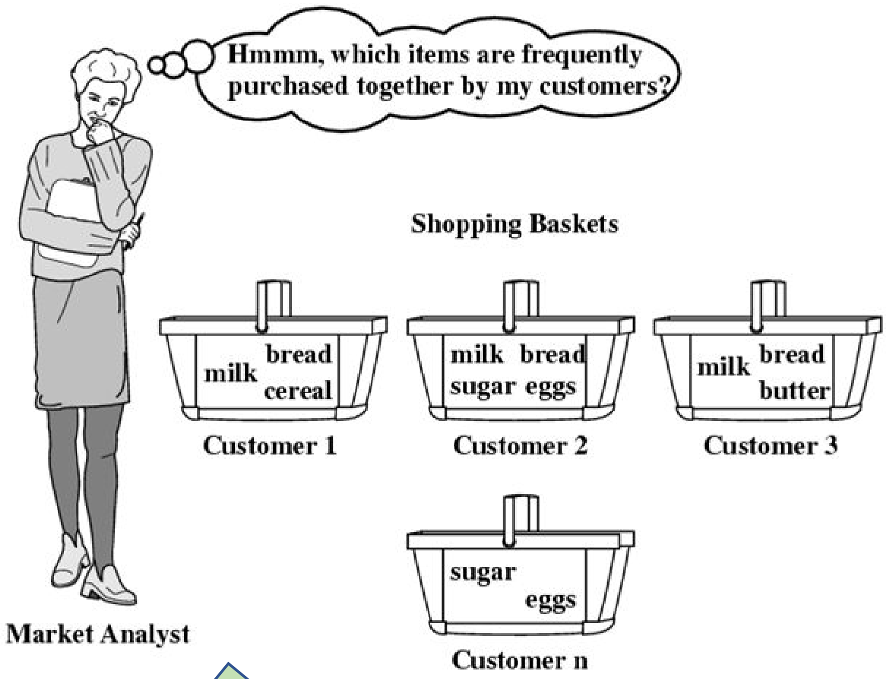
Background :

Considering a retail store,

Goal : Increase revenue by pitching one or more product with other products

How : Uncover association between frequently bought items under a set of transactions



Benefit :

- Product Placement
- Promotional Pricing

Use Association Rule

Association Rule :

- Rule-based machine learning method for **discovering interesting relations between variables** in large databases using some **measures of interestingness.**
- Works on **If – Then relationship** between variables

Association Rule :

- Rule-based machine learning method for **discovering interesting relations between variables** in large databases using some **measures of interestingness.**
- Works on **If – Then relationship** between variables

{A} -> {B}

If - Antecedent Then - Consequent

{A,B} -> {C}

If - Antecedent Then - Consequent

Association Rule :

- Rule-based machine learning method for **discovering interesting relations between variables** in large databases using some **measures of interestingness.**
- Works on **If – Then relationship** between variables

{A} -> {B}

If - Antecedent Then - Consequent

{A,B} -> {C}

If - Antecedent Then - Consequent

Trans ID	Items
151	Veggies, Spaghetti, Chicken
152	Spaghetti, Ketchup, Veggies
153	Eggs, Bread, Sausage, Milk
154	Coke, Bread, Milk, Eggs
155	Cheese, Eggs, Milk
156	Ice Cream, Brownie

Example

{Brownie} -> {Ice Cream}

Antecedent : Brownie
Consequent : Ice Cream

If a customer buys Brownie he/she is likely to buy Ice cream

Association Rule :

- Rule-based machine learning method for **discovering interesting relations between variables** in large databases using some **measures of interestingness.**
- Works on **If – Then relationship** between variables

{A} -> {B}

If - Antecedent Then - Consequent

{A,B} -> {C}

If - Antecedent Then - Consequent

Trans ID	Items
151	Veggies, Spaghetti, Chicken
152	Spaghetti, Ketchup, Veggies
153	Eggs, Bread, Sausage, Milk
154	Coke, Bread, Milk, Eggs
155	Cheese, Eggs, Milk
156	Ice Cream, Brownie

Example

{Brownie} -> {Ice Cream}

Antecedent : Brownie
Consequent : Ice Cream

If a customer buys Brownie he/she is likely to buy Ice cream

Association Rule :

Definitions:

Eggs $\Rightarrow$ Bread

- **Item set:** Collection of items to one or more items

Item set = {Eggs, Bread}

Trans ID	Items
151	Veggies, Spaghetti, Chicken
152	Spaghetti, Ketchup, Veggies
153	Eggs, Bread, Sausage, Milk
154	Coke, Bread, Milk, Eggs
155	Cheese, Eggs, Milk
156	Ice Cream, Brownie

