Association Rule Learning

Association Rule Learning (ARL) is a fascinating technique in the realm of **unsupervised machine learning**. Let's delve into its intricacies:

1. **What Is Association Rule Learning?**
    - ARL is a rule-based method that uncovers interesting relationships or associations between variables within large datasets. It aims to identify strong rules using measures of interestingness .
    - Unlike supervised learning, where we predict outcomes based on labeled data, ARL focuses on discovering dependencies among data items without predefined target labels.

Association rule mining finds interesting associations and relationships among large sets of data items. This rule shows how frequently a itemset occurs in a transaction. A typical example is a Market Based Analysis. Market Based Analysis is one of the key techniques used by large relations to show associations between items.It allows retailers to identify relationships between the items that people buy together frequently. Given a set of transactions, we can find rules that will predict the occurrence of an item based on the occurrences of other items in the transaction.

Association rule learning is a type of unsupervised learning technique that checks for the dependency of one data item on another data item and maps accordingly so that it can be more profitable. It tries to find some interesting relations or associations among the variables of dataset. It is based on different rules to discover the interesting relations between variables in the database.

The association rule learning is one of the very important concepts of machine learning, and it is employed in **Market Basket analysis, Web usage mining, continuous production, etc.** Here market basket analysis is a technique used by the various big retailer to discover the associations between items. We can understand it by taking an example of a supermarket, as in a supermarket, all products that are purchased together are put together.

For example, if a customer buys bread, he most likely can also buy butter, eggs, or milk, so these products are stored within a shelf or mostly nearby. Consider the below diagram:

Association rule learning can be divided into three types of algorithms:

1. **Apriori**
2. **Eclat**
3. **F-P Growth Algorithm**

How does Association Rule Learning work?

Association rule learning works on the concept of If and Else Statement, such as if A then B.

Here the If element is called **antecedent**, and then statement is called as **Consequent**. These types of relationships where we can find out some association or relation between two items is known *as single cardinality*. It is all about creating rules, and if the number of items increases, then cardinality also increases accordingly. So, to measure the associations between thousands of data items, there are several metrics. These metrics are given below:

- **Support**
- **Confidence**
- **Lift**

Support

Support is the frequency of A or how frequently an item appears in the dataset. It is defined as the fraction of the transaction T that contains the itemset X. If there are X datasets, then for transactions T, it can be written as:

$$\text{Supp(X)}=\frac{Freq(X)}{T}$$

Confidence

Confidence indicates how often the rule has been found to be true. Or how often the items X and Y occur together in the dataset when the occurrence of

X is already given. It is the ratio of the transaction that contains X and Y to the number of records that contain X.

$$\text{Confidence} = \frac{Freq(X,Y)}{Freq(X)}$$

Lift

It is the strength of any rule, which can be defined as below formula:

$$\text{Lift} = \frac{Supp(X,Y)}{Supp(X) \times Supp(Y)}$$

It is the ratio of the observed support measure and expected support if X and Y are independent of each other. It has three possible values:

- If **Lift= 1**: The probability of occurrence of antecedent and consequent is independent of each other.
- **Lift>1**: It determines the degree to which the two itemsets are dependent to each other.
- **Lift<1**: It tells us that one item is a substitute for other items, which means one item has a negative effect on another.

Example:

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Before we start defining the rule, let us first see the basic definitions.

Support Count(σ) – Frequency of occurrence of a itemset.

```
Hereσ
({Milk, Bread, Diaper})=2
```

Frequent Itemset – An itemset whose support is greater than or equal to minsup threshold. **Association Rule –** An implication expression of the form X -> Y, where X and Y are any 2 itemsets.

```
Example: {Milk, Diaper}->{Beer}
```

Rule Evaluation Metrics –

- **Support(s) –** The number of transactions that include items in the {X} and {Y} parts of the rule as a percentage of the total number of transaction.It is a measure of how frequently the collection of items occur together as a percentage of all transactions.

- **Support =σ(X+Y)÷Total [** It is interpreted as fraction of transactions that contain both X and Y.]

S=σ({Milk, Diaper, Beer}) ÷ **|T|**

= 2/5

= 0.4

Conf(X=>Y) = Supp(X U Y) ÷ Supp(X) **[**It measures how often each item in Y appears in transactions that contains items in X also.]

C= σ (Milk, Diaper, Beer)÷ σ (Milk, Diaper)

= 2/3

= 0.67

- **Lift(l) –** The lift of the rule X=>Y is the confidence of the rule divided by the expected confidence, assuming that the itemsets X and Y are independent of each other.The expected confidence is the confidence divided by the frequency of {Y}.

Lift(X=>Y) = Conf(X=>Y) ÷ Supp(Y) **[**Lift value near 1 indicates X and Y almost often appear together as expected, greater than 1 means they appear together more than expected and less than 1 means they appear less than expected. Greater lift values indicate stronger association.]

l= Supp({Milk, Diaper, Beer}) ÷ Supp({Milk, Diaper})*Supp({Beer})

= 0.4 / (0.6 x 0.6)

= 1.11

The Association rule is very useful in analyzing datasets. The data is collected using bar-code scanners in supermarkets. Such databases consist of a large number of transaction records which list all items bought by a customer on a single purchase. So the manager could know if certain groups of items are consistently purchased together and use this data for adjusting store layouts, cross-selling, promotions based on statistics.

Apriori algorithm

Apriori is an algorithm for frequent item set mining and association rule learning over relational databases. It proceeds by identifying the frequent individual items in the database and extending them to larger and larger item sets as long as those item sets appear sufficiently often in the database. The frequent item sets determined by Apriori can be used to determine association rules which highlight general trends in the database: this has applications in domains such as market basket analysis.

