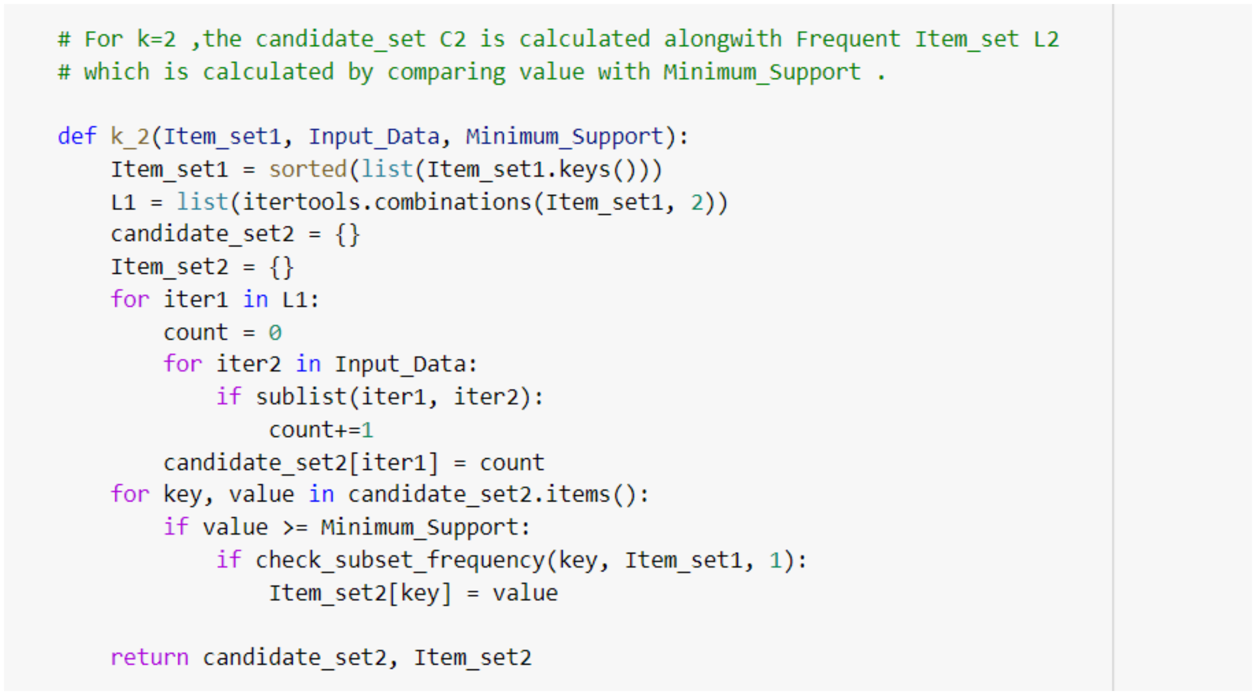
```
Input Transactions :  [["A Beginner's Guide", 'Java: The Complete Reference', 'Java For Dummies', 'nan', 'nan'], ["A Beginner's
Guide", 'Java: The Complete Reference', 'Java For Dummies', 'Android Programming: The Big Nerd Ranch', 'Head First Java 2nd Edi
tion'], ['Android Programming: The Big Nerd Ranch', 'Head First Java 2nd Edition', 'Beginning Programming with Java', 'nan', 'n
an'], ['Android Programming: The Big Nerd Ranch', 'Beginning Programming with Java', 'Java 8 Pocket Guide', 'nan', 'nan'], ['A
Beginner?s Guide', 'Android Programming: The Big Nerd Ranch', 'Head First Java 2nd Edition', 'nan', 'nan'], ['A Beginner?s Guid
e', 'Head First Java 2nd Edition', 'Beginning Programming with Java', 'nan', 'nan'], ['Java: The Complete Reference', 'Java For
Dummies', 'Android Programming: The Big Nerd Ranch', 'nan', 'nan'], ['Java For Dummies', 'Android Programming: The Big Nerd Ran
ch', 'Head First Java 2nd Edition', 'Beginning Programming with Java', 'nan'], ['Beginning Programming with Java', 'Java 8 Pock
et Guide', 'C++ Programming in Easy Steps', 'nan', 'nan'], ['A Beginner?s Guide', 'Java: The Complete Reference', 'Java For Dum
mies', 'Android Programming: The Big Nerd Ranch', 'nan']]
```

For k=1 to 4 , candidate_set and the frequent ItemSet are calculated by comparing the support count of each item in the list to the Minimum Support value. We check if all the subsets in itemset are frequent using the check subset frequency function and if not, we remove respective itemset from the list by comparing the length of the two possible lists using the sublist function.

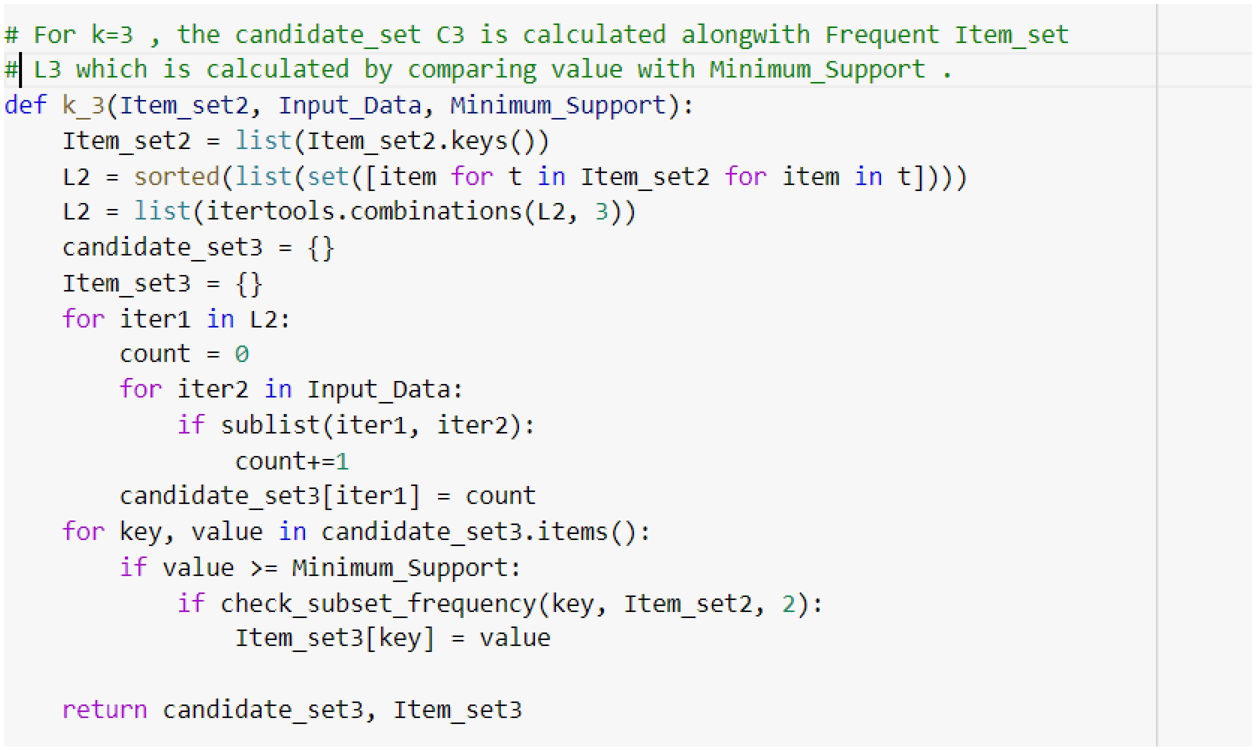
We are printing all the Item_Sets generated after calculations.



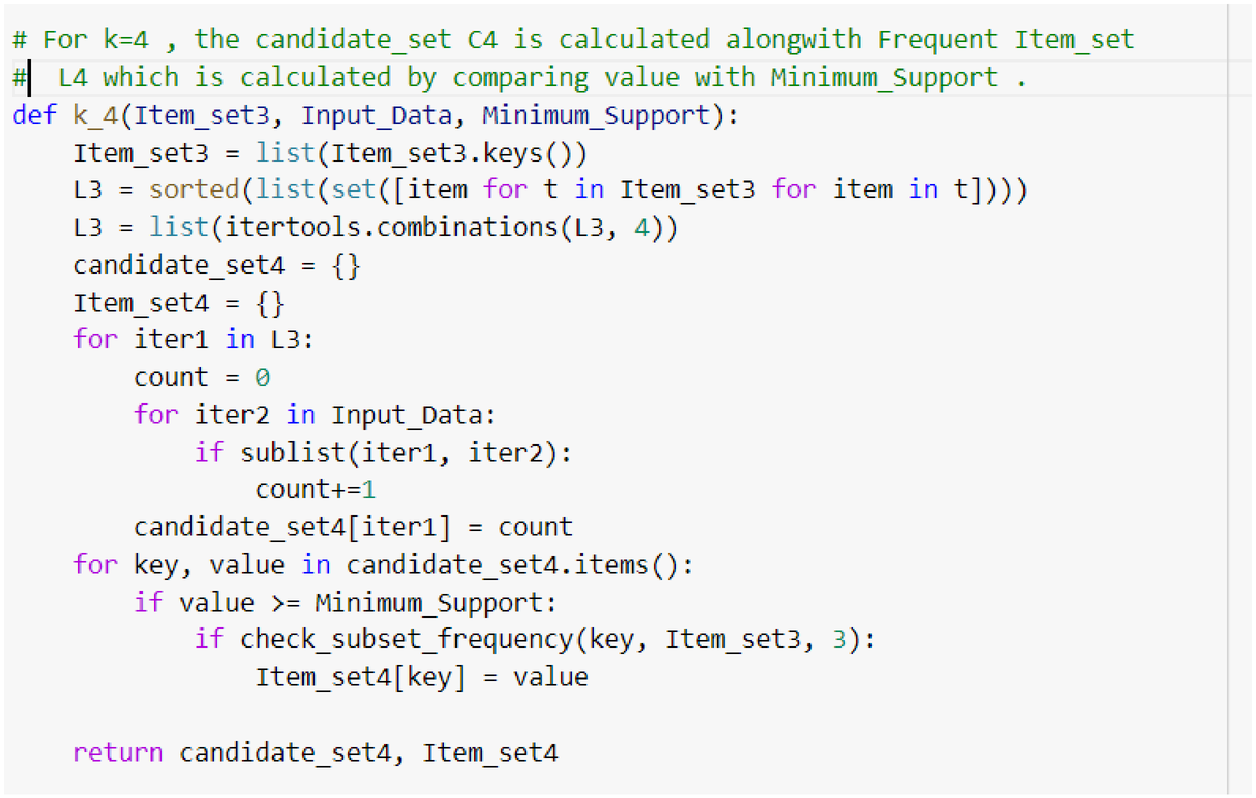
```
[ ]
    # For k=1 , the candidate_set C1 is calculated alongwith Item_set  L1 which is our desired
    # output.

    def k_1(items, Minimum_Support):
        candidate_set1 = {i:items.count(i) for i in items}
        Item_set1 = {}
        for key, value in candidate_set1.items():
            if value >= Minimum_Support:
               Item_set1[key] = value

        return candidate_set1, Item_set1
```



```
# For k=2 ,the candidate_set C2 is calculated alongwith Frequent Item_set L2
# which is calculated by comparing value with Minimum_Support .

def k_2(Item_set1, Input_Data, Minimum_Support):
    Item_set1 = sorted(list(Item_set1.keys()))
    L1 = list(itertools.combinations(Item_set1, 2))
    candidate_set2 = {}
    Item_set2 = {}
    for iter1 in L1:
        count = 0
        for iter2 in Input_Data:
            if sublist(iter1, iter2):
                count+=1
        candidate_set2[iter1] = count
    for key, value in candidate_set2.items():
        if value >= Minimum_Support:
            if check_subset_frequency(key, Item_set1, 1):
                Item_set2[key] = value

    return candidate_set2, Item_set2
```



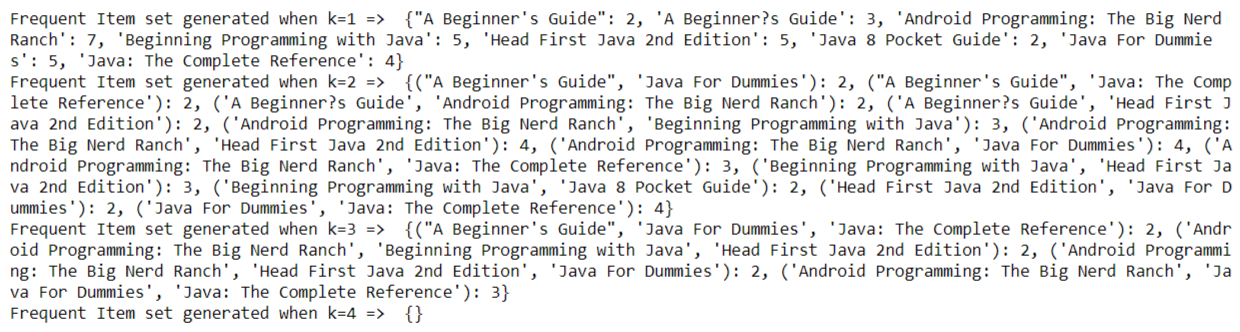
```python
# For k=3 , the candidate_set C3 is calculated alongwith Frequent Item_set
# L3 which is calculated by comparing value with Minimum_Support .
def k_3(Item_set2, Input_Data, Minimum_Support):
    Item_set2 = list(Item_set2.keys())
    L2 = sorted(list(set([item for t in Item_set2 for item in t])))
    L2 = list(itertools.combinations(L2, 3))
    candidate_set3 = {}
    Item_set3 = {}
    for iter1 in L2:
        count = 0
        for iter2 in Input_Data:
            if sublist(iter1, iter2):
                count+=1
        candidate_set3[iter1] = count
    for key, value in candidate_set3.items():
        if value >= Minimum_Support:
            if check_subset_frequency(key, Item_set2, 2):
                Item_set3[key] = value

    return candidate_set3, Item_set3
```



```python
# For k=4 , the candidate_set C4 is calculated alongwith Frequent Item_set
#  L4 which is calculated by comparing value with Minimum_Support .
def k_4(Item_set3, Input_Data, Minimum_Support):
    Item_set3 = list(Item_set3.keys())
    L3 = sorted(list(set([item for t in Item_set3 for item in t])))
    L3 = list(itertools.combinations(L3, 4))
    candidate_set4 = {}
    Item_set4 = {}
    for iter1 in L3:
        count = 0
        for iter2 in Input_Data:
            if sublist(iter1, iter2):
                count+=1
        candidate_set4[iter1] = count
    for key, value in candidate_set4.items():
        if value >= Minimum_Support:
            if check_subset_frequency(key, Item_set3, 3):
                Item_set4[key] = value

    return candidate_set4, Item_set4
```

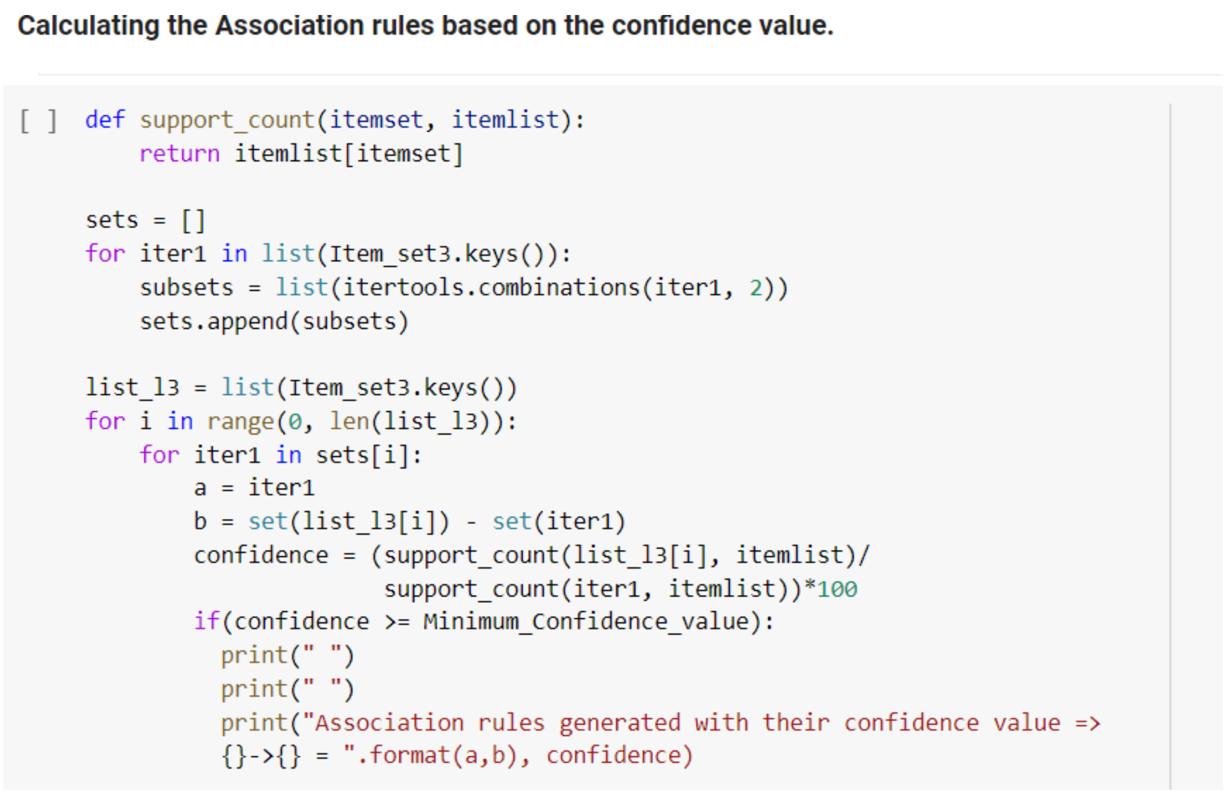
```python
def sublist(lst1, lst2):
    return set(lst1) <= set(lst2)

def check_subset_frequency(itemset, l, n):
    if n>1:
        subsets = list(itertools.combinations(itemset, n))
    else:
        subsets = itemset
    for iter1 in subsets:
        if not iter1 in l:
            return False
    return True


candidate_set1, Item_set1 = k_1(items, Minimum_Support)
candidate_set2, Item_set2 = k_2(Item_set1, Input_Data, Minimum_Support)
candidate_set3, Item_set3 = k_3(Item_set2, Input_Data, Minimum_Support)
candidate_set4, Item_set4 = k_4(Item_set3, Input_Data, Minimum_Support)
print(" ")
print(" ")
print("Frequent Item set generated when k=1 => ", Item_set1)
print("Frequent Item set generated when k=2 => ", Item_set2)
print("Frequent Item set generated when k=3 => ", Item_set3)
print("Frequent Item set generated when k=4 => ", Item_set4)


itemlist = {**Item_set1, **Item_set2, **Item_set3, **Item_set4}
```

Frequent Item sets created :



```
Frequent Item set generated when k=1 =>  {"A Beginner's Guide": 2, 'A Beginner?s Guide': 3, 'Android Programming: The Big Nerd
Ranch': 7, 'Beginning Programming with Java': 5, 'Head First Java 2nd Edition': 5, 'Java 8 Pocket Guide': 2, 'Java For Dummie
s': 5, 'Java: The Complete Reference': 4}
Frequent Item set generated when k=2 =>  {("A Beginner's Guide", 'Java For Dummies'): 2, ("A Beginner's Guide", 'Java: The Comp
lete Reference'): 2, ('A Beginner?s Guide', 'Android Programming: The Big Nerd Ranch'): 2, ('A Beginner?s Guide', 'Head First J
ava 2nd Edition'): 2, ('Android Programming: The Big Nerd Ranch', 'Beginning Programming with Java'): 3, ('Android Programming:
The Big Nerd Ranch', 'Head First Java 2nd Edition'): 4, ('Android Programming: The Big Nerd Ranch', 'Java For Dummies'): 4, ('A
ndroid Programming: The Big Nerd Ranch', 'Java: The Complete Reference'): 3, ('Beginning Programming with Java', 'Head First Ja
va 2nd Edition'): 3, ('Beginning Programming with Java', 'Java 8 Pocket Guide'): 2, ('Head First Java 2nd Edition', 'Java For D
ummies'): 2, ('Java For Dummies', 'Java: The Complete Reference'): 4}
Frequent Item set generated when k=3 =>  {("A Beginner's Guide", 'Java For Dummies', 'Java: The Complete Reference'): 2, ('Andr
oid Programming: The Big Nerd Ranch', 'Beginning Programming with Java', 'Head First Java 2nd Edition'): 2, ('Android Programmi
ng: The Big Nerd Ranch', 'Head First Java 2nd Edition', 'Java For Dummies'): 2, ('Android Programming: The Big Nerd Ranch', 'Ja
va For Dummies', 'Java: The Complete Reference'): 3}
Frequent Item set generated when k=4 =>  {}
```

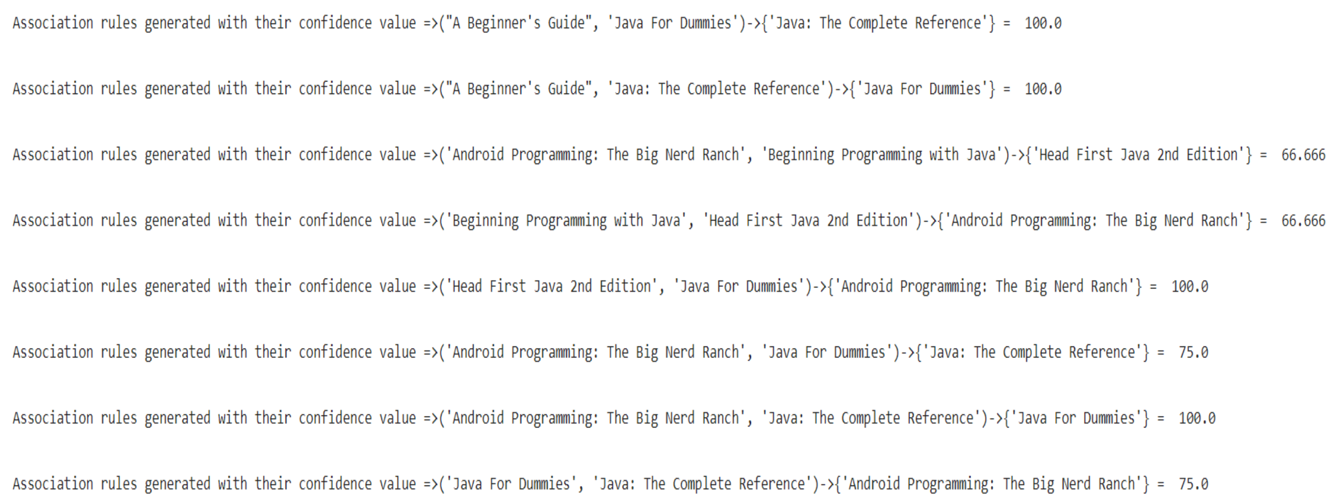
(**Resultset was too wide for screenshot so zoom in to see the output clearly)



Calculating the Association rules based on the confidence value.

```
[ ]  def support_count(itemset, itemlist):
         return itemlist[itemset]

     sets = []
     for iter1 in list(Item_set3.keys()):
         subsets = list(itertools.combinations(iter1, 2))
         sets.append(subsets)

     list_l3 = list(Item_set3.keys())
     for i in range(0, len(list_l3)):
         for iter1 in sets[i]:
             a = iter1
             b = set(list_l3[i]) - set(iter1)
             confidence = (support_count(list_l3[i], itemlist)/
                           support_count(iter1, itemlist))*100
             if(confidence >= Minimum_Confidence_value):
               print(" ")
               print(" ")
               print("Association rules generated with their confidence value =>
               {}->{} = ".format(a,b), confidence)
```

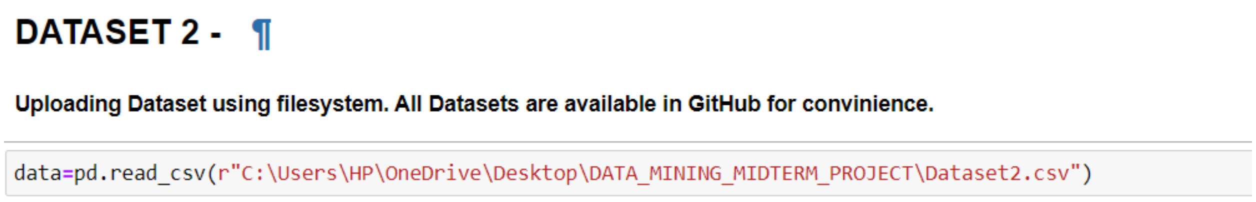
Association Rules with Confidence values :



```
Association rules generated with their confidence value =>("A Beginner's Guide", 'Java For Dummies')->{'Java: The Complete Reference'} =  100.0

Association rules generated with their confidence value =>("A Beginner's Guide", 'Java: The Complete Reference')->{'Java For Dummies'} =  100.0

Association rules generated with their confidence value =>('Android Programming: The Big Nerd Ranch', 'Beginning Programming with Java')->{'Head First Java 2nd Edition'} =  66.666

Association rules generated with their confidence value =>('Beginning Programming with Java', 'Head First Java 2nd Edition')->{'Android Programming: The Big Nerd Ranch'} =  66.666

Association rules generated with their confidence value =>('Head First Java 2nd Edition', 'Java For Dummies')->{'Android Programming: The Big Nerd Ranch'} =  100.0

Association rules generated with their confidence value =>('Android Programming: The Big Nerd Ranch', 'Java For Dummies')->{'Java: The Complete Reference'} =  75.0

Association rules generated with their confidence value =>('Android Programming: The Big Nerd Ranch', 'Java: The Complete Reference')->{'Java For Dummies'} =  100.0

Association rules generated with their confidence value =>('Java For Dummies', 'Java: The Complete Reference')->{'Android Programming: The Big Nerd Ranch'} =  75.0
```

(**Resultset was too wide for screenshot so zoom in to see the output clearly)

Dataset 2 (Best Buy) :

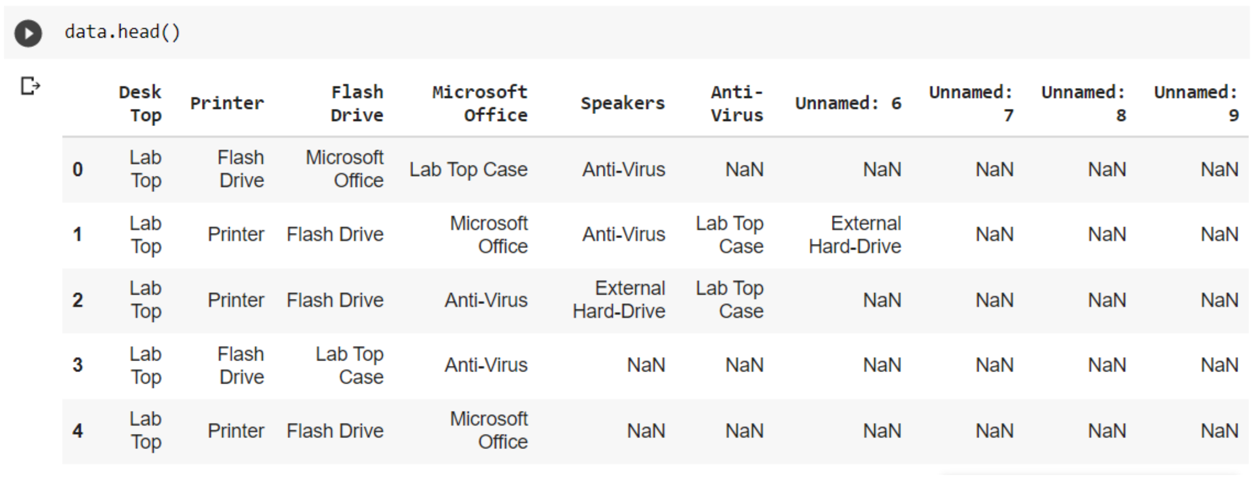


DATASET 2 - ¶

Uploading Dataset using filesystem. All Datasets are available in GitHub for convinience.

