Association Rules

Association rules are used to discover interesting relationships or patterns between items in large datasets. These rules are often used in Market Basket Analysis to identify products that frequently co-occur in transactions. An association rule is typically expressed in the form $\{A\} \rightarrow \{B\}$, meaning that if item A is purchased, item B is likely to be purchased as well.

Source: Kaggle

https://www.kaggle.com/datasets/ahmtcnbs/datasets-for-appiori

```
!pip install apyori #Installing apriori library
```

```
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=622b3b53fc90929a9a6423f5b629e2cee4fbf14f41b25f7ec4ca
  Stored in directory: /root/.cache/pip/wheels/c4/1a/79/20f55c470a50bb3702a8cb7c94d8ada15573538c7f4baebe2d
Successfully built apyori
Installing collected packages: apyori
Successfully installed apyori-1.1.2
```

```
import numpy as np #NumPy is a powerful tool for numerical computations in Python.
import pandas as pd  #Pandas is a powerful library for data manipulation and analysis.
import seaborn as sns #Seaborn is a statistical data visualization library based on Matplotlib.
import matplotlib.pyplot as plt #Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Pytł
```

```
df = pd.read_csv('basket_analysis.csv')
```

```
df.head() #Displays the first 5 rows of the dataset.
```

	Unnamed: 0	Apple	Bread	Butter	Cheese	Corn	Dill	Eggs	Ice cream	Kidney Beans	Milk	Nutme
0	0	False	True	False	False	True	True	False	True	False	False	Fals
1	1	False	False	False	False	False	False	False	False	False	True	Fals
2	2	True	False	True	False	False	True	False	True	False	True	Fals
3	3	False	False	True	True	False	True	False	False	False	True	Tru

Next steps: Generate code with df View recommended plots

```
df.columns #Displays columns names of the dataset.
```

```
Index(['Unnamed: 0', 'Apple', 'Bread', 'Butter', 'Cheese', 'Corn', 'Dill',
       'Eggs', 'Ice cream', 'Kidney Beans', 'Milk', 'Nutmeg', 'Onion', 'Sugar',
       'Unicorn', 'Yogurt', 'chocolate'],
      dtype='object')
```

```
df.shape #Displays the total count of the Rows and Columns respectively.
```

```
(999, 17)
```

```
df = df.drop('Unnamed: 0', axis=1)
```

```
df.shape #Displays the total count of the Rows and Columns respectively.
```

```
(999, 16)
```

```
df.info() # Displays the total count of values present in the particular column along with the null count and data type.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 999 entries, 0 to 998
Data columns (total 16 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Apple         999 non-null    bool
 1   Bread         999 non-null    bool
 2   Butter        999 non-null    bool
 3   Cheese        999 non-null    bool
 4   Corn          999 non-null    bool
 5   Dill          999 non-null    bool
 6   Eggs          999 non-null    bool
 7   Ice cream     999 non-null    bool
 8   Kidney Beans  999 non-null    bool
 9   Milk          999 non-null    bool
 10  Nutmeg        999 non-null    bool
 11  Onion         999 non-null    bool
 12  Sugar         999 non-null    bool
 13  Unicorn       999 non-null    bool
 14  Yogurt        999 non-null    bool
 15  chocolate     999 non-null    bool
dtypes: bool(16)
memory usage: 15.7 KB
```

```
df.isnull().sum() # Displays the total count of the null values in the particular columns.
```

```
Apple           0
Bread           0
Butter          0
Cheese          0
Corn            0
Dill            0
Eggs            0
Ice cream       0
Kidney Beans    0
Milk            0
Nutmeg          0
Onion           0
Sugar           0
Unicorn         0
Yogurt          0
chocolate       0
dtype: int64
```

As we can check there is no Null value in the dataset.

```
from mlxtend.frequent_patterns import apriori, association_rules
apriori(df, min_support=0.15)[1:25]
```

