Name: Rushikesh Jyoti

Division: A

Roll no: 27

SRN: 201901139

In [ ]:

```
print("Hello")
```

```
Hello
```

In [ ]:

```
import pandas as pd
import seaborn as sb
from matplotlib import pyplot as plot

from sklearn.preprocessing import LabelEncoder
# from apyori import apriori
from efficient_apriori import apriori
```

In [ ]:

```
df = pd.read_csv("./retail_dataset.csv")
encoder = LabelEncoder()

df.head()
```

Out[ ]:

	0	1	2	3	4	5	6
0	Bread	Wine	Eggs	Meat	Cheese	Pencil	Diaper
1	Bread	Cheese	Meat	Diaper	Wine	Milk	Pencil
2	Cheese	Meat	Eggs	Milk	Wine	NaN	NaN
3	Cheese	Meat	Eggs	Milk	Wine	NaN	NaN
4	Meat	Pencil	Wine	NaN	NaN	NaN	NaN

In [ ]:

```
for col in df.columns:
    # df[col] = encoder.fit_transform(df[col])
    pass

df.head()
```

Out[ ]:

	0	1	2	3	4	5	6
0	Bread	Wine	Eggs	Meat	Cheese	Pencil	Diaper
1	Bread	Cheese	Meat	Diaper	Wine	Milk	Pencil
2	Cheese	Meat	Eggs	Milk	Wine	NaN	NaN
3	Cheese	Meat	Eggs	Milk	Wine	NaN	NaN
4	Meat	Pencil	Wine	NaN	NaN	NaN	NaN

In [ ]:

```
txns2 = df.stack().groupby(level=0).apply(list).tolist()
```

Support 1% and Confidence 30%

In [ ]:

```
itemsets, rules = apriori(txns2, min_support=0.01, min_confidence=0.3, verbosity=1)
```

```
Generating itemsets.
 Counting itemsets of length 1.
  Found 9 candidate itemsets of length 1.
  Found 9 large itemsets of length 1.
 Counting itemsets of length 2.
  Found 36 candidate itemsets of length 2.
  Found 36 large itemsets of length 2.
 Counting itemsets of length 3.
  Found 84 candidate itemsets of length 3.
  Found 84 large itemsets of length 3.
 Counting itemsets of length 4.
  Found 126 candidate itemsets of length 4.
  Found 126 large itemsets of length 4.
 Counting itemsets of length 5.
  Found 126 candidate itemsets of length 5.
  Found 120 large itemsets of length 5.
 Counting itemsets of length 6.
  Found 69 candidate itemsets of length 6.
  Found 45 large itemsets of length 6.
 Counting itemsets of length 7.
  Found 4 candidate itemsets of length 7.
  Found 2 large itemsets of length 7.
 Counting itemsets of length 8.
  Found 0 candidate itemsets of length 8.
Itemset generation terminated.

Generating rules from itemsets.
 Generating rules of size 2.
 Generating rules of size 3.
 Generating rules of size 4.
 Generating rules of size 5.
 Generating rules of size 6.
 Generating rules of size 7.
Rule generation terminated.
```

In [ ]:

```
for item in sorted(rules, key=lambda item: (item.lift), reverse=True)[:5]:
#  print(f"{item.lhs} -> {item.rhs}")
    print(item)

print(f"There are {len(rules)} rules")
```

```
{Diaper, Eggs, Meat, Pencil} -> {Bagel, Cheese, Wine} (conf: 0.364, supp: 0.013, lift: 3.369,
conv: 1.402)
{Cheese, Diaper, Eggs, Meat, Pencil} -> {Bagel, Wine} (conf: 0.571, supp: 0.013, lift: 3.333,
conv: 1.933)
{Meat, Milk, Pencil} -> {Bread, Cheese, Wine} (conf: 0.400, supp: 0.032, lift: 2.800, conv:
1.429)
{Cheese, Meat, Milk, Pencil} -> {Bread, Eggs, Wine} (conf: 0.333, supp: 0.016, lift: 2.763, c
onv: 1.319)
{Bagel, Cheese, Meat, Pencil, Wine} -> {Diaper, Eggs} (conf: 0.444, supp: 0.013, lift: 2.745,
conv: 1.509)
There are 2681 rules
```