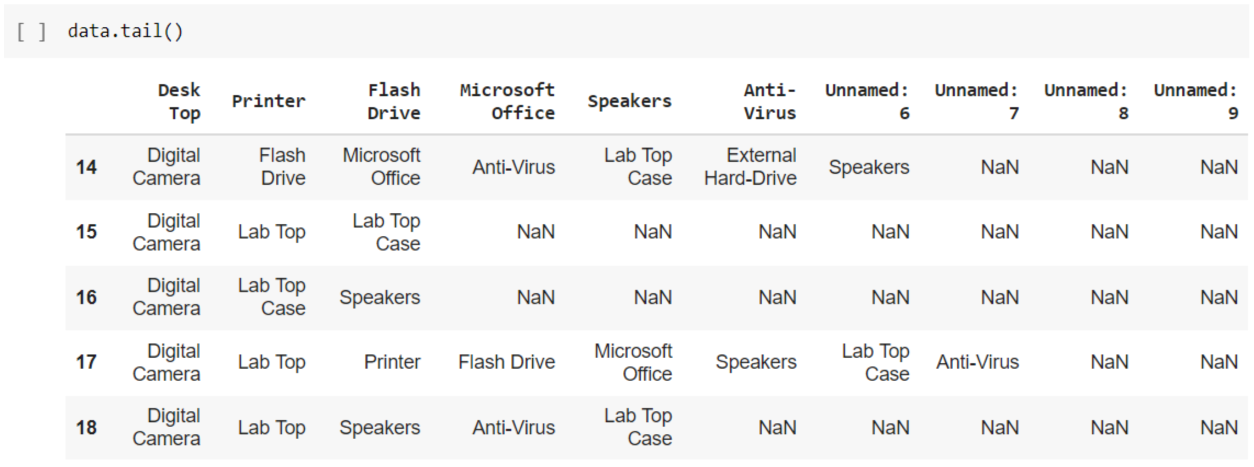
```
data=pd.read_csv(r"C:\Users\HP\OneDrive\Desktop\DATA_MINING_MIDTERM_PROJECT\Dataset2.csv")
```

Data Representation :



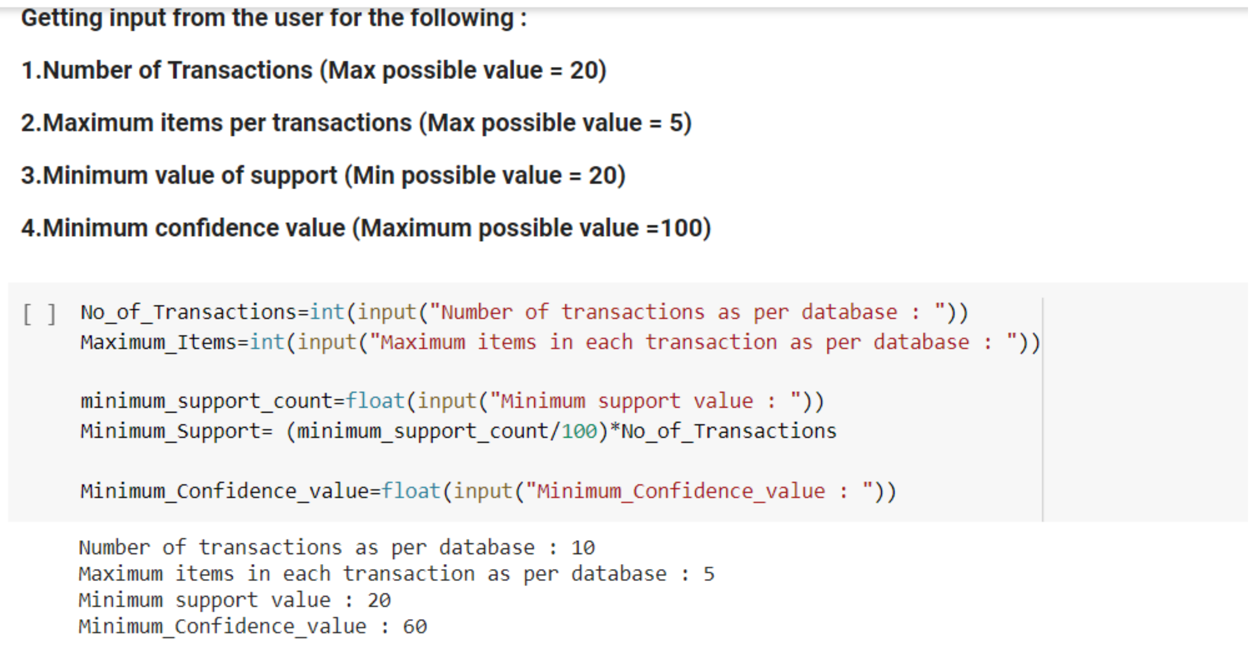
```
data.head()
```

	Desk Top	Printer	Flash Drive	Microsoft Office	Speakers	Anti-Virus	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9
0	Lab Top	Flash Drive	Microsoft Office	Lab Top Case	Anti-Virus	NaN	NaN	NaN	NaN	NaN
1	Lab Top	Printer	Flash Drive	Microsoft Office	Anti-Virus	Lab Top Case	External Hard-Drive	NaN	NaN	NaN
2	Lab Top	Printer	Flash Drive	Anti-Virus	External Hard-Drive	Lab Top Case	NaN	NaN	NaN	NaN
3	Lab Top	Flash Drive	Lab Top Case	Anti-Virus	NaN	NaN	NaN	NaN	NaN	NaN
4	Lab Top	Printer	Flash Drive	Microsoft Office	NaN	NaN	NaN	NaN	NaN	NaN



```
[ ] data.tail()
```

	Desk Top	Printer	Flash Drive	Microsoft Office	Speakers	Anti-Virus	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9
14	Digital Camera	Flash Drive	Microsoft Office	Anti-Virus	Lab Top Case	External Hard-Drive	Speakers	NaN	NaN	NaN
15	Digital Camera	Lab Top	Lab Top Case	NaN	NaN	NaN	NaN	NaN	NaN	NaN
16	Digital Camera	Lab Top Case	Speakers	NaN	NaN	NaN	NaN	NaN	NaN	NaN
17	Digital Camera	Lab Top	Printer	Flash Drive	Microsoft Office	Speakers	Lab Top Case	Anti-Virus	NaN	NaN
18	Digital Camera	Lab Top	Speakers	Anti-Virus	Lab Top Case	NaN	NaN	NaN	NaN	NaN



Getting input from the user for the following :

1.Number of Transactions (Max possible value = 20)

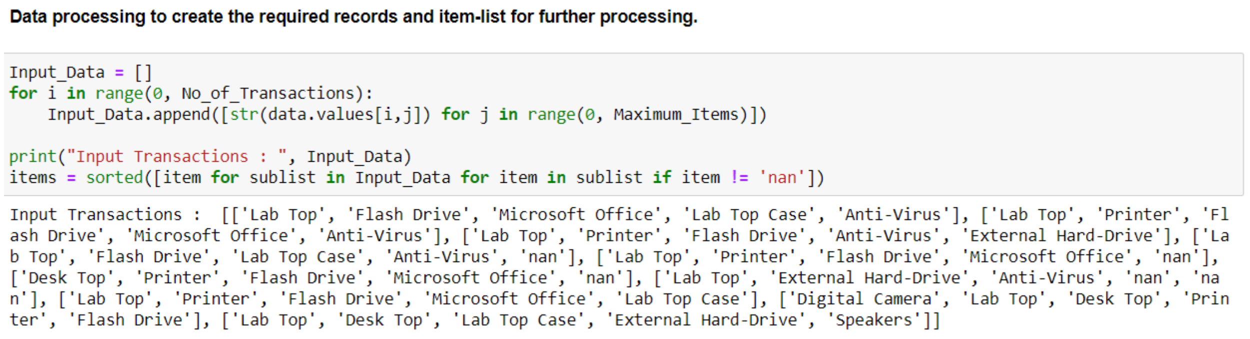
2.Maximum items per transactions (Max possible value = 5)

3.Minimum value of support (Min possible value = 20)

4.Minimum confidence value (Maximum possible value =100)

```
[ ] No_of_Transactions=int(input("Number of transactions as per database : "))
    Maximum_Items=int(input("Maximum items in each transaction as per database : "))

    minimum_support_count=float(input("Minimum support value : "))
    Minimum_Support= (minimum_support_count/100)*No_of_Transactions

    Minimum_Confidence_value=float(input("Minimum_Confidence_value : "))
```

```
Number of transactions as per database : 10
Maximum items in each transaction as per database : 5
Minimum support value : 20
Minimum_Confidence_value : 60
```



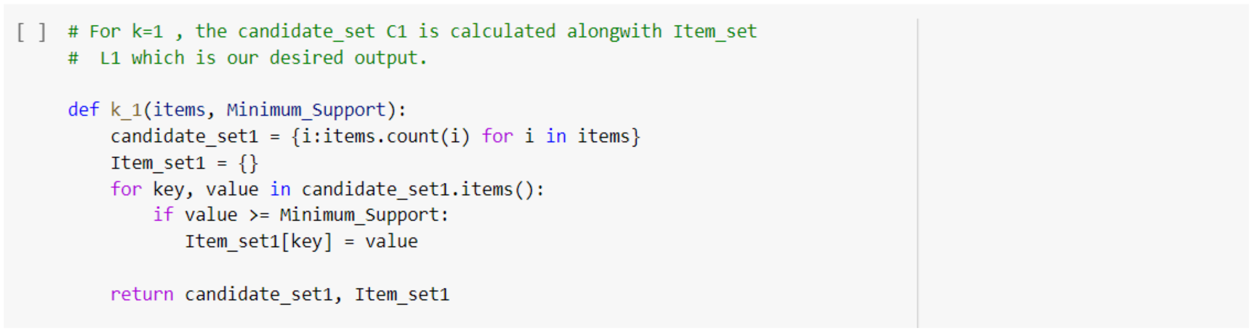
Data processing to create the required records and item-list for further processing.

```
Input_Data = []
for i in range(0, No_of_Transactions):
    Input_Data.append([str(data.values[i,j]) for j in range(0, Maximum_Items)])

print("Input Transactions : ", Input_Data)
items = sorted([item for sublist in Input_Data for item in sublist if item != 'nan'])
```

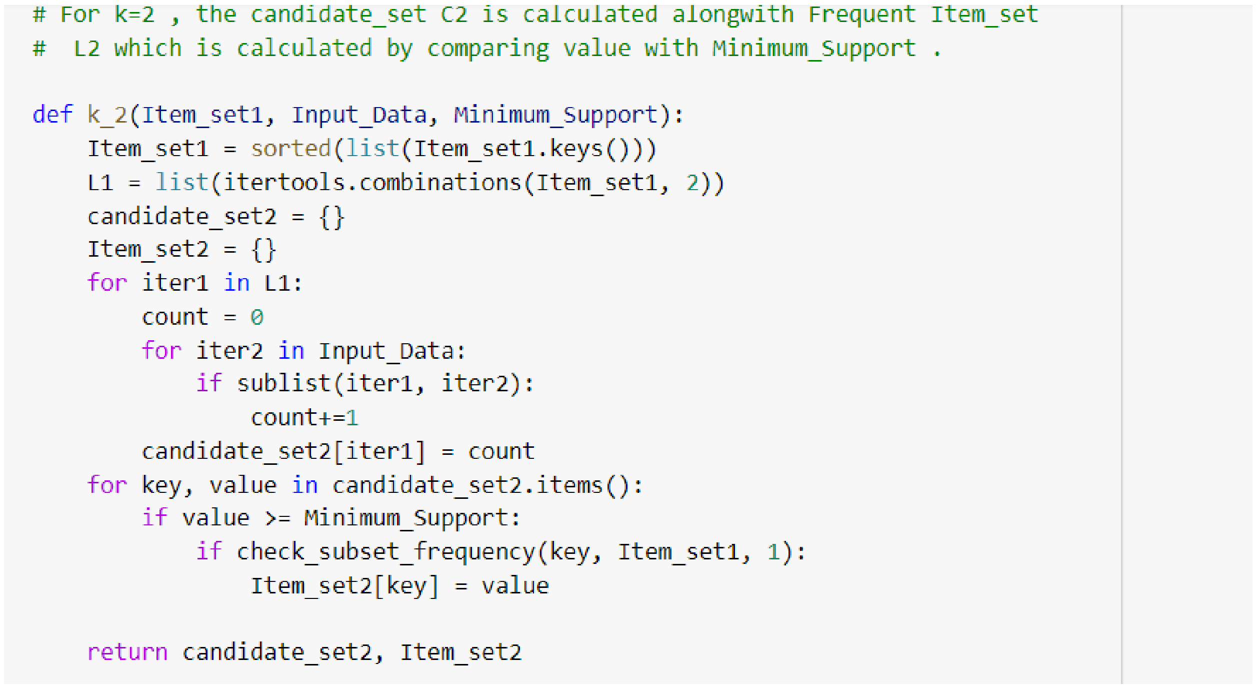
```
Input Transactions :  [['Lab Top', 'Flash Drive', 'Microsoft Office', 'Lab Top Case', 'Anti-Virus'], ['Lab Top', 'Printer', 'Fl
ash Drive', 'Microsoft Office', 'Anti-Virus'], ['Lab Top', 'Printer', 'Flash Drive', 'Anti-Virus', 'External Hard-Drive'], ['La
b Top', 'Flash Drive', 'Lab Top Case', 'Anti-Virus', 'nan'], ['Lab Top', 'Printer', 'Flash Drive', 'Microsoft Office', 'nan'],
['Desk Top', 'Printer', 'Flash Drive', 'Microsoft Office', 'nan'], ['Lab Top', 'External Hard-Drive', 'Anti-Virus', 'nan', 'na
n'], ['Lab Top', 'Printer', 'Flash Drive', 'Microsoft Office', 'Lab Top Case'], ['Digital Camera', 'Lab Top', 'Desk Top', 'Prin
ter', 'Flash Drive'], ['Lab Top', 'Desk Top', 'Lab Top Case', 'External Hard-Drive', 'Speakers']]
```

For k=1 to 4 , candidate_set and the frequent ItemSet are calculated by comparing the support count of each item in the list to the Minimum Support value. We check if all the subsets in itemset are frequent using the check subset frequency function and if not, we remove respective itemset from the list by comparing the length of the two possible lists using the sublist function.

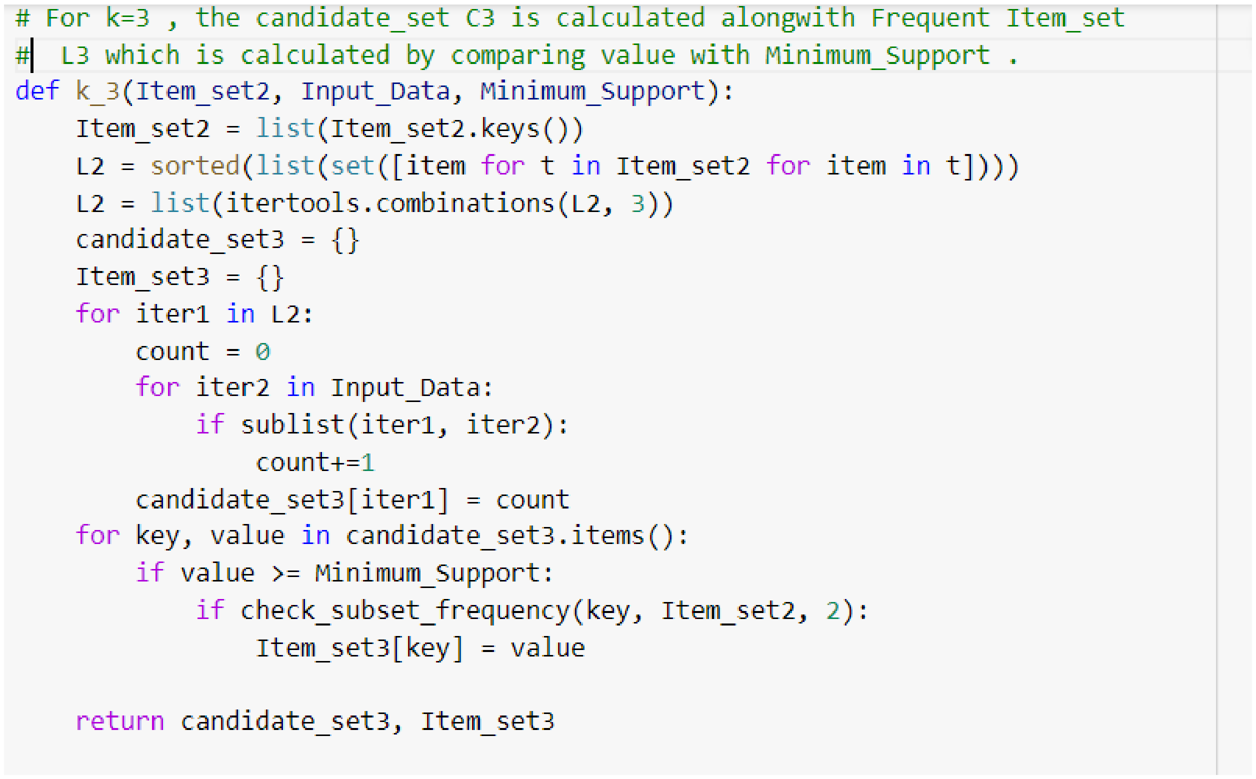
We are printing all the Item_Sets generated after calculations.



