Association Rules

Example with the **Online Retail** dataset, from UCI

Code inspired from this link

In [ ]:
```
# pip install openpyxl # for Excel file handling
```

In [ ]:
```
# pip install mlxtend # for association rules
```

In [4]:
```
import pandas as pd
import numpy as np
from mlxtend.frequent_patterns import apriori, association_rules
```

1 Upload the file `'Online-Retail-France.xlsx'`.

It is a MS Excel file, you can read it with the Pandas' function `read_excel`.

Inspect its content. It is a transactional database where the role of transaction identifier is played by the column `InvoiceNo` and the items are in the column `Description`.

In [5]:
```
# url = 'http://archive.ics.uci.edu/ml/machine-learning-databases/00352/Online%2
url = '../../data/Online-Retail-France.xlsx'
df0 = pd.read_excel(url
                    # , dtype = {  'InvoiceNo': 'str'
                    #              , 'StockCode':'str'
                    #              , 'Description':'str'
                    #              , 'CustomerID':'Int64'
                    #              }
                    )
print("There are {} rows and {} columns".format(df0.shape[0], df0.shape[1]))
```

```
There are 8557 rows and 7 columns
```

In [6]:
```
df0.head()
```

Out[6]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID
0	536370	22728	ALARM CLOCK BAKELIKE PINK	24	2010-12-01 08:45:00	3.75	12583.0
1	536370	22727	ALARM CLOCK BAKELIKE RED	24	2010-12-01 08:45:00	3.75	12583.0
2	536370	22726	ALARM CLOCK BAKELIKE GREEN	12	2010-12-01 08:45:00	3.75	12583.0
3	536370	21724	PANDA AND BUNNIES STICKER SHEET	12	2010-12-01 08:45:00	0.85	12583.0
4	536370	21883	STARS GIFT TAPE	24	2010-12-01 08:45:00	0.65	12583.0

In [7]:
```
df0.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8557 entries, 0 to 8556
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   InvoiceNo    8557 non-null   object 
 1   StockCode    8557 non-null   object 
 2   Description  8557 non-null   object 
 3   Quantity     8557 non-null   int64  
 4   InvoiceDate  8557 non-null   object 
 5   UnitPrice    8557 non-null   float64
 6   CustomerID   8491 non-null   float64
dtypes: float64(2), int64(1), object(4)
memory usage: 468.1+ KB
```

2 Print the number of unique Description values

In [8]:
```
print("The number of unique Description values in the input file is {}".format(l
```

```
The number of unique Description values in the input file is 1565
```

3 Use Pandas function str.strip() Print the number of unique Description values after this cleaning

In [9]:
```
df1 = df0
df1['Description'] = df0['Description'].str.strip()
```

In [10]:
```
print("After cleaning, the number of unique Description values in the input file
```

```
After cleaning, the number of unique Description values in the input file is 1564
```

4 Clean missing `InvoiceNo`

Some rows may not have an `InvoiceNo` and must be removed, because they cannot be used.

Check if there are such that rows and in case remove them.

In [11]:
```
print("Rows with missing InvoiceNo before removing")
df1[df1['InvoiceNo'].isna()]
```

```
Rows with missing InvoiceNo before removing
```

Out[11]:

InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID

In [12]:
```
df2 = df1.dropna(axis=0, subset=['InvoiceNo'])
```

In [13]:
```
print("Rows with missing InvoiceNo after removing")
df2[df2['InvoiceNo'].isna()]
```

```
Rows with missing InvoiceNo after removing
```

Out[13]:

InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID

5 Some InvoiceNo start with a `C`. They are "credit transactions" and must be removed.

Check the number of rows containing `C` in `InvoiceNo` and remove them. At the moment the column `InvoiceNo` is a generic object, in order to be able to use string functions, such as `contains`, it must be transformed into `str` with `astype`.

In [14]:
```
print("There are {} rows containing 'C' in 'InvoiceNo'"\
      .format(sum(df2['InvoiceNo'].str.contains('C'))))
      # .format(sum(df2['InvoiceNo'].astype('str').str.contains('C'))))
```

```
There are 149 rows containing 'C' in 'InvoiceNo'
```

In [15]:
```
df3 = df2[~df2['InvoiceNo'].str.contains('C')]
```

In [16]:
```
print("After removal, there are {} rows containing 'C' in 'InvoiceNo'"\
      .format(sum(df3['InvoiceNo'].str.contains('C'))))
```

```
After removal, there are 0 rows containing 'C' in 'InvoiceNo'
```

6 Clean `POSTAGE`

Several transactions include the item `'POSTAGE'`, which represents the mailing expenses. In this analysis we are not interested in it, therefore the rows with `'POSTAGE'` will be removed.

