# Mahmoud Kahack Home Work 3

**2**

a.

s({e}) = $\frac{8}{10}$ = 0.8
s({b,d}) = $\frac{2}{10}$ = 0.2
s({b,d,e}) = 0.2

b.

c({b,d} -> {e}) = $\frac{s(b,d,e)}{s(b,d)}$ = $\frac{0.2}{0.2}$ = 1
c({e} -> {b,d}) = $\frac{0.2}{0.8}$ = 0.25
confidence is clearly not a symmetric metric

c.

s({e}) = $\frac{4}{5}$ = 0.8
s({b,d}) = $\frac{5}{5}$ = 1
s({b,d,e}) = $\frac{4}{5}$ = 0.8

d.

c({b,d} -> {e}) = 0.8
c({e} -> {b,d}) = 1

e.

In [3]:
```
import numpy as np
import pandas as pd

from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_
rules

%matplotlib inline
```

# Problem 1

Load the *Online Retaildataset* (**Online Retail.xlsx**) from the **UCI Machine Learning Repository** into Python using a Pandas dataframe. Using the apriori module from the **MLxtend** library, generate Frequent Itemsets for all trans-actions for the country of France. What itemset has the largest support? Set the minimum support threshold to 5% and extract frequent itemsets, and use them as input to the association rules module. Use each of the confidence and lift metrics to extract the association rules with the highest values, respectively. What are the antecedents and consequents of each rule? Is the rule with the highest confidence the same as the rule with the highest lift? Why or why not?

## Answer :-

The itemset with the largest support is {'RABBIT NIGHT LIGHT'}.

The rule with the highest confidence is {'SET/20 RED RETROSPOT PAPER NAPKINS','SET/6 RED SPOTTY PAPER PLATES'} -> {'SET/6 RED SPOTTY PAPER CUPS'}.

The rule with the highest lift is {'PACK OF 6 SKULL PAPER CUPS'} -> {'PACK OF 6 SKULL PAPER PLATES'}.

The rule with the highest confidence and the rule with the highest lift are different. This because the lift takes into account the joint support of the rule and both the support of the antecedents and consequents support. Confidence only takes into account joint support of the rule and the antecedent's support.

In [92]:
```
df = pd.read_excel('http://archive.ics.uci.edu/ml/machine-learning-databases/00352/Online%20Retail.xlsx')
df.head()
```

Out[92]:

| | InvoiceNo | StockCode | Description | Quantity | Inv |
|---|---|---|---|---|---|
| **0** | 536365 | 85123A | WHITE HANGING HEART T-LIGHT HOLDER | 6 | 201 08: |
| **1** | 536365 | 71053 | WHITE METAL LANTERN | 6 | 201 08: |
| **2** | 536365 | 84406B | CREAM CUPID HEARTS COAT HANGER | 8 | 201 08: |
| **3** | 536365 | 84029G | KNITTED UNION FLAG HOT WATER BOTTLE | 6 | 201 08: |
| **4** | 536365 | 84029E | RED WOOLLY HOTTIE WHITE | 6 | 201 08: |

In [93]:
```
df.describe()
```

Out[93]:

| | Quantity | UnitPrice | Customer |
|---|---|---|---|
| **count** | 541909.000000 | 541909.000000 | 406829.0000 |
| **mean** | 9.552250 | 4.611114 | 15287.69057 |
| **std** | 218.081158 | 96.759853 | 1713.600303 |
| **min** | -80995.000000 | -11062.060000 | 12346.00000 |
| **25%** | 1.000000 | 1.250000 | 13953.00000 |
| **50%** | 3.000000 | 2.080000 | 15152.00000 |
| **75%** | 10.000000 | 4.130000 | 16791.00000 |
| **max** | 80995.000000 | 38970.000000 | 18287.00000 |

In [94]:
```
df['Description'] = df['Description'].str.strip()
df.dropna(axis = 0, subset=['InvoiceNo'], inplace
= True)
df['InvoiceNo'] = df['InvoiceNo'].astype('str')
df = df[~df['InvoiceNo'].str.contains('C')]
```