Association Rule :

Definitions:

Eggs $\Longrightarrow$ Bread

- **Item set:** Collection of items to one or more items

  Item set = {Eggs, Bread}

- **Support (s):** Gives an idea of **how frequent an itemset** is in all the transactions. A high value means that the items involves a great part of database.

$$s(Eggs, Bread) = \frac{count(Eggs,Bread)}{Nof\ transactions} = 2/6\ (33.33\%)$$

Trans ID	Items
151	Veggies, Spaghetti, Chicken
152	Spaghetti, Ketchup, Veggies
153	Eggs, Bread, Sausage, Milk
154	Coke, Bread, Milk, Eggs
155	Cheese, Eggs, Milk
156	Ice Cream, Brownie

No Bread

Association Rule :

Definitions:

Eggs $\Longrightarrow$ Bread

- **Item set:** Collection of items to one or more items

Item set = {Eggs, Bread}

- **Support (s):** Gives an idea of **how frequent an itemset** is in all the transactions. A high value means that the items involves a great part of database.

$$s(Eggs, Bread) = \frac{count(Eggs,Bread)}{N of\ transactions} = 2/6\ (33.33\%)$$

- **Confidence (c)**: Measure of entries with consequent where antecedent appears in the transaction.

$$c(Eggs, Bread) = \frac{count(Eggs,Bread)}{count(Eggs)} = 2/3\ (66.66\%)$$

Trans ID	Items
151	Veggies, Spaghetti, Chicken
152	Spaghetti, Ketchup, Veggies
153	Eggs, Bread, Sausage, Milk
154	Coke, Bread, Milk, Eggs
155	Cheese, Eggs, Milk
156	Ice Cream, Brownie

Association Rule :

Definitions:

Eggs ⟹ Bread

- **Lift:** Indicates whether there is a relationship between items, or whether the two items are occurring randomly

$$Lift(Eggs, Bread) = \frac{s(Eggs,\ \ Bread)}{s(Eggs)\ *\ s(Bread)} = \frac{2/6}{3/6 * 2/6} = 2$$

- lift = 1 implies **no relationship** between A and B

  (ie: A and B occur together only by chance)
- lift > 1 implies that there is a **positive relationship**

  (ie: A and B occur together more often than random)
- lift < 1 implies that there is a **negative relationship**

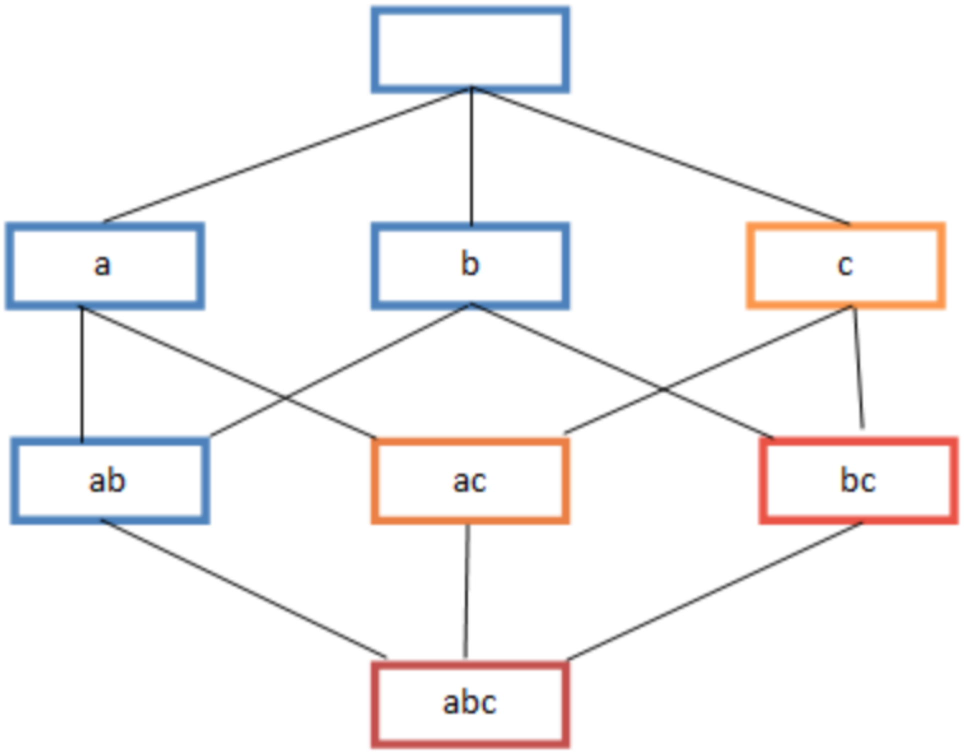
  (ie: A and B occur together less often than random)