The Apriori algorithm was proposed by Agrawal and Srikant in 1994. Apriori is designed to operate on databases containing transactions (for example, collections of items bought by customers, or details of a website frequentation or IP addresses). Apriori uses a "bottom up" approach, where frequent subsets are extended one item at a time (a step known as *candidate generation*), and groups of candidates are tested against the data. The algorithm terminates when no further successful extensions are found.

Apriori uses breadth-first search and a Hash tree structure to count candidate item sets efficiently. It generates candidate item sets of length k from item sets of length k-1. Then it prunes the candidates which have an infrequent sub pattern. According to the downward closure lemma, the candidate set contains all frequent k-length item sets. After that, it scans the transaction database to determine frequent item sets among the candidates.

The pseudo code for the algorithm is given below for a transaction database T, and a support threshold of ε. Usual set theoretic notation is employed, though note that T is a multiset. C_k is the candidate set for level k. At each step, the algorithm is assumed to generate the candidate sets from the large item sets of the preceding level, heeding the downward closure lemma. $\mathrm{count}[c]$ accesses a field of the data structure that represents candidate set c, which is initially assumed to be zero. Many details are omitted below, usually the most important part of the implementation is the data structure used for storing the candidate sets, and counting their frequencies.

```
Apriori(T, ε)
    L_1 ← {large 1 - itemsets}
    k ← 2
    while L_k-1 is not empty
        C_k ← Apriori_gen(L_k-1, k)
        for transactions t in T
            D_t ← {c in C_k : c ⊆ t}
            for candidates c in D_t
                count[c] ← count[c] + 1

        L_k ← {c in C_k : count[c] ≥ ε}
        k ← k + 1

    return Union(L_k)

Apriori_gen(L, k)
    result ← list()
    for all p ∈ L, q ∈ L where p_1 = q_1, p_2 = q_2, ..., p_k-2 = q_k-2 and p_k-1 < q_k-1
        c = p ∪ {q_k-1}
        if u ∈ L for all u ⊆ c where |u| = k-1
            result.add(c)
    return result
```

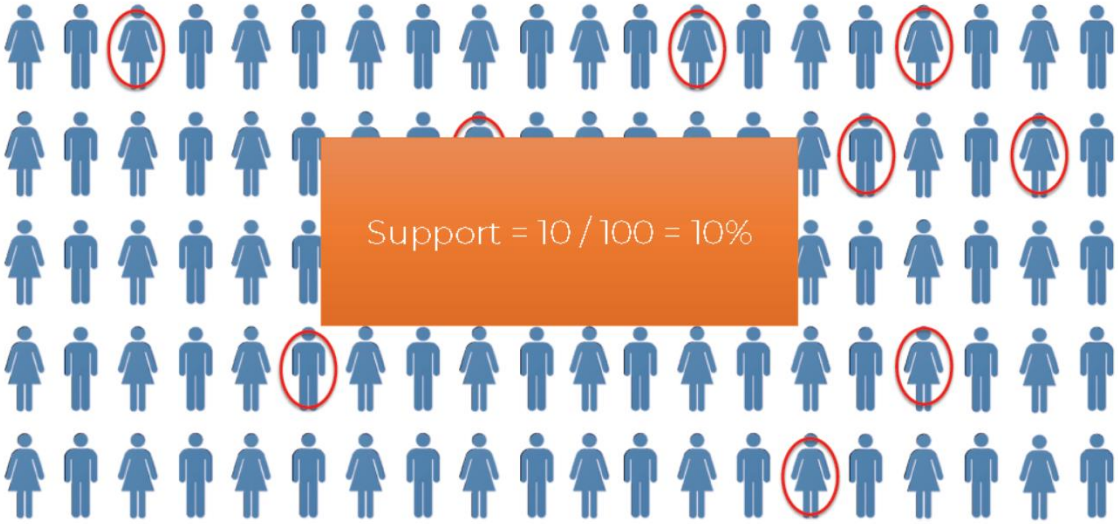
Apriori Algorithm people who bought something to bought something else or who did something did something else.

Apriori - Support

Movie Recommendation: $$\text{support}(M) = \frac{\#\text{ user watchlists containing } M}{\#\text{ user watchlists}}$$

Market Basket Optimisation: $$\text{support}(I) = \frac{\#\text{ transactions containing } I}{\#\text{ transactions}}$$

Apriori - Support



Number of peoples(here 100) watched movies. One of my favorite movies is Sholay. How many of the people have actually seen the movie Sholay. So, support is 10%.

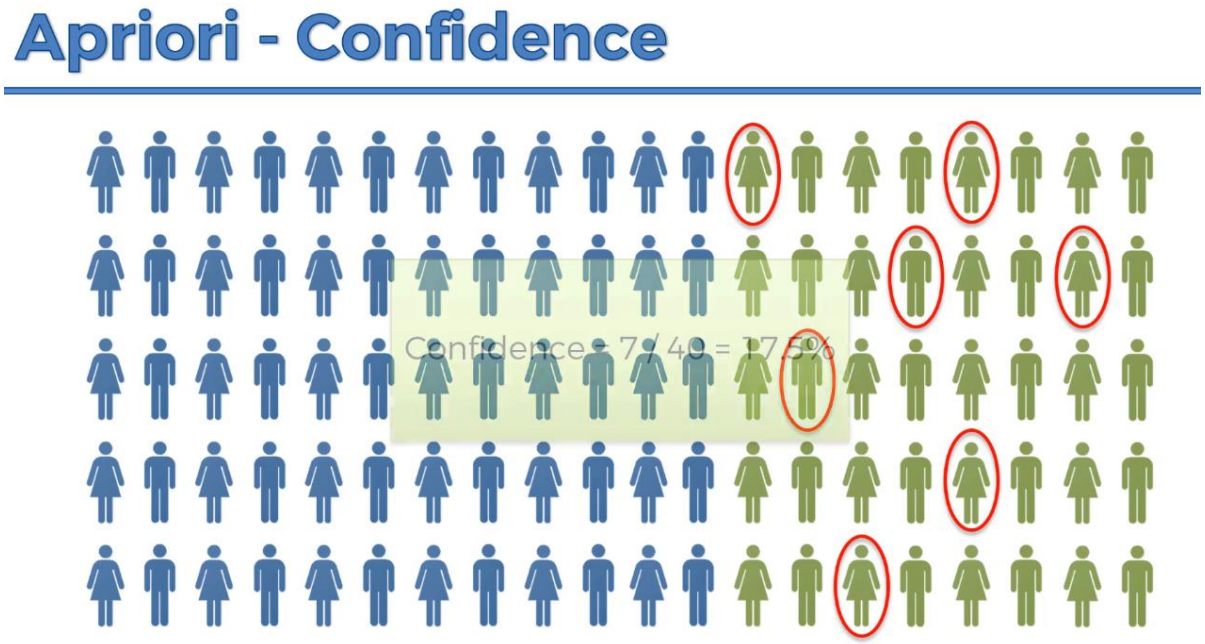
Apriori-Confidence:

Movie Recommendation: $\text{confidence}(M_1 \rightarrow M_2) = \frac{\#\text{ user watchlists containing } M_1 \text{ and } M_2}{\#\text{ user watchlists containing } M_1}$

Market Basket Optimisation: $\text{confidence}(I_1 \rightarrow I_2) = \frac{\#\text{ transactions containing } I_1 \text{ and } I_2}{\#\text{ transactions containing } I_1}$

What is the confidence? Confidence define number of people have seen M1 and M2 moves divided by number of people have seen M1 movie.

Here we are going to assume people have seen Sholay. Hypothesis people have seen Sholay they have also like Sholay. People have seen Sholay also like to see Mohabbatein. Lets say 40 people have seen Sholay. Now we want to how many people have seen Mohabbatein those who have seen Sholay. i.e. seven people(7).



Apriori - Confidence

So, confidence here is 17.5%

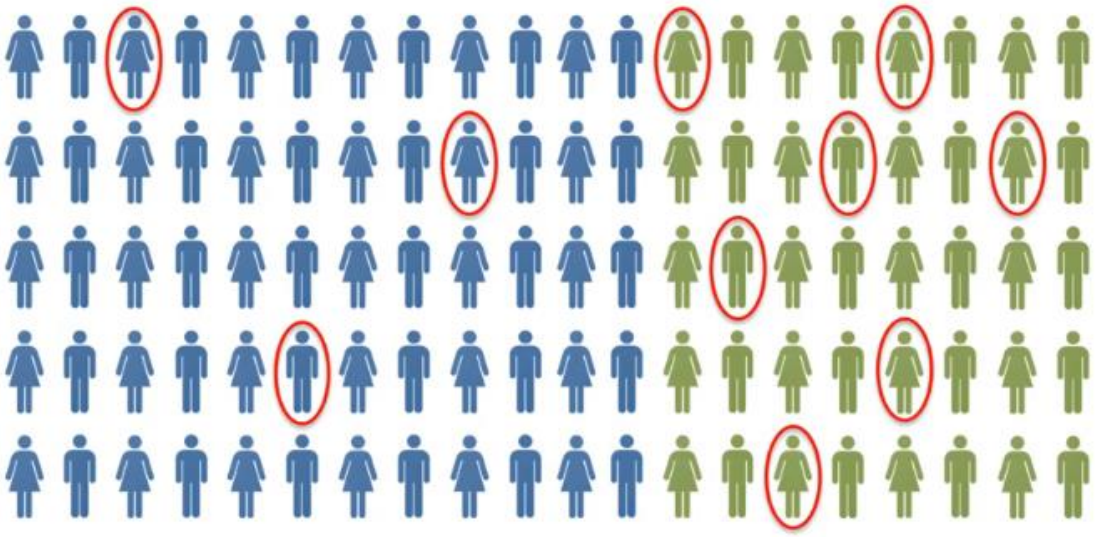
Apriori - Lift

Movie Recommendation: $$\text{lift}(M_1 \rightarrow M_2) = \frac{\text{confidence}(M_1 \rightarrow M_2)}{\text{support}(M_2)}$$

Market Basket Optimisation: $$\text{lift}(I_1 \rightarrow I_2) = \frac{\text{confidence}(I_1 \rightarrow I_2)}{\text{support}(I_2)}$$

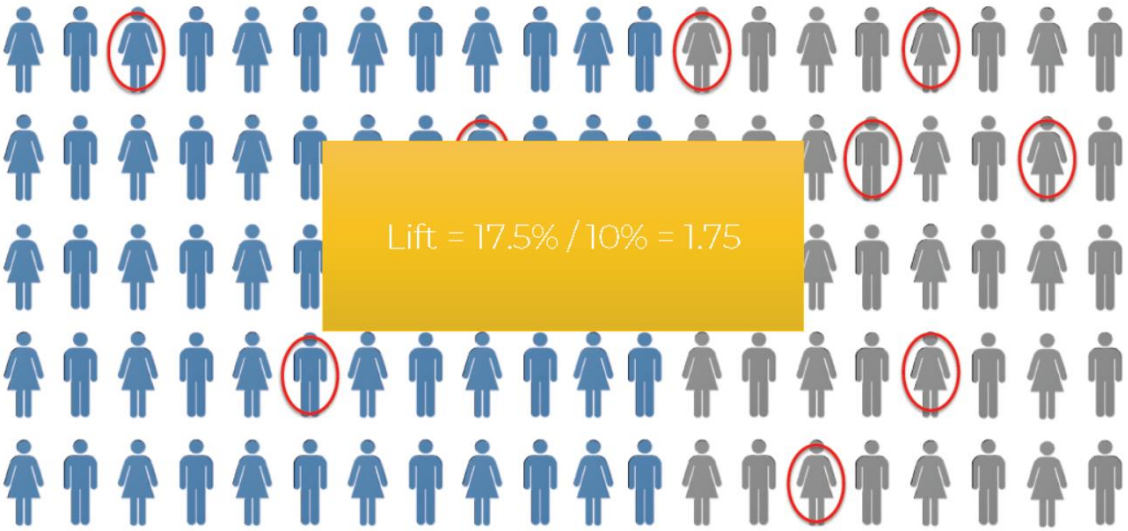
Lift is basically Naïve Bayes classifier. Lift is basically confidence by support. So, step-2 divided by step-1. Here our population :