In [95]:
```
df.describe()
```

Out[95]:

| | Quantity | UnitPrice | Customer |
|---|---|---|---|
| **count** | 532621.000000 | 532621.000000 | 397924.0000 |
| **mean** | 10.239972 | 3.847621 | 15294.31517 |
| **std** | 159.593551 | 41.758023 | 1713.169877 |
| **min** | -9600.000000 | -11062.060000 | 12346.00000 |
| **25%** | 1.000000 | 1.250000 | 13969.00000 |
| **50%** | 3.000000 | 2.080000 | 15159.00000 |
| **75%** | 10.000000 | 4.130000 | 16795.00000 |
| **max** | 80995.000000 | 13541.330000 | 18287.00000 |

In [96]:
```
france_df = df[df["Country"] == "France"]
france_df.describe()
```

Out[96]:

| | Quantity | UnitPrice | CustomerID |
|---|---|---|---|
| **count** | 8408.000000 | 8408.000000 | 8342.000000 |
| **mean** | 13.333016 | 4.399713 | 12678.377128 |
| **std** | 21.069963 | 65.505260 | 277.279060 |
| **min** | 1.000000 | 0.000000 | 12413.000000 |
| **25%** | 6.000000 | 1.250000 | 12571.000000 |
| **50%** | 10.000000 | 1.790000 | 12678.000000 |
| **75%** | 12.000000 | 3.750000 | 12689.000000 |
| **max** | 912.000000 | 4161.060000 | 14277.000000 |

In [97]:
```
french_basket = (france_df
          .groupby(['InvoiceNo', 'Description'])['
Quantity']
          .sum().unstack().reset_index().fillna(0)
          .set_index('InvoiceNo'))

french_basket.tail()
```

Out[97]:

| Description | 10 COLOUR SPACEBOY PEN | 12 COLOURED PARTY BALLOONS | 12 EGG HOUSE PAINTED WOOD | EN |
|---|---|---|---|---|
| InvoiceNo | | | | |
| 580986 | 0.0 | 0.0 | 0.0 | 0.( |
| 581001 | 0.0 | 0.0 | 0.0 | 0.( |
| 581171 | 0.0 | 0.0 | 0.0 | 0.( |
| 581279 | 0.0 | 0.0 | 0.0 | 0.( |
| 581587 | 0.0 | 0.0 | 0.0 | 0.( |

5 rows × 1563 columns

In [99]:

```
def sum_to_boolean(x):
    if x<=0:
        return 0
    else:
        return 1

french_basket_sg = french_basket.applymap(sum_to_b
oolean)
```

In [100]:
```
french_basket_sg.drop('POSTAGE', inplace=True, axis=1)
french_basket_sg.describe()
```

Out[100]:

| Description | 10 COLOUR SPACEBOY PEN | 12 COLOURED PARTY BALLOONS | 12 EGG HOUSE PAINTED WOOD |
|---|---|---|---|
| **count** | 392.000000 | 392.000000 | 392.000000 |
| **mean** | 0.030612 | 0.015306 | 0.002551 |
| **std** | 0.172485 | 0.122924 | 0.050508 |
| **min** | 0.000000 | 0.000000 | 0.000000 |
| **25%** | 0.000000 | 0.000000 | 0.000000 |
| **50%** | 0.000000 | 0.000000 | 0.000000 |
| **75%** | 0.000000 | 0.000000 | 0.000000 |
| **max** | 1.000000 | 1.000000 | 1.000000 |

8 rows × 1562 columns

In [101]:
```
frequent_itemsets = apriori(french_basket_sg, min_
support = 0.05, use_colnames = True)

frequent_itemsets.sort_values('support', ascending
= False).head()
```

Out[101]:

| | support | itemsets |
|---|---|---|
| **46** | 0.188776 | (RABBIT NIGHT LIGHT) |
| **52** | 0.181122 | (RED TOADSTOOL LED NIGHT LIGHT) |
| **44** | 0.170918 | (PLASTERS IN TIN WOODLAND ANIMALS) |
| **40** | 0.168367 | (PLASTERS IN TIN CIRCUS PARADE) |
| **59** | 0.158163 | (ROUND SNACK BOXES SET OF4 WOODLAND) |

In [102]:
```
frequent_itemsets.sort_values('support', ascending
= False).tail()
```