Trans ID	Items
151	Veggies, Spaghetti, Chicken
152	Spaghetti, Ketchup, Veggies
153	Eggs, Bread, Sausage, Milk
154	Coke, Bread, Milk, Eggs
155	Cheese, Eggs, Milk
156	Ice Cream, Brownie

$$\boldsymbol{c(A \rightarrow B) \neq c(B \rightarrow A)}$$
$$\boldsymbol{lift(A \rightarrow B) = lift(B \rightarrow A)}$$

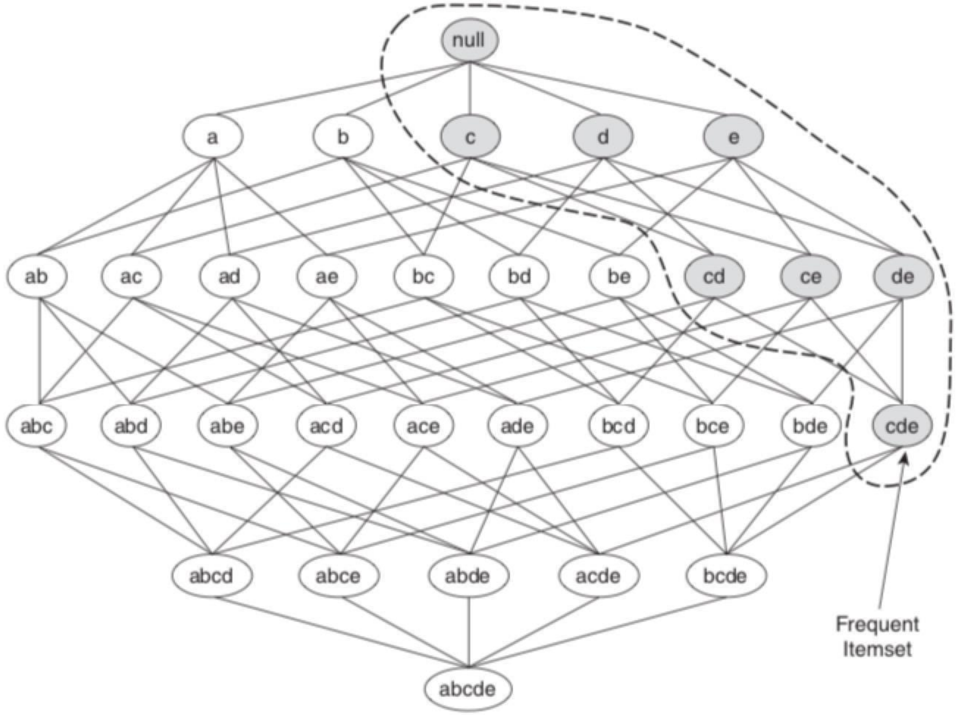
Association Rule :

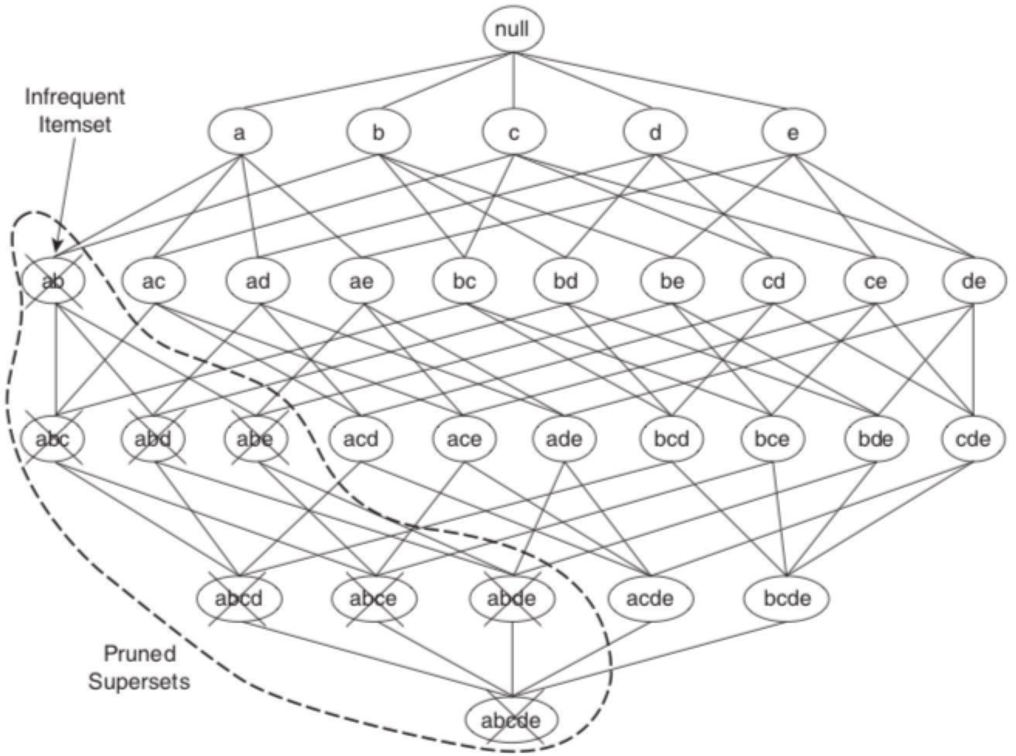
- For ‘n’ unique items 2^n -1 itemset can be obtained (Excluding the empty item set). **Analyzing all sets is computationally expensive**
- Hence, we set required (minimum) support and confidence threshold and perform model on those item sets only