In [17]:
```
container = 'Description'
target = 'POSTAGE'
```

```
print("There are {} rows containing {} in {}"\
      .format(sum(df3[container].str.contains(target)), target, container))
```

```
There are 300 rows containing POSTAGE in Description
```

In [18]:
```
df = df3[~df3[container].str.contains(target)]
print("After cleaning there are {} rows containing {} in {}"\
      .format(sum(df[container].str.contains(target)), target, container))
```

```
After cleaning there are 0 rows containing POSTAGE in Description
```

In [19]:
```
df.describe(include='all')
```

Out[19]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	Cu
count	8108	8108	8108	8108.000000	8108	8108.000000	80
unique	387	1541	1562	NaN	388	NaN	
top	570672	23084	RABBIT NIGHT LIGHT	NaN	2011-10-11 14:52:00	NaN	
freq	259	74	74	NaN	259	NaN	
mean	NaN	NaN	NaN	13.724593	NaN	3.839639	126
std	NaN	NaN	NaN	21.353250	NaN	66.583173	2
min	NaN	NaN	NaN	1.000000	NaN	0.000000	124
25%	NaN	NaN	NaN	6.000000	NaN	1.000000	125
50%	NaN	NaN	NaN	10.000000	NaN	1.650000	126
75%	NaN	NaN	NaN	12.000000	NaN	2.950000	126
max	NaN	NaN	NaN	912.000000	NaN	4161.060000	142

7 Consolidate the items into 1 transaction per row with each product 1 hot encoded

After the cleanup, we need to consolidate the items into 1 transaction per row with each product *one-hot encoded*.

Actions:

1. group by `['InvoiceNo', 'Description']` computing a sum on `['Quantity']`
2. use the `unstack` function to move the items from rows to columns
3. reset the index
4. fill the missing with zero (`fillna(0)`)
5. store the result in the new dataframe `basket` and inspect it

In [20]:
```
basket = (df
          .groupby(['InvoiceNo', 'Description'])['Quantity']
          .sum().unstack().reset_index().fillna(0)
```

```
          .set_index('InvoiceNo')) # in this way, InvoiceNo is not a column anym
basket.head()
```

Out[20]:

Description	10 COLOUR SPACEBOY PEN	12 COLOURED PARTY BALLOONS	12 EGG HOUSE PAINTED WOOD	12 MESSAGE CARDS WITH ENVELOPES	12 PENCIL SMALL TUBE WOODLAND	12 PENCILS SMALL TUBE RED RETROSPOT
InvoiceNo						
536370	0.0	0.0	0.0	0.0	0.0	0.0
536852	0.0	0.0	0.0	0.0	0.0	0.0
536974	0.0	0.0	0.0	0.0	0.0	0.0
537065	0.0	0.0	0.0	0.0	0.0	0.0
537463	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 1562 columns

In [21]:
```
basket.describe()
```

Out[21]:

Description	10 COLOUR SPACEBOY PEN	12 COLOURED PARTY BALLOONS	12 EGG HOUSE PAINTED WOOD	12 MESSAGE CARDS WITH ENVELOPES	12 PENCIL SMALL TUBE WOODLAND	12 PENCILS SMALL TUBE RED RETROSPOT
count	387.000000	387.000000	387.000000	387.000000	387.000000	387.000000
mean	0.868217	0.310078	0.005168	0.062016	0.509044	0.385013
std	5.109242	2.474110	0.101666	0.861544	5.445138	2.978173
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
max	48.000000	20.000000	2.000000	12.000000	96.000000	24.000000

8 rows × 1562 columns

8 Conversion to binary values

There are a lot of zeros in the data but we also need to make sure any positive values are converted to a 1 and anything less the 0 is set to 0.

You can define a function `encode_units` which takes a number and returns 0 if the number is 0 or less, 1 if the number is 1 or more. The function can be applied to

`basket` with the Pandas' function `applymap`, the result is stored in the variable `basket_sets`

Inspect the structure and the correctness of the result

This step will complete the one hot encoding of the data.

In [22]:
```
def encode_units(x):
    if x <= 0:
        return False
    if x >= 1:
        return True

basket_sets = basket.map(encode_units)
basket_sets.describe()
```

Out[22]:

Description	10 COLOUR SPACEBOY PEN	12 COLOURED PARTY BALLOONS	12 EGG HOUSE PAINTED WOOD	12 MESSAGE CARDS WITH ENVELOPES	12 PENCIL SMALL TUBE WOODLAND	12 PENCILS SMALL TUBE RED RETROSPOT
count	387	387	387	387	387	387
unique	2	2	2	2	2	2
top	False	False	False	False	False	False
freq	375	381	386	385	381	380

4 rows × 1562 columns

9 find the maximum value of min_support such that the number of rules generated from the frequent itemsets with lift not less than 1 is at least 20.

Show the value obtained for min_support and show the rules.