Out[102]:

| | support | itemsets |
|---|---|---|
| **6** | 0.05102 | (BLUE HARMONICA IN BOX) |
| **32** | 0.05102 | (MINI LIGHTS WOODLAND MUSHROOMS) |
| **34** | 0.05102 | (PACK OF 20 NAPKINS RED APPLES) |
| **70** | 0.05102 | (SPACEBOY CHILDRENS BOWL) |
| **79** | 0.05102 | (ALARM CLOCK BAKELIKE RED, ROUND SNACK BOXES S... |

In [112]:
```
con_rules = association_rules(frequent_itemsets, m
etric = "confidence", min_threshold = 0.7)
con_rules.sort_values('confidence',ascending = Fal
se)
```

Out[112]:

| | antecedents | consequents | antecedent support | conse su |
|---|---|---|---|---|
| **25** | (SET/6 RED SPOTTY PAPER PLATES, SET/20 RED RET... | (SET/6 RED SPOTTY PAPER CUPS) | 0.102041 | 0.137 |
| **24** | (SET/6 RED SPOTTY PAPER CUPS, SET/20 RED RETRO... | (SET/6 RED SPOTTY PAPER PLATES) | 0.102041 | 0.127 |
| **18** | (SET/6 RED SPOTTY PAPER PLATES) | (SET/6 RED SPOTTY PAPER CUPS) | 0.127551 | 0.137 |
| **7** | (CHILDRENS CUTLERY SPACEBOY) | (CHILDRENS CUTLERY DOLLY GIRL) | 0.068878 | 0.071 |
| **10** | (PACK OF 6 SKULL PAPER | (PACK OF 6 SKULL PAPER | 0.056122 | 0.063 |

In [111]:
```
lift_rules = association_rules(frequent_itemsets,
metric = "lift", min_threshold = 1)
lift_rules.sort_values('lift',ascending = False).h
ead()
```

Out[111]:

| | antecedents | consequents | antecedent support | consec sup |
|---|---|---|---|---|
| **38** | (PACK OF 6 SKULL PAPER CUPS) | (PACK OF 6 SKULL PAPER PLATES) | 0.063776 | 0.0561 |
| **39** | (PACK OF 6 SKULL PAPER PLATES) | (PACK OF 6 SKULL PAPER CUPS) | 0.056122 | 0.0637 |
| **8** | (CHILDRENS CUTLERY DOLLY GIRL) | (CHILDRENS CUTLERY SPACEBOY) | 0.071429 | 0.0688 |
| **9** | (CHILDRENS CUTLERY SPACEBOY) | (CHILDRENS CUTLERY DOLLY GIRL) | 0.068878 | 0.0714 |
| **69** | (ALARM CLOCK BAKELIKE GREEN, ALARM CLOCK | (ALARM CLOCK BAKELIKE RED) | 0.073980 | 0.0943 |

# Problem 2

Load the *Extended Bakery* dataset (**75000-out2-binary.csv**) into Python using a Pandas dataframe. Calculate the binary correlation coefficient Φ for the *Chocolate Coffee* and *Chocolate Cake* items. Are these two items symmetric binary variables? Provide supporting calculations. Would the association rules {**Chocolate Coffee**} ⇒ {**Chocolate Cake**} have the same value for Φ as {**Chocolate Cake**} ⇒ {**Chocolate Coffee**}?

## Answer :-

The binary correlation is 0.4856. These items are assymmetric binary variables.

s({Chocolate Cake}) = 0.0835
s({Chocolate Coffee}) = 0.08314
s({Chocolate Cake},{Chocolate Coffee}) = 0.4404

The correlation coefficient is the same for both rules {Chocolate Coffee} -> {Chocolate Cake} and {Chocolate Cake} -> {Chocolate Coffee} because the correlation coefficient is invariant.

In [4]:
```
intial_bakery_df = pd.read_csv("75000-out2-binary.csv")
intial_bakery_df.describe()
```

Out[4]:

| | Transaction Number | Chocolate Cake | Lemon Cake |
|---|---|---|---|
| **count** | 75000.000000 | 75000.000000 | 75000.000000 |
| **mean** | 37500.500000 | 0.083533 | 0.083613 |
| **std** | 21650.779432 | 0.276689 | 0.276809 |
| **min** | 1.000000 | 0.000000 | 0.000000 |
| **25%** | 18750.750000 | 0.000000 | 0.000000 |
| **50%** | 37500.500000 | 0.000000 | 0.000000 |
| **75%** | 56250.250000 | 0.000000 | 0.000000 |
| **max** | 75000.000000 | 1.000000 | 1.000000 |