Rule generation: Grouping itemset with if-then condition with
(***support > min support)*** & (***confidence > min confidence)***

Apriori Algorithm:





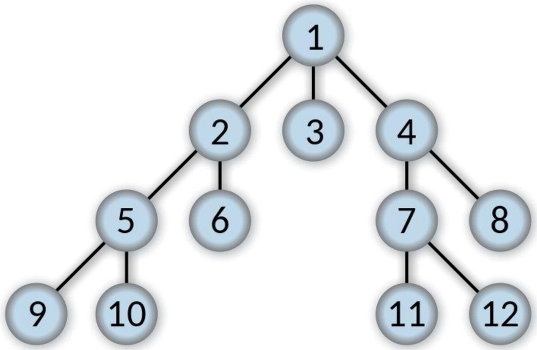
- Any **subset of a frequent itemset** must be **frequent**
- **Supersets of infrequent itemset must also be infrequent**

A transaction containing {eggs, milk, bread} also contains {bread, milk}

If {Eggs, Milk, Bread} is frequent → {Egg, Milk} must also be frequent

Apriori Algorithm:

- Breadth-first search strategy to count the support of item sets
- A candidate generation function exploits the downward closure property of support.



For *k* products

1. User sets a **minimum support criterion**
2. Algorithm **generates a list of one-item sets** that meet the support criterion
3. Use this list of one-item sets to **generate list of two-item sets** that meet the support criterion
4. Use list of two-item sets to **generate list of three-item sets**
5. Continue up **through *k*-item sets**

Example: User input -> min support=2

Itemset	Support
Veggies	2
Spaghetti	2
Chicken	1
Ketchup	1
Eggs	3
Bread	2
Sausage	1
Milk	3
Coke	1
Cheese	1
Ice Cream	1
Brownie	1

Trans ID	Items
151	Veggies, Spaghetti, Chicken
152	Spaghetti, Ketchup, Veggies
153	Eggs, Bread, Sausage, Milk
154	Coke, Bread, Milk, Eggs
155	Cheese, Eggs, Milk
156	Ice Cream, Brownie

Example: User input -> min support=2

Itemset	Support
Veggies	2
Spaghetti	2
Chicken	1
Ketchup	1
Eggs	3
Bread	2
Sausage	1
Milk	3
Coke	1
Cheese	1
Ice Cream	1
Brownie	1

Itemset	Support
Veggies, Spaghetti	1
Veggies, Eggs	0
Veggies, Bread	0
Veggies, Milk	0
Spaghetti, Eggs	0
Spaghetti, Bread	0
Spaghetti, Milk	0
Eggs, Bread	2
Eggs, Milk	3
Bread, Milk	2

Trans ID	Items
151	Veggies, Spaghetti, Chicken
152	Spaghetti, Ketchup, Veggies
153	Eggs, Bread, Sausage, Milk
154	Coke, Bread, Milk, Eggs
155	Cheese, Eggs, Milk
156	Ice Cream, Brownie

Example: User input -> min support=2

Itemset	Support
Veggies	2
Spaghetti	2
Chicken	1
Ketchup	1
Eggs	3
Bread	2
Sausage	1
Milk	3
Coke	1
Cheese	1
Ice Cream	1
Brownie	1