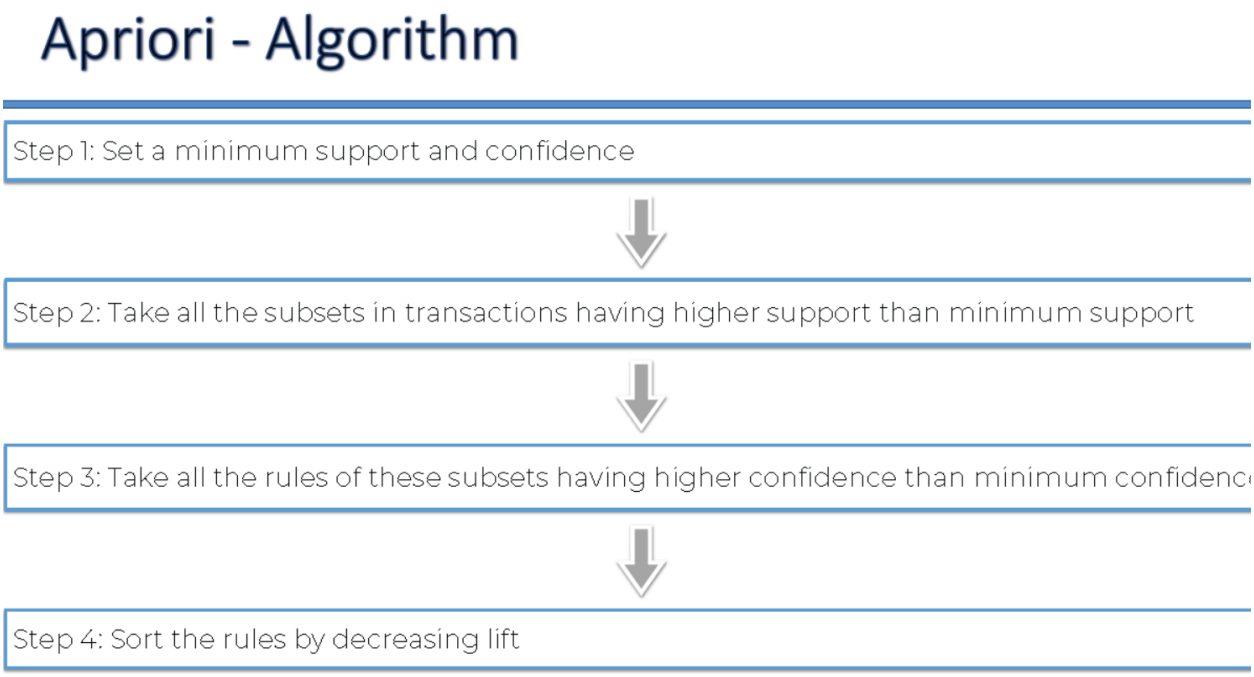
Apriori - Lift



If we take random population likelihood from fresh population. What is the likelihood to like the movie? Likelihood is 10%, Out of 100 people only 10 people like that movie. Question is can we improve the result by use some prior knowledge as algorithm Apriori.

Apriori- Lift





We need to setup minimum support and confidence. Apriori quite slow algorithm because goes through all of the different combination. So, lots of lots of combinations pair…5..6..7 item combinations. We need to set some kind limitation so set minimum support. We might not consider support not less than 20%. Don't want to waste time building a model something that only has success rate 20%. Same way confident might not consider less than 12% because not a strong enough factor or rule. We can sort the rules by decreasing lift which we can find the highest lift and also get the top 5 or 10 lift. Those things consider for improving the business decision. That's how Apriori algorithm work.

Some good fun here

Amazon, Netflix some good example for using Apriori but of course there much more sophisticated algorithms not just Apriori there much more specific design combination of algorithms. Apriori just a kind of basic approach, straight forward algorithm solving the problem.

