AKHIL RAJESH SHARMA
22201271

ASSOCIATION RULE REPORT

Association Rule Assignment

Q1 - Outline and discuss market basket analysis.

Market basket analysis is a data mining technique used by companies to boost their sales by understanding consumer patterns by analyzing past data collected from various systems. These systems generate raw data on each transaction which needs to be cleaned and processed to find relevant insights. Finding out items that are likely to be purchased together is the key function of Market basket analysis. It is also known as association analysis or affinity analysis where our analysis findings are called association rules.

The main objective of this analysis is to use the insights to develop effective product placement, cross-selling strategies, and discounts. Strategic decisions like product placement can be done using this analysis where products that are purchased together are kept in the vicinity of each other to induce more sales. Bundling is also a concept used in a lot of places where items usually bought together are sold in a package deal that benefits the customer and increases sales. Many other decisions can also be made from the results of market basket analysis. (Li, 2017).

The main terminologies used in Market Basket analysis are as follows which will be explained using an example. Let's assume that there are 100 customers out of which 10 bought bread, 6 bought butter and 4 bought both of them. Keeping this example in mind we will look at some of the key concepts used in market basket analysis.

1. Support: The number of times a transaction is seen as compared to the total transactions is called the support for that particular transaction. It is basically used to check the frequency of a particular transaction occurring out of the total available data. Decisions/rules on items that higher support affects a larger section of the data.

From the above example, the support for bread is given by 10/100 = 0.1

2. Confidence: The certainty of an item being purchased if the other is purchased. In technical terms, it measures the probability of a consequent item being purchased given that an antecedent item is in the transaction.

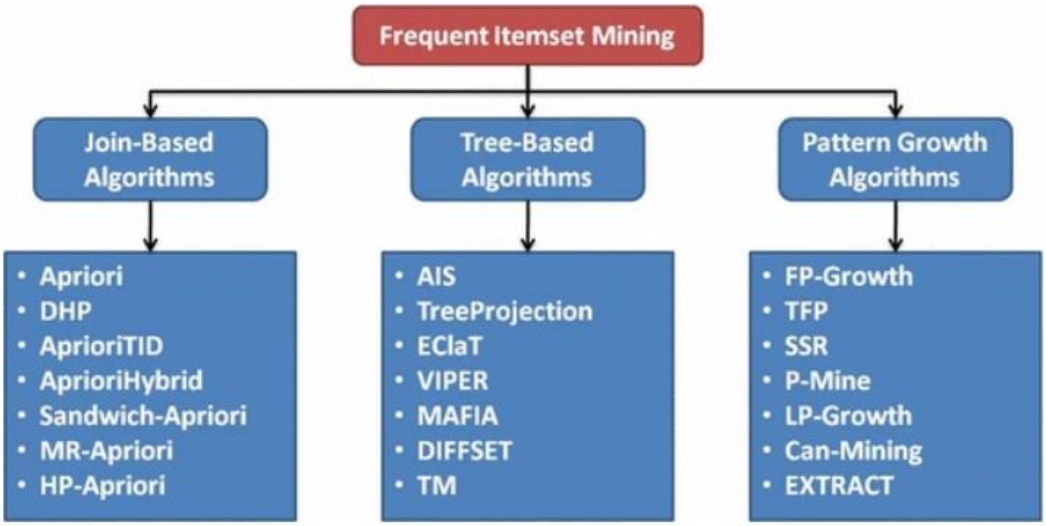
From the above example, if we keep the antecedent as butter and if we check the confidence of butter => bread, we will get it as
Confidence = Support (Bread&Butter)/Support(bread)=4/10=40%

This shows moderate to good confidence in this rule.

3. Lift: It is the measure of the correlation between two items keeping into account the frequency of occurrence of the two items independently. A positive value

indicates that the correlation is positive, and it is not chance. A zero value indicates that they are independent and a negative value means the buying of one can negatively impact the purchase of the other.