Hint: use a loop with an initial value min_support=1 and decrease it in steps -0.01

Hint: In apriori set the parameter use_colnames=True.

In [23]:
```
step = 0.01
min_support = 1
min_rules = 20
metric = 'lift'
min_threshold = 1
while True:
    frequent_itemsets = apriori(basket_sets, min_support=min_support, use_colnam
    if frequent_itemsets.shape[0]>0:
        rules = association_rules(frequent_itemsets, metric=metric, min_threshol
    if frequent_itemsets.shape[0] >0 and rules.shape[0]>=min_rules:
        break
    else:
        min_support -= step
```

10. Generate the rules with `association_rules` using `metric=lift` and `min_threshold=1`

In [24]:
```
print("'min_support'={:4.2f}, 'metric'={}, 'min_threshold'={}, number of frequen
      .format(min_support, metric, min_threshold, frequent_itemsets.shape[0], ru
```

```
'min_support'=0.08, 'metric'=lift, 'min_threshold'=1, number of frequent itemsets
=41, number of rules=20
```

In [2]:
```
print(rules)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[2], line 1
----> 1 print(rules)

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

11 Plot the rules

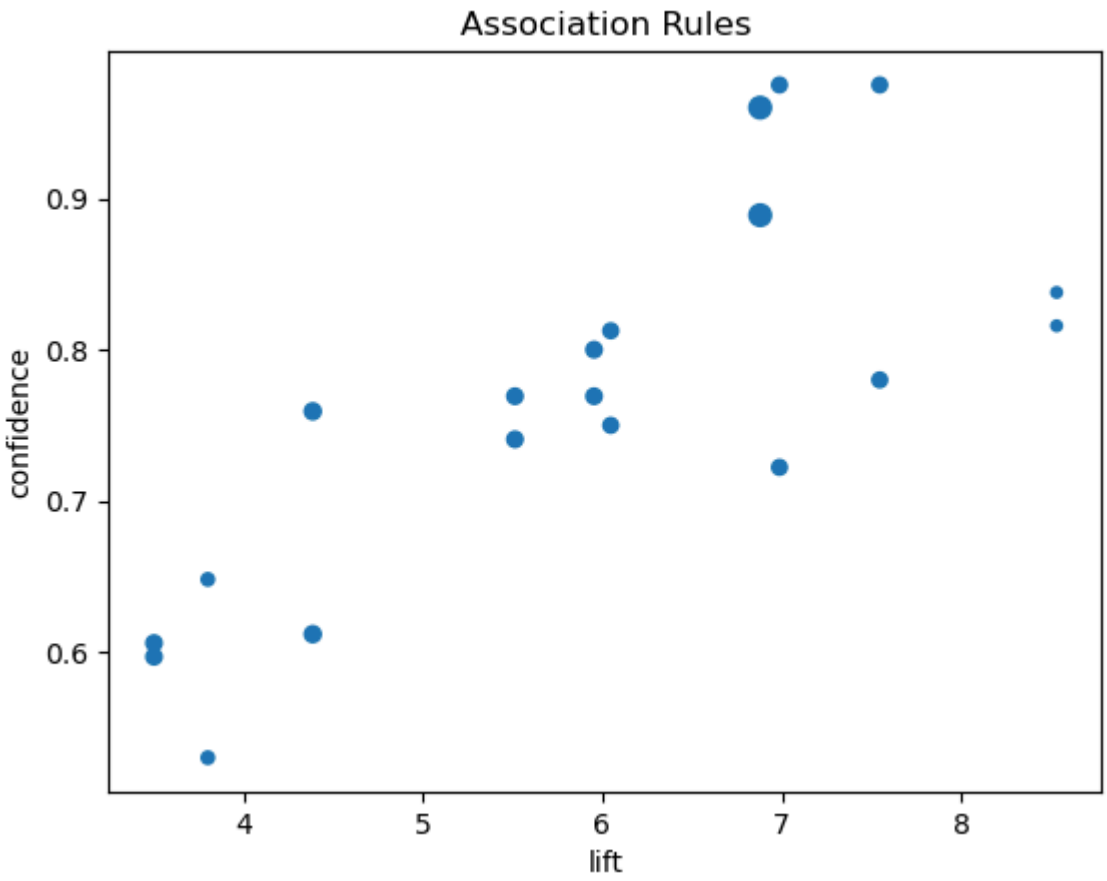
In order to plot the rules, it is better to sort them according to some metrics. We will sort on descending lift, support and confidence and scatter plot `'lift'` in x axis, `'support'` in y axis and `'confdence'` as the dot size.