Itemset	Support
Veggies, Spaghetti	1
Veggies, Eggs	0
Veggies, Bread	0
Veggies, Milk	0
Spaghetti, Eggs	0
Spaghetti, Bread	0
Spaghetti, Milk	0
Eggs, Bread	2
Eggs, Milk	3
Bread, Milk	2

Itemset	Support
Eggs, Bread, Milk	2

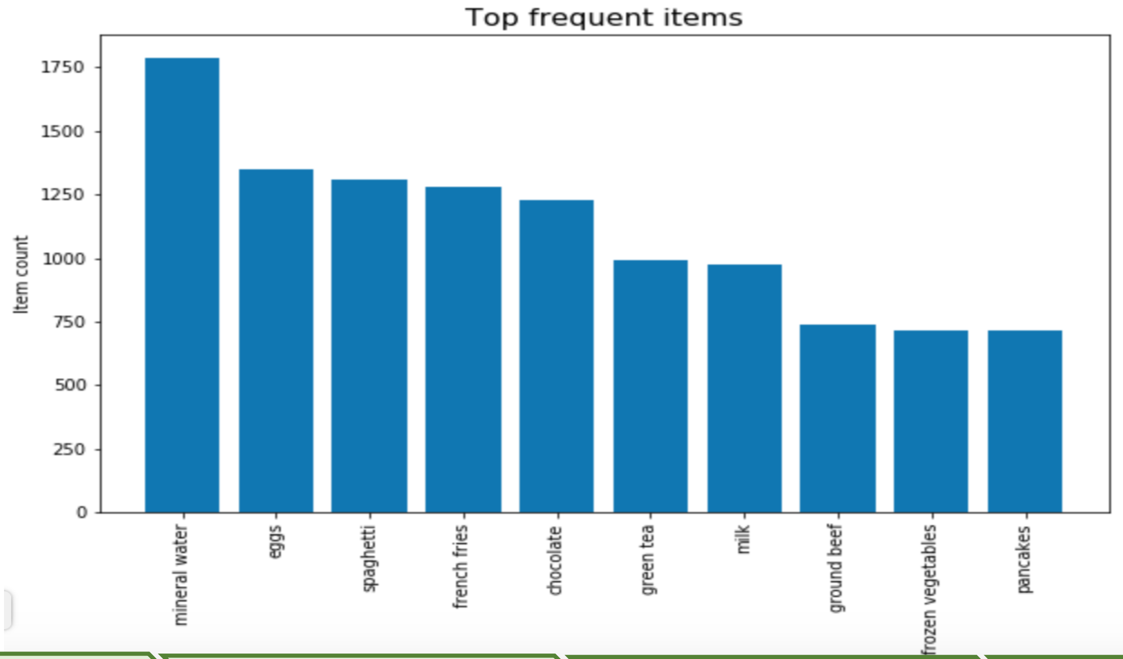
Trans ID	Items
151	Veggies, Spaghetti, Chicken
152	Spaghetti, Ketchup, Veggies
153	Eggs, Bread, Sausage, Milk
154	Coke, Bread, Milk, Eggs
155	Cheese, Eggs, Milk
156	Ice Cream, Brownie

Modelling :

Gather transaction data

```
print (df_list)  # List contains NAN Values
```

```
[['shrimp', 'almonds', 'avocado', 'vegetables mix', 'green grapes', 'w
ergy drink', 'tomato juice', 'low fat yogurt', 'green tea', 'honey', '
juice', 'frozen smoothie', 'spinach', 'olive oil'], ['burgers', 'meatb
n', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'na
an', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'n
['turkey', 'avocado', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan',
n', 'nan', 'nan', 'nan', 'nan'], ['mineral water', 'milk', 'energy bar
n', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'na
an', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'n
'nan', 'nan'], ['whole wheat pasta', 'french fries', 'nan', 'nan', 'na
'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan'], ['soup
n', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'na
ables', 'spaghetti', 'green tea', 'nan', 'nan', 'nan', 'nan', 'nan', '
'nan', 'nan', 'nan', 'nan', 'nan'], ['french fries', 'nan', 'nan', 'na
'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan'],
n', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'nan', 'na
```