	support	itemsets
1	0.384384	(1)
2	0.420420	(2)
3	0.404404	(3)
4	0.407407	(4)
5	0.398398	(5)
6	0.384384	(6)
7	0.410410	(7)
8	0.408408	(8)
9	0.405405	(9)
10	0.401401	(10)
11	0.403403	(11)
12	0.409409	(12)
13	0.389389	(13)
14	0.420420	(14)
15	0.421421	(15)
16	0.154154	(0, 1)
17	0.188188	(0, 2)
18	0.162162	(0, 3)
19	0.186186	(0, 4)
20	0.179179	(0, 5)
21	0.156156	(0, 6)
22	0.172172	(0, 7)
23	0.176176	(0, 8)
24	0.184184	(0, 9)

```
df.mean() #The df.mean() function in pandas is used to calculate the mean (average) value of each column in a DataFrame.
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell`
  and should_run_async(code)
Apple           0.383383
Bread           0.384384
Butter          0.420420
Cheese          0.404404
Corn            0.407407
Dill            0.398398
Eggs            0.384384
Ice cream       0.410410
Kidney Beans    0.408408
Milk            0.405405
Nutmeg          0.401401
Onion           0.403403
Sugar           0.409409
Unicorn         0.389389
Yogurt          0.420420
chocolate       0.421421
dtype: float64
```

Calculate Mean: For each numeric column in the DataFrame, it computes the mean value.

Return Result: It returns a pandas Series containing the mean values, with the column names as the index.

```
# Compute frequent itemsets using the Apriori algorithm
frequent_itemsets = apriori(df,
                            min_support = .006,
                            max_len = 3,
                            use_colnames = True)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipykernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell`
  and should_run_async(code)
```

Using the apriori function from the mlxtend library to perform Market Basket Analysis by finding frequent itemsets in the DataFrame df.

min_support = 0.006: The minimum support threshold for the itemsets to be considered frequent. Support is the proportion of transactions that contain the itemset.

max_len = 3: The maximum length (number of items) of the itemsets to be considered.

use_colnames = True: This indicates that the column names should be used to represent items in the itemsets.

```
frequent_itemsets
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipykernel.py:283: DeprecationWarning: `
  and should_run_async(code)
```

	support	itemsets
0	0.383383	(Apple)
1	0.384384	(Bread)
2	0.420420	(Butter)
3	0.404404	(Cheese)
4	0.407407	(Corn)
...	...	...
691	0.098098	(Yogurt, Onion, chocolate)
692	0.087087	(Yogurt, Sugar, Unicorn)
693	0.090090	(Sugar, Unicorn, chocolate)
694	0.095095	(Yogurt, Sugar, chocolate)
695	0.086086	(Yogurt, Unicorn, chocolate)

696 rows × 2 columns

Next steps: Generate code with frequent_itemsets View recommended plots

```
# Compute all association rules for frequent_itemsets
rules = association_rules(frequent_itemsets, #This is the input DataFrame containing the frequent itemsets that were previously generated us
                          metric = 'support', #This parameter specifies the metric used to evaluate the association rules. In this case, '
                          min_threshold=0.1) #This parameter sets the minimum threshold for the chosen metric. Only the rules that meet or
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipykernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell`
  and should_run_async(code)
```

Double-click (or enter) to edit

```
rules
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `
  and should_run_async(code)
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	le
0	(Bread)	(Apple)	0.384384	0.383383	0.154154	0.401042	1.046059	0.
1	(Apple)	(Bread)	0.383383	0.384384	0.154154	0.402089	1.046059	0.
2	(Apple)	(Butter)	0.383383	0.420420	0.188188	0.490862	1.167549	0.
3	(Butter)	(Apple)	0.420420	0.383383	0.188188	0.447619	1.167549	0.
4	(Apple)	(Cheese)	0.383383	0.404404	0.162162	0.422977	1.045925	0.
...	...	...	...	...	...	...	...	
433	(Yogurt, chocolate)	(Milk)	0.198198	0.405405	0.104104	0.525253	1.295623	0.
434	(Milk, chocolate)	(Yogurt)	0.211211	0.420420	0.104104	0.492891	1.172376	0.
435	(Yogurt)	(Milk, chocolate)	0.420420	0.211211	0.104104	0.247619	1.172376	0.
436	(Milk)	(Yogurt, chocolate)	0.405405	0.198198	0.104104	0.256790	1.295623	0.

Next steps: Generate code with `rules` View recommended plots

What the Code Does:

Apply Multiple Filtering Conditions:

The code applies all the specified filtering conditions to the rules DataFrame using the & (logical AND) operator. This ensures that only the rules meeting all the conditions are selected.

Create a New DataFrame filtered_rules:

The filtered rules are stored in a new DataFrame called filtered_rules.