Support 2% and Confidence 40%

In [ ]:

```
itemsets2, rules2 = apriori(txns2, min_support=0.02, min_confidence=0.4, verbosity=1)
```

```
Generating itemsets.
 Counting itemsets of length 1.
  Found 9 candidate itemsets of length 1.
  Found 9 large itemsets of length 1.
 Counting itemsets of length 2.
  Found 36 candidate itemsets of length 2.
  Found 36 large itemsets of length 2.
 Counting itemsets of length 3.
  Found 84 candidate itemsets of length 3.
  Found 84 large itemsets of length 3.
 Counting itemsets of length 4.
  Found 126 candidate itemsets of length 4.
  Found 126 large itemsets of length 4.
 Counting itemsets of length 5.
  Found 126 candidate itemsets of length 5.
  Found 95 large itemsets of length 5.
 Counting itemsets of length 6.
  Found 27 candidate itemsets of length 6.
  Found 10 large itemsets of length 6.
 Counting itemsets of length 7.
  Found 0 candidate itemsets of length 7.
Itemset generation terminated.

Generating rules from itemsets.
 Generating rules of size 2.
 Generating rules of size 3.
 Generating rules of size 4.
 Generating rules of size 5.
 Generating rules of size 6.
Rule generation terminated.
```

In [ ]:

```
for item in sorted(rules2, key=lambda item: (item.lift), reverse=True)[:5]:
#  print(f"{item.lhs} -> {item.rhs}")
    print(item)

print(f"There are {len(rules2)} rules")
```

```
{Meat, Milk, Pencil} -> {Bread, Cheese, Wine} (conf: 0.400, supp: 0.032, lift: 2.800, conv:
1.429)
{Cheese, Meat, Milk, Pencil} -> {Bread, Wine} (conf: 0.667, supp: 0.032, lift: 2.727, conv:
2.267)
{Meat, Milk, Pencil, Wine} -> {Bread, Cheese} (conf: 0.625, supp: 0.032, lift: 2.625, conv:
2.032)
{Milk, Pencil, Wine} -> {Bread, Eggs} (conf: 0.467, supp: 0.044, lift: 2.492, conv: 1.524)
{Cheese, Meat, Milk, Pencil} -> {Bread, Eggs} (conf: 0.467, supp: 0.022, lift: 2.492, conv:
1.524)
There are 1361 rules
```

Support 3% and Confidence 50%

In [ ]:

```
itemsets3, rules3 = apriori(txns2, min_support=0.03, min_confidence=0.5, verbosity=1)
```

```
Generating itemsets.
 Counting itemsets of length 1.
  Found 9 candidate itemsets of length 1.
  Found 9 large itemsets of length 1.
 Counting itemsets of length 2.
  Found 36 candidate itemsets of length 2.
  Found 36 large itemsets of length 2.
 Counting itemsets of length 3.
  Found 84 candidate itemsets of length 3.
  Found 84 large itemsets of length 3.
 Counting itemsets of length 4.
  Found 126 candidate itemsets of length 4.
  Found 123 large itemsets of length 4.
 Counting itemsets of length 5.
  Found 113 candidate itemsets of length 5.
  Found 48 large itemsets of length 5.
 Counting itemsets of length 6.
  Found 7 candidate itemsets of length 6.
  Found 1 large itemsets of length 6.
 Counting itemsets of length 7.
  Found 0 candidate itemsets of length 7.
Itemset generation terminated.

Generating rules from itemsets.
 Generating rules of size 2.
 Generating rules of size 3.
 Generating rules of size 4.
 Generating rules of size 5.
 Generating rules of size 6.
Rule generation terminated.
```

In [ ]:

```python
for item in sorted(rules3, key=lambda item: (item.lift), reverse=True)[:5]:
 print(item)
len(rules3)
```

```
{Cheese, Meat, Milk, Pencil} -> {Bread, Wine} (conf: 0.667, supp: 0.032, lift: 2.727, conv:
2.267)
{Meat, Milk, Pencil, Wine} -> {Bread, Cheese} (conf: 0.625, supp: 0.032, lift: 2.625, conv:
2.032)
{Cheese, Milk, Pencil, Wine} -> {Bread, Meat} (conf: 0.500, supp: 0.032, lift: 2.423, conv:
1.587)
{Diaper, Meat, Milk} -> {Bread, Cheese} (conf: 0.550, supp: 0.035, lift: 2.310, conv: 1.693)
{Meat, Milk, Pencil} -> {Bread, Wine} (conf: 0.560, supp: 0.044, lift: 2.291, conv: 1.717)
```

Out[ ]:

```
693
```

Collect the lift, confidence, lhs and rhs to draw plots

In [ ]:

```python
parameters = {
    'Rules': [],
    'Item1': [],
    'Item2': [],
    'Confidence': [],
    'Support': [],
    'Lift': []
}

data = pd.DataFrame(parameters)
temp_rules = sorted(rules, key=lambda item: (item.lift, item.conviction), reverse=True)

for item in temp_rules:
    data.loc[len(data.index)] = [(item.lhs+item.rhs), item.lhs,
                                 item.rhs, item.confidence, item.support, item.lift]
```

```
C:\Users\Rushikesh\AppData\Local\Programs\Python\Python39\lib\site-packages\numpy\core\_asarr
ay.py:102: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which
is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprec
ated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray.
  return array(a, dtype, copy=False, order=order)
```

