Q4. Run Apriori algorithm to find frequent item sets and association rules

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
import pandas as pd
import numpy as np
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
import matplotlib.pyplot as plt
```

```
from csv import reader
# read csv file as a list of lists
with open('/content/gdrive/MyDrive/groceries.csv', 'r') as read_obj:
  # pass the file object to reader() to get the reader object
  csv_reader = reader(read_obj)
  # Pass reader object to list() to get a list of lists
  groceries = list(csv_reader)
  print(groceries)
```

```
[['citrus fruit', 'semi-finished bread', 'margarine', 'ready soups'], ['tropical fruit',
```

```
te = TransactionEncoder()
```

Automatic saving failed. This file was updated remotely or in another tab. Show diff

```
#groceries.replace([None], np.nan, inplace=True)
groceries = pd.DataFrame(te_ary, columns=te.columns_)
print(groceries)
```

```
      Instant food products  UHT-milk  ...  yogurt  zwieback
0                     False     False  ...   False     False
1                     False     False  ...    True     False
2                     False     False  ...   False     False
3                     False     False  ...    True     False
4                     False     False  ...   False     False
...                     ...       ...  ...     ...       ...
9830                  False     False  ...   False     False
9831                  False     False  ...   False     False
9832                  False     False  ...    True     False
9833                  False     False  ...   False     False
9834                  False     False  ...   False     False

[9835 rows x 169 columns]
```

```
frequent_items = apriori(groceries, min_support=0.1, use_colnames=True)
print(frequent_items)
```

```
    support            itemsets
0  0.110524     (bottled water)
1  0.193493  (other vegetables)
2  0.183935        (rolls/buns)
3  0.108998   (root vegetables)
4  0.174377              (soda)
5  0.104931    (tropical fruit)
6  0.255516        (whole milk)
7  0.139502            (yogurt)
```

```
association_rules(frequent_items, metric="confidence", min_threshold=0.5)
```

antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage

```
frequent_items = apriori(groceries, min_support=0.05, use_colnames=True)
print(frequent_items)
```

```
    support                      itemsets
0  0.052466                        (beef)
1  0.080529                 (bottled beer)
2  0.110524                (bottled water)
3  0.064870                  (brown bread)
4  0.055414                       (butter)
5  0.077682                 (canned beer)
6  0.082766                 (citrus fruit)
7  0.058058                       (coffee)
8  0.053279                         (curd)
9  0.063447                (domestic eggs)
```



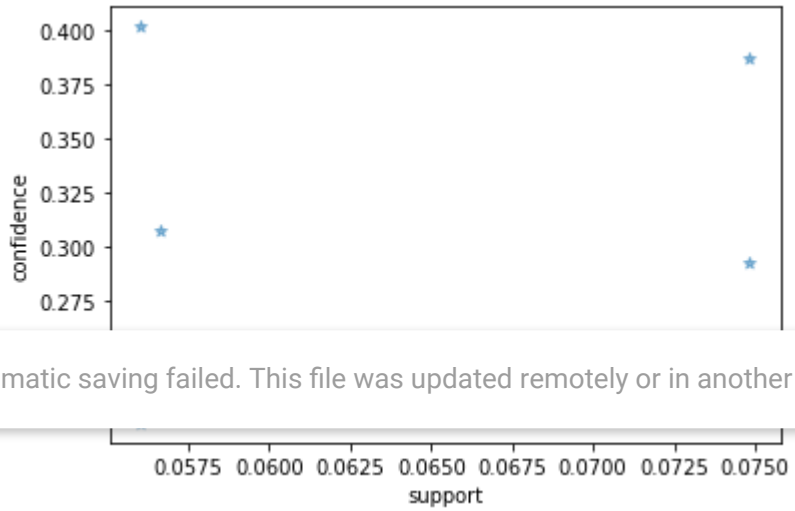
```
13  0.052364                      (napkins)
14  0.079817                   (newspapers)
15  0.193493             (other vegetables)
16  0.088968                       (pastry)
17  0.075648                    (pip fruit)
18  0.057651                         (pork)
19  0.183935                   (rolls/buns)
20  0.108998              (root vegetables)
21  0.093950                      (sausage)
22  0.098526                (shopping bags)
23  0.174377                        (soda)
24  0.104931               (tropical fruit)
25  0.071683           (whipped/sour cream)
26  0.255516                   (whole milk)
27  0.139502                       (yogurt)
28  0.074835  (whole milk, other vegetables)
29  0.056634        (whole milk, rolls/buns)
30  0.056024            (whole milk, yogurt)
```

```
rules=association_rules(frequent_items, metric="confidence", min_threshold=0.06)
print(rules)
#print(rules.keys())
```