In [26]:
```
sorted_rules=rules.sort_values(by=['lift','confidence'],ascending=False).reset_i
sorted_rules.head()
```

Out[26]:

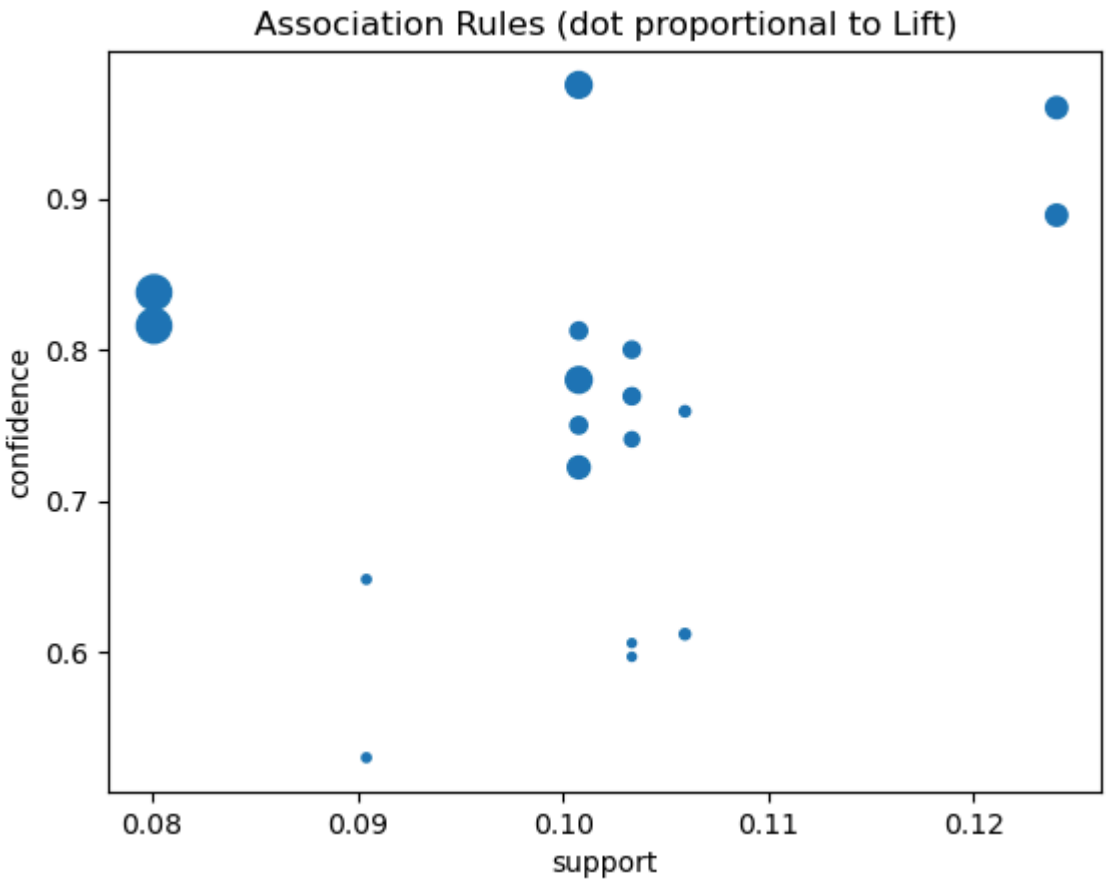
	antecedents	consequents	antecedent support	consequent support	support	confidence	lift
0	(ALARM CLOCK BAKELIKE RED)	(ALARM CLOCK BAKELIKE GREEN)	0.095607	0.098191	0.080103	0.837838	8.532717
1	(ALARM CLOCK BAKELIKE GREEN)	(ALARM CLOCK BAKELIKE RED)	0.098191	0.095607	0.080103	0.815789	8.532717
2	(SET/6 RED SPOTTY PAPER CUPS, SET/20 RED RETRO...	(SET/6 RED SPOTTY PAPER PLATES)	0.103359	0.129199	0.100775	0.975000	7.546500
3	(SET/6 RED SPOTTY PAPER PLATES)	(SET/6 RED SPOTTY PAPER CUPS, SET/20 RED RETRO...	0.129199	0.103359	0.100775	0.780000	7.546500
4	(SET/6 RED SPOTTY PAPER PLATES, SET/20 RED RET...	(SET/6 RED SPOTTY PAPER CUPS)	0.103359	0.139535	0.100775	0.975000	6.987500

In [27]:
```
sorted_rules.plot.scatter(x='lift',y='confidence',s=3**(sorted_rules['support']*
```



You find below a three dimensional plot, where the dot size is proportional to the lift, obtained using `plot.scatter`.

In [28]:
```
s = [1.8**n for n in rules.lift]
rules.plot.scatter(x='support',
                   y='confidence',
                   title='Association Rules (dot proportional to Lift)',
                   s=s);
```


Association Rules (dot proportional to Lift)
confidence
0.9
0.8
0.7
0.6
0.08
0.09
0.10
0.11
0.12
support