# INFS4203/7203 Data Mining Tutorial 2

Let's see how we can apply the external library to find frequent itemsets

# 1. Install mlxtend library

The mlxtend package can be installed in either of these two ways:

1. running **conda install -c conda-forge mlxtend** on your *Anaconda command promote*

OR

1. running **pip install mlxtend** on your *terminal* .

For more information, please check the documentation of mlxtend at http://rasbt.github.io/mlxtend/user_guide/frequent_patterns/apriori/ (http://rasbt.github.io/mlxtend/user_guide/frequent_patterns/apriori/)

# 2. Load data

We load the provided data *groceries.csv*.

In [1]:

```
# Import the libraries and load data
import mlxtend
import pandas as pd
import numpy as np

# load data from groceries.csv
df = pd.read_csv('groceries.csv')
# give a sign when the task finished
print("loading successful")
```

```
loading successful
```

In [2]:

```
# show the first three transactions
# transaction 0: citrus fruit, semi-finished bread, margarine, ready soups
# transaction 1: tropical fruit, yogurt, coffee
# transaction 2: whole milk
print(df.head(3))

# try by yourself
# print(df.head(4))
# print(df.head(100))
```

```
              item1                item2      item3        item4 item5 item6  \
0    citrus fruit  semi-finished bread  margarine  ready soups   NaN   NaN
1  tropical fruit               yogurt     coffee          NaN   NaN   NaN
2      whole milk                  NaN        NaN          NaN   NaN   NaN

  item7 item8 item9 item10  ... item21 item22 item23 item24 item25 item26  \
0   NaN   NaN   NaN    NaN  ...    NaN    NaN    NaN    NaN    NaN    NaN
1   NaN   NaN   NaN    NaN  ...    NaN    NaN    NaN    NaN    NaN    NaN
2   NaN   NaN   NaN    NaN  ...    NaN    NaN    NaN    NaN    NaN    NaN

  item27 item28 item29 item30
0    NaN    NaN    NaN    NaN
1    NaN    NaN    NaN    NaN
2    NaN    NaN    NaN    NaN

[3 rows x 30 columns]
```

# 3. Change the data format

From the documentation, we can see the required data format is like:

| | Type 1 | Type 2 | ... | Type N |
|---|---|---|---|---|
| 0 | True | False | ... | True |
| 1 | False | True | ... | True |
| 2 | True | True | ... | False |
| 3 | True | True | ... | True |

So next, we learn how to format the original transaction data into this one.

**We first make each transaction into a "list"**, where "list" is a prespecified data structure in Python. (See here for more on list: https://docs.python.org/3/tutorial/introduction.html#lists (https://docs.python.org/3/tutorial/introduction.html#lists))

In [3]:
```
# change the data to list
dataset = df.values.tolist()
cleanList = []

for trans in dataset: # for each transaction
    cleanTrans = []
    for x in trans: # for each element in the transaction
        if str(x) != 'nan': # if the item is not 'nan', put it in the list
            cleanTrans.append(x)
    cleanList.append(cleanTrans)
dataset = np.asarray(cleanList)

# give a sign when the task finished
print('Done')
```

```
Done
```

In [4]:
```
# let's see the dataset
print(dataset)
```

```
[list(['citrus fruit', 'semi-finished bread', 'margarine', 'ready soups'])
 list(['tropical fruit', 'yogurt', 'coffee']) list(['whole milk']) ...
 list(['chicken', 'citrus fruit', 'other vegetables', 'butter', 'yogurt', 'fr
ozen dessert', 'domestic eggs', 'rolls/buns', 'rum', 'cling film/bags'])
 list(['semi-finished bread', 'bottled water', 'soda', 'bottled beer'])
 list(['chicken', 'tropical fruit', 'other vegetables', 'vinegar', 'shopping
bags'])]
```

**We then change the list into a mlxtend required format use the function *TransactionEncoder()*.**

In [5]:
```
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori

# This part is not required to be understood. You can just run the code and sk
ip it.
te = TransactionEncoder() # a pre-defined function to transfer data
te_ary = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_ary, columns=te.columns_) # fit the transferred data back
into a pandas data format

# give a sign when the task finished
print('Done!')
```

```
Done!
```

```
In [6]: # let's see the first three transactions of the current data
        print(df.head(3))
```

```
   Instant food products  UHT-milk  abrasive cleaner  artif. sweetener  \
0                  False     False             False             False   
1                  False     False             False             False   
2                  False     False             False             False   

   baby cosmetics  baby food   bags  baking powder  bathroom cleaner   beef  
\
0           False      False  False          False             False  False   
1           False      False  False          False             False  False   
2           False      False  False          False             False  False   

   ...  turkey  vinegar  waffles  whipped/sour cream  whisky  white bread  \
0  ...   False    False    False               False   False        False   
1  ...   False    False    False               False   False        False   
2  ...   False    False    False               False   False        False   

   white wine  whole milk  yogurt  zwieback  
0       False       False   False     False  
1       False       False    True     False  
2       False        True   False     False  

[3 rows x 169 columns]
```

Great, you have done it!