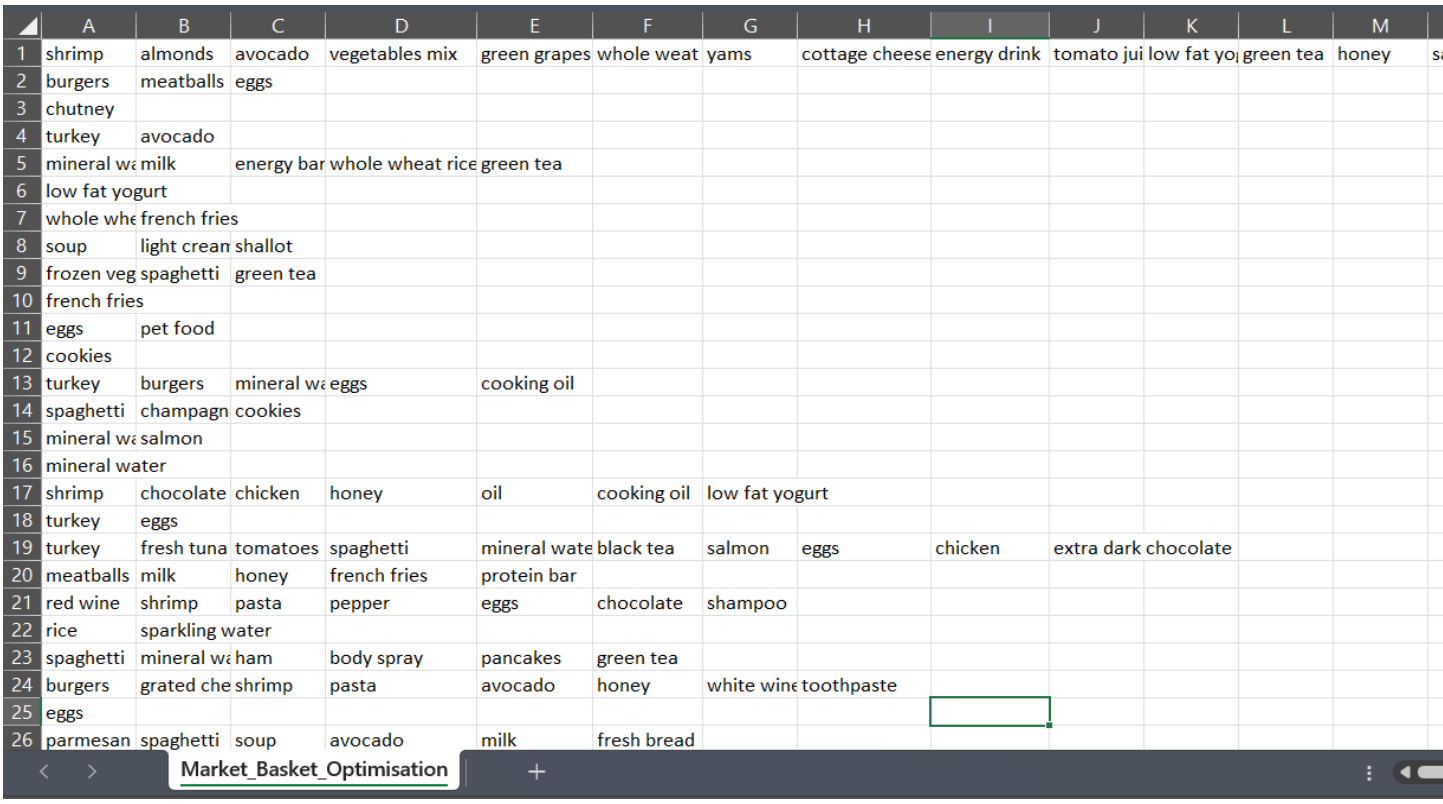
Identify best Association rule among the different product bought by the customer. Very famous deal to buy this and get that product if you buy this then get that product free. Use association rule to find strong relation among the different products. Association is used to market basket analyze and optimization.

Let’s describe what the dataset about:

Imagine you are the business owner of the shop and you would like to optimize and boost the sale after some new great deal. You have to identify best association rules among the different products bought by the customer. Offer to customer buy this and get that product free. Infect this product they are very likely to get other product. Therefore, what did owner did? he must know the data science or hire data scientist. So that learn the association rule and owner offer to customer best deal.

Come to dataset each row of the dataset corresponds to the different transaction mean different customer. Each of the transaction different product, customer did the transaction to purchase. Here 7500 row means where collected one week customer purchased. Owner recoded all the transaction give them to you, data scientist. So, you learn the association rule. Mission, you return to owner best possible association rule of two elements. So, that owner can find the best deal to his client.

Dataset:



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	shrimp	almonds	avocado	vegetables mix	green grapes	whole weat	yams	cottage cheese	energy drink	tomato jui	low fat yo	green tea	honey
2	burgers	meatballs	eggs										
3	chutney												
4	turkey	avocado											
5	mineral wa	milk	energy bar	whole wheat rice	green tea								
6	low fat yogurt												
7	whole whe	french fries											
8	soup	light crean	shallot										
9	frozen veg	spaghetti	green tea										
10	french fries												
11	eggs	pet food											
12	cookies												
13	turkey	burgers	mineral wa	eggs	cooking oil								
14	spaghetti	champagn	cookies										
15	mineral wa	salmon											
16	mineral water												
17	shrimp	chocolate	chicken	honey	oil	cooking oil	low fat yogurt						
18	turkey	eggs											
19	turkey	fresh tuna	tomatoes	spaghetti	mineral wate	black tea	salmon	eggs	chicken	extra dark chocolate			
20	meatballs	milk	honey	french fries	protein bar								
21	red wine	shrimp	pasta	pepper	eggs	chocolate	shampoo						
22	rice	sparkling water											
23	spaghetti	mineral wa	ham	body spray	pancakes	green tea							
24	burgers	grated che	shrimp	pasta	avocado	honey	white wine	toothpaste					
25	eggs												
26	parmesan	spaghetti	soup	avocado	milk	fresh bread							

Market_Basket_Optimisation

Apriori

```
!pip install apyori

O/P:
Collecting apyori
  Downloading apyori-1.1.2.tar.gz (8.6 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: apyori
  Building wheel for apyori (setup.py) ... done
  Created wheel for apyori: filename=apyori-1.1.2-py3-none-any.whl
size=5955
sha256=78294c6efdf6ca2c7661015c082260046856d7d6e8aa22d8ea973099e5f8bb41
  Stored in directory:
/root/.cache/pip/wheels/c4/1a/79/20f55c470a50bb3702a8cb7c94d8ada15573538c7
f4baebe2d
Successfully built apyori
Installing collected packages: apyori
Successfully installed apyori-1.1.2
```

