```
import numpy as np
import pandas as pd
from collections import defaultdict
from collections import Counter
from itertools import combinations, chain
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
```

```
from google.colab import files
uploaed = files.upload()
```

Choose Files 2 files

- **Online Retail.xlsx**(application/vnd.openxmlformats-officedocument.spreadsheetml.sheet) - 23715344 byt 3/16/2021 - 100% done
- **store_data.csv**(application/vnd.ms-excel) - 431066 bytes, last modified: 3/16/2021 - 100% done

```
Saving Online Retail.xlsx to Online Retail.xlsx
Saving store_data.csv to store_data.csv
```

1. Get online Retail dataset from http://archive.ics.uci.edu/ml/datasets/Online+Retail

It consists of 541909 instances with 8 attributes.

```
# Upload the dataset,create a dataframe from the dataset
online_data = pd.read_excel("Online Retail.xlsx")
```

```
online_data.head(20)
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	Custo
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART	6	2010-12-01	3.39	17

Pre Processing data

```
def CheckMissingData(df):
  """Checking for missing data"""
  for col in df.columns:
      pct_missing = np.mean(df[col].isnull())
      print('{} - {}%'.format(col, round(pct_missing*100)))
```

HAND WARMER RED 2010-12-01

```
CheckMissingData(online_data)
```

```
InvoiceNo - 0%
StockCode - 0%
Description - 0%
Quantity - 0%
InvoiceDate - 0%
UnitPrice - 0%
CustomerID - 25%
Country - 0%
```

```
online_data.shape
```

```
(541909, 8)
```

```
# Drop the rows with missing data
pp_data = online_data.dropna()
```

```
# Stripping extra spaces in the description
pp_data['Description'] = pp_data['Description'].str.strip()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
```

```
CheckMissingData(pp_data)
```

```
InvoiceNo - 0%
StockCode - 0%
Description - 0%
Quantity - 0%
InvoiceDate - 0%
UnitPrice - 0%
CustomerID - 0%
Country - 0%
```

```
pp_data.shape
```

```
(406829, 8)
```

```
len(pp_data.InvoiceNo.unique())
```

```
22190
```

a. Write a function to choose unique ‘k’ items from the dataset.

```
def uniqueItem(k):
  """ This finction returns  unique ‘k’ items,StockCode from the dataset"""
  uniq_ls = pp_data.StockCode.unique()
  test_list = uniq_ls[0:k]
  test_list = [str(i) for i in test_list]
  return test_list
```

```
item_list = uniqueItem(7) # Return 8 unique items list
item_list
```

```
['85123A', '71053', '84406B', '84029G', '84029E', '22752', '21730']
```

b. Write a function to find the customers who bought items from a given list of ‘k’ items and output in the form of a transaction matrix.

```
# Create dictionary with custmer ID as key and items brought by them as values
cust_list = defaultdict(list)
customers = list(pp_data["CustomerID"])
items  = list(pp_data["StockCode"])
```

```
for i in range(len(pp_data)):
  cust_list[customers[i]].append(str(items[i]))


def CustItemDf(custdic,il):
    """
    """
    item_list = sorted(il)
    ## Create data for the dataframe
    df_Total_data = []
    for i in custdic:
      df_data = []
      df_data.append(i)
      for j in range(len(item_list)):
        if item_list[j] in custdic[i]:
          df_data.append(1)
        else:
          df_data.append(0)
      df_Total_data.append(df_data)
    column_names = ["Customer"] + item_list
    return column_names , df_Total_data




def CustomerItem(item_list):
  cust_result = []
  for item in item_list:
    for i in cust_list:
      if item in cust_list[i]:
        cust_result.append(i)
  cust_result = list(set(cust_result))

  cust_list_subset = {key: value for key, value in cust_list.items() if key in cust_result}
  column_names , data = CustItemDf(cust_list_subset,item_list)
  # Create tranaction data frame
  df = pd.DataFrame(data,columns=column_names)
  print(df)




item_list

     ['85123A', '71053', '84406B', '84029G', '84029E', '22752', '21730']


CustomerItem(item_list)

            Customer  21730  22752  71053  84029E  84029G  84406B  85123A
     0      17850.0      1      1      1       1       1       1       1
     1      13047.0      0      0      0       0       0       0       1
     2      15291.0      0      0      0       0       0       1       0
```

```
3      15311.0      0      1      0       0       0       1       0
4      14527.0      0      0      1       1       0       0       1
...        ...    ...    ...    ...     ...     ...     ...     ...
1301   15318.0      0      1      0       0       0       0       0
1302   12650.0      0      1      0       0       0       0       0
1303   14578.0      0      0      0       1       1       0       0
1304   15471.0      0      0      1       0       0       0       1
1305   13298.0      0      0      0       0       1       0       0

[1306 rows x 8 columns]
```

2. Frequent k-itemset.- L(k)

a. Given a support threshold ‘s’, write a function to find the k-itemset having support greater than ‘s’.