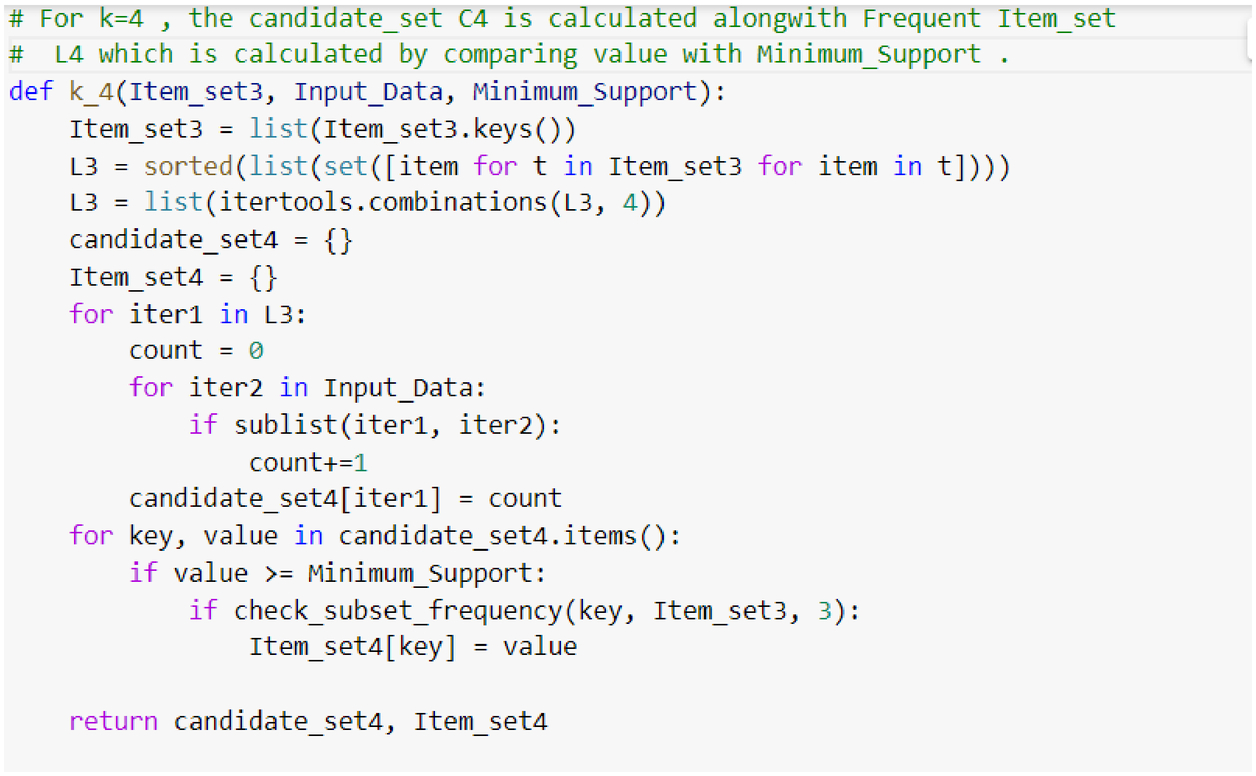
```
[ ]  # For k=1 , the candidate_set C1 is calculated alongwith Item_set
     #  L1 which is our desired output.

     def k_1(items, Minimum_Support):
         candidate_set1 = {i:items.count(i) for i in items}
         Item_set1 = {}
         for key, value in candidate_set1.items():
             if value >= Minimum_Support:
                Item_set1[key] = value

         return candidate_set1, Item_set1
```



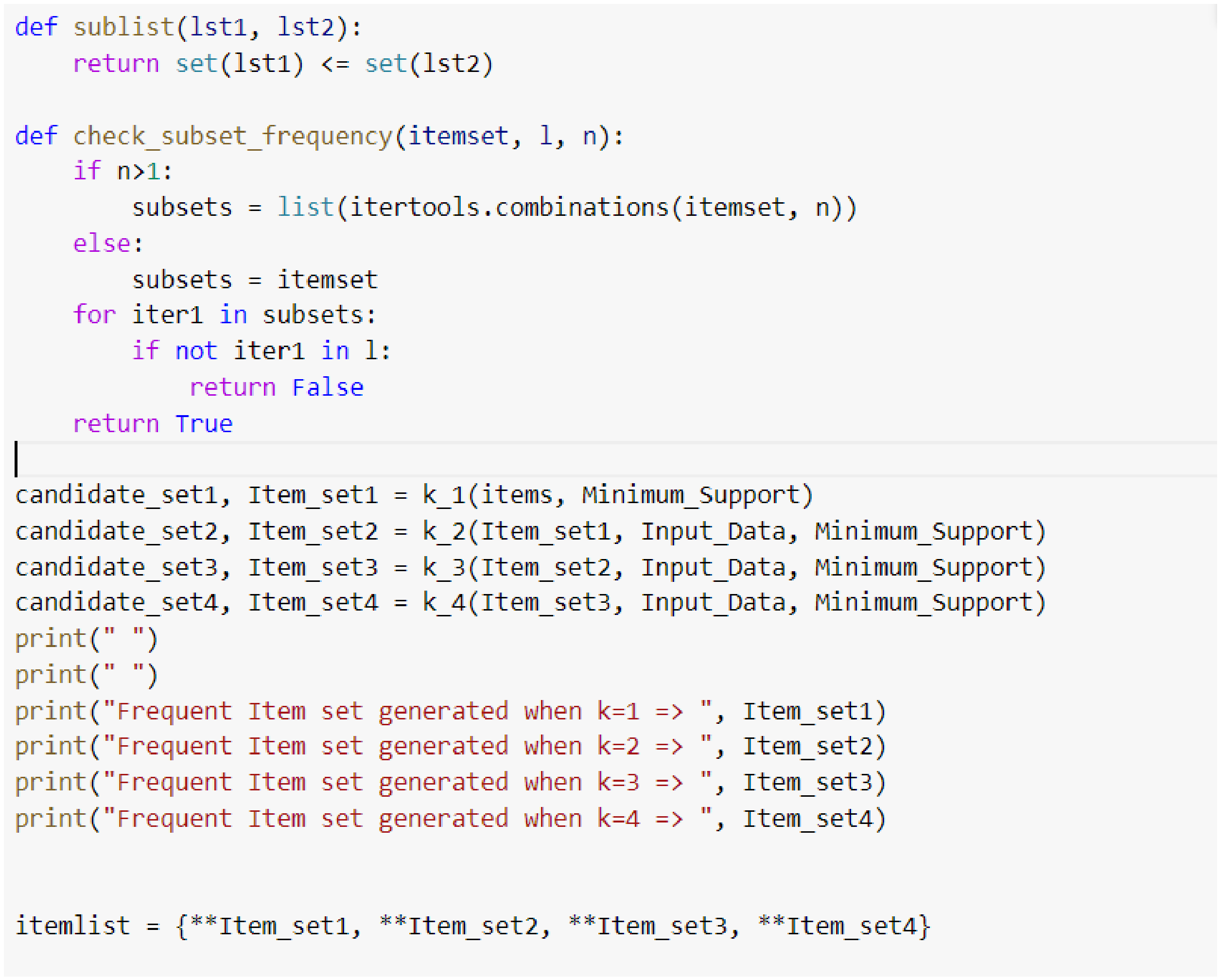
```
# For k=2 , the candidate_set C2 is calculated alongwith Frequent Item_set
#  L2 which is calculated by comparing value with Minimum_Support .

def k_2(Item_set1, Input_Data, Minimum_Support):
    Item_set1 = sorted(list(Item_set1.keys()))
    L1 = list(itertools.combinations(Item_set1, 2))
    candidate_set2 = {}
    Item_set2 = {}
    for iter1 in L1:
        count = 0
        for iter2 in Input_Data:
            if sublist(iter1, iter2):
                count+=1
        candidate_set2[iter1] = count
    for key, value in candidate_set2.items():
        if value >= Minimum_Support:
            if check_subset_frequency(key, Item_set1, 1):
                Item_set2[key] = value

    return candidate_set2, Item_set2
```



```
# For k=3 , the candidate_set C3 is calculated alongwith Frequent Item_set
#  L3 which is calculated by comparing value with Minimum_Support .
def k_3(Item_set2, Input_Data, Minimum_Support):
    Item_set2 = list(Item_set2.keys())
    L2 = sorted(list(set([item for t in Item_set2 for item in t])))
    L2 = list(itertools.combinations(L2, 3))
    candidate_set3 = {}
    Item_set3 = {}
    for iter1 in L2:
        count = 0
        for iter2 in Input_Data:
            if sublist(iter1, iter2):
                count+=1
        candidate_set3[iter1] = count
    for key, value in candidate_set3.items():
        if value >= Minimum_Support:
            if check_subset_frequency(key, Item_set2, 2):
                Item_set3[key] = value

    return candidate_set3, Item_set3
```

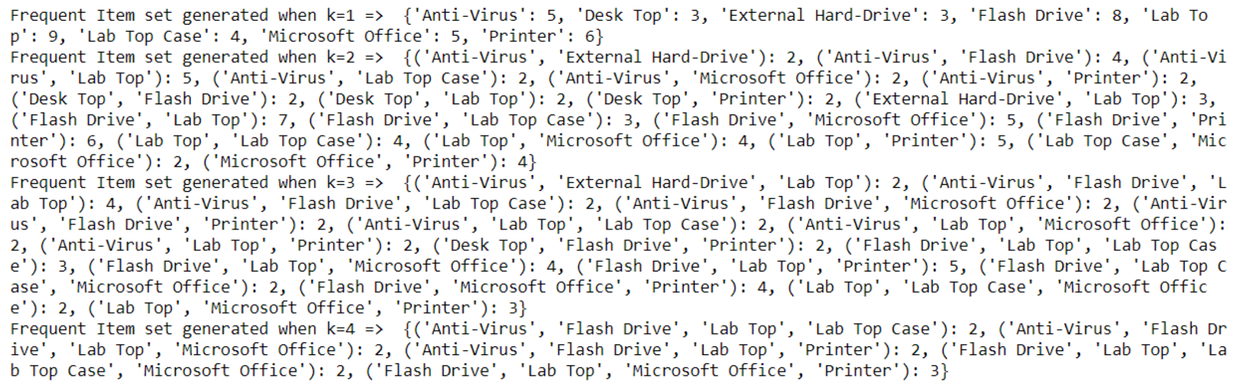


```
# For k=4 , the candidate_set C4 is calculated alongwith Frequent Item_set
#  L4 which is calculated by comparing value with Minimum_Support .
def k_4(Item_set3, Input_Data, Minimum_Support):
    Item_set3 = list(Item_set3.keys())
    L3 = sorted(list(set([item for t in Item_set3 for item in t])))
    L3 = list(itertools.combinations(L3, 4))
    candidate_set4 = {}
    Item_set4 = {}
    for iter1 in L3:
        count = 0
        for iter2 in Input_Data:
            if sublist(iter1, iter2):
                count+=1
        candidate_set4[iter1] = count
    for key, value in candidate_set4.items():
        if value >= Minimum_Support:
            if check_subset_frequency(key, Item_set3, 3):
                Item_set4[key] = value

    return candidate_set4, Item_set4
```



```python
def sublist(lst1, lst2):
    return set(lst1) <= set(lst2)

def check_subset_frequency(itemset, l, n):
    if n>1:
        subsets = list(itertools.combinations(itemset, n))
    else:
        subsets = itemset
    for iter1 in subsets:
        if not iter1 in l:
            return False
    return True

candidate_set1, Item_set1 = k_1(items, Minimum_Support)
candidate_set2, Item_set2 = k_2(Item_set1, Input_Data, Minimum_Support)
candidate_set3, Item_set3 = k_3(Item_set2, Input_Data, Minimum_Support)
candidate_set4, Item_set4 = k_4(Item_set3, Input_Data, Minimum_Support)
print(" ")
print(" ")
print("Frequent Item set generated when k=1 => ", Item_set1)
print("Frequent Item set generated when k=2 => ", Item_set2)
print("Frequent Item set generated when k=3 => ", Item_set3)
print("Frequent Item set generated when k=4 => ", Item_set4)


itemlist = {**Item_set1, **Item_set2, **Item_set3, **Item_set4}
```

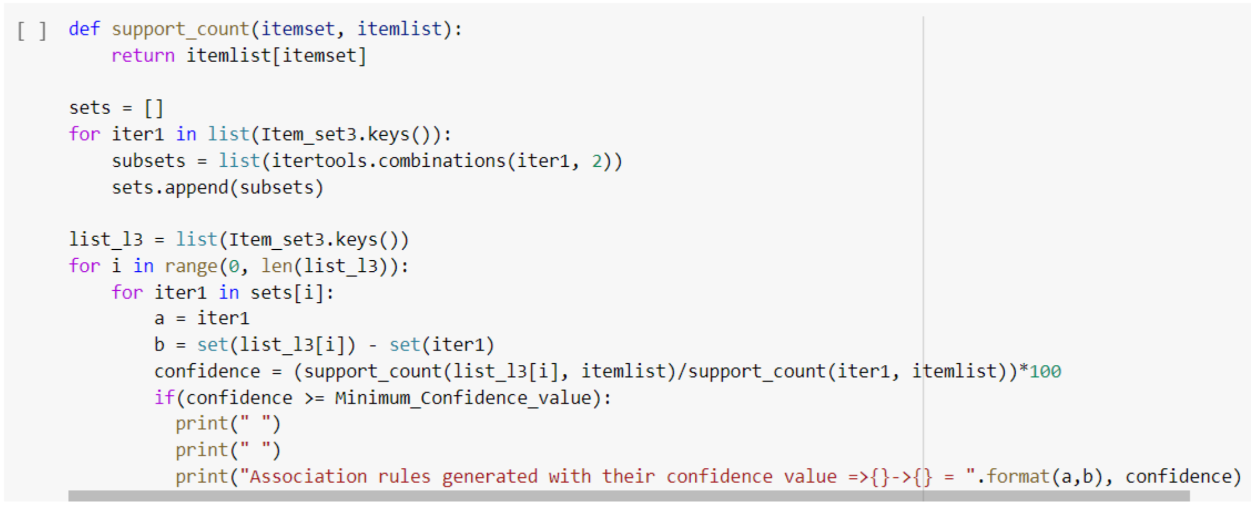
Frequent Itemset created:



```
Frequent Item set generated when k=1 =>  {'Anti-Virus': 5, 'Desk Top': 3, 'External Hard-Drive': 3, 'Flash Drive': 8, 'Lab To
p': 9, 'Lab Top Case': 4, 'Microsoft Office': 5, 'Printer': 6}
Frequent Item set generated when k=2 =>  {('Anti-Virus', 'External Hard-Drive'): 2, ('Anti-Virus', 'Flash Drive'): 4, ('Anti-Vi
rus', 'Lab Top'): 5, ('Anti-Virus', 'Lab Top Case'): 2, ('Anti-Virus', 'Microsoft Office'): 2, ('Anti-Virus', 'Printer'): 2,
('Desk Top', 'Flash Drive'): 2, ('Desk Top', 'Lab Top'): 2, ('Desk Top', 'Printer'): 2, ('External Hard-Drive', 'Lab Top'): 3,
('Flash Drive', 'Lab Top'): 7, ('Flash Drive', 'Lab Top Case'): 3, ('Flash Drive', 'Microsoft Office'): 5, ('Flash Drive', 'Pri
nter'): 6, ('Lab Top', 'Lab Top Case'): 4, ('Lab Top', 'Microsoft Office'): 4, ('Lab Top', 'Printer'): 5, ('Lab Top Case', 'Mic
rosoft Office'): 2, ('Microsoft Office', 'Printer'): 4}
Frequent Item set generated when k=3 =>  {('Anti-Virus', 'External Hard-Drive', 'Lab Top'): 2, ('Anti-Virus', 'Flash Drive', 'L
ab Top'): 4, ('Anti-Virus', 'Flash Drive', 'Lab Top Case'): 2, ('Anti-Virus', 'Flash Drive', 'Microsoft Office'): 2, ('Anti-Vir
us', 'Flash Drive', 'Printer'): 2, ('Anti-Virus', 'Lab Top', 'Lab Top Case'): 2, ('Anti-Virus', 'Lab Top', 'Microsoft Office'):
2, ('Anti-Virus', 'Lab Top', 'Printer'): 2, ('Desk Top', 'Flash Drive', 'Printer'): 2, ('Flash Drive', 'Lab Top', 'Lab Top Cas
e'): 3, ('Flash Drive', 'Lab Top', 'Microsoft Office'): 4, ('Flash Drive', 'Lab Top', 'Printer'): 5, ('Flash Drive', 'Lab Top C
ase', 'Microsoft Office'): 2, ('Flash Drive', 'Microsoft Office', 'Printer'): 4, ('Lab Top', 'Lab Top Case', 'Microsoft Offic
e'): 2, ('Lab Top', 'Microsoft Office', 'Printer'): 3}
Frequent Item set generated when k=4 =>  {('Anti-Virus', 'Flash Drive', 'Lab Top', 'Lab Top Case'): 2, ('Anti-Virus', 'Flash Dr
ive', 'Lab Top', 'Microsoft Office'): 2, ('Anti-Virus', 'Flash Drive', 'Lab Top', 'Printer'): 2, ('Flash Drive', 'Lab Top', 'La
b Top Case', 'Microsoft Office'): 2, ('Flash Drive', 'Lab Top', 'Microsoft Office', 'Printer'): 3}
```

(**Resultset was too wide for screenshot so zoom in to see the output clearly)

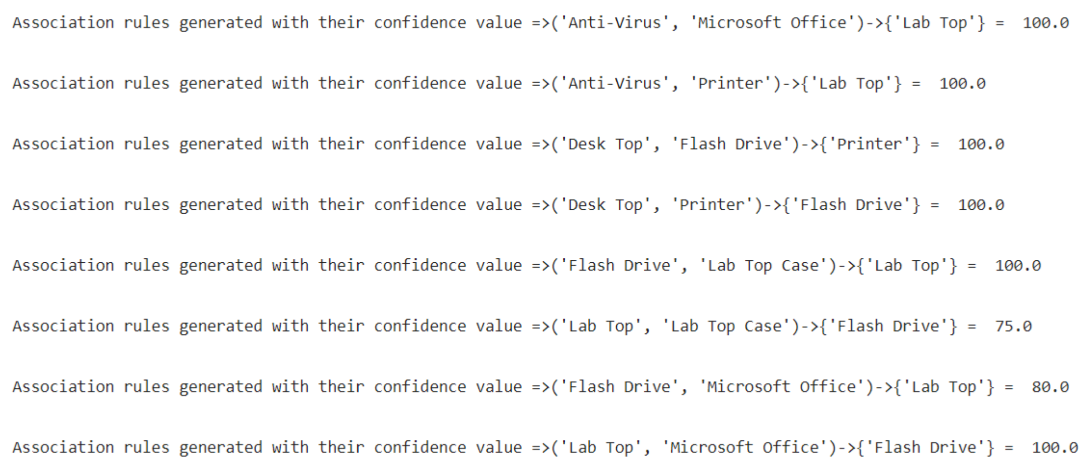
Calculating the Association rules based on the confidence value.



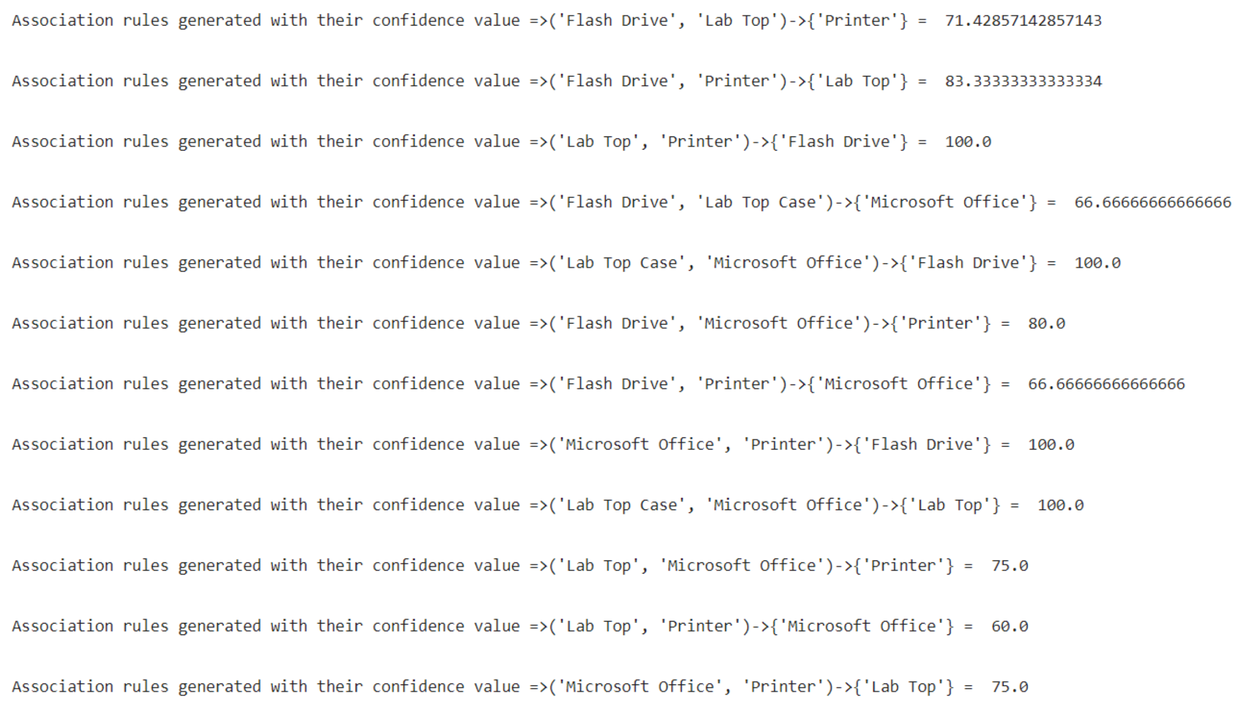
```
[ ] def support_count(itemset, itemlist):
        return itemlist[itemset]

    sets = []
    for iter1 in list(Item_set3.keys()):
        subsets = list(itertools.combinations(iter1, 2))
        sets.append(subsets)

    list_l3 = list(Item_set3.keys())
    for i in range(0, len(list_l3)):
        for iter1 in sets[i]:
            a = iter1
            b = set(list_l3[i]) - set(iter1)
            confidence = (support_count(list_l3[i], itemlist)/support_count(iter1, itemlist))*100
            if(confidence >= Minimum_Confidence_value):
              print(" ")
              print(" ")
              print("Association rules generated with their confidence value =>{}->{} = ".format(a,b), confidence)
```

Association Rules created with Confidence values :

```
Association rules generated with their confidence value =>('Anti-Virus', 'External Hard-Drive')->{'Lab Top'} =  100.0

Association rules generated with their confidence value =>('External Hard-Drive', 'Lab Top')->{'Anti-Virus'} =  66.66666666666666

Association rules generated with their confidence value =>('Anti-Virus', 'Flash Drive')->{'Lab Top'} =  100.0

Association rules generated with their confidence value =>('Anti-Virus', 'Lab Top')->{'Flash Drive'} =  80.0

Association rules generated with their confidence value =>('Anti-Virus', 'Lab Top Case')->{'Flash Drive'} =  100.0

Association rules generated with their confidence value =>('Flash Drive', 'Lab Top Case')->{'Anti-Virus'} =  66.66666666666666

Association rules generated with their confidence value =>('Anti-Virus', 'Microsoft Office')->{'Flash Drive'} =  100.0

Association rules generated with their confidence value =>('Anti-Virus', 'Printer')->{'Flash Drive'} =  100.0
```



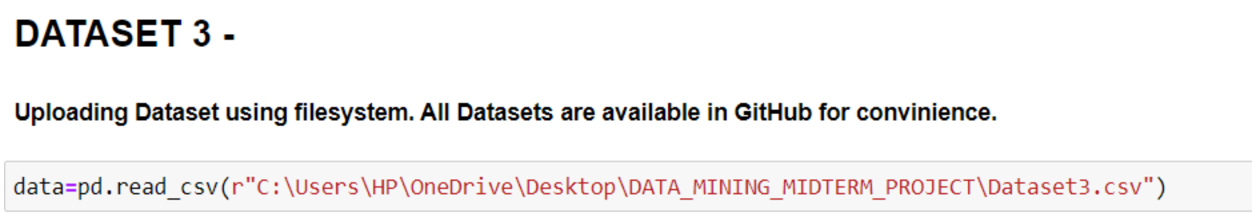
```
Association rules generated with their confidence value =>('Anti-Virus', 'Microsoft Office')->{'Lab Top'} =  100.0

Association rules generated with their confidence value =>('Anti-Virus', 'Printer')->{'Lab Top'} =  100.0

Association rules generated with their confidence value =>('Desk Top', 'Flash Drive')->{'Printer'} =  100.0

Association rules generated with their confidence value =>('Desk Top', 'Printer')->{'Flash Drive'} =  100.0

Association rules generated with their confidence value =>('Flash Drive', 'Lab Top Case')->{'Lab Top'} =  100.0

Association rules generated with their confidence value =>('Lab Top', 'Lab Top Case')->{'Flash Drive'} =  75.0

Association rules generated with their confidence value =>('Flash Drive', 'Microsoft Office')->{'Lab Top'} =  80.0

Association rules generated with their confidence value =>('Lab Top', 'Microsoft Office')->{'Flash Drive'} =  100.0
```



```
Association rules generated with their confidence value =>('Flash Drive', 'Lab Top')->{'Printer'} =  71.42857142857143

Association rules generated with their confidence value =>('Flash Drive', 'Printer')->{'Lab Top'} =  83.33333333333334

Association rules generated with their confidence value =>('Lab Top', 'Printer')->{'Flash Drive'} =  100.0

Association rules generated with their confidence value =>('Flash Drive', 'Lab Top Case')->{'Microsoft Office'} =  66.66666666666666

Association rules generated with their confidence value =>('Lab Top Case', 'Microsoft Office')->{'Flash Drive'} =  100.0

Association rules generated with their confidence value =>('Flash Drive', 'Microsoft Office')->{'Printer'} =  80.0

Association rules generated with their confidence value =>('Flash Drive', 'Printer')->{'Microsoft Office'} =  66.66666666666666

Association rules generated with their confidence value =>('Microsoft Office', 'Printer')->{'Flash Drive'} =  100.0

Association rules generated with their confidence value =>('Lab Top Case', 'Microsoft Office')->{'Lab Top'} =  100.0

Association rules generated with their confidence value =>('Lab Top', 'Microsoft Office')->{'Printer'} =  75.0

Association rules generated with their confidence value =>('Lab Top', 'Printer')->{'Microsoft Office'} =  60.0

Association rules generated with their confidence value =>('Microsoft Office', 'Printer')->{'Lab Top'} =  75.0
```

(**Resultset was too wide for screenshot so zoom in to see the output clearly)

Dataset 3 (K-Mart) :

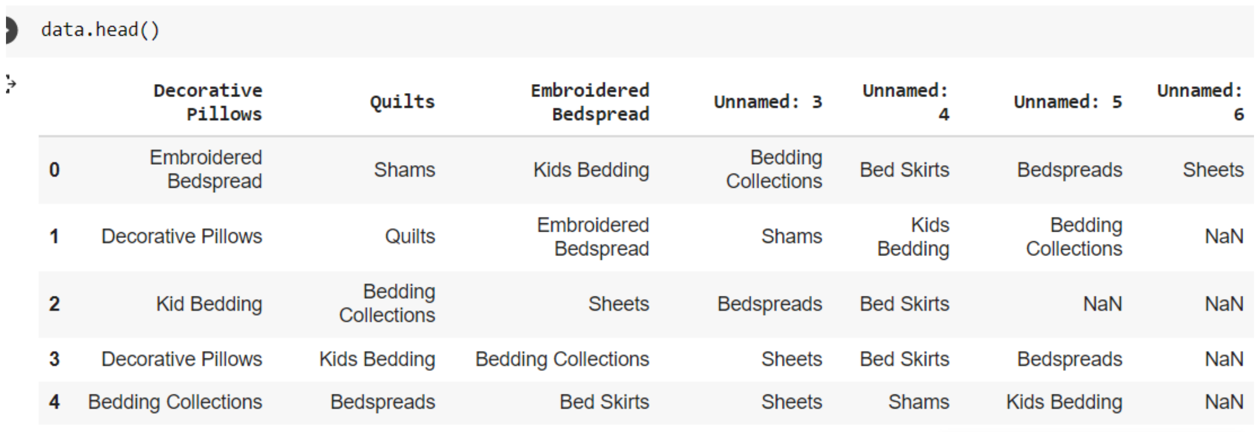


DATASET 3 -

Uploading Dataset using filesystem. All Datasets are available in GitHub for convinience.