Importing the Libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

** Importing libraries same however first time we won't use Scikit Learn. sciKit Learn model doesn't include Apriori Module or classes. So, we not include scikit learn to train the model. Actually use another library Apriori. Apriori.py is the python implement containg all Apriori algorithm that what we will get and use trained our whole dataset but exceptionally, Goole colab contains most of the library pre-installed but doesn't include ariori module. You have to installed it. pip command download first and then it is installed it in particular notebook. In our case it is apyori and version is `apyori-1.1.2.`

Data Preprocessing

```
dataset = pd.read_csv('Market_Basket_Optimisation.csv', header = None)
transactions = []
for i in range(0, 7501):
  transactions.append([str(dataset.values[i,j]) for j in range(0, 20)])
```

** In Dataset total 19 column that's why range (0,20). We can't directly access cell of dataset that's why we are using value as dataset.values and training the Apriori model, all elements must be string otherwise Apriori model won't be learn the row. Makesur products are string so this value must be inside string function i.e. str(dataset.values[i,j]). Here name of the dataset is Market_Basket_Optimisation.csv; don't try to open the dataset here because dataset is very large size. We can specify indeed that there are no header mean no column name that why header = None; Now first row dataset take as first transaction.

Training the Apriori model on the dataset

```
from apyori import apriori
rules = apriori(transactions = transactions, min_support = 0.003,
min_confidence = 0.2, min_lift = 3, min_length = 2, max_length = 2)
```

** Now we need indeed to upload Apriori function, belongs to apyori package: from apyori import apriori;
Apriori function takes some very intuitive arguments: 1st one dataset name transactions(transactions list which we crated right format, next argument

```
is minimum support, here 7501 transaction among this most relevant rule
like support if a product 3 times per day in transaction then total
transaction in week is 21. Actually support is number of product appear in
the transaction divided by total number of transactions. Here consider
minimum support 3 time 7 divided by 7501(total transaction over the week.
So 3*7/7501=0.00279=0.003(round up).
Next argument is minimum confidence. Here we choose 0.2 for min_confidence
(no rule thump we choose as per business requirement.
Next parameter is minimum lift. Generally good lift is at least 3. Life
below 3 make the rule that is no relevant.
Two last arguments infect compulsory for business problem. Fact is buy a
product A and get the product B free. One product is left hand side rule
and other product is right hand rule. We need two more arguments here
min_elnght and then max_length. Min_lenght is minimum number of elements
in your left to right and max_length is maximum number elements in your
left to right. Set both are 2. Image buy two products and get one product
free the both would be 3. Just as per your business case set the minimum
length.
```

Visualising the results

Displaying the first results coming directly from the output of the apriori function

```
results = list(rules)
results


** results=list(rules)→Result variable as list of rules which will just
put this rules into a list. results→ display the all the rules.
```

O/P:

```
[RelationRecord(items=frozenset({'light cream', 'chicken'}), support=0.004532728969470737, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'light cream'}), items_add=frozenset({'chicken'}), confidence=0.29059829059829057,
lift=4.84395061728395)]),
 RelationRecord(items=frozenset({'escalope', 'mushroom cream sauce'}), support=0.005732568990801226, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'mushroom cream sauce'}), items_add=frozenset({'escalope'}), confidence=0.3006993006993007,
lift=3.790832696715049)]),
 RelationRecord(items=frozenset({'escalope', 'pasta'}), support=0.005865884548726837, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'pasta'}), items_add=frozenset({'escalope'}), confidence=0.3728813559322034, lift=4.700811850163794)]),
 RelationRecord(items=frozenset({'honey', 'fromage blanc'}), support=0.003332888948140248, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'fromage blanc'}), items_add=frozenset({'honey'}), confidence=0.2450980392156863,
lift=5.164270764485569)]),
 RelationRecord(items=frozenset({'ground beef', 'herb & pepper'}), support=0.015997866951073192, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'herb & pepper'}), items_add=frozenset({'ground beef'}), confidence=0.3234501347708895,
lift=3.2919938411349285)]),
 RelationRecord(items=frozenset({'ground beef', 'tomato sauce'}), support=0.005332622317024397, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'tomato sauce'}), items_add=frozenset({'ground beef'}), confidence=0.3773584905660377,
lift=3.840659481324083)]),
 RelationRecord(items=frozenset({'light cream', 'olive oil'}), support=0.003199573390214638, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'light cream'}), items_add=frozenset({'olive oil'}), confidence=0.20512820512820515,
lift=3.1147098515519573)]),
 RelationRecord(items=frozenset({'whole wheat pasta', 'olive oil'}), support=0.007998933475536596, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'whole wheat pasta'}), items_add=frozenset({'olive oil'}), confidence=0.2714932126696833,
lift=4.122410097642296)]),
 RelationRecord(items=frozenset({'pasta', 'shrimp'}), support=0.005065991201173177, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'pasta'}), items_add=frozenset({'shrimp'}), confidence=0.3220338983050847, lift=4.506672147735896)])]
```

**** starting with the 1st one: 1st row we see that two product in the rule light cream and chicken ('light cream', 'chicken'). It's not order base but it is items_base→ light cream and items_add→ chicken mean lefthand side light cream and right hand side chicken. So, it some one buy light cream then they will have high chance to buy chicken. High chance measure by the confidence that is 0.29. If customer buy light cream that will be 29% chance to buy chicken. Here support=0.0045 confidence= 0.29.59… and lift =4.8439…**

Putting the results well organised into a Pandas DataFrame

```
def inspect(results):
    lhs         = [tuple(result[2][0][0])[0] for result in results]
    rhs         = [tuple(result[2][0][1])[0] for result in results]
    supports    = [result[1] for result in results]
    confidences = [result[2][0][2] for result in results]
    lifts       = [result[2][0][3] for result in results]
    return list(zip(lhs, rhs, supports, confidences, lifts))
resultsinDataFrame = pd.DataFrame(inspect(results), columns = ['Left Hand
Side', 'Right Hand Side', 'Support', 'Confidence', 'Lift'])
```

****Inspect function which will return rules, mean this rule well organized data frame pandas and sort the rules descending order. We will be able to sort easily the rules by the lift. Each of the row take the separately lhs, rhs, supports, confidences and lifts.**
lhs→ left hand side row; rhs→ right hand side row; inspect function give the output of the column name beside as the column name Left Hand Side, Right Hand Side, Support, Confidece and Lift.