```
# Get unique items, item list
item_list = list(basket_sets.columns)
for i in range(len(item_list)):
  txt = str(item_list[i])
  item_list[i]= txt.split("_")[1]
item_list = set(item_list)
item_list = sorted(item_list)
#Number of unique items
len(item_list)
```

```
120
```

```
# Get the tranasction list
data = data.fillna('')
txn_list  = data.values.tolist()
for i in range(len(txn_list)):
  a =  [x for x in txn_list[i] if x != '']
  txn_list[i] = a
# Sample tranction element
txn_list[1]
```

```
['burgers', 'meatballs', 'eggs']
```

```
# candiate dic
c = {}
# Frequent Item Dic
l = {}
# Add candidate item set for 1
discarded = {1:[]}
c.update({1:item_list})
```

```
supp_count_l = {}
min_support = 0.05
```

```
# This function counts the number of occurance of an item set
def count_occurences(itemset,tranactions):
  count = 0
  for i in range(len(tranactions)):
    if set(itemset).issubset(set(tranactions[i])):
      count += 1
  return count



# This function returns the number of frequent item set, thier support and the list of item s
def get_freqItemSet(item_set,transactions,min_support,prev_discarded):
  L = [] # List of feq item
  supp_count = []
  new_discarded = []

  k = len(prev_discarded.keys()) # could be list
  for s in range(len(item_set)):
    discarded_before = False
    if k>0:
      for it in prev_discarded[k]:
        if set(it).issubset(set(item_set[s])):
          discarded_before = True
          break
    if not discarded_before:
      count = count_occurences(item_set[s],transactions)
      if count/len(transactions) >= min_support:
        L.append(item_set[s])
        supp_count.append(count)
      else:
        new_discarded.append(item_set[s])
  return L,supp_count,new_discarded




# Function Call
f,sup,new_discarded = get_freqItemSet(c[1],txn_list,min_support,discarded)


discarded.update({1:new_discarded})
l.update({1:f})
supp_count_l.update({1:sup})


# Display the frequent item set
print("The frequent item set : \n" ,l[1])
```

```
The frequent item set :
 ['burgers', 'cake', 'chicken', 'chocolate', 'cookies', 'cooking oil', 'eggs', 'escalope
```

```
print("Total number of item set ", len(item_list))
print("Number of freq item set " ,len(l[1]))
```

```
Total number of item set  120
Number of freq item set  25
```

3 . Candidate itemset

a. Given two frequent k-itemset, L(k), generate L(k+1)

```
def Candidate_itemsets(set_of_items):
  c = []
  for i in range(len(set_of_items)):
    for j in range(i+1, len(set_of_items)):
      it_out = join_two_itemsets(set_of_items[i], set_of_items[j])
      if len(it_out)>0:
        c.append(it_out)
  return c


def join_two_itemsets(it1,it2):
  it1 = sorted([it1])
  it2 = sorted([it2])
  for i in range(len(it1)-1):
    if it1[i] != it2[i]:
       return []

  return it1 + [it2[-1]]



 c.update({2:Candidate_itemsets(l[1])})


# Candidate item list C2:
print("List of candidate item set : \n")
c[2]
```

```
      ['ground beef', 'spaghetti'],
      ['ground beef', 'tomatoes'],
      ['ground beef', 'turkey'],
      ['ground beef', 'whole wheat rice'],
      ['low fat yogurt', 'milk'],
      ['low fat yogurt', 'mineral water'],
      ['low fat yogurt', 'olive oil'],
      ['low fat yogurt', 'pancakes'],
      ['low fat yogurt', 'shrimp'],
      ['low fat yogurt', 'soup'],
      ['low fat yogurt', 'spaghetti'],
      ['low fat yogurt', 'tomatoes'],
      ['low fat yogurt', 'turkey'],
      ['low fat yogurt', 'whole wheat rice'],
```