```
             antecedents         consequents  ...  leverage  conviction
0         (whole milk)  (other vegetables)  ...  0.025394    1.140548
1  (other vegetables)         (whole milk)  ...  0.025394    1.214013
2         (whole milk)        (rolls/buns)  ...  0.009636    1.048452
3        (rolls/buns)         (whole milk)  ...  0.009636    1.075696
4         (whole milk)           (yogurt)  ...  0.020379    1.102157
5             (yogurt)         (whole milk)  ...  0.020379    1.244132

[6 rows x 9 columns]
```

```
support=rules['support']
confidence=rules['confidence']
```

```
plt.scatter(support, confidence,   alpha=0.5, marker="*")
plt.xlabel('support')
plt.ylabel('confidence')
plt.show()
```



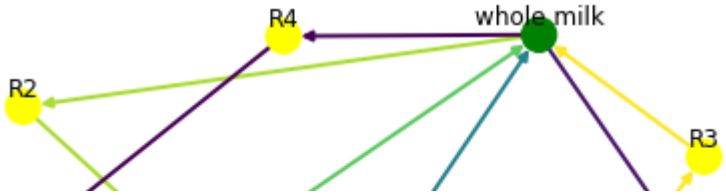

```
help(nx.draw())
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-13-1944a6f24330> in <module>()
----> 1 help(nx.draw())

NameError: name 'nx' is not defined
```

SEARCH STACK OVERFLOW

```
def draw_graph(rules, rules_to_show):
  import networkx as nx
```

```
    G1 = nx.DiGraph()

    color_map=[]
    N = 50
    colors = np.random.rand(N)
    strs=['R0', 'R1', 'R2', 'R3', 'R4', 'R5', 'R6', 'R7', 'R8', 'R9', 'R10', 'R11']


    for i in range (rules_to_show):
      G1.add_nodes_from(["R"+str(i)])
      for a in rules.iloc[i]['antecedents']:
          G1.add_nodes_from([a])
          G1.add_edge(a, "R"+str(i), color=colors[i] , weight = 2)

      for c in rules.iloc[i]['consequents']:
              G1.add_nodes_from([a])
              G1.add_edge("R"+str(i), c, color=colors[i],  weight=2)

    for node in G1:
         found_a_string = False
         for item in strs:
             if node==item:
                  found_a_string = True
         if found_a_string:
              color_map.append('yellow')
         else:
              color_map.append('green')

    edge = G1.edges()
    colors = [G1[u][v]['color'] for u,v in edge]
    weights = [G1[u][v]['weight'] for u,v in edge]

    Automatic saving failed. This file was updated remotely or in another tab. Show diff
                                                                                    font_size=16, wi
    for p in pos:  # raise text positions
             pos[p][1] += 0.07
    nx.draw_networkx_labels(G1, pos)
    plt.show()

draw_graph (rules, 6)
```



4.1 Use minimum support as 50% and minimum confidence as 75%

```
frequent_itemsets = apriori(groceries, min_support=0.5, use_colnames=True)
print(frequent_itemsets)
```

```
Empty DataFrame
Columns: [support, itemsets]
Index: []
```

```
association_rules(frequent_itemsets, metric="confidence", min_threshold=0.75)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-16-33869913e3b1> in <module>()
----> 1 association_rules(frequent_itemsets, metric="confidence", min_threshold=0.75)

3 frames

/usr/local/lib/python3.7/dist-packages/numpy/lib/function_base.py in
_get_ufunc_and_otypes(self, func, args)
   2140             args = [asarray(arg) for arg in args]
   2141             if builtins.any(arg.size == 0 for arg in args):
-> 2142                 raise ValueError('cannot call `vectorize` on size 0 inputs '
   2143                                  'unless `otypes` is set')
   2144
```

Automatic saving failed. This file was updated remotely or in another tab. Show diff

set

SEARCH STACK OVERFLOW

4.2 Use minimum support as 60% and minimum confidence as 60%

```
frequent_items = apriori(groceries, min_support=0.6, use_colnames=True)
frequent_items
```

	support	itemsets

```
association_rules(frequent_items, metric="confidence", min_threshold=0.6)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-19-dae2de37e5e8> in <module>()
----> 1 association_rules(frequent_items, metric="confidence", min_threshold=0.6)

 3 frames
/usr/local/lib/python3.7/dist-packages/numpy/lib/function_base.py in
_get_ufunc_and_otypes(self, func, args)
   2140             args = [asarray(arg) for arg in args]
   2141             if builtins.any(arg.size == 0 for arg in args):
-> 2142                 raise ValueError('cannot call `vectorize` on size 0 inputs '
   2143                                  'unless `otypes` is set')
   2144

ValueError: cannot call `vectorize` on size 0 inputs unless `otypes` is set
```

SEARCH STACK OVERFLOW

Automatic saving failed. This file was updated remotely or in another tab. Show diff