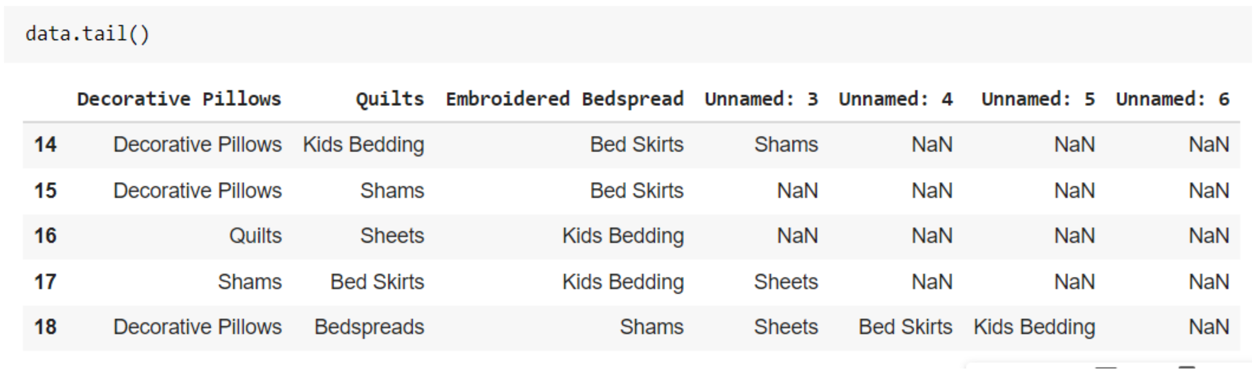
```
data=pd.read_csv(r"C:\Users\HP\OneDrive\Desktop\DATA_MINING_MIDTERM_PROJECT\Dataset3.csv")
```

Data Representation :



```
data.head()
```

	Decorative Pillows	Quilts	Embroidered Bedspread	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6
0	Embroidered Bedspread	Shams	Kids Bedding	Bedding Collections	Bed Skirts	Bedspreads	Sheets
1	Decorative Pillows	Quilts	Embroidered Bedspread	Shams	Kids Bedding	Bedding Collections	NaN
2	Kid Bedding	Bedding Collections	Sheets	Bedspreads	Bed Skirts	NaN	NaN
3	Decorative Pillows	Kids Bedding	Bedding Collections	Sheets	Bed Skirts	Bedspreads	NaN
4	Bedding Collections	Bedspreads	Bed Skirts	Sheets	Shams	Kids Bedding	NaN



```
data.tail()
```

	Decorative Pillows	Quilts	Embroidered Bedspread	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6
14	Decorative Pillows	Kids Bedding	Bed Skirts	Shams	NaN	NaN	NaN
15	Decorative Pillows	Shams	Bed Skirts	NaN	NaN	NaN	NaN
16	Quilts	Sheets	Kids Bedding	NaN	NaN	NaN	NaN
17	Shams	Bed Skirts	Kids Bedding	Sheets	NaN	NaN	NaN
18	Decorative Pillows	Bedspreads	Shams	Sheets	Bed Skirts	Kids Bedding	NaN

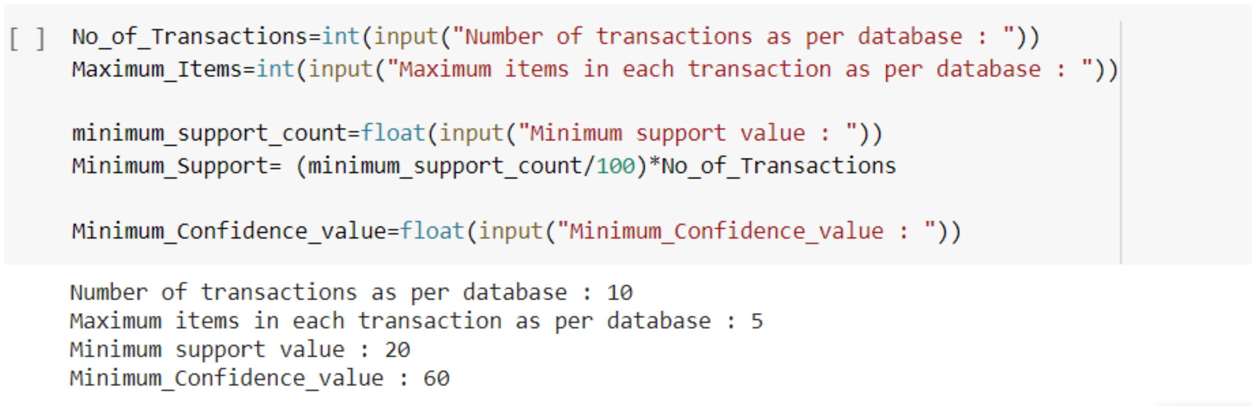
Getting input from the user for the following :

1.Number of Transactions (Max possible value = 20)

2.Maximum items per transactions (Max possible value = 5)

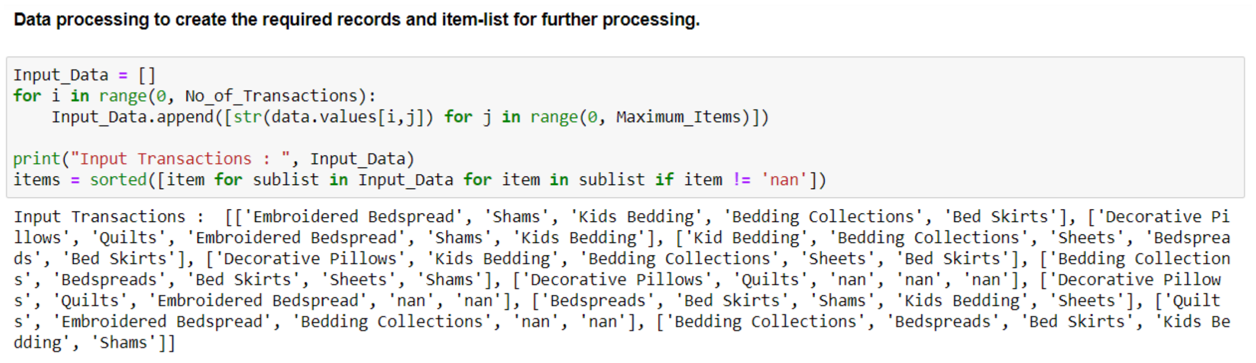
3.Minimum value of support (Min possible value = 20)

4.Minimum confidence value (Maximum possible value =100)



```
[ ]  No_of_Transactions=int(input("Number of transactions as per database : "))
     Maximum_Items=int(input("Maximum items in each transaction as per database : "))

     minimum_support_count=float(input("Minimum support value : "))
     Minimum_Support= (minimum_support_count/100)*No_of_Transactions

     Minimum_Confidence_value=float(input("Minimum_Confidence_value : "))
```

```
     Number of transactions as per database : 10
     Maximum items in each transaction as per database : 5
     Minimum support value : 20
     Minimum_Confidence_value : 60
```



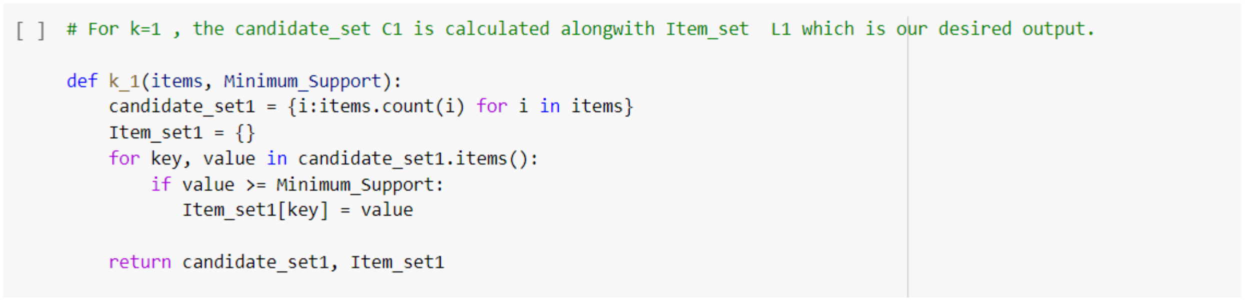
Data processing to create the required records and item-list for further processing.

```
Input_Data = []
for i in range(0, No_of_Transactions):
    Input_Data.append([str(data.values[i,j]) for j in range(0, Maximum_Items)])

print("Input Transactions : ", Input_Data)
items = sorted([item for sublist in Input_Data for item in sublist if item != 'nan'])
```

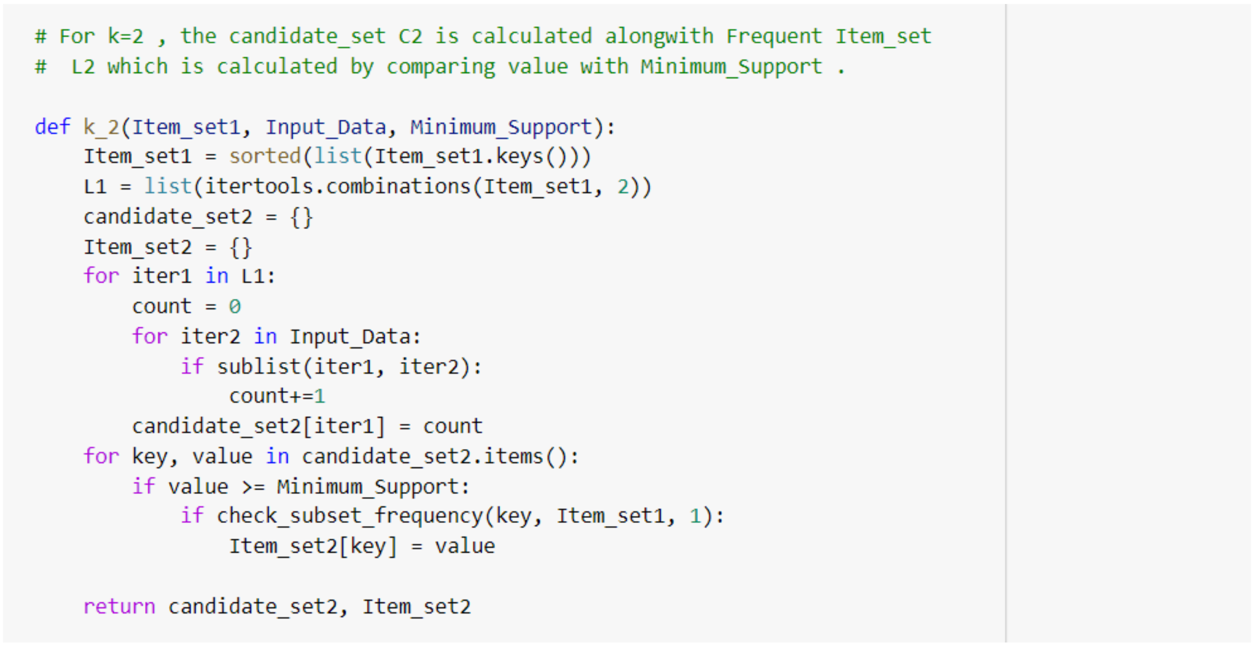
```
Input Transactions :  [['Embroidered Bedspread', 'Shams', 'Kids Bedding', 'Bedding Collections', 'Bed Skirts'], ['Decorative Pi
llows', 'Quilts', 'Embroidered Bedspread', 'Shams', 'Kids Bedding'], ['Kid Bedding', 'Bedding Collections', 'Sheets', 'Bedsprea
ds', 'Bed Skirts'], ['Decorative Pillows', 'Kids Bedding', 'Bedding Collections', 'Sheets', 'Bed Skirts'], ['Bedding Collection
s', 'Bedspreads', 'Bed Skirts', 'Sheets', 'Shams'], ['Decorative Pillows', 'Quilts', 'nan', 'nan', 'nan'], ['Decorative Pillow
s', 'Quilts', 'Embroidered Bedspread', 'nan', 'nan'], ['Bedspreads', 'Bed Skirts', 'Shams', 'Kids Bedding', 'Sheets'], ['Quilt
s', 'Embroidered Bedspread', 'Bedding Collections', 'nan', 'nan'], ['Bedding Collections', 'Bedspreads', 'Bed Skirts', 'Kids Be
dding', 'Shams']]
```

For k=1 to 4 , candidate_set and the frequent ItemSet are calculated by comparing the support count of each item in the list to the Minimum Support value. We check if all the subsets in itemset are frequent using the check subset frequency function and if not, we remove respective itemset from the list by comparing the length of the two possible lists using the sublist function.

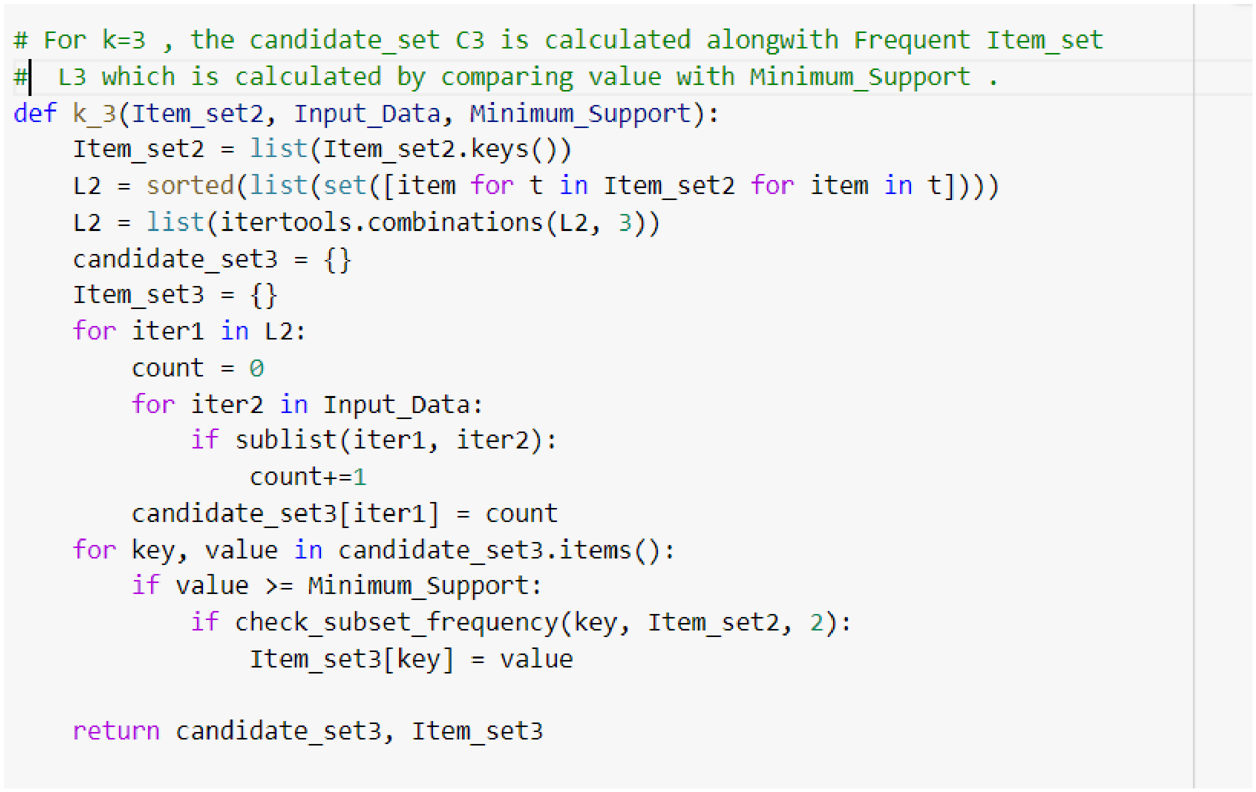
We are printing all the Item_Sets generated after calculations.



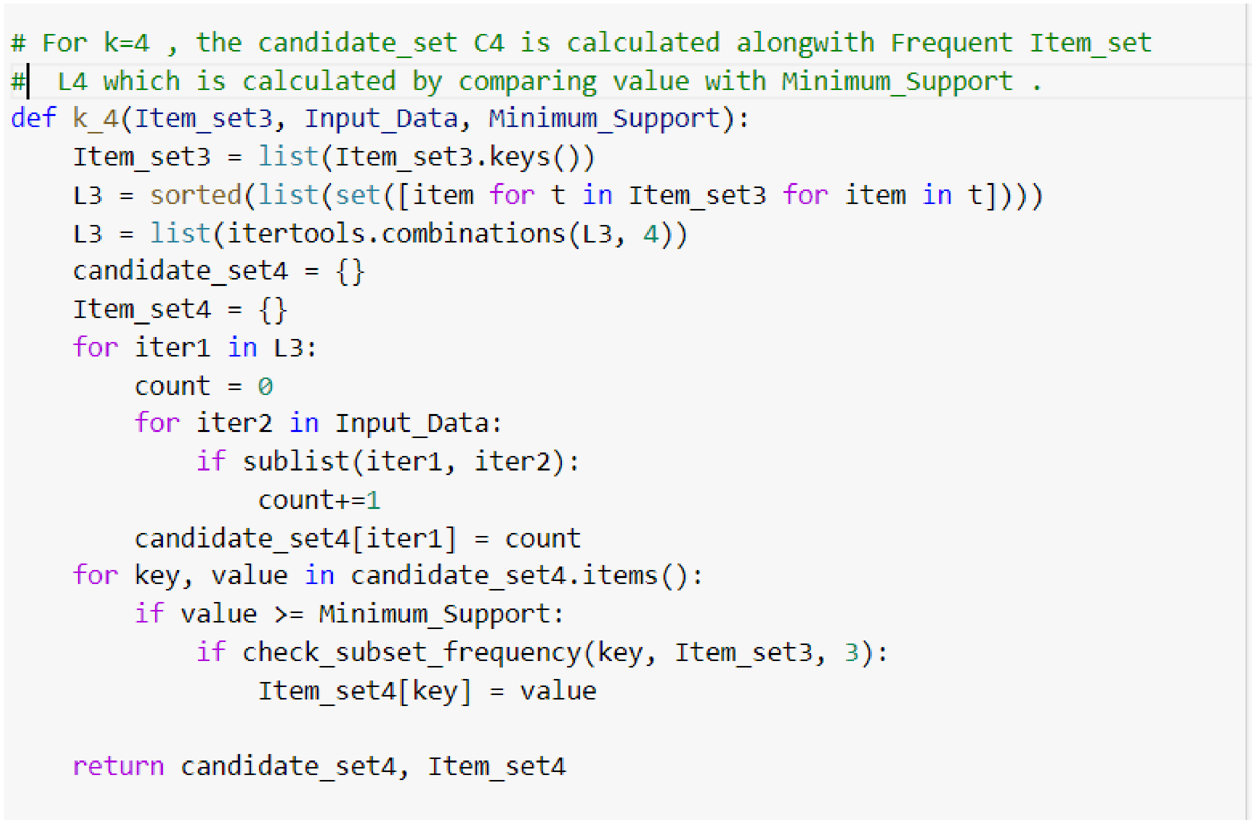
```
[ ] # For k=1 , the candidate_set C1 is calculated alongwith Item_set  L1 which is our desired output.

    def k_1(items, Minimum_Support):
        candidate_set1 = {i:items.count(i) for i in items}
        Item_set1 = {}
        for key, value in candidate_set1.items():
            if value >= Minimum_Support:
               Item_set1[key] = value

        return candidate_set1, Item_set1
```



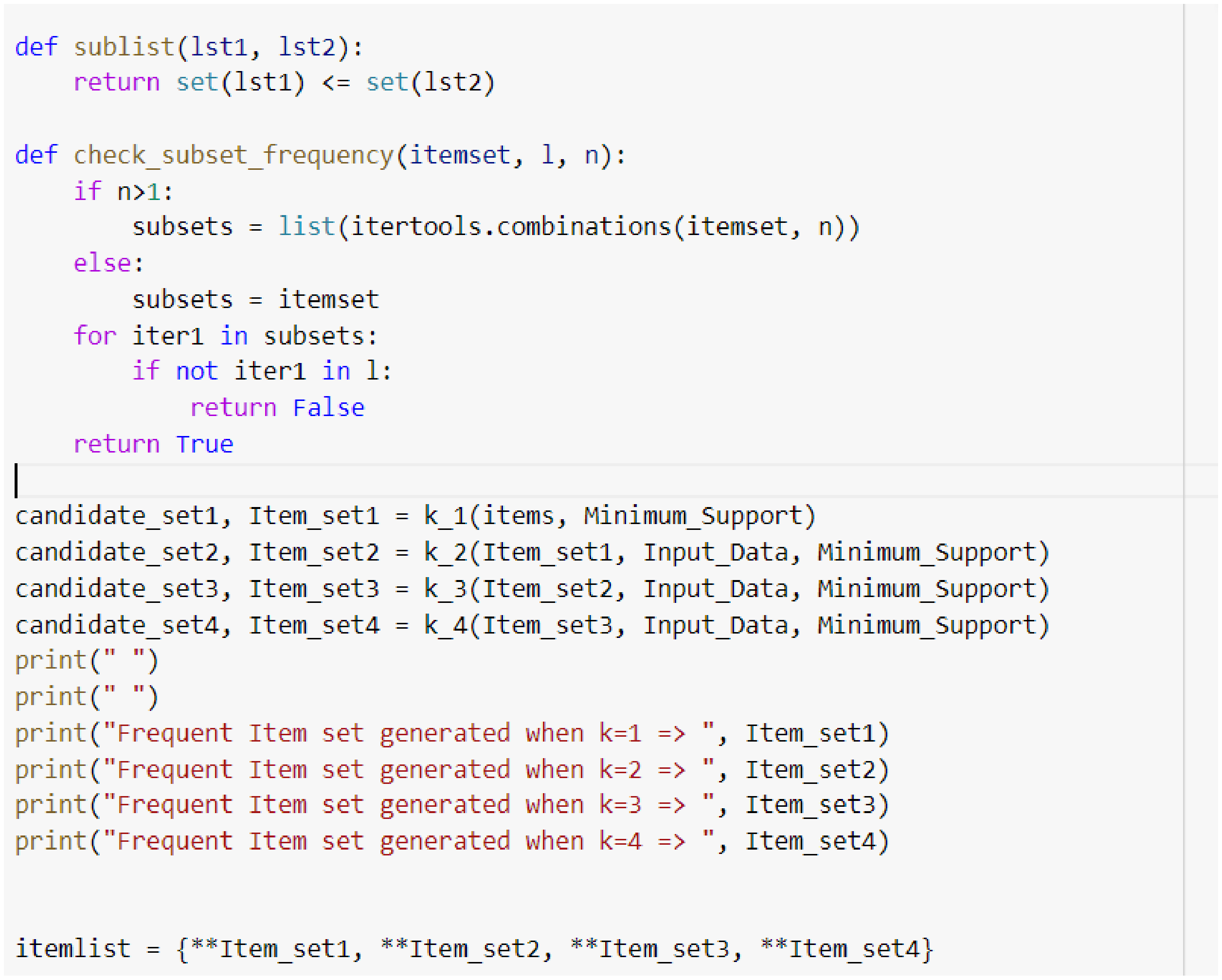
```
# For k=2 , the candidate_set C2 is calculated alongwith Frequent Item_set
#  L2 which is calculated by comparing value with Minimum_Support .

def k_2(Item_set1, Input_Data, Minimum_Support):
    Item_set1 = sorted(list(Item_set1.keys()))
    L1 = list(itertools.combinations(Item_set1, 2))
    candidate_set2 = {}
    Item_set2 = {}
    for iter1 in L1:
        count = 0
        for iter2 in Input_Data:
            if sublist(iter1, iter2):
                count+=1
        candidate_set2[iter1] = count
    for key, value in candidate_set2.items():
        if value >= Minimum_Support:
            if check_subset_frequency(key, Item_set1, 1):
                Item_set2[key] = value

    return candidate_set2, Item_set2
```



```python
# For k=3 , the candidate_set C3 is calculated alongwith Frequent Item_set
#  L3 which is calculated by comparing value with Minimum_Support .
def k_3(Item_set2, Input_Data, Minimum_Support):
    Item_set2 = list(Item_set2.keys())
    L2 = sorted(list(set([item for t in Item_set2 for item in t])))
    L2 = list(itertools.combinations(L2, 3))
    candidate_set3 = {}
    Item_set3 = {}
    for iter1 in L2:
        count = 0
        for iter2 in Input_Data:
            if sublist(iter1, iter2):
                count+=1
        candidate_set3[iter1] = count
    for key, value in candidate_set3.items():
        if value >= Minimum_Support:
            if check_subset_frequency(key, Item_set2, 2):
                Item_set3[key] = value

    return candidate_set3, Item_set3
```