These concepts are then used in several algorithms that are used to get the results for Market Basket Analysis. Some of these algorithms are as follows each of them having its own pros and cons.



Q2 - Compare and contrast the Apriori and FP-Growth algorithms.

Apriori: Apriori is an algorithm used in the Market Basket Analysis that considers how frequent the items are to produce a frequent itemset. The items are said to be frequent if they are above a minimum support threshold.

The algorithm is an iteration of joining (set building) and pruning (checking for minimum support and filtering).(*Apriori Algorithm in Data Mining: Implementation With Examples*, 2023)

STEPS:

1. Initially, the algorithm takes 1-itemsets candidate. This is basically taking the count of each item itself in the list.
2. Now out of these itemsets, we pick the ones that are frequent and above the minimum support threshold to be used further. Others are removed (pruned).
3. In the next step all the remaining items with the minimum support are joined to create the 2-itemset which is formed by combining items with itself from the previous set.
4. In this manner, the join step to create further itemsets and the prune step to remove items having less than minimum support are repeated and the algorithm stops when the most frequent itemset is achieved.

FP-Growth: Unlike having a step-by-step iteration, FPGrowth algorithm saves the data in a tree structure. We call this a FP-tree and is responsible for maintaining the association information between the frequent itemsets rather than producing a step-by-step process.

STEPS:

1. After creating the initial FP-tree, it is segregated into a set of further conditionally divided FP-Trees for every frequent item.
2. These tree branches can be measured individually and do the same process till the root node is reached.
3. At each level the minimum threshold is used to filter out the process to go for further branches.

Differences: (Verma, 2021)

Apriori Algorithm	FP Growth Algorithm
Apriori generates frequent patterns by making the itemsets using pairing. It has a frequent itemset building step(joining) and removing infrequent itemset based on minimum threshold(pruning).	FP Growth generates an FP-Tree for making frequent patterns. It generates an FP for each item.
Apriori uses candidate generation where frequent subsets are extended one item at a time. So they start with a 1-item itemset, then goes on to be 2-item itemset and so on.	FP-growth generates a conditional FP-Tree for every item in the data. It is stored in a list format with each item having its own tree structure.
Since apriori scans the database in each of its steps it becomes time-consuming for data where the number of items is larger. For smaller support threshold, it is slower.	FP-tree requires only one scan of the database in its beginning steps so it consumes less time. However, it is slower when the FP generation of the data is complex.
A converted version of the database is saved in the memory	A Set of conditional FP-tree for every item is saved in the memory
It uses a breadth-first search	It uses a depth-first search.

A small observation seen in the analysis is as follows:

```
# Generate frequent itemsets with a minimum support of 0.1%
df_itemsets = apriori(df_onehot, min_support=0.001, use_colnames=True)
df_itemsetsf = fpgrowth(df_onehot, min_support=0.001, use_colnames=True)
print('The itemsets generated by Apriori for 0.1% minimum support are',df_itemsets.shape[0])
print('The itemsets generated by FPGrowth for 0.1% minimum support are',df_itemsetsf.shape[0])
#%timeit apriori(df_onehot, min_support=0.05, use_colnames=True)
#%timeit fpgrowth(df_onehot, min_support=0.05, use_colnames=True)

# Generate frequent itemsets with a minimum support of 0.5%
df_itemsets = apriori(df_onehot, min_support=0.005, use_colnames=True)
df_itemsetsf = fpgrowth(df_onehot, min_support=0.005, use_colnames=True)
print('The itemsets generated by Apriori for 0.5% minimum support are',df_itemsets.shape[0])
print('The itemsets generated by FPGrowth for 0.5% minimum support are',df_itemsetsf.shape[0])
#%timeit apriori(df_onehot, min_support=0.05, use_colnames=True)
#%timeit fpgrowth(df_onehot, min_support=0.05, use_colnames=True)

df_itemsets.sort_values(by=['support'], ascending=False)
```

```
The itemsets generated by Apriori for 10% minimum support are 4
The itemsets generated by FPGrowth for 10% minimum support are 4
The itemsets generated by Apriori for 1% minimum support are 68
The itemsets generated by FPGrowth for 1% minimum support are 68
The itemsets generated by Apriori for 0.1% minimum support are 762
The itemsets generated by FPGrowth for 0.1% minimum support are 762
The itemsets generated by Apriori for 0.5% minimum support are 133
The itemsets generated by FPGrowth for 0.5% minimum support are 133
```

It is seen that both algorithms generate equal outcomes. Hence the main difference between the algorithms is time and memory consumed.