```
        ['milk', 'mineral water'],
        ['milk', 'olive oil'],
        ['milk', 'pancakes'],
        ['milk', 'shrimp'],
        ['milk', 'soup'],
        ['milk', 'spaghetti'],
        ['milk', 'tomatoes'],
        ['milk', 'turkey'],
        ['milk', 'whole wheat rice'],
        ['mineral water', 'olive oil'],
        ['mineral water', 'pancakes'],
        ['mineral water', 'shrimp'],
        ['mineral water', 'soup'],
        ['mineral water', 'spaghetti'],
        ['mineral water', 'tomatoes'],
        ['mineral water', 'turkey'],
        ['mineral water', 'whole wheat rice'],
        ['olive oil', 'pancakes'],
        ['olive oil', 'shrimp'],
        ['olive oil', 'soup'],
        ['olive oil', 'spaghetti'],
        ['olive oil', 'tomatoes'],
        ['olive oil', 'turkey'],
        ['olive oil', 'whole wheat rice'],
        ['pancakes', 'shrimp'],
        ['pancakes', 'soup'],
        ['pancakes', 'spaghetti'],
        ['pancakes', 'tomatoes'],
        ['pancakes', 'turkey'],
        ['pancakes', 'whole wheat rice'],
        ['shrimp', 'soup'],
        ['shrimp', 'spaghetti'],
        ['shrimp', 'tomatoes'],
        ['shrimp', 'turkey'],
        ['shrimp', 'whole wheat rice'],
        ['soup', 'spaghetti'],
        ['soup', 'tomatoes'],
        ['soup', 'turkey'],
        ['soup', 'whole wheat rice'],
        ['spaghetti', 'tomatoes'],
        ['spaghetti', 'turkey'],
        ['spaghetti', 'whole wheat rice'],
        ['tomatoes', 'turkey'],
        ['tomatoes', 'whole wheat rice'],
        ['turkey', 'whole wheat rice']]
```

```
f,sup,new_discarded = get_freqItemSet(c[2],txn_list,min_support,discarded)


discarded.update({2:new_discarded})
l.update({2:f})
supp_count_l.update({2:sup})


# Frequent Item set
print("Frequent Item set : L2 : ")
```

```
print('Frequent Item set : L2 :')
l[2]
```

```
Frequent Item set : L2 :
[['chocolate', 'mineral water'],
 ['eggs', 'mineral water'],
 ['mineral water', 'spaghetti']]
```

4. Given an itemset with cardinality ‘T’, and confidence threshold ‘c’, write a function to output the possible association rules with confidence greater than ‘c’.

```
def powerset(s):
  return list(chain.from_iterable(combinations([s],r) for r in range(1,len([s])+1)))


def write_rules(x,x_s,s,conf,supp,lift,num_trans):
  out_rules  = ""
  out_rules +="Freq Itemset  \n".format(x)
  out_rules += "  rule {} -> {}\n ".format(list(s),list(x_s))
  out_rules += "  conf : {0:2.3f} ".format(conf)
  out_rules += "  supp : {0:2.3f} ".format(supp/num_trans)
  out_rules += "  conf : {0:2.3f} ".format(lift)
  print(x)
  return out_rules


num_trans = len(data)
min_conf = 0.01


assoc_rules_str = ""
for i in range(1,len(l)):
  for j in range(len(l[i])):
    a  = [l[i][j]]
    s = list(powerset(a))
    s.pop()
    for z in s:
      s = set(z)
      x = set(l[i][j])
      x_s = set(x-s)
      sup_x = count_occurences(x,txn_list)
      sup_x_s = count_occurences(x_s,txn_list)
      conf = sup_x/count_occurences(s,txn_list)
      lift = conf/(sup_x_s/num_trans)
      if conf >= min_conf and sup_x >= min_support:
        assoc_rules_str += write_rules(x,x_s,s,conf,sup_x,lift,num_trans)
```

```
assoc_rules_str
```

```
''
```

6. Follow https://www.geeksforgeeks.org/implementing-apriori-algorithm-in-python/ to use in-built apriori algorithm.

Apriori Inbuilt Function - Online Store Dataset