Cleaning the data

```
print (clean_df_list)  # List cleaned
```

```
[['shrimp', 'almonds', 'avocado', 'vegetables mix', 'green grapes', 'w
ergy drink', 'tomato juice', 'low fat yogurt', 'green tea', 'honey', '
juice', 'frozen smoothie', 'spinach', 'olive oil'], ['burgers', 'meatb
o'], ['mineral water', 'milk', 'energy bar', 'whole wheat rice', 'gree
a', 'french fries'], ['soup', 'light cream', 'shallot'], ['frozen vege
ies'], ['eggs', 'pet food'], ['cookies'], ['turkey', 'burgers', 'miner
i', 'champagne', 'cookies'], ['mineral water', 'salmon'], ['mineral wa
y', 'oil', 'cooking oil', 'low fat yogurt'], ['turkey', 'eggs'], ['tur
ineral water', 'black tea', 'salmon', 'eggs', 'chicken', 'extra dark c
nch fries', 'protein bar'], ['red wine', 'shrimp', 'pasta', 'pepper',
kling water'], ['spaghetti', 'mineral water', 'ham', 'body spray', 'pa
se', 'shrimp', 'pasta', 'avocado', 'honey', 'white wine', 'toothpaste'
'soup', 'avocado', 'milk', 'fresh bread'], ['ground beef', 'spaghetti'
tea', 'salmon', 'frozen smoothie', 'escalope'], ['sparkling water'], [
e', 'french fries'], ['frozen vegetables', 'spaghetti', 'yams', 'miner
'light cream', 'magazines'], ['mineral water', 'chocolate', 'avocado',
```

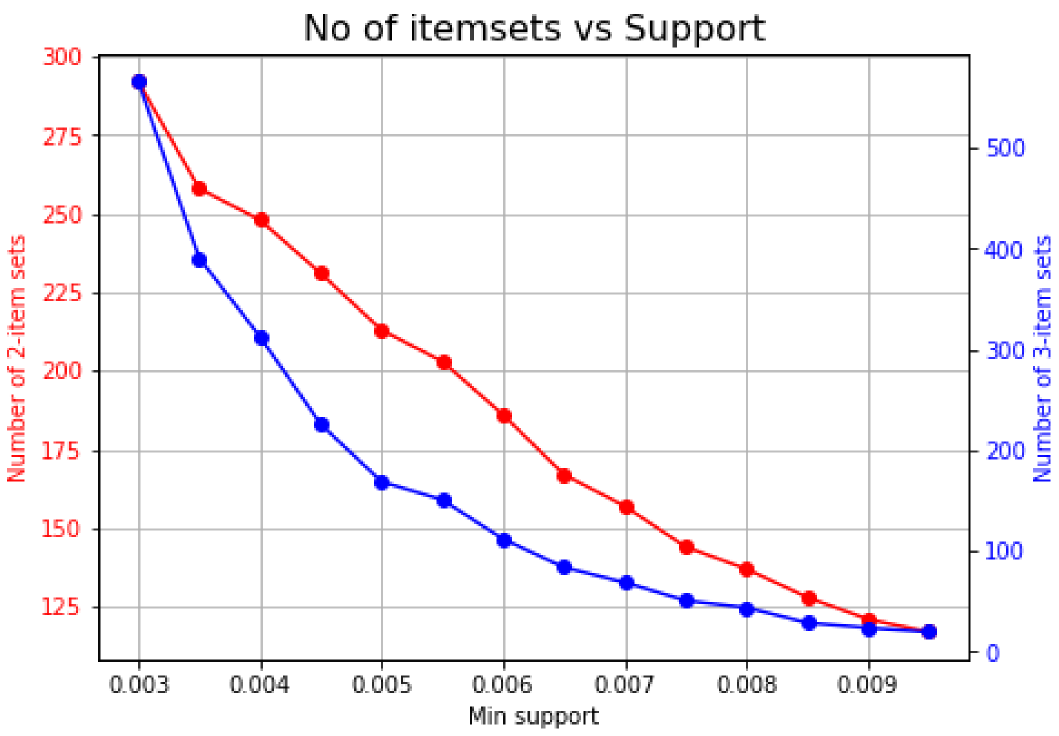
Performing Association Rule in training data