We compared the time taken which was seen by

```
#%timeit apriori(df_onehot, min_support=0.05, use_colnames=True)
#%timeit fpgrowth(df_onehot, min_support=0.05, use_colnames=True)
```

In the result, it was seen that Apriori was faster than FPgrowth. It was however seen that the gap between the two was much higher for a smaller dataset(7500 records) than it was for a relatively larger dataset(15000) records.

Also over the lower support threshold, Apriori takes more items which makes it slower.

Q3 - Code Overview

The first step is to import all the relevant libraries. We have used libraries like permutations to see how many rules can be generated. We have used Pandas to use dataframes and other functions. We have used mlxtend to use the apriori, fpgrowth, etc. I have also implemented some visualizations for which I have used matplotlib. The code comment below it was me trying to experiment with different ways I can do the same code.

```
# import required packages here
import pandas as pd
import collections
from ast import literal_eval
from itertools import permutations
!pip install mlxtend
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import fpgrowth
from mlxtend.frequent_patterns import association_rules

!pip install matplotlib
import matplotlib.pyplot as plt

#for experimenting with the data
!pip install apyori
!pip install pyspark
#from apyori import apriori
#import pyspark
#from pyspark.ml.fpm import FPGrowth
```

The corresponding steps in the code are just to understand the data that we have after importing it from a CSV (Comma separated values) file to a dataframe using Pandas. We can see the shape of the data (rows and columns) using the functions given to us by Pandas.

```
#loading our data into a dataframe using pandas
df = pd.read_csv('basket.csv')

print(df.shape) #to see overall shape
print(df.shape[1]) #column
print(df.shape[0]) #rows
print(df.columns)
```

```
# Use df.head() to see the first 8 rows of the df
df.head(8)
```

```
# Set the index of the df to the Transaction_ID column
df.set_index('ID', inplace=True)
# Use df.head() to see the first 3 rows of the df
df.head(3)
```

In the next step instead of keeping a data frame of 15,000 transactions, we create a list of lists. Here the inner list consists of each transaction individually and the outer list is the list of all these transactions.

If we had to take an example, a college would be a list of lists. Here the student names inside a class would be the inner list and the list of all the possible classes(departments) would be the outer list.

```
# create a list of lists, where each list contains the items in a given transaction
cols = [i for i in df.columns]
print(cols) #just to verify and check numberic values of columns

df_split = pd.DataFrame(df.values.tolist(),columns=cols)
df_split.head()
```

In this step instead of having a list of lists, we consolidate it into one giant list with all the items. This flattened list is then used to aggregate the count of each item. This is then sorted to see which item is bought the most to see the support of 1-item candidate set. We also see how many unique items are these in the list.

```
# from your list of lists, create a flattened list that contains all items purchased
flattened = [item for transaction in transactions for item in transaction]

print(flattened[:2])

len(flattened)
print(len(flattened)) # view the legnth of the list
```

```
# create a list of unique items from the flattened list
# use the set() method because a set by definition contains only unique items
items = list(set(flattened))

# print the count of unique items which is the length of the list
print('# of items:',len(items))

# sort items alphabeitcally
items.sort()
print(items)# print out how many unique items we have
```

In Apriori, in each step, a (1+k) itemset is made in a permutation with itself and in each round itemset with support less than minimum support threshold is removed and the process continues. Itemset permutations can be taken in the following method.

```
# generate the itemset permutations up to 2-itemsets
rules = list(permutations(items, 2))
print('# of rules:',len(rules)) # print out the number of rules
print(rules[:3]) # print some of the elements of the list of rules

#Experiment to generate itemsets upto 3-itemsets
rules1a = list(permutations(items, 3))
print('# of rules:',len(rules1a)) # print out the number of rules
print(rules1a[:3]) # print some of the elements of the list of rules
```