# 4. Apply the Apriori algorithm

After we have the data organized as the requirement, we can apply the apriori algorithm:

```
In [7]: # define the MIN_SUPP
        MIN_SUPP = 0.02

        # apply the defined apriori algorithm
        freq_set = apriori(df, min_support=MIN_SUPP,use_colnames=True)

        print('Done!')
```

```
Done!
```

In [8]:
```
# let's see our result
freq_set
```

Out[8]:

|  | support | itemsets |
|---|---|---|
| **0** | 0.033452 | (UHT-milk) |
| **1** | 0.052466 | (beef) |
| **2** | 0.033249 | (berries) |
| **3** | 0.026029 | (beverages) |
| **4** | 0.080529 | (bottled beer) |
| **...** | ... | ... |
| **117** | 0.032232 | (whole milk, whipped/sour cream) |
| **118** | 0.020742 | (yogurt, whipped/sour cream) |
| **119** | 0.056024 | (whole milk, yogurt) |
| **120** | 0.023183 | (whole milk, other vegetables, root vegetables) |
| **121** | 0.022267 | (whole milk, other vegetables, yogurt) |

122 rows × 2 columns

OK. Now we have 122 frequent itemsets, sorted according to the support.

# How to check the i-th frequent itemset?

In [9]:
```
# check the 10th frequent itemset
freq_set.loc[[10], ['support', 'itemsets']]
```

Out[9]:

|  | support | itemsets |
|---|---|---|
| **10** | 0.077682 | (canned beer) |

# How to check whether an itemset is frequent?

If it is frequent, provide the location of the itemset in **feq_set**; otherwise provide "Not frequent".

### Check whether 'beef' is frequent

In [10]:
```
# specify the itemset you want to check
check_set = ['beef']

# Select the idx from the frequent set based on the given check_set
itemset_idx = freq_set.index[freq_set['itemsets'] == frozenset(check_set)].tol
ist()
if itemset_idx==[]: # given check_set does not exist in the frequent set
    print('Not frequent!')
else:
    print('Found at location '+str(itemset_idx[0]))
```

```
Found at location 1
```

## Check whether 'whole milk, yogurt' is frequent

In [11]:
```
check_set = ['yogurt','whole milk']

# Select the idx from the frequent set based on the given check_set
itemset_idx = freq_set.index[freq_set['itemsets'] == frozenset(check_set)].tol
ist()
if itemset_idx==[]: # given check_set does not exist in the frequent set
    print('Not frequent!')
else:
    print('Found at location '+str(itemset_idx[0]))
```

```
Found at location 119
```

## Check whether 'university, queensland' is frequent

In [12]:
```
check_set = ['university','queensland']

# Select the idx from the frequent set based on the given check_set
itemset_idx = freq_set.index[freq_set['itemsets'] == frozenset(check_set)].tol
ist()
if itemset_idx==[]:
    print('Not frequent!') # given check_set does not exist in the frequent se
t
else:
    print('Found at location '+str(itemset_idx[0]))
```

```
Not frequent!
```

Great! You can play with your own sets and see if they are frequent.

**Execrise**: after we find an itemset is frequent, how can we have its support? (hint: use the location of the frequent itemset.)

## Section 4.1 Calcualte the confidence

In [13]:

```
"""
Return the support of the given itemset X
"""
def get_itemset_support(freq_set, X):
    # Select the idx from the frequent set based on the given check_set
    itemset_idx = freq_set.index[freq_set['itemsets'] == frozenset(X)].tolist
()
    if itemset_idx==[]:
        return None # Request itemset X does not exist in the frequent itemset
    else:
        return freq_set.loc[itemset_idx[0],['support']] # Return the correspon
ding support

"""
Print the confidence of the given itemset {X} -> {Y}
"""
def get_rule_confidence(freq_set, X, Y):

    itemset = X + Y # join itemset X and itemset Y
    x_support = get_itemset_support(freq_set, X) # get support of X
    joint_support = get_itemset_support(freq_set, itemset) # get support of X
 joint Y

    if joint_support is None or x_support is None:
        return "Make sure the X, Y and X+Y are in the frequent list."

    print("The confidence of rule {%s} -> {%s} is: %3f"%(X, Y, joint_support/x
_support))
```

### Now, let's calculate the confidence of rule {X} -> {Y}

In [14]:

```
# Specify the content of X and Y
X = ['yogurt', 'whole milk']
Y = ['other vegetables']

# Get the confidence
get_rule_confidence(freq_set, X, Y)
```

```
The confidence of rule {['yogurt', 'whole milk']} -> {['other vegetables']} i
s: 0.397459
```

In [15]:

```
# Specify the content of X and Y
X = ['queensland']
Y = ['university']

# Get the confidence
get_rule_confidence(freq_set, X, Y)
```

Out[15]:

```
'Make sure the X, Y and X+Y are in the frequent list.'
```

This is the end of the tutorial. You can play with your own rules now.

```
In [ ]:
```