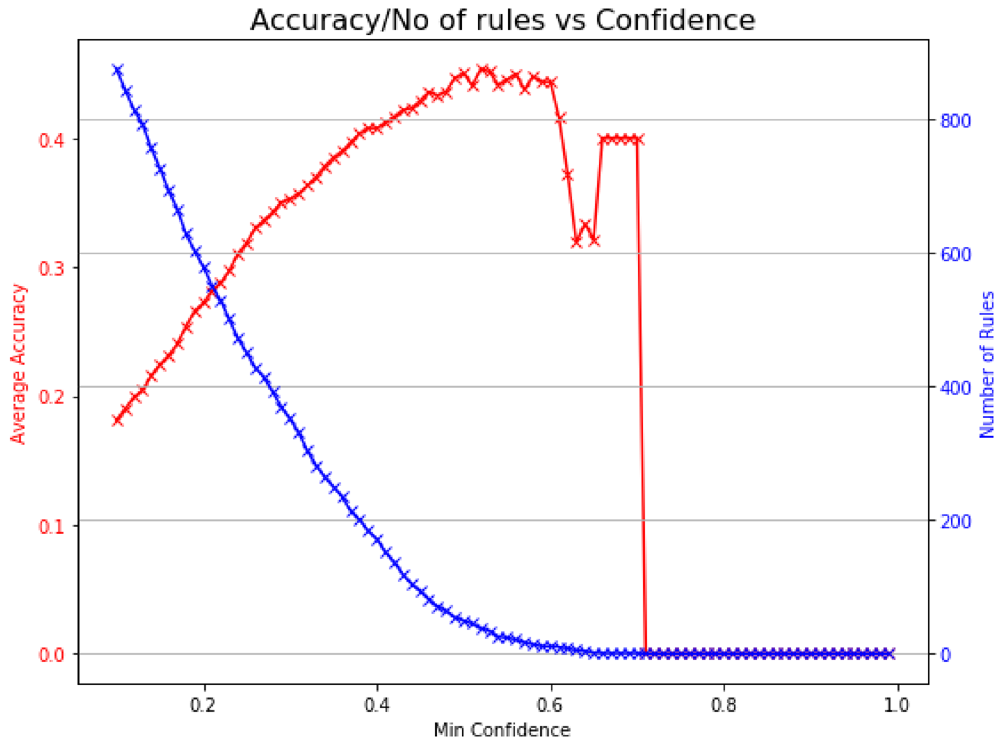
```
association_rules = apriori(train, min_support=0.0045, min_confidence=0.2, min_lift=3, min_length=2)
association_results = list(association_rules)
```

Generated Rules

```
Rule: ['light cream'] -> ['chicken']
Support: 0.004952380952380952
Confidence: 0.2988505747126437
Lift: 4.9338538278030795
=====================================
Rule: ['mushroom cream sauce'] -> ['escalope']
Support: 0.005523809523809524
Confidence: 0.3118279569892473
Lift: 3.879376242164806
=====================================
Rule: ['pasta'] -> ['escalope']
Support: 0.005142857142857143
Confidence: 0.33749999999999997
Lift: 4.198755924170616
=====================================
Rule: ['fresh tuna'] -> ['pancakes']
Support: 0.006285714285714286
Confidence: 0.28695652173913044
Lift: 3.0496391480373175
```

How to select min_support, min_confidence??