```
data = pp_data.copy()
```

```
data.Country.unique()
```

```
array(['United Kingdom', 'France', 'Australia', 'Netherlands', 'Germany',
       'Norway', 'EIRE', 'Switzerland', 'Spain', 'Poland', 'Portugal',
       'Italy', 'Belgium', 'Lithuania', 'Japan', 'Iceland',
       'Channel Islands', 'Denmark', 'Cyprus', 'Sweden', 'Austria',
       'Israel', 'Finland', 'Greece', 'Singapore', 'Lebanon',
       'United Arab Emirates', 'Saudi Arabia', 'Czech Republic', 'Canada',
       'Unspecified', 'Brazil', 'USA', 'European Community', 'Bahrain',
       'Malta', 'RSA'], dtype=object)
```

```
data.head(5)
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Cou
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	U Kinç
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	U Kinç

```
# Splitting the data according to the region of transaction

# Transactions done in France
basket_France = (data[data['Country'] =="France"]
          .groupby(['InvoiceNo', 'Description'])['Quantity']
          .sum().unstack().reset_index().fillna(0)
          .set_index('InvoiceNo'))
```

```
# Transactions done in the United Kingdom
basket_UK = (data[data['Country'] =="United Kingdom"]
          .groupby(['InvoiceNo', 'Description'])['Quantity']
          .sum().unstack().reset_index().fillna(0)
          .set_index('InvoiceNo'))

# Transactions done in Portugal
basket_Por = (data[data['Country'] =="Portugal"]
          .groupby(['InvoiceNo', 'Description'])['Quantity']
          .sum().unstack().reset_index().fillna(0)
          .set_index('InvoiceNo'))

basket_Sweden = (data[data['Country'] =="Sweden"]
          .groupby(['InvoiceNo', 'Description'])['Quantity']
          .sum().unstack().reset_index().fillna(0)
          .set_index('InvoiceNo'))



# Hot encoding the Data
# Defining the hot encoding function to make the data suitable
# for the concerned libraries
def hot_encode(x):
  if(x<= 0):
    return 0
  if(x>= 1):
    return 1

# Encoding the datasets
basket_encoded = basket_France.applymap(hot_encode)
basket_France = basket_encoded

basket_encoded = basket_UK.applymap(hot_encode)
basket_UK = basket_encoded

basket_encoded = basket_Por.applymap(hot_encode)
basket_Por = basket_encoded

basket_encoded = basket_Sweden.applymap(hot_encode)
basket_Sweden = basket_encoded



# Buliding the models and analyzing the results


# a) France
# Building the model
frq_items = apriori(basket_France, min_support = 0.05, use_colnames = True)

# Collecting the inferred rules in a dataframe
rules = association_rules(frq_items, metric ="lift", min_threshold = 1)
```

```
rules = rules.sort_values(['confidence', 'lift'], ascending =[False, False])
print(rules.head())
```

```
                                           antecedents  ... conviction
30                         (JUMBO BAG WOODLAND ANIMALS)  ...        inf
208  (SET/20 RED RETROSPOT PAPER NAPKINS, SET/6 RED...  ...  35.633188
209  (SET/20 RED RETROSPOT PAPER NAPKINS, SET/6 RED...  ...  35.283843
215  (SET/20 RED RETROSPOT PAPER NAPKINS, SET/6 RED...  ...  29.397380
216  (SET/20 RED RETROSPOT PAPER NAPKINS, SET/6 RED...  ...  29.109170

[5 rows x 9 columns]
```

Apriori Inbuilt Function - Store data Dataset

```
data = pd.read_csv("store_data.csv",header=None)
```

```
data.head()
```

	0	1	2	3	4	5	6	7	8	9	1
0	shrimp	almonds	avocado	vegetables mix	green grapes	whole weat flour	yams	cottage cheese	energy drink	tomato juice	lo f yogu
1	burgers	meatballs	eggs	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
2	chutney	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
3	turkey	avocado	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
4	mineral water	milk	energy bar	whole wheat rice	green tea	NaN	NaN	NaN	NaN	NaN	Na

```
# Data Conversion
basket_sets = pd.get_dummies(data)
basket_sets.head()
```

```
apriori(basket_sets, min_support=0.02,use_colnames = True)
```

	support	itemsets
0	0.076790	(0_burgers)
1	0.052126	(0_chocolate)
2	0.035995	(0_cookies)
3	0.037195	(0_eggs)
4	0.032529	(0_french fries)
5	0.049727	(0_frozen vegetables)

```
df = basket_sets

frequent_itemsets = apriori(basket_sets, min_support=0.002, use_colnames=True)

frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(lambda x: len(x))
frequent_itemsets
```