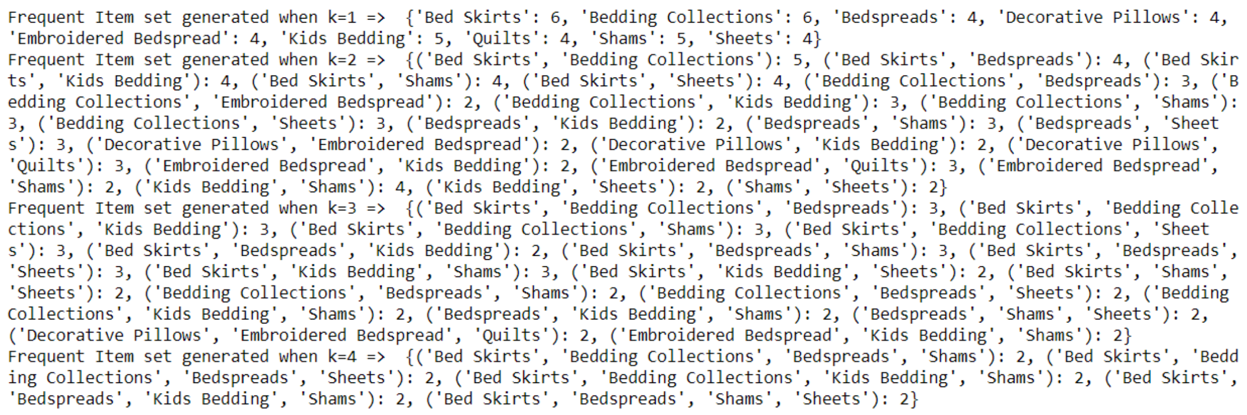


```python
# For k=4 , the candidate_set C4 is calculated alongwith Frequent Item_set
#  L4 which is calculated by comparing value with Minimum_Support .
def k_4(Item_set3, Input_Data, Minimum_Support):
    Item_set3 = list(Item_set3.keys())
    L3 = sorted(list(set([item for t in Item_set3 for item in t])))
    L3 = list(itertools.combinations(L3, 4))
    candidate_set4 = {}
    Item_set4 = {}
    for iter1 in L3:
        count = 0
        for iter2 in Input_Data:
            if sublist(iter1, iter2):
                count+=1
        candidate_set4[iter1] = count
    for key, value in candidate_set4.items():
        if value >= Minimum_Support:
            if check_subset_frequency(key, Item_set3, 3):
                Item_set4[key] = value

    return candidate_set4, Item_set4
```



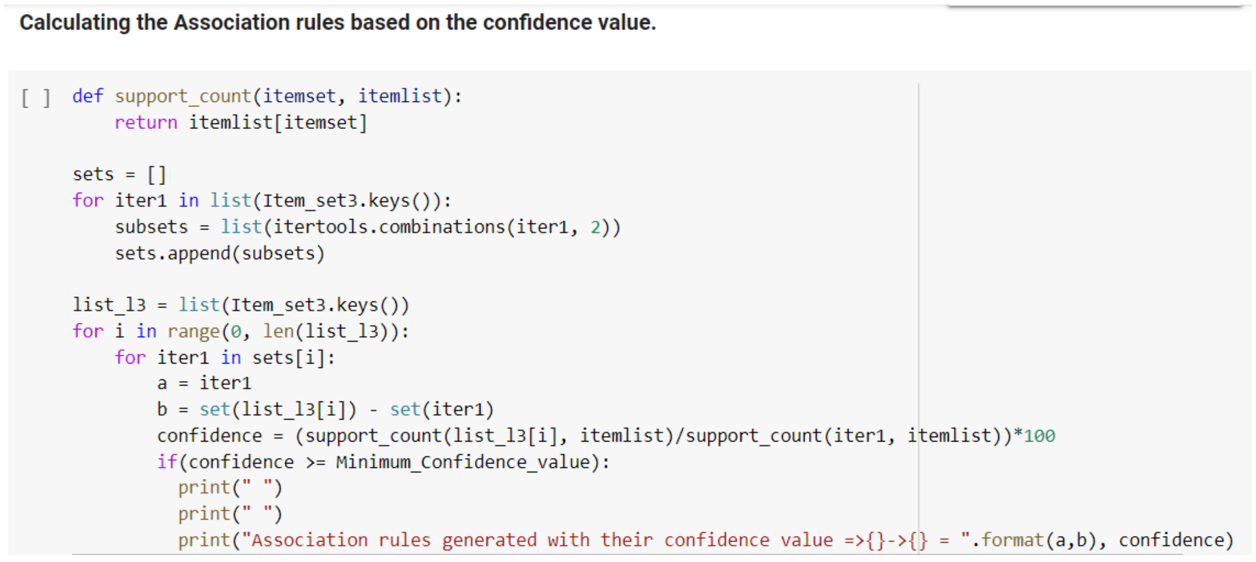
```python
def sublist(lst1, lst2):
    return set(lst1) <= set(lst2)

def check_subset_frequency(itemset, l, n):
    if n>1:
        subsets = list(itertools.combinations(itemset, n))
    else:
        subsets = itemset
    for iter1 in subsets:
        if not iter1 in l:
            return False
    return True

candidate_set1, Item_set1 = k_1(items, Minimum_Support)
candidate_set2, Item_set2 = k_2(Item_set1, Input_Data, Minimum_Support)
candidate_set3, Item_set3 = k_3(Item_set2, Input_Data, Minimum_Support)
candidate_set4, Item_set4 = k_4(Item_set3, Input_Data, Minimum_Support)
print(" ")
print(" ")
print("Frequent Item set generated when k=1 => ", Item_set1)
print("Frequent Item set generated when k=2 => ", Item_set2)
print("Frequent Item set generated when k=3 => ", Item_set3)
print("Frequent Item set generated when k=4 => ", Item_set4)


itemlist = {**Item_set1, **Item_set2, **Item_set3, **Item_set4}
```

Frequent Itemsets created :



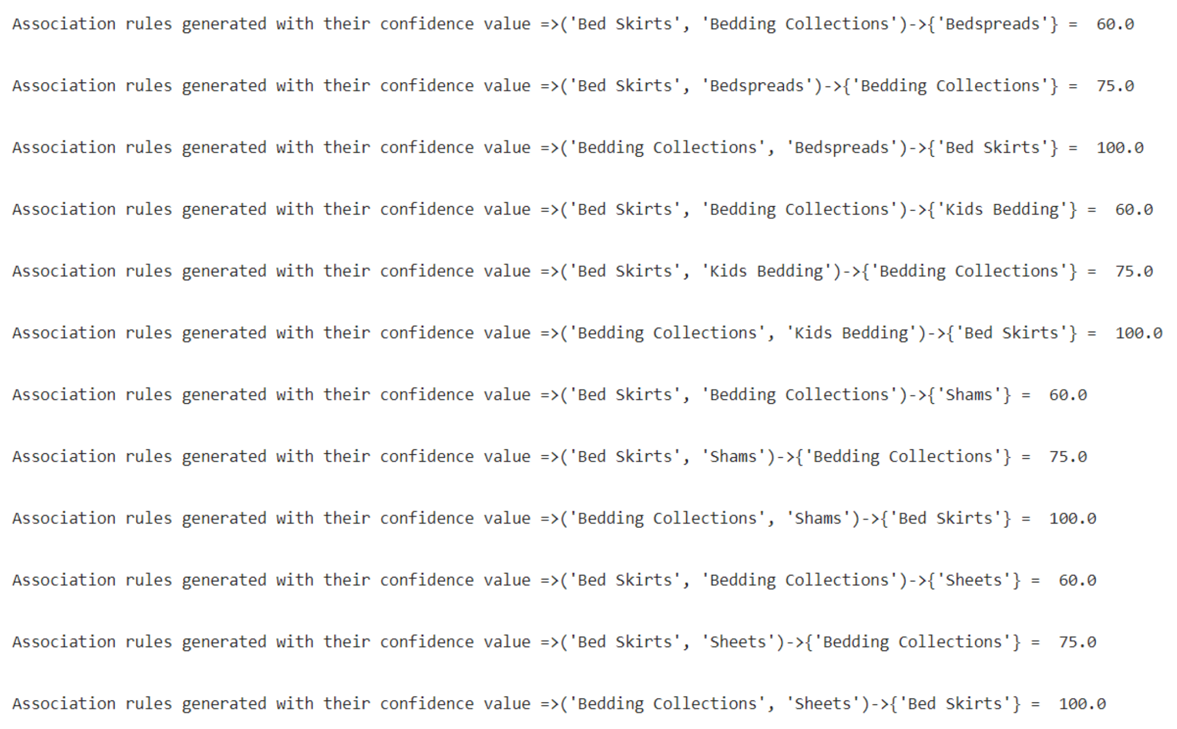
```
Frequent Item set generated when k=1 =>  {'Bed Skirts': 6, 'Bedding Collections': 6, 'Bedspreads': 4, 'Decorative Pillows': 4,
'Embroidered Bedspread': 4, 'Kids Bedding': 5, 'Quilts': 4, 'Shams': 5, 'Sheets': 4}
Frequent Item set generated when k=2 =>  {('Bed Skirts', 'Bedding Collections'): 5, ('Bed Skirts', 'Bedspreads'): 4, ('Bed Skir
ts', 'Kids Bedding'): 4, ('Bed Skirts', 'Shams'): 4, ('Bed Skirts', 'Sheets'): 4, ('Bedding Collections', 'Bedspreads'): 3, ('B
edding Collections', 'Embroidered Bedspread'): 2, ('Bedding Collections', 'Kids Bedding'): 3, ('Bedding Collections', 'Shams'):
3, ('Bedding Collections', 'Sheets'): 3, ('Bedspreads', 'Kids Bedding'): 2, ('Bedspreads', 'Shams'): 3, ('Bedspreads', 'Sheet
s'): 3, ('Decorative Pillows', 'Embroidered Bedspread'): 2, ('Decorative Pillows', 'Kids Bedding'): 2, ('Decorative Pillows',
'Quilts'): 3, ('Embroidered Bedspread', 'Kids Bedding'): 2, ('Embroidered Bedspread', 'Quilts'): 3, ('Embroidered Bedspread',
'Shams'): 2, ('Kids Bedding', 'Shams'): 4, ('Kids Bedding', 'Sheets'): 2, ('Shams', 'Sheets'): 2}
Frequent Item set generated when k=3 =>  {('Bed Skirts', 'Bedding Collections', 'Bedspreads'): 3, ('Bed Skirts', 'Bedding Colle
ctions', 'Kids Bedding'): 3, ('Bed Skirts', 'Bedding Collections', 'Shams'): 3, ('Bed Skirts', 'Bedding Collections', 'Sheet
s'): 3, ('Bed Skirts', 'Bedspreads', 'Kids Bedding'): 2, ('Bed Skirts', 'Bedspreads', 'Shams'): 3, ('Bed Skirts', 'Bedspreads',
'Sheets'): 3, ('Bed Skirts', 'Kids Bedding', 'Shams'): 3, ('Bed Skirts', 'Kids Bedding', 'Sheets'): 2, ('Bed Skirts', 'Shams',
'Sheets'): 2, ('Bedding Collections', 'Bedspreads', 'Shams'): 2, ('Bedding Collections', 'Bedspreads', 'Sheets'): 2, ('Bedding
Collections', 'Kids Bedding', 'Shams'): 2, ('Bedspreads', 'Kids Bedding', 'Shams'): 2, ('Bedspreads', 'Shams', 'Sheets'): 2,
('Decorative Pillows', 'Embroidered Bedspread', 'Quilts'): 2, ('Embroidered Bedspread', 'Kids Bedding', 'Shams'): 2}
Frequent Item set generated when k=4 =>  {('Bed Skirts', 'Bedding Collections', 'Bedspreads', 'Shams'): 2, ('Bed Skirts', 'Bedd
ing Collections', 'Bedspreads', 'Sheets'): 2, ('Bed Skirts', 'Bedding Collections', 'Kids Bedding', 'Shams'): 2, ('Bed Skirts',
'Bedspreads', 'Kids Bedding', 'Shams'): 2, ('Bed Skirts', 'Bedspreads', 'Shams', 'Sheets'): 2}
```

(**Resultset was too wide for screenshot so zoom in to see the output clearly)

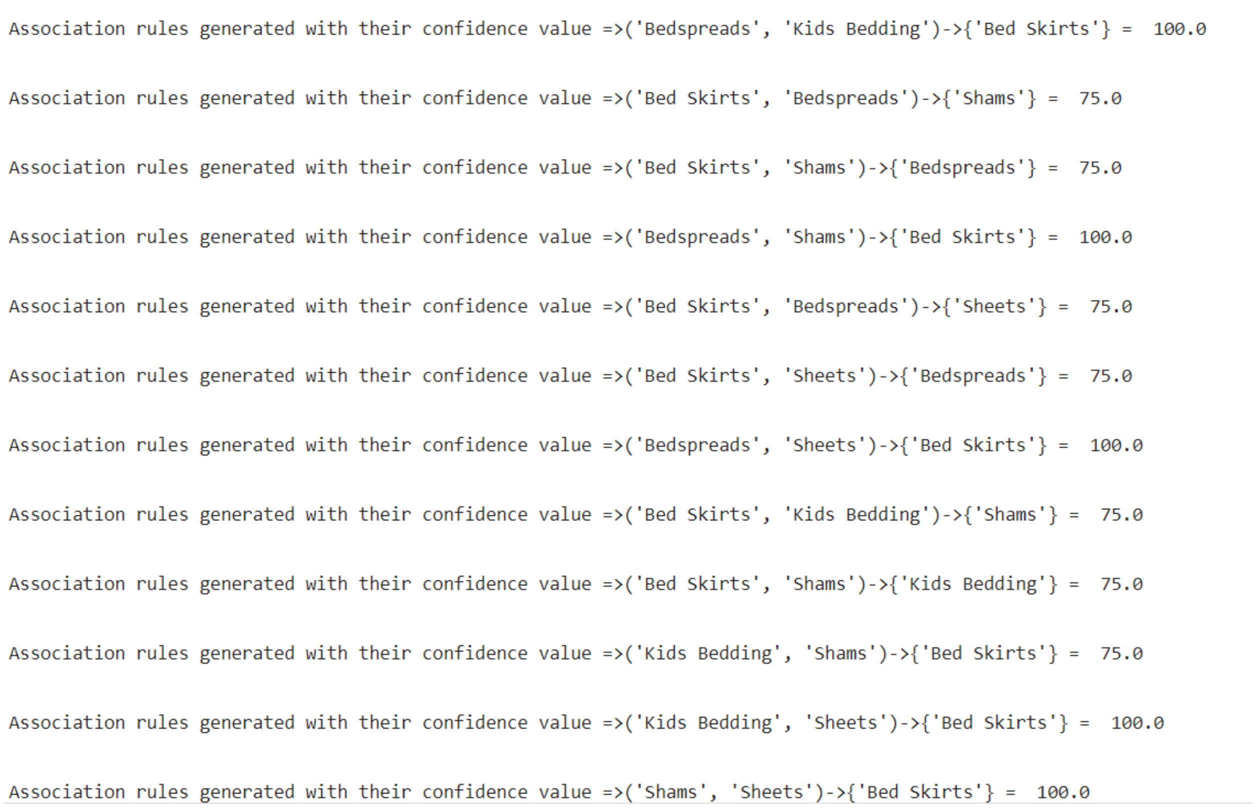

Calculating the Association rules based on the confidence value.

```
[ ]  def support_count(itemset, itemlist):
         return itemlist[itemset]

     sets = []
     for iter1 in list(Item_set3.keys()):
         subsets = list(itertools.combinations(iter1, 2))
         sets.append(subsets)

     list_l3 = list(Item_set3.keys())
     for i in range(0, len(list_l3)):
         for iter1 in sets[i]:
             a = iter1
             b = set(list_l3[i]) - set(iter1)
             confidence = (support_count(list_l3[i], itemlist)/support_count(iter1, itemlist))*100
             if(confidence >= Minimum_Confidence_value):
               print(" ")
               print(" ")
               print("Association rules generated with their confidence value =>{}->{} = ".format(a,b), confidence)
```

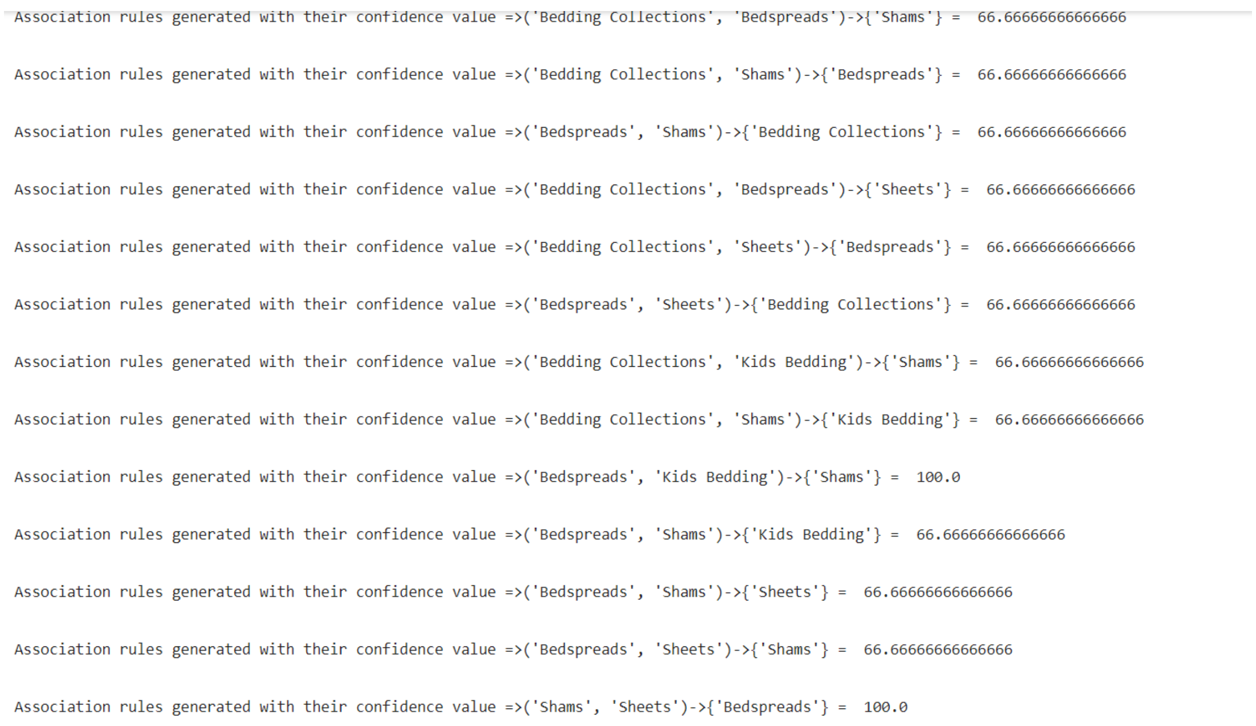
Association Rules with Confidence values :



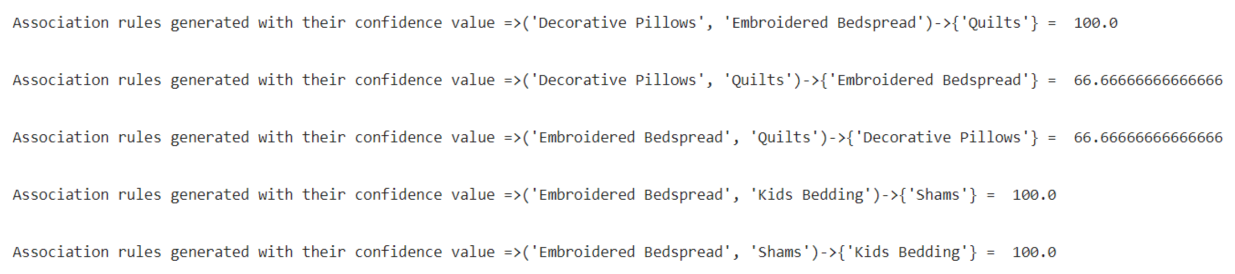
```
Association rules generated with their confidence value =>('Bed Skirts', 'Bedding Collections')->{'Bedspreads'} =  60.0

Association rules generated with their confidence value =>('Bed Skirts', 'Bedspreads')->{'Bedding Collections'} =  75.0

Association rules generated with their confidence value =>('Bedding Collections', 'Bedspreads')->{'Bed Skirts'} =  100.0

Association rules generated with their confidence value =>('Bed Skirts', 'Bedding Collections')->{'Kids Bedding'} =  60.0

Association rules generated with their confidence value =>('Bed Skirts', 'Kids Bedding')->{'Bedding Collections'} =  75.0

Association rules generated with their confidence value =>('Bedding Collections', 'Kids Bedding')->{'Bed Skirts'} =  100.0

Association rules generated with their confidence value =>('Bed Skirts', 'Bedding Collections')->{'Shams'} =  60.0

Association rules generated with their confidence value =>('Bed Skirts', 'Shams')->{'Bedding Collections'} =  75.0

Association rules generated with their confidence value =>('Bedding Collections', 'Shams')->{'Bed Skirts'} =  100.0

Association rules generated with their confidence value =>('Bed Skirts', 'Bedding Collections')->{'Sheets'} =  60.0

Association rules generated with their confidence value =>('Bed Skirts', 'Sheets')->{'Bedding Collections'} =  75.0

Association rules generated with their confidence value =>('Bedding Collections', 'Sheets')->{'Bed Skirts'} =  100.0
```



```
Association rules generated with their confidence value =>('Bedspreads', 'Kids Bedding')->{'Bed Skirts'} =  100.0

Association rules generated with their confidence value =>('Bed Skirts', 'Bedspreads')->{'Shams'} =  75.0

Association rules generated with their confidence value =>('Bed Skirts', 'Shams')->{'Bedspreads'} =  75.0

Association rules generated with their confidence value =>('Bedspreads', 'Shams')->{'Bed Skirts'} =  100.0

Association rules generated with their confidence value =>('Bed Skirts', 'Bedspreads')->{'Sheets'} =  75.0

Association rules generated with their confidence value =>('Bed Skirts', 'Sheets')->{'Bedspreads'} =  75.0

Association rules generated with their confidence value =>('Bedspreads', 'Sheets')->{'Bed Skirts'} =  100.0

Association rules generated with their confidence value =>('Bed Skirts', 'Kids Bedding')->{'Shams'} =  75.0

Association rules generated with their confidence value =>('Bed Skirts', 'Shams')->{'Kids Bedding'} =  75.0

Association rules generated with their confidence value =>('Kids Bedding', 'Shams')->{'Bed Skirts'} =  75.0

Association rules generated with their confidence value =>('Kids Bedding', 'Sheets')->{'Bed Skirts'} =  100.0

Association rules generated with their confidence value =>('Shams', 'Sheets')->{'Bed Skirts'} =  100.0
```