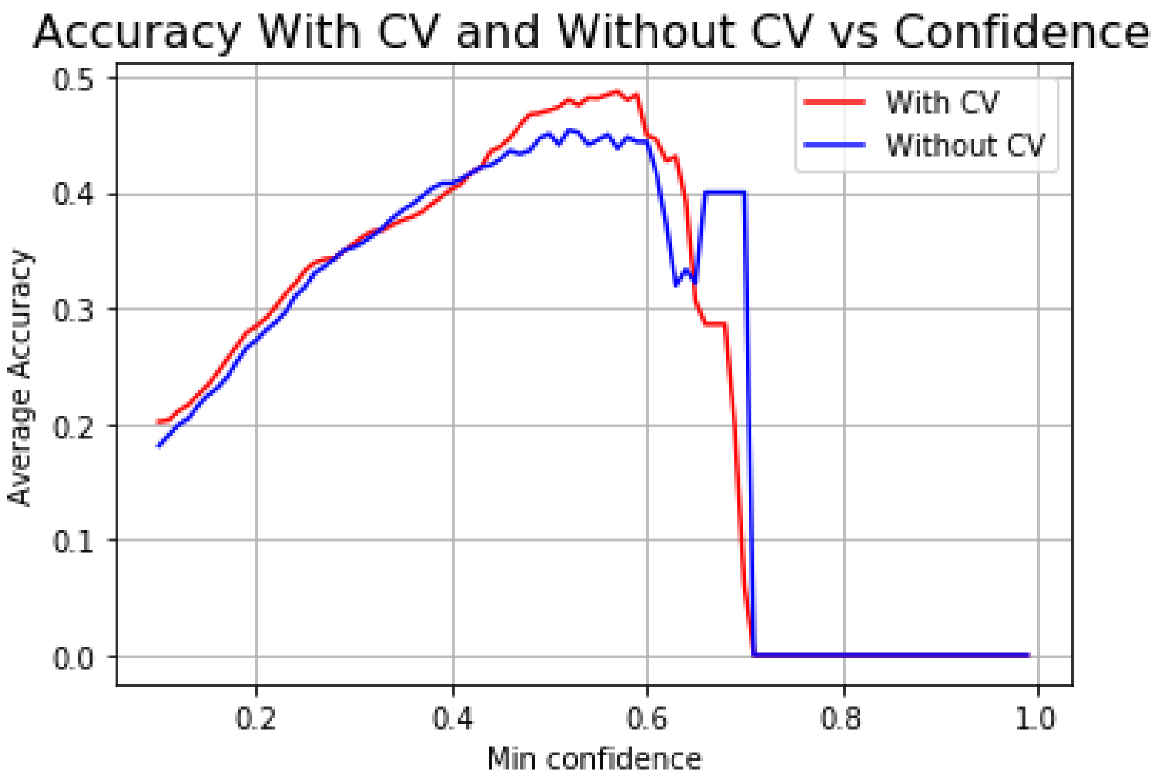




The number of item sets decrease with increase in support value. Therefore, to have optimum transactions to check for confidence, we will choose the least value of support

There is a trade-off between model accuracy and number of rules, at different min_confidence values.

Results:



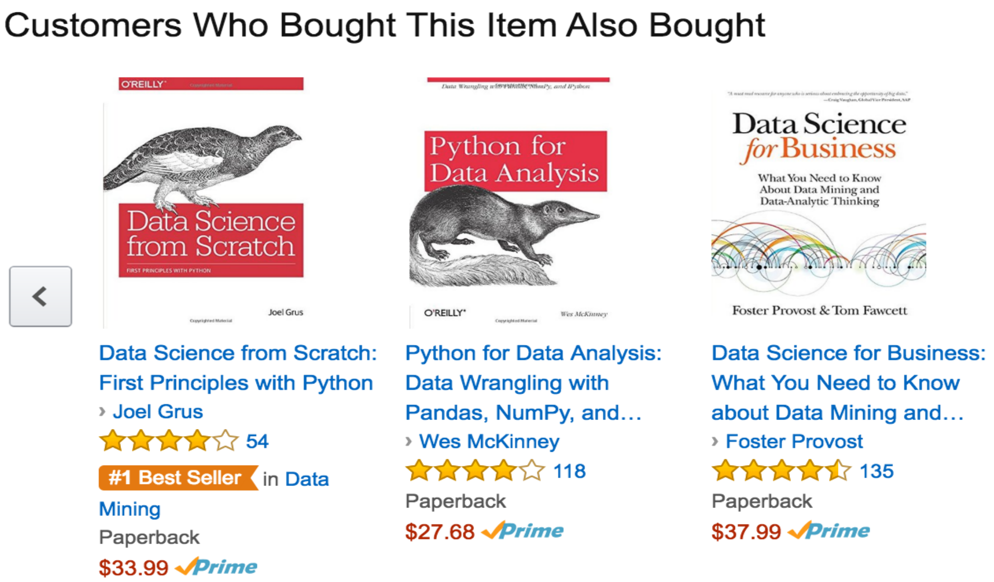
Top 5 rules after parameter tuning

```
Rule #1 ['cake', 'turkey'] -> ['eggs']
Support: 0.004
Confidence: 0.525
Lift: 2.8532608695652177
=====================================
Rule #2 ['cake', 'soup'] -> ['mineral water']
Support: 0.004
Confidence: 0.5833333333333334
Lift: 2.4539262820512824
=====================================
Rule #3 ['cereals', 'milk'] -> ['mineral water']
Support: 0.004
Confidence: 0.525
Lift: 2.208533653846154
=====================================
Rule #4 ['ground beef', 'chicken'] -> ['spaghetti']
Support: 0.004761904761904762
Confidence: 0.49019607843137264
Lift: 2.812600450016072
=====================================
Rule #5 ['soup', 'chocolate'] -> ['mineral water']
Support: 0.007238095238095238
Confidence: 0.6440677966101694
Lift: 2.709419817470665
=====================================
```

```
print ("The Accuracy at support: {}, confidence: {} and lift: {} is: {}"
print ("The Number of rules are: {}".format(len(rul_res)))
```

```
The Accuracy at support: 0.004, confidence: 0.48 and lift: 2.2 is: 0.461
The Number of rules are: 42
```

Applications:



1. Gain **insight about merchandise** (FMCG)
   - Which products are brought together
   - Products to be offered promotions
   - Designing store layouts, etc.,
2. Unusual **combinations of insurance** can be a sign of fraud and can spark further investigation.
3. **Medical patient histories** can give indications of likely complications based on certain combinations of treatments.
4. Plagiarism check of various documents.
5. **Suggestions on related searches** (Google, Facebook ads)