In [ ]:

```
data.head()
```

Out[ ]:

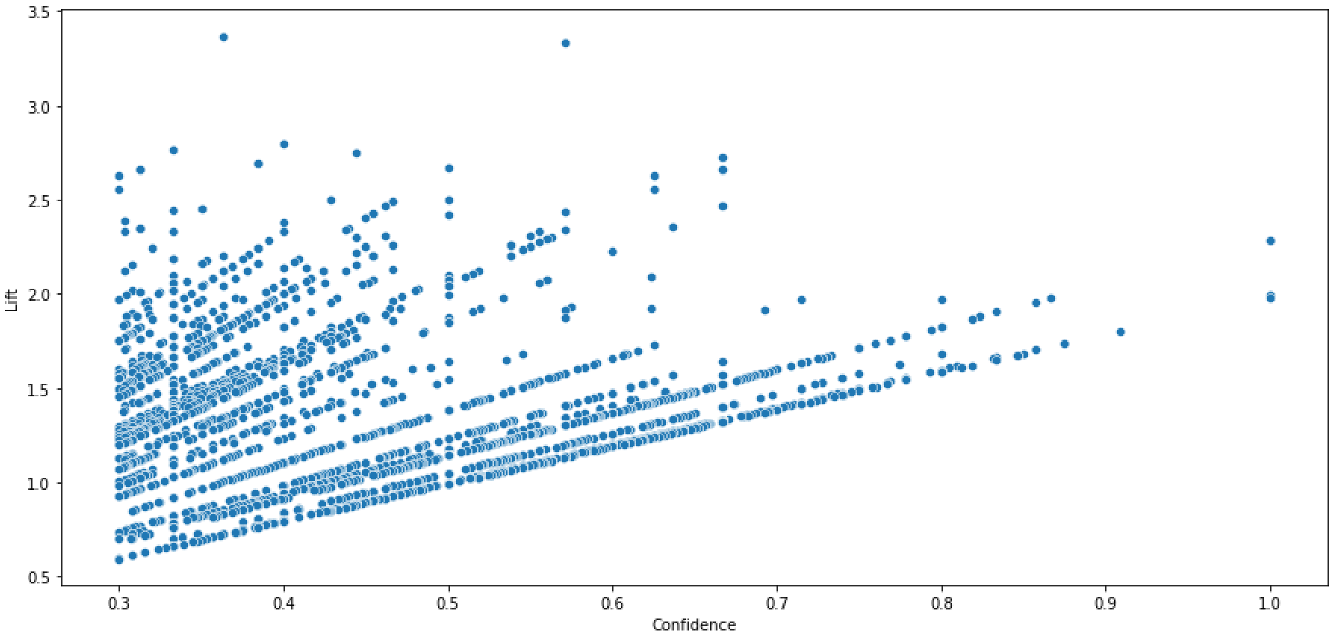
	Rules	Item1	Item2	Confidence	Support	Lift
0	(Diaper, Eggs, Meat, Pencil, Bagel, Cheese, Wine)	(Diaper, Eggs, Meat, Pencil)	(Bagel, Cheese, Wine)	0.363636	0.012698	3.368984
1	(Cheese, Diaper, Eggs, Meat, Pencil, Bagel, Wine)	(Cheese, Diaper, Eggs, Meat, Pencil)	(Bagel, Wine)	0.571429	0.012698	3.333333
2	(Meat, Milk, Pencil, Bread, Cheese, Wine)	(Meat, Milk, Pencil)	(Bread, Cheese, Wine)	0.400000	0.031746	2.800000
3	(Cheese, Meat, Milk, Pencil, Bread, Eggs, Wine)	(Cheese, Meat, Milk, Pencil)	(Bread, Eggs, Wine)	0.333333	0.015873	2.763158
4	(Bagel, Cheese, Meat, Pencil, Wine, Diaper, Eggs)	(Bagel, Cheese, Meat, Pencil, Wine)	(Diaper, Eggs)	0.444444	0.012698	2.745098

In [ ]:

```
plot.figure(1, (15, 7))
sb.scatterplot(data=data, x='Confidence', y='Lift')
```

Out[ ]:

```
<AxesSubplot:xlabel='Confidence', ylabel='Lift'>
```


Lift
Confidence