```
filtered_rules = rules[(rules['antecedent support'] > 0.02)& #This condition filters rules where the support for the antecedent itemset is g
                       (rules['consequent support'] >0.01) & #This condition filters rules where the support for the consequent itemset is
                       (rules['confidence'] > 0.2)  #This condition filters rules where the confidence is greater than 0.2 (or 20%). Confic
                       (rules['lift'] > 1.0)] #This condition filters rules where the lift is greater than 1. Lift greater than 1.0 (indica
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell`
  and should_run_async(code)
```

Lift measures the strength of the association between the antecedent and the consequent. A lift value greater than 1 indicates that the antecedent and consequent occur together more frequently than would be expected if they were independent.

```
filtered_rules
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipykernel.py:283: DeprecationWarning: `
  and should_run_async(code)
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	le
0	(Bread)	(Apple)	0.384384	0.383383	0.154154	0.401042	1.046059	0.
1	(Apple)	(Bread)	0.383383	0.384384	0.154154	0.402089	1.046059	0.
2	(Apple)	(Butter)	0.383383	0.420420	0.188188	0.490862	1.167549	0.
3	(Butter)	(Apple)	0.420420	0.383383	0.188188	0.447619	1.167549	0.
4	(Apple)	(Cheese)	0.383383	0.404404	0.162162	0.422977	1.045925	0.
...	...	...	...	...	...	...	...	
433	(Yogurt, chocolate)	(Milk)	0.198198	0.405405	0.104104	0.525253	1.295623	0.
434	(Milk, chocolate)	(Yogurt)	0.211211	0.420420	0.104104	0.492891	1.172376	0.
435	(Yogurt)	(Milk, chocolate)	0.420420	0.211211	0.104104	0.247619	1.172376	0.
436	(Milk)	(Yogurt, chocolate)	0.405405	0.198198	0.104104	0.256790	1.295623	0.

Next steps: Generate code with filtered_rules View recommended plots

```
filtered_rules.sort_values('confidence',ascending=False)
```

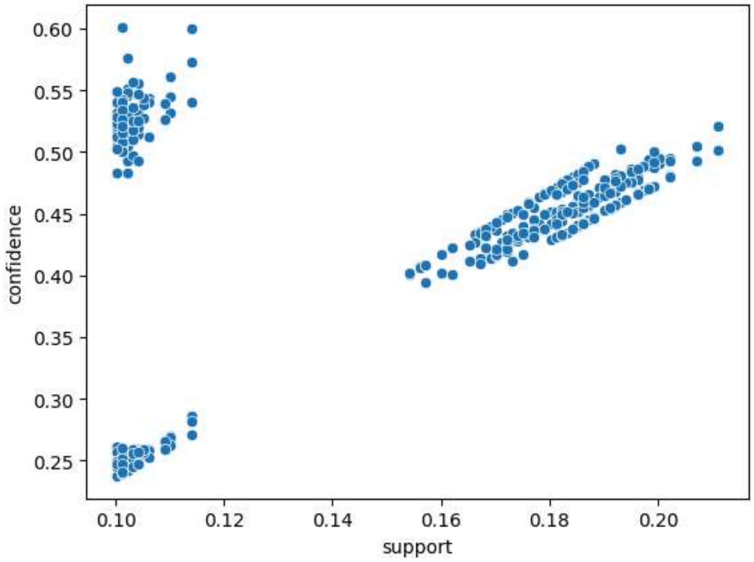
```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipykernel.py:283: DeprecationWarning: `
  and should_run_async(code)
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	le
402	(Dill, Unicorn)	(chocolate)	0.168168	0.421421	0.101101	0.601190	1.426578	0.
390	(Dill, Milk)	(chocolate)	0.190190	0.421421	0.114114	0.600000	1.423753	0.
325	(Dill, Cheese)	(Onion)	0.177177	0.403403	0.102102	0.576271	1.428523	0.
391	(Dill, chocolate)	(Milk)	0.199199	0.405405	0.114114	0.572864	1.413065	0.
258	(Kidney Beans, Ice cream)	(Butter)	0.196196	0.420420	0.110110	0.561224	1.334913	0.
...	...	...	...	...	...	...	...	
245	(Butter)	(Apple, Sugar)	0.420420	0.182182	0.100100	0.238095	1.306907	0.
322	(Yogurt)	(Nutmeg, Butter)	0.420420	0.198198	0.100100	0.238095	1.201299	0.

The filtered_rules.sort_values('confidence', ascending=False) code sorts the filtered association rules by the confidence metric in descending order. This allows you to easily identify the rules with the highest confidence, which can be interpreted as the most reliable or significant rules. Rules with higher confidence indicate a stronger association between the antecedent and consequent, making them more useful for decision-making and analysis.