Explain:

lhs = [tuple(result[2][0][0])[0] for result in results]

for result in results: each of the row inside the list. Each of the rules is the full list of the rules and access each of the elements separately; start with left hand side basically display 'light cream'.

RelationRecord(items=frozenset({'light cream', 'chicken'}): this element index 0;
support=0.004532728969470737 : index 1;
ordered_statistics=[OrderedStatistic(items_base=frozenset({'light cream'}),
items_add=frozenset({'chicken'}), confidence=0.29059829059829057, lift=4.84395061728395)]) :
index 2;
`result[2][0]:` OrderedStatistic(items_base=frozenset({'light cream'}),
items_add=frozenset({'chicken'}), confidence=0.29059829059829057, lift=4.84395061728395) mean square bracket;
`result[2][0][0]:` items_base=frozenset({'light cream'} which is left hand side and that is light cream;
same way rhs: `result[2][0][1]=` items_add=frozenset({'chicken'} which is right hand side and that is chicken;
`supports = [result[1] for result in results] : full row and index 1 related to` support=0.004532728969470737;

`confidences = [result[2][0][2]: for result in results]=`
confidence=0.29059829059829057 1st row index 2 then index 0 then index 2 `for confidence;`
`same way lift;`

Displaying the results non sorted

```
resultsinDataFrame

**different element separate column;
```

O/P:

	Left Hand Side	Right Hand Side	Support	Confidence	Lift
0	light cream	chicken	0.004533	0.290598	4.843951
1	mushroom cream sauce	escalope	0.005733	0.300699	3.790833
2	pasta	escalope	0.005866	0.372881	4.700812
3	fromage blanc	honey	0.003333	0.245098	5.164271
4	herb & pepper	ground beef	0.015998	0.323450	3.291994
5	tomato sauce	ground beef	0.005333	0.377358	3.840659
6	light cream	olive oil	0.003200	0.205128	3.114710
7	whole wheat pasta	olive oil	0.007999	0.271493	4.122410
8	pasta	shrimp	0.005066	0.322034	4.506672

** 1st row if customer buying light cream then they will have actually 29% chance of buying chicken and this rule appear 0.4% transaction and lift 4.84 is also good. 2nd, 3rd rows…same rule..

Displaying the results sorted by descending lifts

```
resultsinDataFrame.nlargest(n = 10, columns = 'Lift')
```

```
**prebuild function pandas library for sort a specific column;
nlargest: is the specific method can take three argument like n= number of
row to return; columns='Lift' mean sort by the value of lift;
```

	Left Hand Side	Right Hand Side	Support	Confidence	Lift
3	fromage blanc	honey	0.003333	0.245098	5.164271
0	light cream	chicken	0.004533	0.290598	4.843951
2	pasta	escalope	0.005866	0.372881	4.700812
8	pasta	shrimp	0.005066	0.322034	4.506672
7	whole wheat pasta	olive oil	0.007999	0.271493	4.122410
5	tomato sauce	ground beef	0.005333	0.377358	3.840659
1	mushroom cream sauce	escalope	0.005733	0.300699	3.790833
4	herb & pepper	ground beef	0.015998	0.323450	3.291994
6	light cream	olive oil	0.003200	0.205128	3.114710

**here the highest lift 5.16 and lowest lift is 3.11 of top 10 lift. Formage blanc and honey is the highest lift strong rule. If I owner the shop then definitely make the buy formage blac and get free honey. Light cream and chicken as second strongest rule.

ECLAT Algorithm

The ECLAT algorithm stands for **Equivalence Class Clustering and bottom-up Lattice Traversal**. It is one of the popular methods of Association Rule mining. It is a more efficient and scalable version of the Apriori algorithm. While the Apriori algorithm works in a horizontal sense imitating the Breadth-First Search of a graph, the ECLAT algorithm works in a vertical manner just like the Depth-First Search of a graph. This vertical approach of the ECLAT algorithm makes it a faster algorithm than the Apriori algorithm.

How the algorithm work? :
The basic idea is to use Transaction Id Sets(tidsets) intersections to compute the support value of a candidate and avoiding the generation of subsets which do not exist in the prefix tree. In the first call of the function, all single items are used along with their tidsets. Then the function is called recursively and in each recursive call, each item-tidset pair is verified and combined with other item-tidset pairs. This process is continued until no candidate item-tidset pairs can be combined.
Let us now understand the above stated working with an example:-

Consider the following transactions record:-

Transaction Id	Bread	Butter	Milk	Coke	Jam
T1	1	1	0	0	1
T2	0	1	0	1	0
T3	0	1	1	0	0
T4	1	1	0	1	0
T5	1	0	1	0	0
T6	0	1	1	0	0
T7	1	0	1	0	0
T8	1	1	1	0	1
T9	1	1	1	0	0

The above-given data is a boolean matrix where for each cell (i, j), the value denotes whether the j'th item is included in the i'th transaction or not. 1 means true while 0 means false.