In one hot encoding, we replace the values of the items as true/false that is we make it as 1 or 0 based on its presence in each transaction.

```
# one-hot encode the data and show the first 5 rows of the resulting df
# don't forget to drop the 'nan' value
encoder = TransactionEncoder().fit(transactions)
onehot = encoder.transform(transactions)
df_onehot = pd.DataFrame(onehot, columns=encoder.columns_).drop('nan', axis=1)
df_onehot.head()
```

In the code above we use the Apriori algorithm and the FPGrowth algorithm to generate frequent itemsets. As the first parameter we pass the one hot encoded (1,0) dataset. In each different scenario we pass different parameters which are used to filter out frequent itemsets which satisfy the minimum support threshold. We observe that the number of frequent itemsets generated decreases with each increase in the minimum threshold value as more items are pruned at the end of each step which do not satisfy the minimum requirement.

```
# Generate frequent itemsets with a minimum support of 10%
df_itemsets = apriori(df_onehot, min_support=0.1, use_colnames=True)
df_itemsetsf = fpgrowth(df_onehot, min_support=0.1, use_colnames=True)
print('The itemsets generated by Apriori for 10% minimum support are',df_itemsets.shape[0])
print('The itemsets generated by FPGrowth for 10% minimum support are',df_itemsetsf.shape[0])
#%timeit apriori(df_onehot, min_support=0.1, use_colnames=True)
#%timeit fpgrowth(df_onehot, min_support=0.1, use_colnames=True)

# Generate frequent itemsets with a minimum support of 1%
df_itemsets = apriori(df_onehot, min_support=0.01, use_colnames=True)
df_itemsetsf = fpgrowth(df_onehot, min_support=0.01, use_colnames=True)
print('The itemsets generated by Apriori for 1% minimum support are',df_itemsets.shape[0])
print('The itemsets generated by FPGrowth for 1% minimum support are',df_itemsetsf.shape[0])
#%timeit apriori(df_onehot, min_support=0.01, use_colnames=True)
#%timeit fpgrowth(df_onehot, min_support=0.01, use_colnames=True)

# Generate frequent itemsets with a minimum support of 0.1%
df_itemsets = apriori(df_onehot, min_support=0.001, use_colnames=True)
df_itemsetsf = fpgrowth(df_onehot, min_support=0.001, use_colnames=True)
print('The itemsets generated by Apriori for 0.1% minimum support are',df_itemsets.shape[0])
print('The itemsets generated by FPGrowth for 0.1% minimum support are',df_itemsetsf.shape[0])
#%timeit apriori(df_onehot, min_support=0.001, use_colnames=True)
#%timeit fpgrowth(df_onehot, min_support=0.001, use_colnames=True)

# Generate frequent itemsets with a minimum support of 0.5%
df_itemsets = apriori(df_onehot, min_support=0.005, use_colnames=True)
df_itemsetsf = fpgrowth(df_onehot, min_support=0.005, use_colnames=True)
print('The itemsets generated by Apriori for 0.5% minimum support are',df_itemsets.shape[0])
print('The itemsets generated by FPGrowth for 0.5% minimum support are',df_itemsetsf.shape[0])
#%timeit apriori(df_onehot, min_support=0.005, use_colnames=True)
#%timeit fpgrowth(df_onehot, min_support=0.005, use_colnames=True)
```

```
The itemsets generated by Apriori for 10% minimum support are 4
The itemsets generated by FPGrowth for 10% minimum support are 4
The itemsets generated by Apriori for 1% minimum support are 68
The itemsets generated by FPGrowth for 1% minimum support are 68
The itemsets generated by Apriori for 0.1% minimum support are 762
The itemsets generated by FPGrowth for 0.1% minimum support are 762
The itemsets generated by Apriori for 0.5% minimum support are 133
The itemsets generated by FPGrowth for 0.5% minimum support are 133
```

Using the itemsets generated with the minimum support for 0.5%, we now create association rules with the minimum confidence threshold of 10%. The total rules generated are 42 based on the criterias of initially creating itemsets with the given minimum support and minimum confidence threshold.