8 rows × 51 columns

In [90]:
```
intial_bakery_df[(intial_bakery_df["Chocolate Cake"] == 1) & (intial_bakery_df["Chocolate Coffee"] == 0)]["Transaction Number"].count()
```

Out[90]: 2962

In [8]:
```
bake_f = apriori(intial_bakery_df.drop(columns="Transaction Number"), min_support = 0.04, use_colnames = True)
bake_f.sort_values('support', ascending = False)
```

Out[8]:

| | support | itemsets |
|---|---|---|
| **7** | 0.109240 | (Coffee Eclair) |
| **45** | 0.102667 | (Hot Coffee) |
| **28** | 0.100747 | (Tuile Cookie) |
| **18** | 0.093160 | (Cherry Tart) |
| **4** | 0.092640 | (Strawberry Cake) |
| **35** | 0.092573 | (Apricot Danish) |
| **42** | 0.091613 | (Orange Juice) |
| **22** | 0.090440 | (Gongolais Cookie) |
| **27** | 0.089773 | (Marzipan Cookie) |
| **14** | 0.084827 | (Berry Tart) |
| **32** | 0.083987 | (Apricot Croissant) |
| **1** | 0.083613 | (Lemon Cake) |
| **0** | 0.083533 | (Chocolate Cake) |
| **46** | 0.083147 | (Chocolate Coffee) |
| **16** | 0.082947 | (Blueberry Tart) |
| **9** | 0.082747 | (Napoleon Cake) |
| **5** | 0.082240 | (Truffle Cake) |
| **33** | 0.082213 | (Cheese Croissant) |

In [9]:
```
ccake_support = bake_f[bake_f["itemsets"] == {'Cho
colate Cake'}]["support"].values[0]
print(ccake_support)
```

```
0.08353333333333333
```

In [10]:
```
ccoffee_support = bake_f[bake_f["itemsets"] == {'C
hocolate Coffee'}]["support"].values[0]
print(ccoffee_support)
```

```
0.08314666666666666
```

In [11]:
```
pair_support = bake_f[bake_f["itemsets"] == {'Choc
olate Coffee','Chocolate Cake'}]["support"].values
[0]
print(pair_support)
```

```
0.04404
```

In [12]:
```
all_n = intial_bakery_df["Transaction Number"].cou
nt()
all_n
```

Out[12]:
```
75000
```

In [20]:
```
fp1 = np.ceil(all_n*ccoffee_support)
fp1
```

Out[20]:
```
6236.0
```

In [21]:
```
f1p = np.ceil(all_n*ccake_support)
f1p
```

Out[21]: 6265.0

In [22]:
```
f11 = np.ceil(all_n*pair_support)
f11
```

Out[22]: 3303.0

In [23]:
```
f01 = fp1 - f11
f01
```

Out[23]: 2933.0

In [24]:
```
f10 = f1p - f11
f10
```

Out[24]: 2962.0

In [25]:
```
f0p = all_n - f1p
f0p
```

Out[25]: 68735.0

In [26]:
```
fp0 = all_n - fp1
fp0
```

Out[26]: 68764.0

In [27]:
```
f00 = fp0 - f10
f00
```

Out[27]: 65802.0

In [33]:
```
f0p - f01 == f00
```

Out[33]: True

In [34]:
```
f00 + f01 == f0p
```

Out[34]: True

In [35]:
```
f00 + f10 == fp0
```

Out[35]: True

In [36]:
```
f11 + f10 == f1p
```

Out[36]: True

In [37]:
```
f11 + f01 == fp1
```

Out[37]: True

In [82]:
```
print("\t","Coco","\t\t","!Coco","\t")
print()
print("Cake", "\t",f11,"\t",f10,"\t",f1p)
print()
print("!Cake","\t",f01,"\t",f00,"\t",f0p)
print()
print("\t", fp1, "\t", fp0, "\t", all_n)
```

```
         Coco            !Coco

Cake     3303.0          2962.0          6265.0

!Cake    2933.0          65802.0         68735.0

         6236.0          68764.0         75000
```

In [86]:
```
corr = (f11*f00 - f10*f01)/np.sqrt(fp1*f1p*fp0*f0p
)
corr
```

Out[86]: 0.4855664925278768