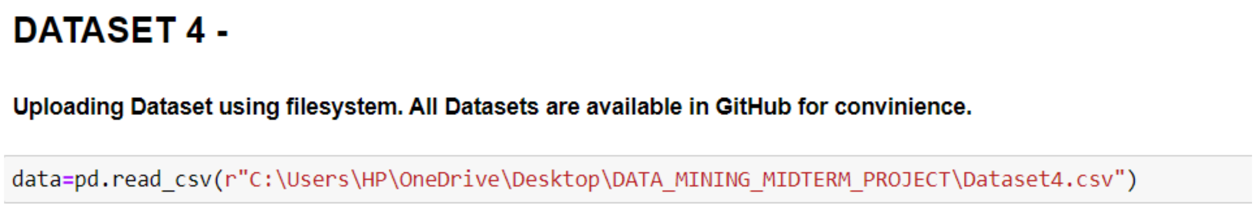
```
Association rules generated with their confidence value =>('Bedding Collections', 'Bedspreads')->{'Shams'} =  66.66666666666666

Association rules generated with their confidence value =>('Bedding Collections', 'Shams')->{'Bedspreads'} =  66.66666666666666

Association rules generated with their confidence value =>('Bedspreads', 'Shams')->{'Bedding Collections'} =  66.66666666666666

Association rules generated with their confidence value =>('Bedding Collections', 'Bedspreads')->{'Sheets'} =  66.66666666666666

Association rules generated with their confidence value =>('Bedding Collections', 'Sheets')->{'Bedspreads'} =  66.66666666666666

Association rules generated with their confidence value =>('Bedspreads', 'Sheets')->{'Bedding Collections'} =  66.66666666666666

Association rules generated with their confidence value =>('Bedding Collections', 'Kids Bedding')->{'Shams'} =  66.66666666666666

Association rules generated with their confidence value =>('Bedding Collections', 'Shams')->{'Kids Bedding'} =  66.66666666666666

Association rules generated with their confidence value =>('Bedspreads', 'Kids Bedding')->{'Shams'} =  100.0

Association rules generated with their confidence value =>('Bedspreads', 'Shams')->{'Kids Bedding'} =  66.66666666666666

Association rules generated with their confidence value =>('Bedspreads', 'Shams')->{'Sheets'} =  66.66666666666666

Association rules generated with their confidence value =>('Bedspreads', 'Sheets')->{'Shams'} =  66.66666666666666

Association rules generated with their confidence value =>('Shams', 'Sheets')->{'Bedspreads'} =  100.0
```



```
Association rules generated with their confidence value =>('Decorative Pillows', 'Embroidered Bedspread')->{'Quilts'} =  100.0

Association rules generated with their confidence value =>('Decorative Pillows', 'Quilts')->{'Embroidered Bedspread'} =  66.66666666666666

Association rules generated with their confidence value =>('Embroidered Bedspread', 'Quilts')->{'Decorative Pillows'} =  66.66666666666666

Association rules generated with their confidence value =>('Embroidered Bedspread', 'Kids Bedding')->{'Shams'} =  100.0

Association rules generated with their confidence value =>('Embroidered Bedspread', 'Shams')->{'Kids Bedding'} =  100.0
```

(**Resultset was too wide for screenshot so zoom in to see the output clearly)

Dataset 4 (Nike) :

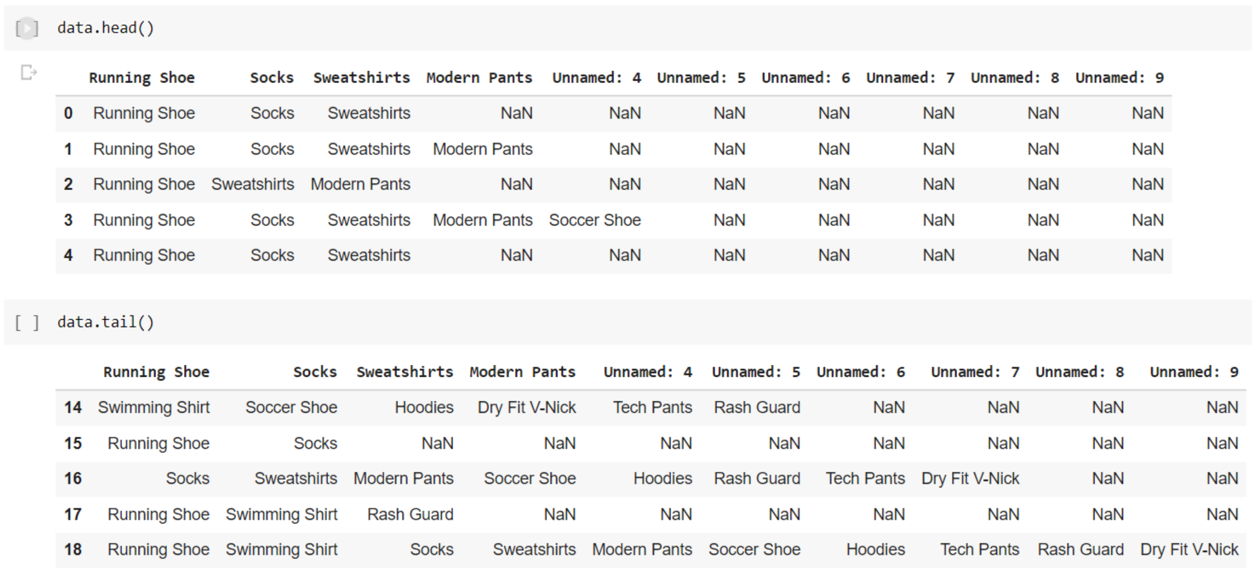


DATASET 4 -

Uploading Dataset using filesystem. All Datasets are available in GitHub for convinience.

```
data=pd.read_csv(r"C:\Users\HP\OneDrive\Desktop\DATA_MINING_MIDTERM_PROJECT\Dataset4.csv")
```

Data Representation :



```
data.head()
```

	Running Shoe	Socks	Sweatshirts	Modern Pants	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9
0	Running Shoe	Socks	Sweatshirts	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	Running Shoe	Socks	Sweatshirts	Modern Pants	NaN	NaN	NaN	NaN	NaN	NaN
2	Running Shoe	Sweatshirts	Modern Pants	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	Running Shoe	Socks	Sweatshirts	Modern Pants	Soccer Shoe	NaN	NaN	NaN	NaN	NaN
4	Running Shoe	Socks	Sweatshirts	NaN	NaN	NaN	NaN	NaN	NaN	NaN

```
[ ] data.tail()
```

	Running Shoe	Socks	Sweatshirts	Modern Pants	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9
14	Swimming Shirt	Soccer Shoe	Hoodies	Dry Fit V-Nick	Tech Pants	Rash Guard	NaN	NaN	NaN	NaN
15	Running Shoe	Socks	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
16	Socks	Sweatshirts	Modern Pants	Soccer Shoe	Hoodies	Rash Guard	Tech Pants	Dry Fit V-Nick	NaN	NaN
17	Running Shoe	Swimming Shirt	Rash Guard	NaN	NaN	NaN	NaN	NaN	NaN	NaN
18	Running Shoe	Swimming Shirt	Socks	Sweatshirts	Modern Pants	Soccer Shoe	Hoodies	Tech Pants	Rash Guard	Dry Fit V-Nick

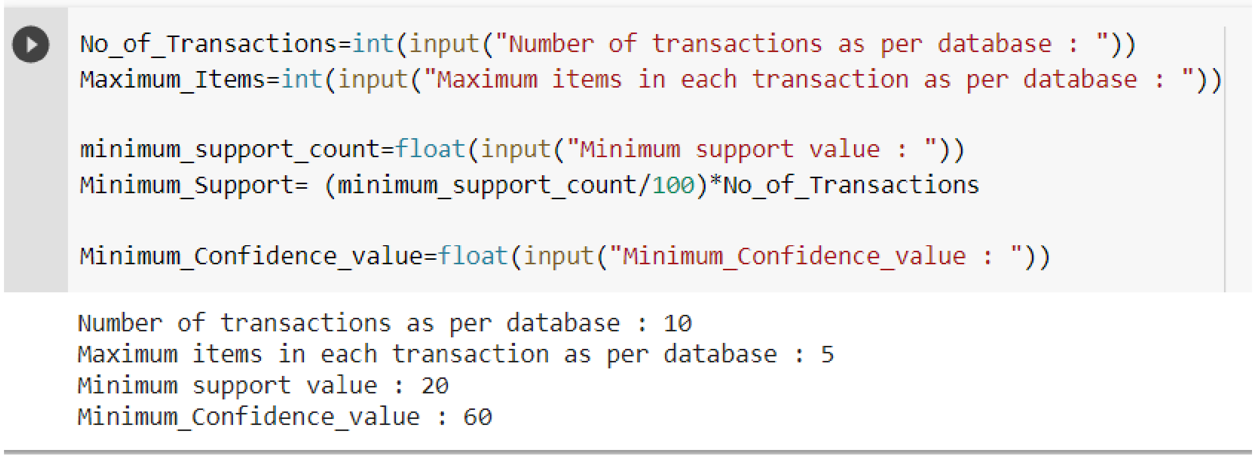
Getting input from the user for the following :

1.Number of Transactions (Max possible value = 20)

2.Maximum items per transactions (Max possible value = 5)

3.Minimum value of support (Min possible value = 20)

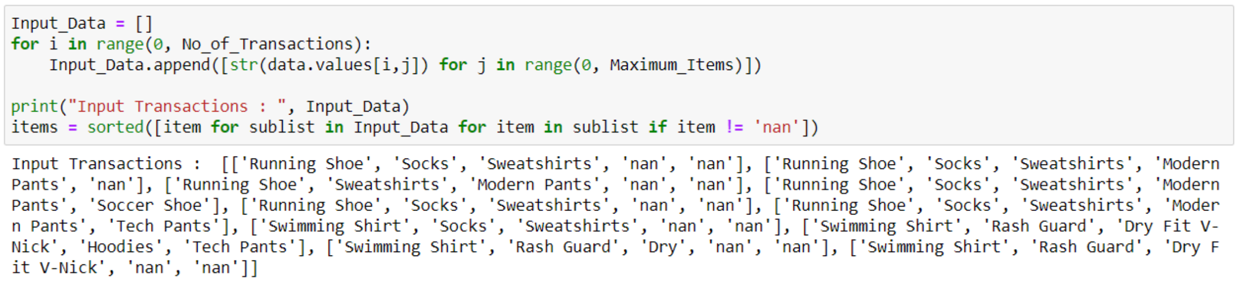
4.Minimum confidence value (Maximum possible value =100)



```
No_of_Transactions=int(input("Number of transactions as per database : "))
Maximum_Items=int(input("Maximum items in each transaction as per database : "))

minimum_support_count=float(input("Minimum support value : "))
Minimum_Support= (minimum_support_count/100)*No_of_Transactions

Minimum_Confidence_value=float(input("Minimum_Confidence_value : "))
```

```
Number of transactions as per database : 10
Maximum items in each transaction as per database : 5
Minimum support value : 20
Minimum_Confidence_value : 60
```

Data processing to create the required records and item-list for further processing.

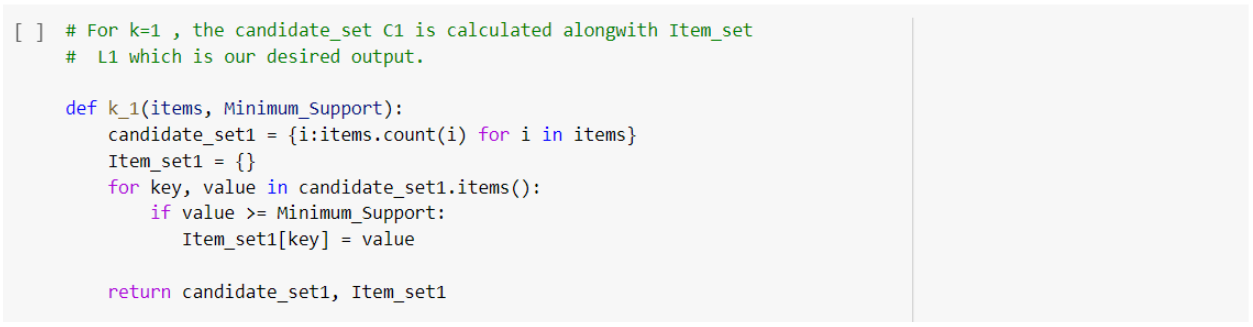


```
Input_Data = []
for i in range(0, No_of_Transactions):
    Input_Data.append([str(data.values[i,j]) for j in range(0, Maximum_Items)])

print("Input Transactions : ", Input_Data)
items = sorted([item for sublist in Input_Data for item in sublist if item != 'nan'])
```

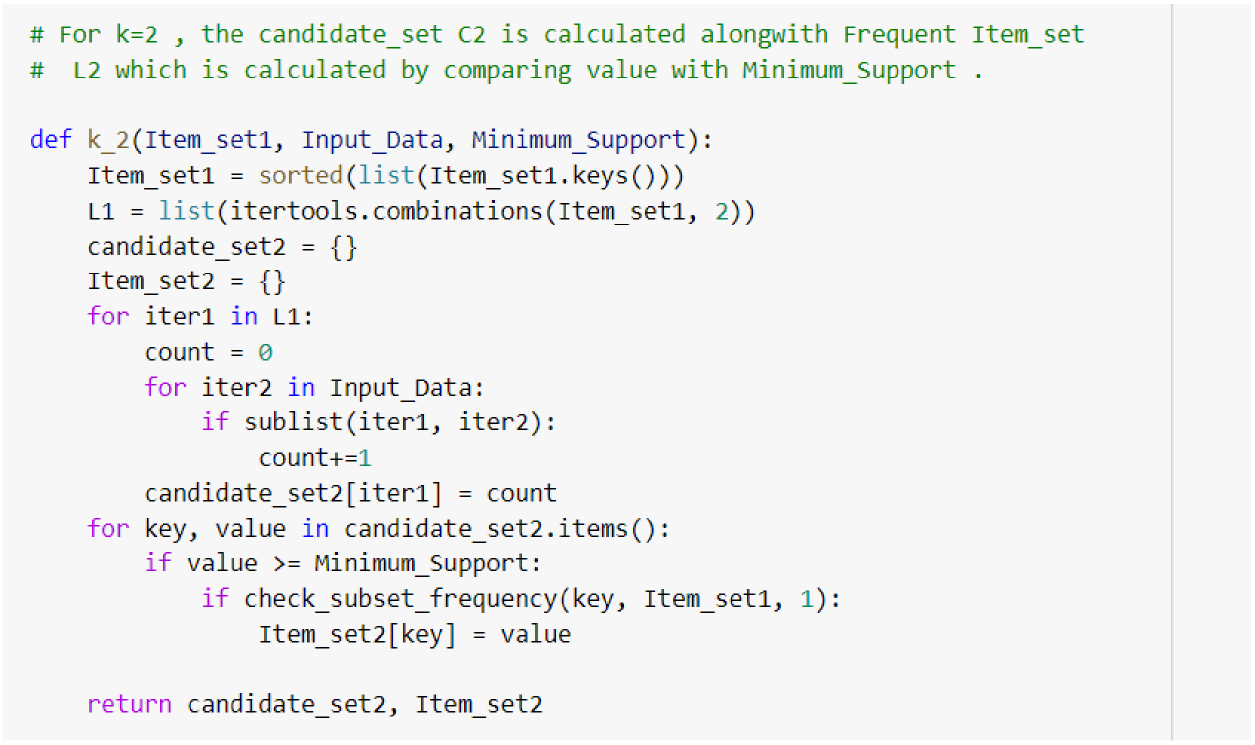
```
Input Transactions :  [['Running Shoe', 'Socks', 'Sweatshirts', 'nan', 'nan'], ['Running Shoe', 'Socks', 'Sweatshirts', 'Modern
Pants', 'nan'], ['Running Shoe', 'Sweatshirts', 'Modern Pants', 'nan', 'nan'], ['Running Shoe', 'Socks', 'Sweatshirts', 'Modern
Pants', 'Soccer Shoe'], ['Running Shoe', 'Socks', 'Sweatshirts', 'nan', 'nan'], ['Running Shoe', 'Socks', 'Sweatshirts', 'Moder
n Pants', 'Tech Pants'], ['Swimming Shirt', 'Socks', 'Sweatshirts', 'nan', 'nan'], ['Swimming Shirt', 'Rash Guard', 'Dry Fit V-
Nick', 'Hoodies', 'Tech Pants'], ['Swimming Shirt', 'Rash Guard', 'Dry', 'nan', 'nan'], ['Swimming Shirt', 'Rash Guard', 'Dry F
it V-Nick', 'nan', 'nan']]
```

For k=1 to 4 , candidate_set and the frequent ItemSet are calculated by comparing the support count of each item in the list to the Minimum Support value. We check if all the subsets in itemset are frequent using the check subset frequency function and if not, we remove respective itemset from the list by comparing the length of the two possible lists using the sublist function.

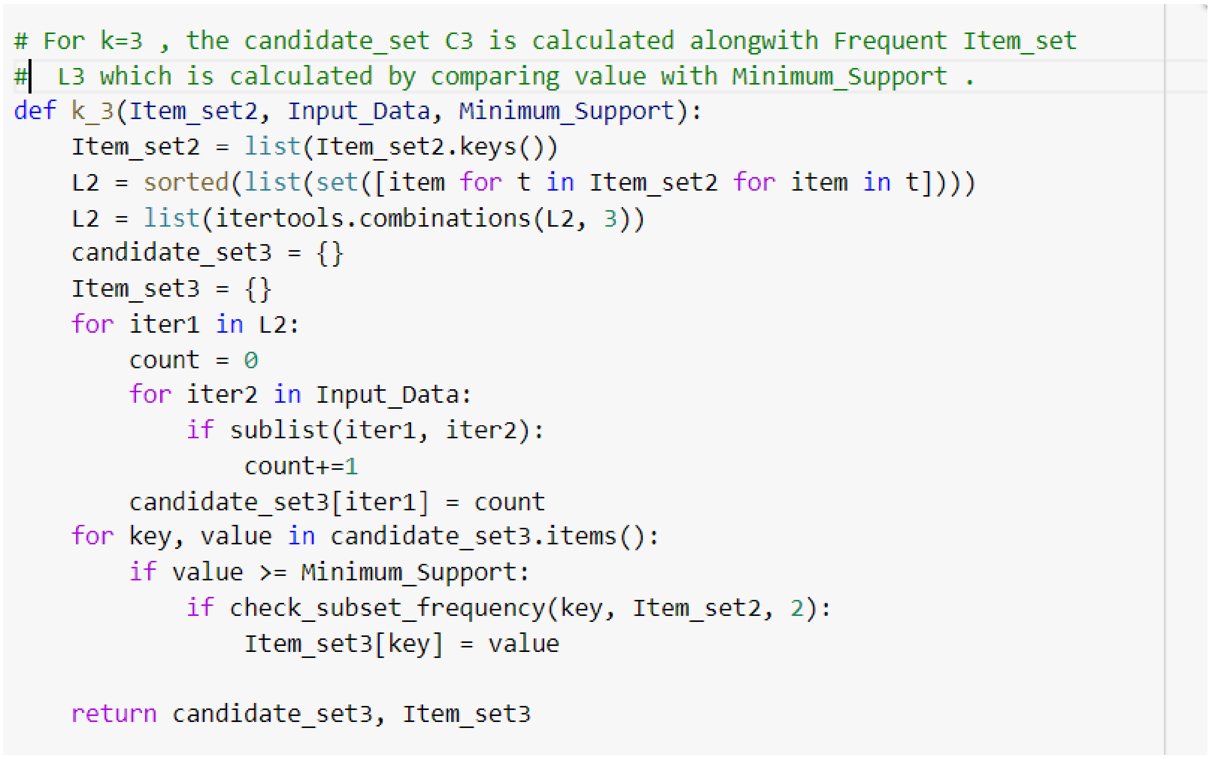
We are printing all the Item_Sets generated after calculations.



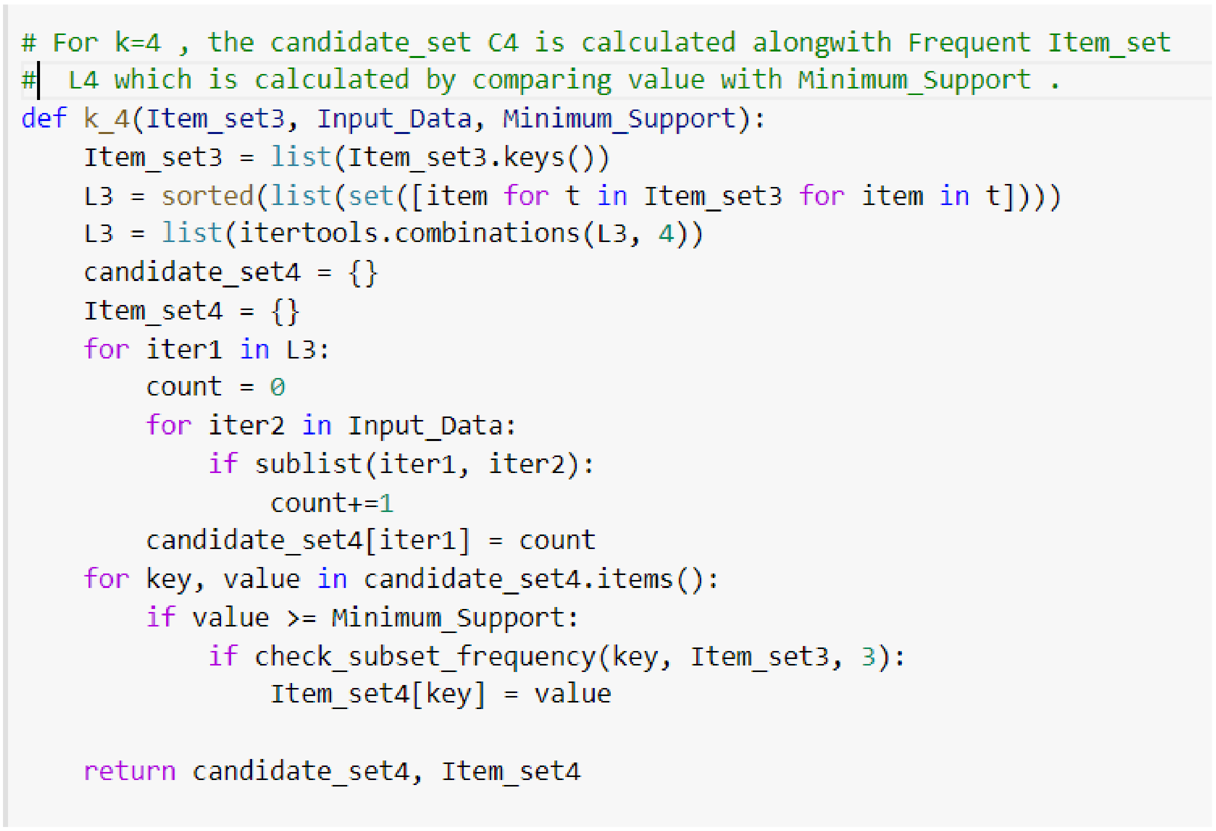
```
[ ] # For k=1 , the candidate_set C1 is calculated alongwith Item_set
    #  L1 which is our desired output.

    def k_1(items, Minimum_Support):
        candidate_set1 = {i:items.count(i) for i in items}
        Item_set1 = {}
        for key, value in candidate_set1.items():
            if value >= Minimum_Support:
                Item_set1[key] = value

        return candidate_set1, Item_set1
```