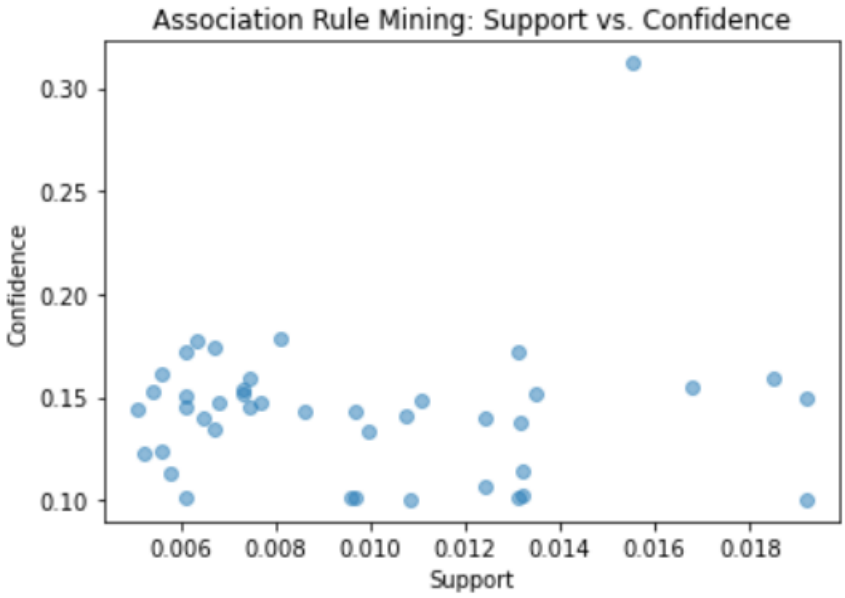
```
# generate association rules with a confidence threshold of 10%
rules_df = association_rules(df_itemsets, metric='confidence', min_threshold=0.1)

# how many rules are generated?
print('Number of rules generated with minimum(s)= 0.5% and minimum(c)=10% is', rules_df.shape[0])
rules_df.head(8)
```

```
Number of rules generated with minimum(s)= 0.5% and minimum(c)=10% is 42
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(beef)	(other vegetables)	0.074510	0.128568	0.009930	0.133274	1.036610	0.000351	1.005431
1	(beef)	(whole milk)	0.074510	0.191109	0.011048	0.148279	0.775887	-0.003191	0.949714
2	(bottled beer)	(berries)	0.045311	0.116336	0.005590	0.123367	1.060442	0.000319	1.008021
3	(bottled water)	(berries)	0.060108	0.116336	0.006116	0.101751	0.874629	-0.000877	0.983763
4	(berries)	(other vegetables)	0.116336	0.128568	0.013218	0.113624	0.883764	-0.001739	0.983140
5	(other vegetables)	(berries)	0.128568	0.116336	0.013218	0.102813	0.883764	-0.001739	0.984928
6	(pastry)	(berries)	0.051098	0.116336	0.005787	0.113256	0.973529	-0.000157	0.996527
7	(soda)	(berries)	0.095226	0.116336	0.009667	0.101519	0.872642	-0.001411	0.983510

In this section we create a scatterplot for the values of support and confidence for the top 10 rules given.