We now call the function for the first time and arrange each item with it's tidset in a tabular fashion:-

k = 1, minimum support = 2

Item	Tidset
Bread	{T1, T4, T5, T7, T8, T9}
Butter	{T1, T2, T3, T4, T6, T8, T9}
Milk	{T3, T5, T6, T7, T8, T9}
Coke	{T2, T4}
Jam	{T1, T8}

We now recursively call the function till no more item-tidset pairs can be combined:-

k = 2

Item	Tidset
{Bread, Butter}	{T1, T4, T8, T9}
{Bread, Milk}	{T5, T7, T8, T9}
{Bread, Coke}	{T4}
{Bread, Jam}	{T1, T8}
{Butter, Milk}	{T3, T6, T8, T9}

Item	Tidset
{Butter, Coke}	{T2, T4}
{Butter, Jam}	{T1, T8}
{Milk, Jam}	{T8}

k = 3

Item	Tidset
{Bread, Butter, Milk}	{T8, T9}
{Bread, Butter, Jam}	{T1, T8}

k = 4

Item	Tidset
{Bread, Butter, Milk, Jam}	{T8}

We stop at k = 4 because there are no more item-tidset pairs to combine.

Since minimum support = 2, we conclude the following rules from the given dataset:-

Items Bought	Recommended Products
Bread	Butter
Bread	Milk
Bread	Jam
Butter	Milk
Butter	Coke

Items Bought	Recommended Products
Butter	Jam
Bread and Butter	Milk
Bread and Butter	Jam

Advantages over Apriori algorithm: -

1. **Memory Requirements:** Since the ECLAT algorithm uses a Depth-First Search approach, it uses less memory than Apriori algorithm.
2. **Speed:** The ECLAT algorithm is typically faster than the Apriori algorithm.
3. **Number of Computations:** The ECLAT algorithm does not involve the repeated scanning of the data to compute the individual support values.

Previous we had Apriori Model Support, Confidence and lift but in ECLAT model, we only have Support.

Eclat - Support

Movie Recommendation: $$\text{support}(M) = \frac{\#\text{ user watchlists containing } M}{\#\text{ user watchlists}}$$

Market Basket Optimisation: $$\text{support}(I) = \frac{\#\text{ transactions containing } I}{\#\text{ transactions}}$$

How frequently set of items occurred. Here M stands for set of two or more movies. Same things for transaction, if you have chips and burgers, 75% all of your order. Some body just buying burgers then they will like to chips, then we will recommend chips and there is 75% chance that they will also be interested

or like to buy chips. That is ECLAT approach and the step involve the minimum support. You want to set your support level then you set all the subset in transaction having higher support to minimum. Basically, at the top you will have strongest combination of items, which you look at may be top 10 or something like that. That’s all the ECLAT model, it’s much easier to understand compare to Apriori model.

Eclat model is actually simplified version of the Apriori Model, because we only deal with the Support. Some business model only interested in doing a support therefore we might be used ECLAT Model. This time we are going to use a Eclat analysis to analyze the highest support of combination of products here, two because the deal buy one get it another product is free. Wo, well, that's the same scenario as in Apriori. Eclat Model is adapting just Apriori package to the Eclat model by only considering the support. From scratch how I turned that Apriori implementation into this new Eclat implementation.

Eclat model same as Apriori only some code change:

Importing the libraries & Data Preprocessing are same.

Training the Eclat model on the dataset

```
from apyori import apriori
rules = apriori(transactions = transactions, min_support = 0.003,
min_confidence = 0.2, min_lift = 3, min_length = 2, max_length = 2)
```

Training the dataset is same only remove the min confidence and min lift here in order to really only consider this port but we still have to keep this. Eclat not considering rules but set of products that because we are only considering the support mean sets of products. which of course the number of transaction containing the products, support set of products like let's say ABC, which is of A,B and C divided by the total number of transaction.

Visualising the results

Displaying the first results coming directly from the output of the apriori function

```
results = list(rules)
results
```

Putting the results well organised into a Pandas DataFrame

```
def inspect(results):
    lhs         = [tuple(result[2][0][0])[0] for result in results]
```

```
    rhs         = [tuple(result[2][0][1])[0] for result in results]
    supports    = [result[1] for result in results]
    return list(zip(lhs, rhs, supports))
resultsinDataFrame = pd.DataFrame(inspect(results), columns = ['Product
1', 'Product 2', 'Support'])
```

****same as Apriori only remove the confidence and lift. Left hand side replaced by Product 1 and right hand side by Product 2. Here absolutely need to sort support. So, that directly display the result sorted by descending support.**

Displaying the results sorted by descending supports

```
resultsinDataFrame.nlargest(n = 10, columns = 'Support')
```

O/P:

	Product 1	Product 2	Support
4	herb & pepper	ground beef	0.015998
7	whole wheat pasta	olive oil	0.007999
2	pasta	escalope	0.005866
1	mushroom cream sauce	escalope	0.005733
5	tomato sauce	ground beef	0.005333
8	pasta	shrimp	0.005066
0	light cream	chicken	0.004533
3	fromage blanc	honey	0.003333
6	light cream	olive oil	0.003200

Here displaying the results sorted buy the descending Support. Indeed, we see the combination of two products. The set of two products highest support 0.0159 which mean 1.6% down to lowest support. For 10 set of products with 10 higher supports. We simply build the Eclat Model by adapting Apriori Model to Eclat Model and returning exact same output as the Eclat Model is supposed give us. Meaning the set of products having the highest support. If you want to perform an analysis with the largest set of products here analysis only two set of products. You just need to change

these parameter max_length of Training the Eclat Model increases some larger set of products. Now we can use this larger set of products. Most frequently purchase set of two products like herb& pepper and ground beef. All these seem very relevant associations leading. So, you have now an extra association rule learning model in our toolkit. The Eclat Mode nicely adapted from Apriori Model.