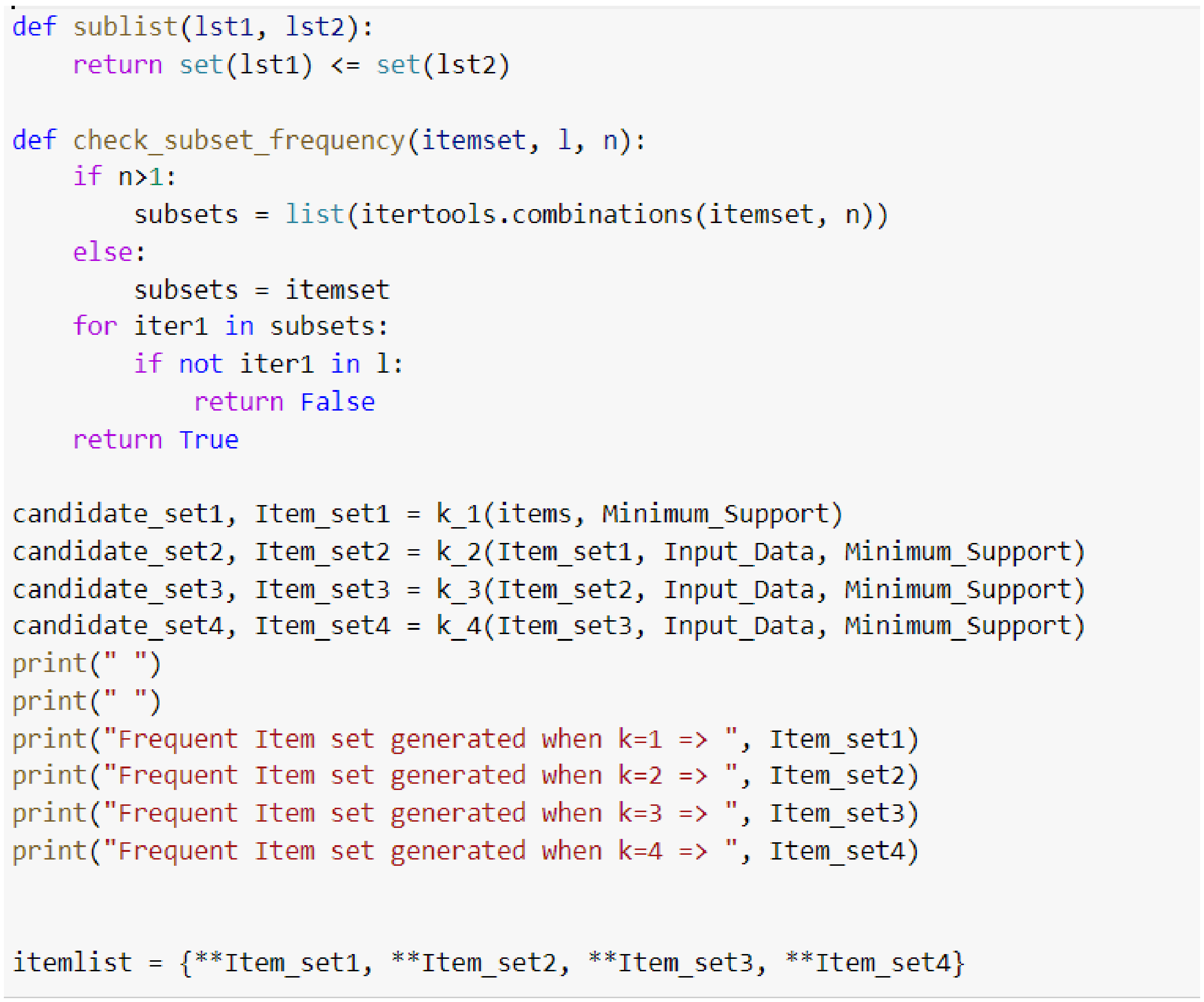
```
# For k=2 , the candidate_set C2 is calculated alongwith Frequent Item_set
#  L2 which is calculated by comparing value with Minimum_Support .

def k_2(Item_set1, Input_Data, Minimum_Support):
    Item_set1 = sorted(list(Item_set1.keys()))
    L1 = list(itertools.combinations(Item_set1, 2))
    candidate_set2 = {}
    Item_set2 = {}
    for iter1 in L1:
        count = 0
        for iter2 in Input_Data:
            if sublist(iter1, iter2):
                count+=1
        candidate_set2[iter1] = count
    for key, value in candidate_set2.items():
        if value >= Minimum_Support:
            if check_subset_frequency(key, Item_set1, 1):
                Item_set2[key] = value

    return candidate_set2, Item_set2
```



```python
# For k=3 , the candidate_set C3 is calculated alongwith Frequent Item_set
#  L3 which is calculated by comparing value with Minimum_Support .
def k_3(Item_set2, Input_Data, Minimum_Support):
    Item_set2 = list(Item_set2.keys())
    L2 = sorted(list(set([item for t in Item_set2 for item in t])))
    L2 = list(itertools.combinations(L2, 3))
    candidate_set3 = {}
    Item_set3 = {}
    for iter1 in L2:
        count = 0
        for iter2 in Input_Data:
            if sublist(iter1, iter2):
                count+=1
        candidate_set3[iter1] = count
    for key, value in candidate_set3.items():
        if value >= Minimum_Support:
            if check_subset_frequency(key, Item_set2, 2):
                Item_set3[key] = value

    return candidate_set3, Item_set3
```

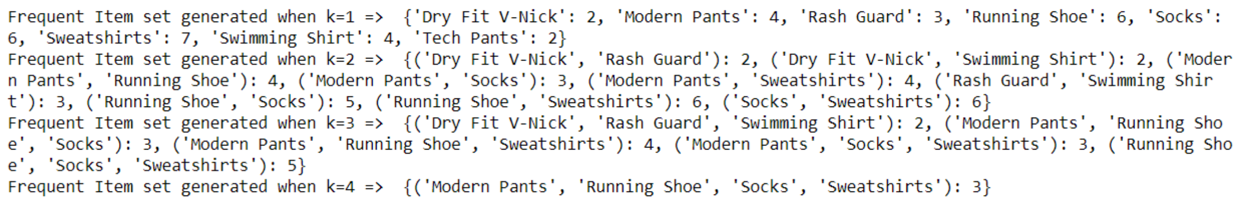


```python
# For k=4 , the candidate_set C4 is calculated alongwith Frequent Item_set
#  L4 which is calculated by comparing value with Minimum_Support .
def k_4(Item_set3, Input_Data, Minimum_Support):
    Item_set3 = list(Item_set3.keys())
    L3 = sorted(list(set([item for t in Item_set3 for item in t])))
    L3 = list(itertools.combinations(L3, 4))
    candidate_set4 = {}
    Item_set4 = {}
    for iter1 in L3:
        count = 0
        for iter2 in Input_Data:
            if sublist(iter1, iter2):
                count+=1
        candidate_set4[iter1] = count
    for key, value in candidate_set4.items():
        if value >= Minimum_Support:
            if check_subset_frequency(key, Item_set3, 3):
                Item_set4[key] = value

    return candidate_set4, Item_set4
```



```
def sublist(lst1, lst2):
    return set(lst1) <= set(lst2)

def check_subset_frequency(itemset, l, n):
    if n>1:
        subsets = list(itertools.combinations(itemset, n))
    else:
        subsets = itemset
    for iter1 in subsets:
        if not iter1 in l:
            return False
    return True

candidate_set1, Item_set1 = k_1(items, Minimum_Support)
candidate_set2, Item_set2 = k_2(Item_set1, Input_Data, Minimum_Support)
candidate_set3, Item_set3 = k_3(Item_set2, Input_Data, Minimum_Support)
candidate_set4, Item_set4 = k_4(Item_set3, Input_Data, Minimum_Support)
print(" ")
print(" ")
print("Frequent Item set generated when k=1 => ", Item_set1)
print("Frequent Item set generated when k=2 => ", Item_set2)
print("Frequent Item set generated when k=3 => ", Item_set3)
print("Frequent Item set generated when k=4 => ", Item_set4)


itemlist = {**Item_set1, **Item_set2, **Item_set3, **Item_set4}
```

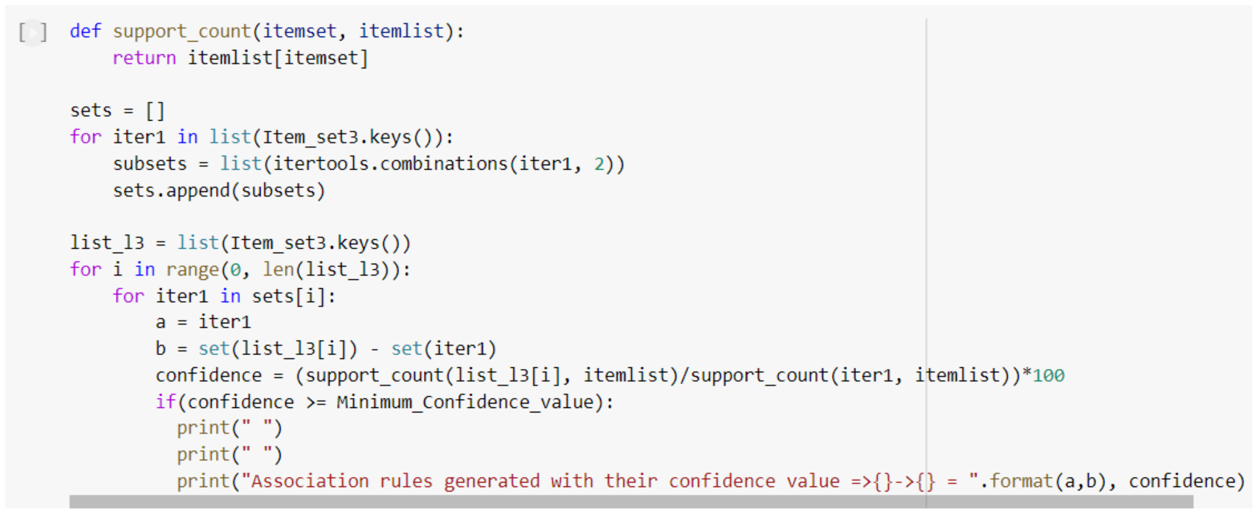
Frequent Itemsets created :



```
Frequent Item set generated when k=1 =>  {'Dry Fit V-Nick': 2, 'Modern Pants': 4, 'Rash Guard': 3, 'Running Shoe': 6, 'Socks':
6, 'Sweatshirts': 7, 'Swimming Shirt': 4, 'Tech Pants': 2}
Frequent Item set generated when k=2 =>  {('Dry Fit V-Nick', 'Rash Guard'): 2, ('Dry Fit V-Nick', 'Swimming Shirt'): 2, ('Moder
n Pants', 'Running Shoe'): 4, ('Modern Pants', 'Socks'): 3, ('Modern Pants', 'Sweatshirts'): 4, ('Rash Guard', 'Swimming Shir
t'): 3, ('Running Shoe', 'Socks'): 5, ('Running Shoe', 'Sweatshirts'): 6, ('Socks', 'Sweatshirts'): 6}
Frequent Item set generated when k=3 =>  {('Dry Fit V-Nick', 'Rash Guard', 'Swimming Shirt'): 2, ('Modern Pants', 'Running Sho
e', 'Socks'): 3, ('Modern Pants', 'Running Shoe', 'Sweatshirts'): 4, ('Modern Pants', 'Socks', 'Sweatshirts'): 3, ('Running Sho
e', 'Socks', 'Sweatshirts'): 5}
Frequent Item set generated when k=4 =>  {('Modern Pants', 'Running Shoe', 'Socks', 'Sweatshirts'): 3}
```

(**Resultset was too wide for screenshot so zoom in to see the output clearly)

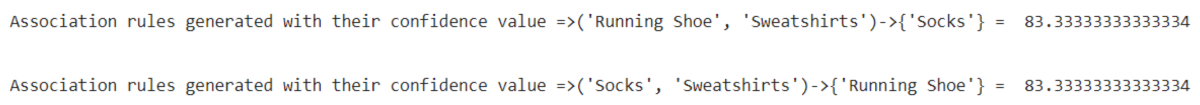
Calculating the Association rules based on the confidence value.



```python
def support_count(itemset, itemlist):
    return itemlist[itemset]

sets = []
for iter1 in list(Item_set3.keys()):
    subsets = list(itertools.combinations(iter1, 2))
    sets.append(subsets)

list_l3 = list(Item_set3.keys())
for i in range(0, len(list_l3)):
    for iter1 in sets[i]:
        a = iter1
        b = set(list_l3[i]) - set(iter1)
        confidence = (support_count(list_l3[i], itemlist)/support_count(iter1, itemlist))*100
        if(confidence >= Minimum_Confidence_value):
          print(" ")
          print(" ")
          print("Association rules generated with their confidence value =>{}->{} = ".format(a,b), confidence)
```

Association Rules with Confidence values :

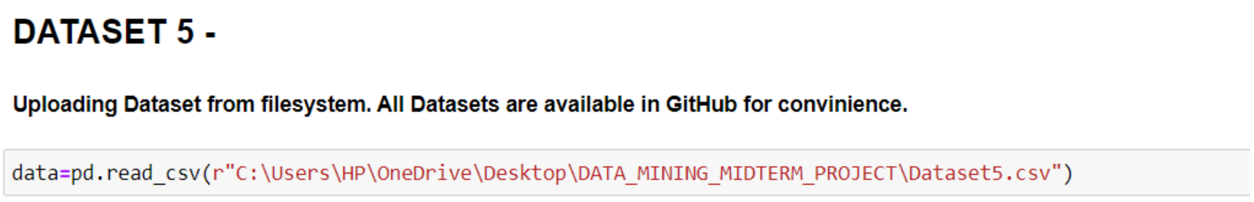
```
Association rules generated with their confidence value =>('Dry Fit V-Nick', 'Rash Guard')->{'Swimming Shirt'} =  100.0

Association rules generated with their confidence value =>('Dry Fit V-Nick', 'Swimming Shirt')->{'Rash Guard'} =  100.0

Association rules generated with their confidence value =>('Rash Guard', 'Swimming Shirt')->{'Dry Fit V-Nick'} =  66.66666666666666

Association rules generated with their confidence value =>('Modern Pants', 'Running Shoe')->{'Socks'} =  75.0

Association rules generated with their confidence value =>('Modern Pants', 'Socks')->{'Running Shoe'} =  100.0

Association rules generated with their confidence value =>('Running Shoe', 'Socks')->{'Modern Pants'} =  60.0

Association rules generated with their confidence value =>('Modern Pants', 'Running Shoe')->{'Sweatshirts'} =  100.0

Association rules generated with their confidence value =>('Modern Pants', 'Sweatshirts')->{'Running Shoe'} =  100.0

Association rules generated with their confidence value =>('Running Shoe', 'Sweatshirts')->{'Modern Pants'} =  66.66666666666666

Association rules generated with their confidence value =>('Modern Pants', 'Socks')->{'Sweatshirts'} =  100.0

Association rules generated with their confidence value =>('Modern Pants', 'Sweatshirts')->{'Socks'} =  75.0

Association rules generated with their confidence value =>('Running Shoe', 'Socks')->{'Sweatshirts'} =  100.0
```



```
Association rules generated with their confidence value =>('Running Shoe', 'Sweatshirts')->{'Socks'} =  83.33333333333334

Association rules generated with their confidence value =>('Socks', 'Sweatshirts')->{'Running Shoe'} =  83.33333333333334
```

(**Resultset was too wide for screenshot so zoom in to see the output clearly)

Dataset 5 (Generic) :

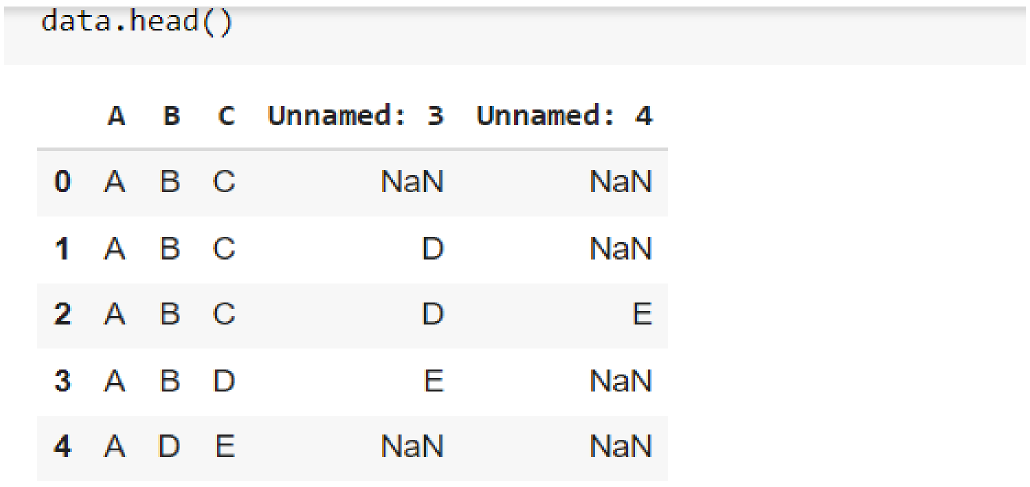


DATASET 5 -

Uploading Dataset from filesystem. All Datasets are available in GitHub for convinience.

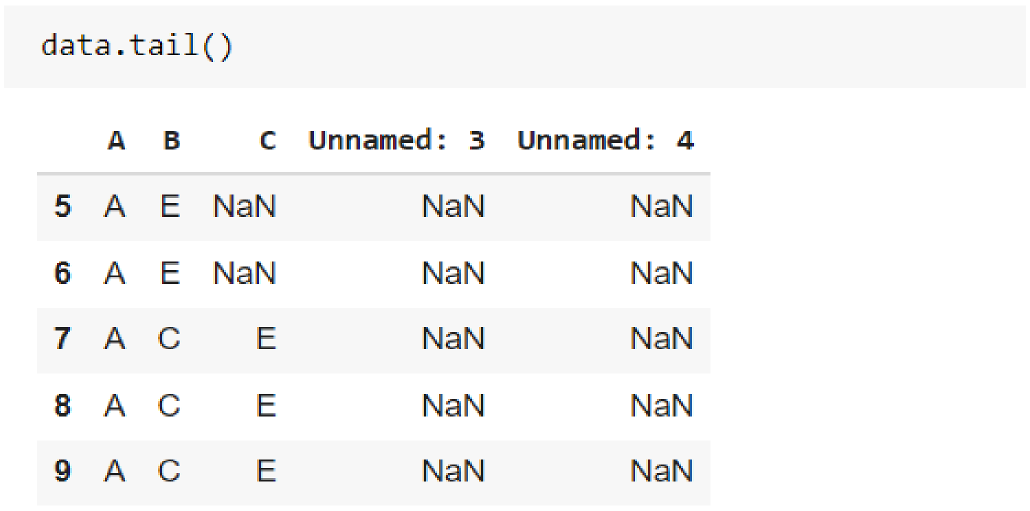
```
data=pd.read_csv(r"C:\Users\HP\OneDrive\Desktop\DATA_MINING_MIDTERM_PROJECT\Dataset5.csv")
```

Data Representation :



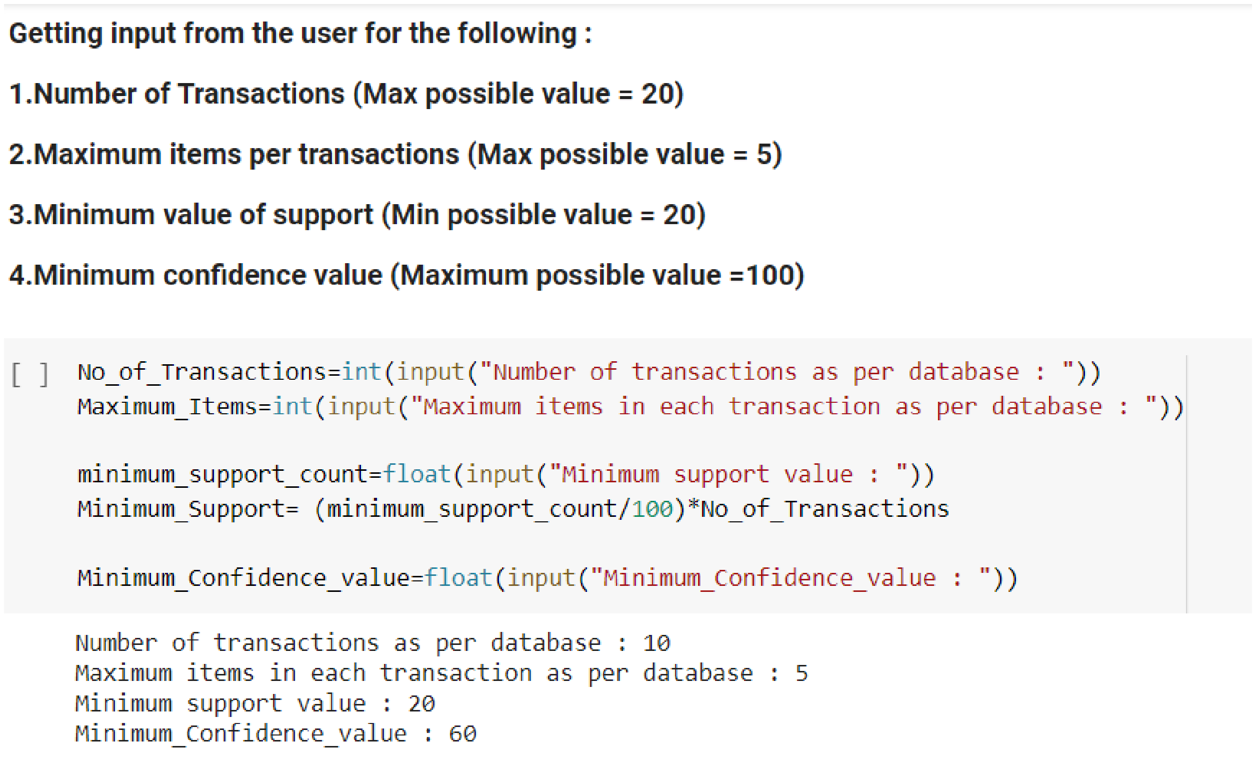
```
data.head()
```

	A	B	C	Unnamed: 3	Unnamed: 4
0	A	B	C	NaN	NaN
1	A	B	C	D	NaN
2	A	B	C	D	E
3	A	B	D	E	NaN
4	A	D	E	NaN	NaN



```
data.tail()
```

	A	B	C	Unnamed: 3	Unnamed: 4
5	A	E	NaN	NaN	NaN
6	A	E	NaN	NaN	NaN
7	A	C	E	NaN	NaN
8	A	C	E	NaN	NaN
9	A	C	E	NaN	NaN



Getting input from the user for the following :

1.Number of Transactions (Max possible value = 20)

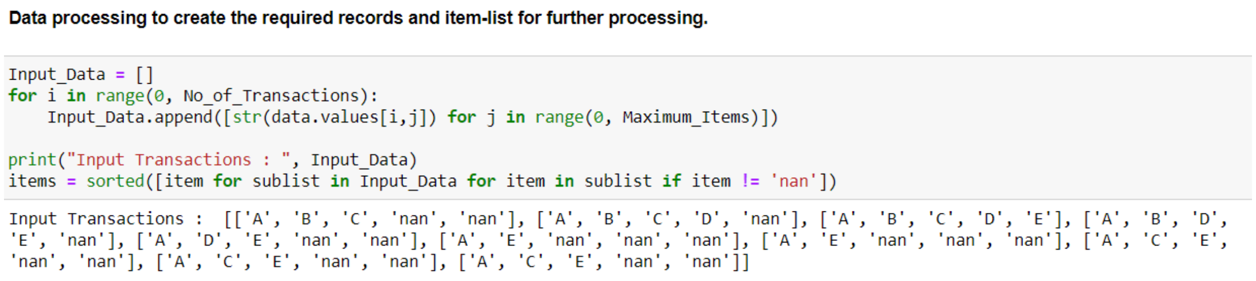
2.Maximum items per transactions (Max possible value = 5)

3.Minimum value of support (Min possible value = 20)

4.Minimum confidence value (Maximum possible value =100)

```
[ ]  No_of_Transactions=int(input("Number of transactions as per database : "))
     Maximum_Items=int(input("Maximum items in each transaction as per database : "))

     minimum_support_count=float(input("Minimum support value : "))
     Minimum_Support= (minimum_support_count/100)*No_of_Transactions

     Minimum_Confidence_value=float(input("Minimum_Confidence_value : "))
```

```
Number of transactions as per database : 10
Maximum items in each transaction as per database : 5
Minimum support value : 20
Minimum_Confidence_value : 60
```



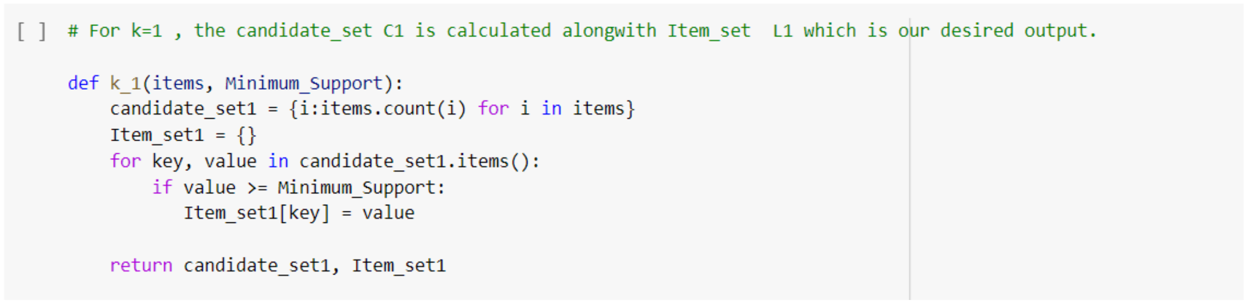
Data processing to create the required records and item-list for further processing.

```
Input_Data = []
for i in range(0, No_of_Transactions):
    Input_Data.append([str(data.values[i,j]) for j in range(0, Maximum_Items)])

print("Input Transactions : ", Input_Data)
items = sorted([item for sublist in Input_Data for item in sublist if item != 'nan'])
```

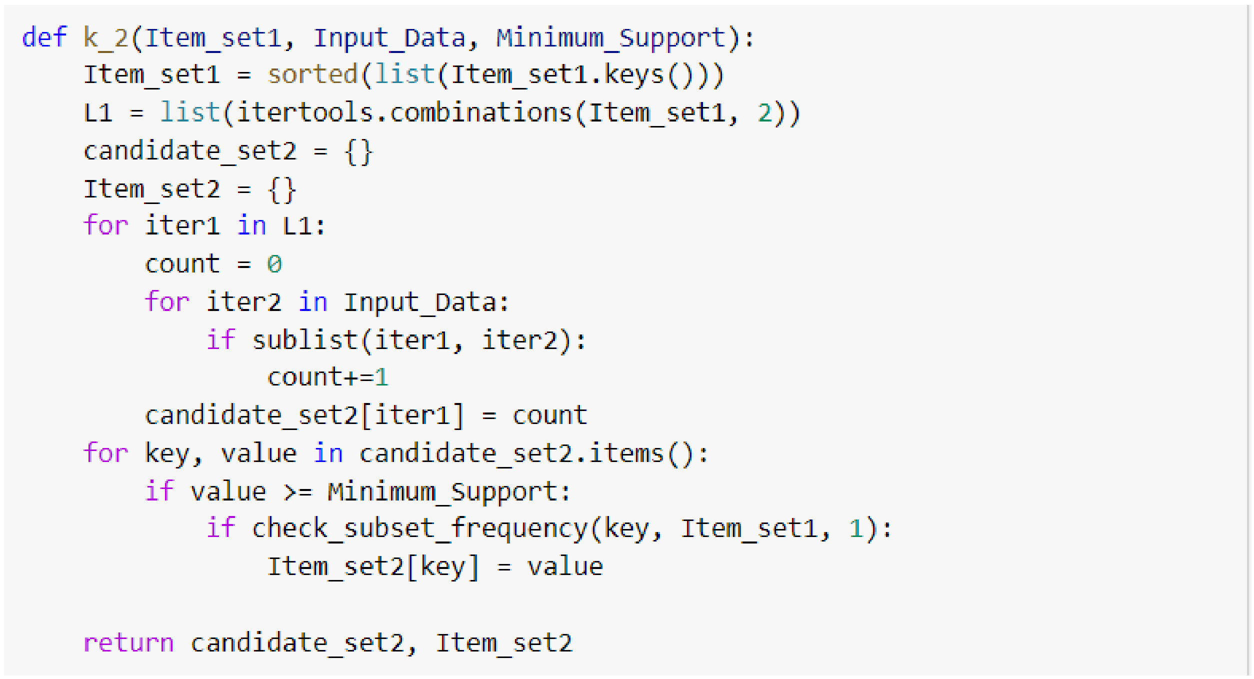
```
Input Transactions :  [['A', 'B', 'C', 'nan', 'nan'], ['A', 'B', 'C', 'D', 'nan'], ['A', 'B', 'C', 'D', 'E'], ['A', 'B', 'D',
'E', 'nan'], ['A', 'D', 'E', 'nan', 'nan'], ['A', 'E', 'nan', 'nan', 'nan'], ['A', 'E', 'nan', 'nan', 'nan'], ['A', 'C', 'E',
'nan', 'nan'], ['A', 'C', 'E', 'nan', 'nan'], ['A', 'C', 'E', 'nan', 'nan']]
```

For k=1 to 4 , candidate_set and the frequent ItemSet are calculated by comparing the support count of each item in the list to the Minimum Support value. We check if all the subsets in itemset are frequent using the check subset frequency function and if not, we remove respective itemset from the list by comparing the length of the two possible lists using the sublist function.