```
# Get the top 10 items sold in association rule
top_items = pd.Series(rules_df['antecedents'].tolist()+rules_df['consequents'].tolist()).value_counts().head(10)

top_items.index = top_items.index.map(lambda x: ', '.join(list(x)))


# Create a bar graph plot for the top 10 items
plt.bar(top_items.index, top_items.values)
plt.xlabel('Item')
plt.ylabel('Number of Sales')
plt.title('Top 10 Items Sold in Association Rule')

# Rotate the x-axis labels for better visibility
plt.xticks(rotation=45)

# Show the plot
plt.show()

import matplotlib.pyplot as plt

# Generate association rules using confidence and minimum support thresholds
rules_df = association_rules(df_itemsets, metric='confidence', min_threshold=0.1)

# Plot scatter plot of support vs. confidence
plt.scatter(rules_df['support'], rules_df['confidence'], alpha=0.5)

# Set axis labels and title
plt.xlabel('Support')
plt.ylabel('Confidence')
plt.title('Association Rule Mining: Support vs. Confidence')

# Show the plot
plt.show()
```



Here we give more importance to values with higher confidence and support as they are associated in a relatively stronger manner as well as show up in a lot of transactions.

Q4 - Result Interpretation

For the step of result interpretation, I have used an additional dataset to explain rules and explain some different outcomes and point out some common misunderstandings.

Dataset 1

```
rules_df.sort_values(by=['lift'], ascending=False).drop(columns=['antecedent support', 'consequent support', 'conviction']).head(
```

	antecedents	consequents	support	confidence	lift	leverage
21	(eggs)	(whole milk)	0.015520	0.312169	1.633464	0.006019
26	(root vegetables)	(other vegetables)	0.013087	0.171997	1.337790	0.003304
25	(other vegetables)	(root vegetables)	0.013087	0.101790	1.337790	0.003304
28	(white bread)	(other vegetables)	0.006116	0.171587	1.334602	0.001533
9	(berries)	(yogurt)	0.012429	0.106840	1.203414	0.002101
10	(yogurt)	(berries)	0.012429	0.140000	1.203414	0.002101
2	(bottled beer)	(berries)	0.005590	0.123367	1.060442	0.000319
20	(eggs)	(other vegetables)	0.006708	0.134921	1.049413	0.000316

Rule 1: Antecedent (Eggs) => Consequent(Whole Milk)

The following rule is the most positive and useful rule out of our current data set with the highest confidence and lift value. With a support of 1.5% it affects that many transactions out of the total. The confidence of 31% is moderate but it is the highest in the current set of rules along with a lift value of 1.6 states that they are not bought together by chance but intentional and hence can be used in bundles as a breakfast sale deal.

Rule 2: Antecedent (root vegetables) => Consequent(other vegetables)

The following rule has a confidence of only 17% which is not too good to work on. A positive lift shows a positive correlation. And a positive leverage shows a better odd than being independent. This rule affects 1.3% of the transactions as the two items are seen together that many times.

However, this insight as well isn't too worthwhile to devise any strategy out of it.

Dataset 2

```
rules_df2.sort_values(by=['lift'], ascending=False).drop(columns=['antecedent support', 'consequent support', 'conviction'])
```

	antecedents	consequents	support	confidence	lift	leverage
16	(pasta)	(escalope)	0.005866	0.372881	4.700812	0.004618
66	(pasta)	(shrimp)	0.005066	0.322034	4.506672	0.003942
210	(herb & pepper, spaghetti)	(ground beef)	0.006399	0.393443	4.004360	0.004801
208	(herb & pepper, mineral water)	(ground beef)	0.006666	0.390625	3.975683	0.004989
30	(tomato sauce)	(ground beef)	0.005333	0.377358	3.840659	0.003944

Rule 3: Antecedent (Pasta) => Consequent (escalope)

The rule shows how having the highest confidence doesn't always ensure the highest lift value which is perfectly seen in this dataset. Here having a lift of 4 is a very high value which basically shows that it is not chance that these items show up together in transactions and they are highly positively correlated and are often bought together. A good idea would be to promote buying these items together by throwing them in a bungle or placing the items close to one another so that the sale of escalope because of pasta grows. Support of 0.58% shows that this affects many transactions out of the total. A positive leverage

Rule 4: Antecedent (herb pepper, spaghetti) =>Consequent (ground beef)

The rule has a support of 0.66% which means this set shows up in 0.66% of the total transactions. A positive leverage above indicates that they are associated to each other strongly and are seen in a single transaction together. The confidence is 39% which is a moderately high value displaying the certainty of this rule.