	support	itemsets	length
0	0.002400	(0_antioxydant juice)	1
1	0.007599	(0_avocado)	1
2	0.003066	(0_brownies)	1
3	0.076790	(0_burgers)	1
4	0.006932	(0_butter)	1
...	...	...	...
658	0.002266	(1_spaghetti, 2_mineral water, 3_soup)	3
659	0.002266	(1_tomatoes, 3_mineral water, 2_spaghetti)	3
660	0.003600	(2_ground beef, 4_mineral water, 3_spaghetti)	3
661	0.002533	(2_mineral water, 4_milk, 3_soup)	3
662	0.002133	(4_milk, 3_mineral water, 2_spaghetti)	3

663 rows × 3 columns

```
frequent_itemsets[frequent_itemsets['length'] >= 3]
```

	support	itemsets	length
650	0.002400	(1_spaghetti, 2_mineral water, 0_frozen vegeta...	3
651	0.003600	(1_spaghetti, 2_mineral water, 0_ground beef)	3
652	0.002400	(1_frozen vegetables, 2_tomatoes, 0_shrimp)	3
653	0.002133	(1_frozen vegetables, 3_spaghetti, 0_shrimp)	3
654	0.002266	(1_mineral water, 0_spaghetti, 2_milk)	3
655	0.002133	(1_frozen vegetables, 2_ground beef, 3_spaghetti)	3
656	0.004399	(2_spaghetti, 3_mineral water, 1_ground beef)	3

Association Rules

Confidence

```
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.5)
rules.head()
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	lev
0	(1_fresh tuna)	(0_burgers)	0.004533	0.076790	0.003066	0.676471	8.809385	0.0
1	(1_ham)	(0_burgers)	0.007599	0.076790	0.003999	0.526316	6.853984	0.0
2	(1_pasta)	(0_escalope)	0.005333	0.019064	0.002666	0.500000	26.227273	0.0
3	(1_burgers)	(0_turkey)	0.010399	0.061059	0.009865	0.948718	15.537846	0.0

Lift

```
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)
rules.head()
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leve
0	(1_chocolate)	(0_burgers)	0.029729	0.076790	0.002533	0.085202	1.109546	0.00
1	(0_burgers)	(1_chocolate)	0.076790	0.029729	0.002533	0.032986	1.109546	0.00
2	(1_eggs)	(0_burgers)	0.040261	0.076790	0.006799	0.168874	2.199176	0.00
3	(0_burgers)	(1_eggs)	0.076790	0.040261	0.006799	0.088542	2.199176	0.00

Lift and Confidence

```
rules[(rules['lift'] >= 5) & (rules['confidence']>= 0.5)]
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift
4	(1_fresh tuna)	(0_burgers)	0.004533	0.076790	0.003066	0.676471	8.809385
12	(1_ham)	(0_burgers)	0.007599	0.076790	0.003999	0.526316	6.853984
100	(1_pasta)	(0_escalope)	0.005333	0.019064	0.002666	0.500000	26.227273
274	(1_burgers)	(0_turkey)	0.010399	0.061059	0.009865	0.948718	15.537846
345	(2_tomatoes)	(1_frozen vegetables)	0.011332	0.031196	0.005733	0.505882	16.216340
460	(3_tomatoes)	(2_frozen vegetables)	0.004933	0.011598	0.003200	0.648649	55.925443
491	(3_soup)	(2_mineral water)	0.009599	0.049993	0.006133	0.638889	12.779481
548	(4_mineral water)	(3_spaghetti)	0.011199	0.022264	0.008399	0.750000	33.687126
560	(4_spaghetti)	(5_mineral water)	0.008266	0.005866	0.004533	0.548387	93.487537
561	(5_mineral water)	(4_spaghetti)	0.005866	0.008266	0.004533	0.772727	93.487537
572	(2_mineral water, 0_ground beef)	(1_spaghetti)	0.004666	0.054793	0.003600	0.771429	14.079041
578	(2_tomatoes, 0_shrimp)	(1_frozen vegetables)	0.002533	0.031196	0.002400	0.947368	30.368421
584	(3_spaghetti, 0_shrimp)	(1_frozen vegetables)	0.003066	0.031196	0.002133	0.695652	22.299517
590	(0_spaghetti, 2_milk)	(1_mineral water)	0.002933	0.064525	0.002266	0.772727	11.975676

```
basket_sets.shape
```

```
(7501, 1269)
```