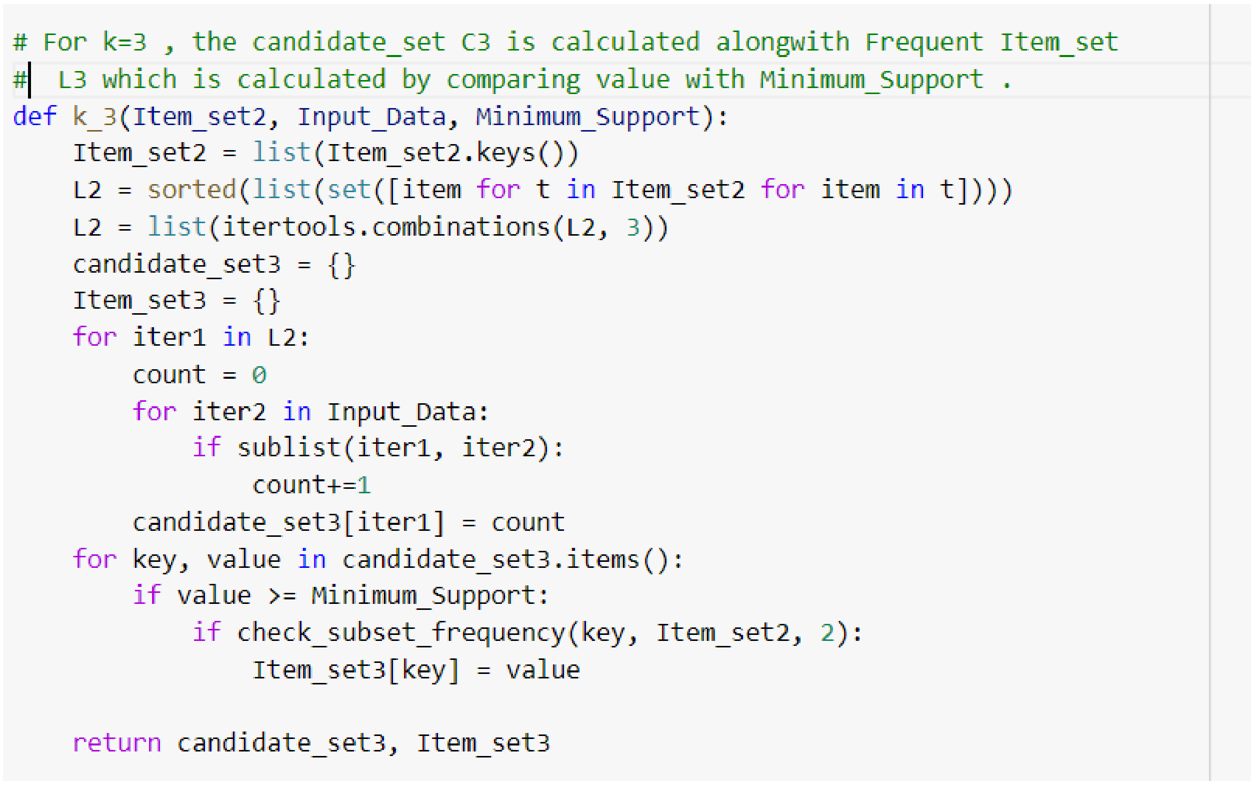
We are printing all the Item_Sets generated after calculations.



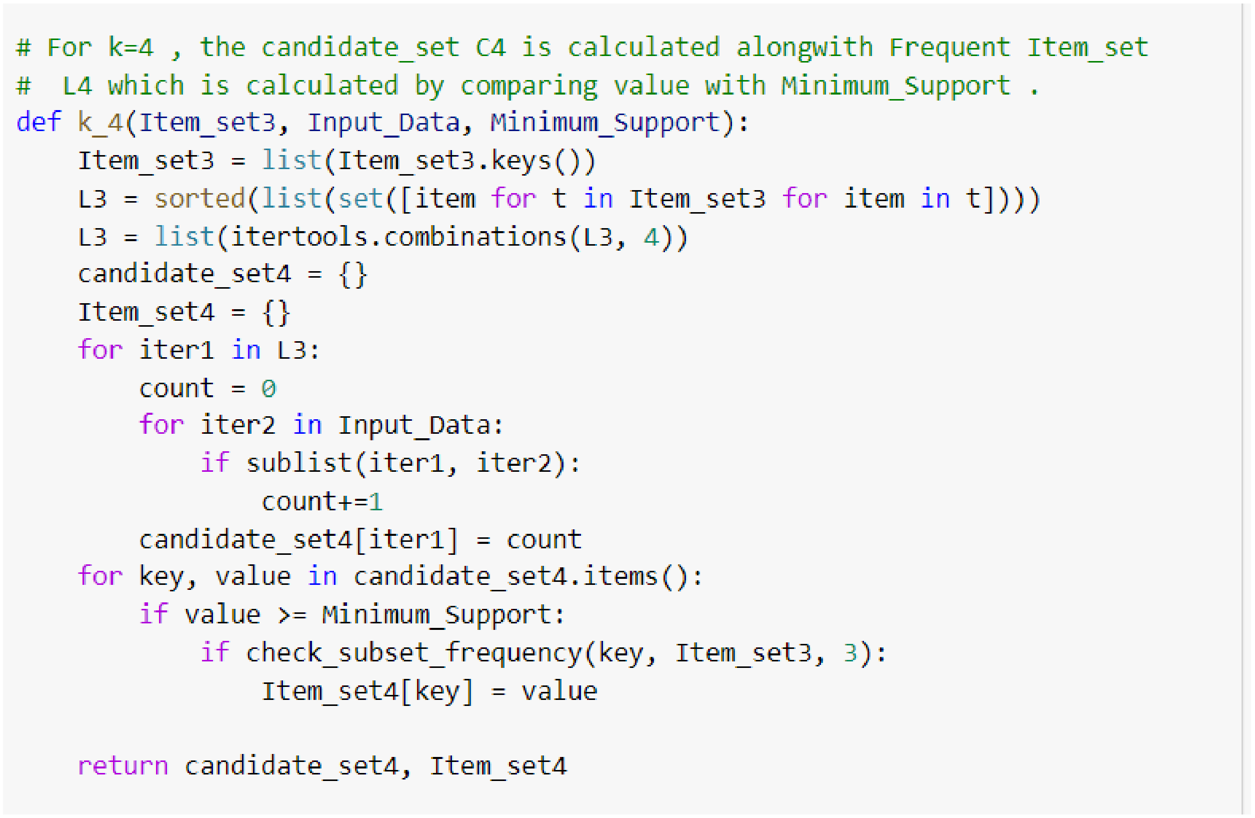
```
[ ] # For k=1 , the candidate_set C1 is calculated alongwith Item_set  L1 which is our desired output.

    def k_1(items, Minimum_Support):
        candidate_set1 = {i:items.count(i) for i in items}
        Item_set1 = {}
        for key, value in candidate_set1.items():
            if value >= Minimum_Support:
              Item_set1[key] = value

        return candidate_set1, Item_set1
```



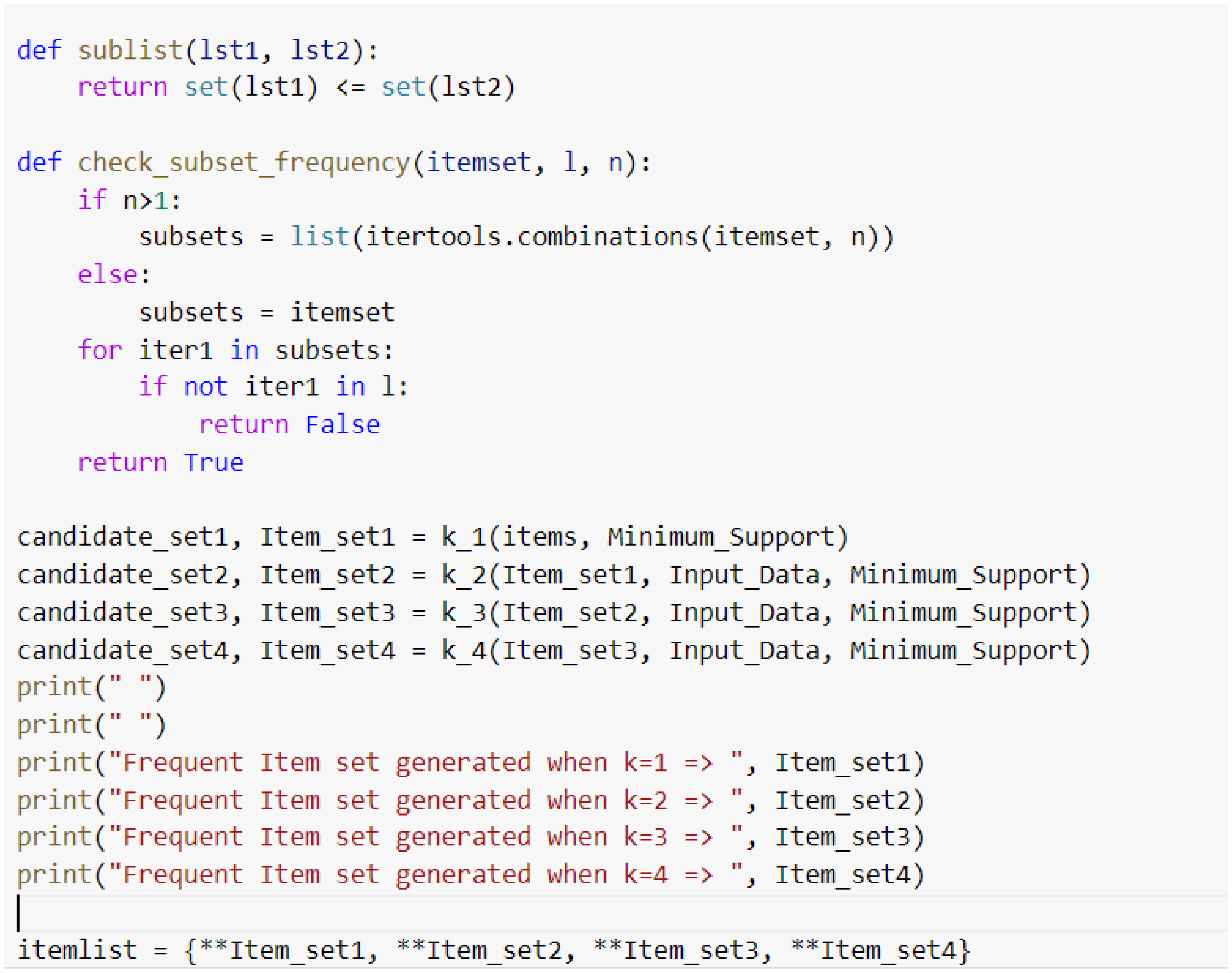
```
def k_2(Item_set1, Input_Data, Minimum_Support):
    Item_set1 = sorted(list(Item_set1.keys()))
    L1 = list(itertools.combinations(Item_set1, 2))
    candidate_set2 = {}
    Item_set2 = {}
    for iter1 in L1:
        count = 0
        for iter2 in Input_Data:
            if sublist(iter1, iter2):
                count+=1
        candidate_set2[iter1] = count
    for key, value in candidate_set2.items():
        if value >= Minimum_Support:
            if check_subset_frequency(key, Item_set1, 1):
                Item_set2[key] = value

    return candidate_set2, Item_set2
```



```python
# For k=3 , the candidate_set C3 is calculated alongwith Frequent Item_set
#  L3 which is calculated by comparing value with Minimum_Support .
def k_3(Item_set2, Input_Data, Minimum_Support):
    Item_set2 = list(Item_set2.keys())
    L2 = sorted(list(set([item for t in Item_set2 for item in t])))
    L2 = list(itertools.combinations(L2, 3))
    candidate_set3 = {}
    Item_set3 = {}
    for iter1 in L2:
        count = 0
        for iter2 in Input_Data:
            if sublist(iter1, iter2):
                count+=1
        candidate_set3[iter1] = count
    for key, value in candidate_set3.items():
        if value >= Minimum_Support:
            if check_subset_frequency(key, Item_set2, 2):
                Item_set3[key] = value

    return candidate_set3, Item_set3
```

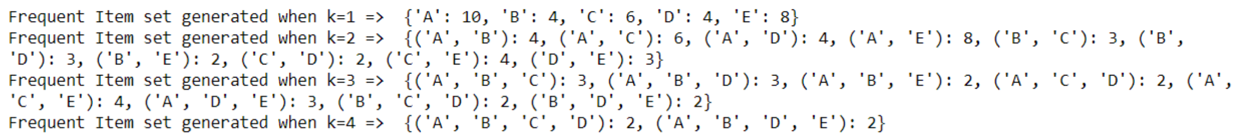


```python
# For k=4 , the candidate_set C4 is calculated alongwith Frequent Item_set
#  L4 which is calculated by comparing value with Minimum_Support .
def k_4(Item_set3, Input_Data, Minimum_Support):
    Item_set3 = list(Item_set3.keys())
    L3 = sorted(list(set([item for t in Item_set3 for item in t])))
    L3 = list(itertools.combinations(L3, 4))
    candidate_set4 = {}
    Item_set4 = {}
    for iter1 in L3:
        count = 0
        for iter2 in Input_Data:
            if sublist(iter1, iter2):
                count+=1
        candidate_set4[iter1] = count
    for key, value in candidate_set4.items():
        if value >= Minimum_Support:
            if check_subset_frequency(key, Item_set3, 3):
                Item_set4[key] = value

    return candidate_set4, Item_set4
```



```
def sublist(lst1, lst2):
    return set(lst1) <= set(lst2)

def check_subset_frequency(itemset, l, n):
    if n>1:
        subsets = list(itertools.combinations(itemset, n))
    else:
        subsets = itemset
    for iter1 in subsets:
        if not iter1 in l:
            return False
    return True

candidate_set1, Item_set1 = k_1(items, Minimum_Support)
candidate_set2, Item_set2 = k_2(Item_set1, Input_Data, Minimum_Support)
candidate_set3, Item_set3 = k_3(Item_set2, Input_Data, Minimum_Support)
candidate_set4, Item_set4 = k_4(Item_set3, Input_Data, Minimum_Support)
print(" ")
print(" ")
print("Frequent Item set generated when k=1 => ", Item_set1)
print("Frequent Item set generated when k=2 => ", Item_set2)
print("Frequent Item set generated when k=3 => ", Item_set3)
print("Frequent Item set generated when k=4 => ", Item_set4)

itemlist = {**Item_set1, **Item_set2, **Item_set3, **Item_set4}
```

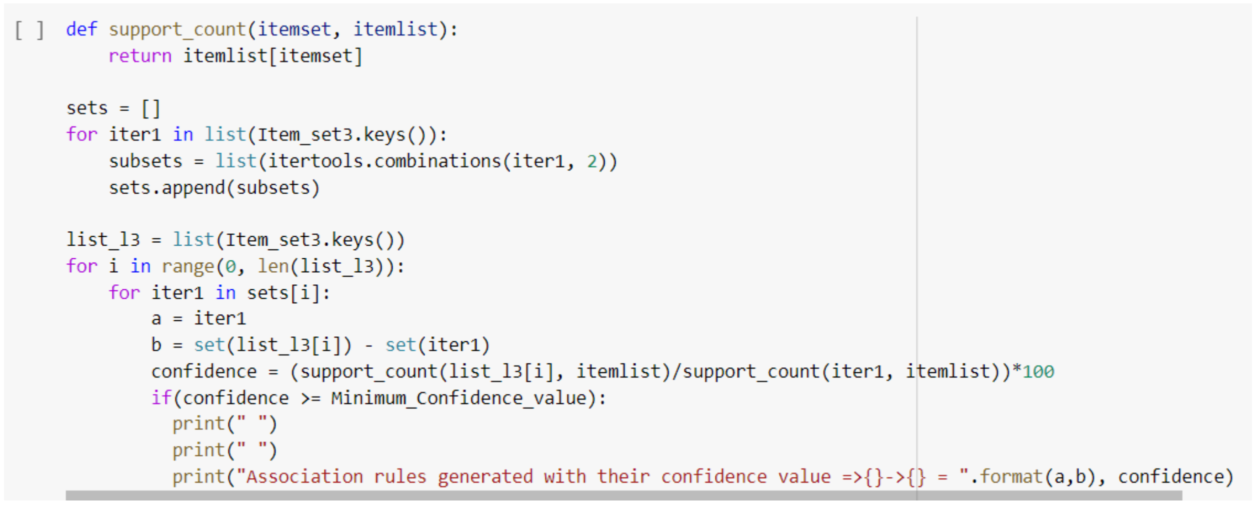
Frequent Itemsets created :



```
Frequent Item set generated when k=1 =>  {'A': 10, 'B': 4, 'C': 6, 'D': 4, 'E': 8}
Frequent Item set generated when k=2 =>  {('A', 'B'): 4, ('A', 'C'): 6, ('A', 'D'): 4, ('A', 'E'): 8, ('B', 'C'): 3, ('B',
'D'): 3, ('B', 'E'): 2, ('C', 'D'): 2, ('C', 'E'): 4, ('D', 'E'): 3}
Frequent Item set generated when k=3 =>  {('A', 'B', 'C'): 3, ('A', 'B', 'D'): 3, ('A', 'B', 'E'): 2, ('A', 'C', 'D'): 2, ('A',
'C', 'E'): 4, ('A', 'D', 'E'): 3, ('B', 'C', 'D'): 2, ('B', 'D', 'E'): 2}
Frequent Item set generated when k=4 =>  {('A', 'B', 'C', 'D'): 2, ('A', 'B', 'D', 'E'): 2}
```

(**Resultset was too wide for screenshot so zoom in to see the output clearly)

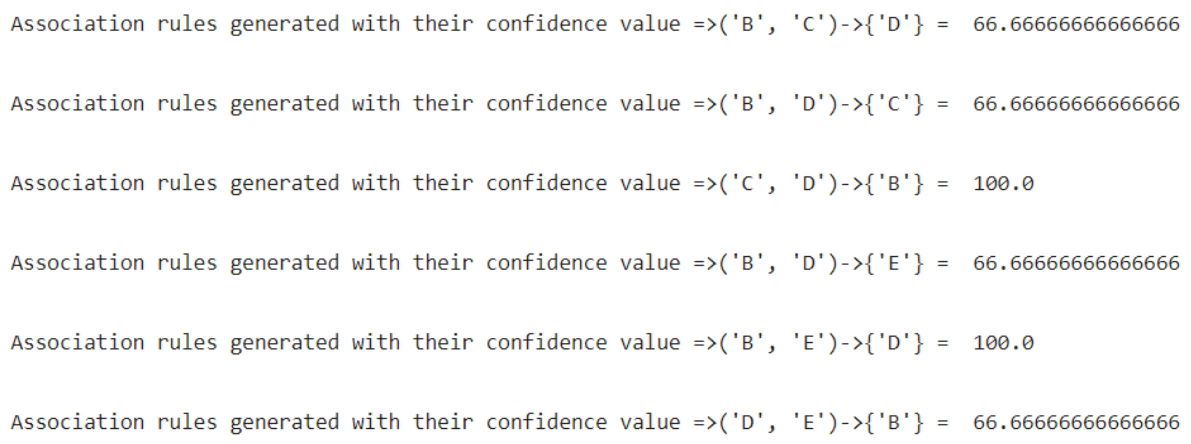
Calculating the Association rules based on the confidence value.



```python
def support_count(itemset, itemlist):
    return itemlist[itemset]

sets = []
for iter1 in list(Item_set3.keys()):
    subsets = list(itertools.combinations(iter1, 2))
    sets.append(subsets)

list_l3 = list(Item_set3.keys())
for i in range(0, len(list_l3)):
    for iter1 in sets[i]:
        a = iter1
        b = set(list_l3[i]) - set(iter1)
        confidence = (support_count(list_l3[i], itemlist)/support_count(iter1, itemlist))*100
        if(confidence >= Minimum_Confidence_value):
          print(" ")
          print(" ")
          print("Association rules generated with their confidence value =>{}->{} = ".format(a,b), confidence)
```

Association Rules with Confidence values :

```
Association rules generated with their confidence value =>('A', 'B')->{'C'} =  75.0

Association rules generated with their confidence value =>('B', 'C')->{'A'} =  100.0

Association rules generated with their confidence value =>('A', 'B')->{'D'} =  75.0

Association rules generated with their confidence value =>('A', 'D')->{'B'} =  75.0

Association rules generated with their confidence value =>('B', 'D')->{'A'} =  100.0

Association rules generated with their confidence value =>('B', 'E')->{'A'} =  100.0

Association rules generated with their confidence value =>('C', 'D')->{'A'} =  100.0

Association rules generated with their confidence value =>('A', 'C')->{'E'} =  66.66666666666666

Association rules generated with their confidence value =>('C', 'E')->{'A'} =  100.0

Association rules generated with their confidence value =>('A', 'D')->{'E'} =  75.0

Association rules generated with their confidence value =>('D', 'E')->{'A'} =  100.0
```



```
Association rules generated with their confidence value =>('B', 'C')->{'D'} =  66.66666666666666

Association rules generated with their confidence value =>('B', 'D')->{'C'} =  66.66666666666666

Association rules generated with their confidence value =>('C', 'D')->{'B'} =  100.0

Association rules generated with their confidence value =>('B', 'D')->{'E'} =  66.66666666666666

Association rules generated with their confidence value =>('B', 'E')->{'D'} =  100.0

Association rules generated with their confidence value =>('D', 'E')->{'B'} =  66.66666666666666
```

(**Resultset was too wide for screenshot so zoom in to see the output clearly)

References :

- For definitions and understanding Apriori in-depth :

  https://www.softwaretestinghelp.com/apriori-algorithm/