```
rules_df2.sort_values(by=['confidence'], ascending=False).drop(columns=['antecedent support', 'consequent support', 'conviction'])
```

	antecedents	consequents	support	confidence	lift	leverage
189	(soup, frozen vegetables)	(mineral water)	0.005066	0.633333	2.656954	0.003159
243	(soup, olive oil)	(mineral water)	0.005199	0.582090	2.441976	0.003070
185	(frozen vegetables, olive oil)	(mineral water)	0.006532	0.576471	2.418404	0.003831
232	(soup, milk)	(mineral water)	0.008532	0.561404	2.355194	0.004909
133	(soup, chocolate)	(mineral water)	0.005599	0.552632	2.318395	0.003184

Rule 5: Antecedent (soup,milk) => Consequent (mineral water)

The above rule has a support of 0.8% which means that 0.8% of the transactions are impacted. The positive lift ratio indicates that this is not by chance and the confidence level of 56% is a moderately high value to consider a action.

Rule 6: Antecedent (Soup, Frozen vegetables) => Consequent(mineral water)

The rule has the highest confidence of 63% which means if Soup and frozen vegetables are purchased then water will also be purchased. It isn't a very high confidence level, but it is the highest in the current dataset which means that it can be looked into. If we see closely we also notice that most of the high confidence consequents are mineral water which might be influencing the outcome.

Good Minimum Support

While experimenting with data, I performed the process with a minimum threshold support of 0.1%

```
rules_df.sort_values(by=['confidence'], ascending=False).drop(columns=['antecedent support', 'consequent support', 'conviction'])
```

	antecedents	consequents	support	confidence	lift	leverage
212	(eggs, whole milk, root vegetables)	(other vegetables)	0.001250	0.826087	6.425309	0.001055
211	(eggs, whole milk, other vegetables)	(root vegetables)	0.001250	0.655172	8.610676	0.001104
181	(chicken, root vegetables)	(other vegetables)	0.001578	0.648649	5.045193	0.001265
196	(white bread, root vegetables)	(other vegetables)	0.002696	0.611940	4.759675	0.002130
184	(eggs, root vegetables)	(other vegetables)	0.002762	0.591549	4.601073	0.002162
214	(whole milk, other vegetables, root vegetables)	(eggs)	0.001250	0.487179	9.799010	0.001122
213	(eggs, other vegetables, root vegetables)	(whole milk)	0.001250	0.452381	2.367139	0.000722
152	(root vegetables, beef)	(other vegetables)	0.003091	0.451923	3.515060	0.002212

Here we see that a very high confidence of 82% is achieved with a highly positive lift but a meagre support of 0.12%. We also found some other transactions with a lift as high as 51 which was ketchup and mustard that shows strong prediction of one item being available if the other is present.

I would like to say that 0.001 is a good minimum support threshold for this data as it gives reasonable insights with a good confidence level for a lot of itemsets as well as the number of itemsets produced and rules generated are manageable.

References:

Apriori Algorithm in Data Mining: Implementation With Examples (2023) *Software Testing Help*. Available at: https://www.softwaretestinghelp.com/apriori-algorithm/ (Accessed: 24 March 2023).

Li, S. (2017) *A Gentle Introduction on Market Basket Analysis — Association Rules*, *Medium*. Available at: https://towardsdatascience.com/a-gentle-introduction-on-market-basket-analysis-association-rules-fa4b986a40ce (Accessed: 24 March 2023).

Verma, Y. (2021) *Apriori vs FP-Growth in Market Basket Analysis - A Comparative Guide*, *Analytics India Magazine*. Available at: https://analyticsindiamag.com/apriori-vs-fp-growth-in-market-basket-analysis-a-comparative-guide/ (Accessed: 24 March 2023).