```
# Generate scatterplot confidence versus support
sns.scatterplot(x = "support", y = "confidence", data = filtered_rules)
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipykernel.py:283: DeprecationWarning: `
  and should_run_async(code)
```



The purpose of this code is to filter the association rules in the rules DataFrame based on specified criteria for various metrics. This filtering process ensures that only the most significant and relevant association rules are retained for further analysis.

```
filtered_rules = rules[(rules['antecedent support'] > 0.02)& #Purpose: To ensure that the rules include antecedent itemsets that appear in m
                       (rules['consequent support'] >0.01) & #Purpose: To ensure that the rules include consequent itemsets that appear in
                       (rules['confidence'] > 0.45) &  #Purpose: To ensure that the rules have a confidence level greater than 0.45 (or 45%
                       (rules['lift'] > 1.0)] #Purpose: To ensure that the rules have a lift greater than 1.
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipykernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell`
  and should_run_async(code)
```

```
filtered_rules
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipykernel.py:283: DeprecationWarning: `
  and should_run_async(code)
```

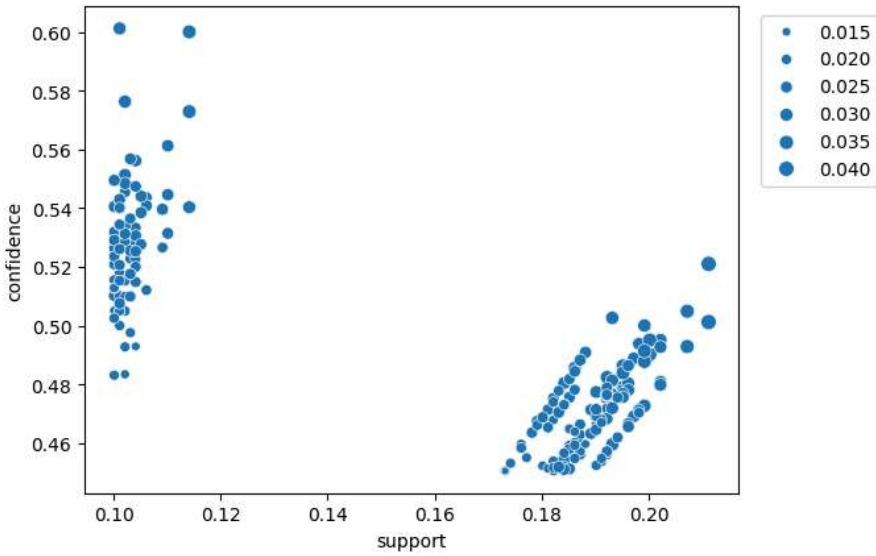
	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	le
2	(Apple)	(Butter)	0.383383	0.420420	0.188188	0.490862	1.167549	0.
6	(Apple)	(Corn)	0.383383	0.407407	0.186186	0.485640	1.192025	0.
7	(Corn)	(Apple)	0.407407	0.383383	0.186186	0.457002	1.192025	0.
9	(Apple)	(Dill)	0.383383	0.398398	0.179179	0.467363	1.173104	0.
14	(Apple)	(Kidney Beans)	0.383383	0.408408	0.176176	0.459530	1.125173	0.
...	...	...	...	...	...	...	...	
427	(Nutmeg, Yogurt)	(Kidney Beans)	0.192192	0.408408	0.101101	0.526042	1.288028	0.
428	(Yogurt, Kidney Beans)	(Nutmeg)	0.194194	0.401401	0.101101	0.520619	1.297002	0.
432	(Yogurt, Milk)	(chocolate)	0.190190	0.421421	0.104104	0.547368	1.298862	0.
	(Yogurt							

Next steps: Generate code with `filtered_rules` View recommended plots

```
# Generate scatterplot confidence versus support

sns.scatterplot(x = "support", y = "confidence", size= 'leverage',data = filtered_rules)
plt.legend(bbox_to_anchor= (1.02, 1), loc='upper left',)
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `
  and should_run_async(code)
```




```
# add extra another rule where support more than 0.2 for given itemset
filtered_rules = rules[(rules['antecedent support'] > 0.02)& #This ensures that the rules include antecedent itemsets that appear in more th
                        (rules['consequent support'] >0.01) & #This ensures that the rules include consequent itemsets that appear in more t
                        (rules['confidence'] > 0.45) & #This ensures that the rules have a confidence level greater than 0.45 (or 45%).
                        (rules['lift'] > 1.0)& #This ensures that the rules have a lift greater than 1.
                        (rules['support']>0.195)] #This ensures that the rules have a support value greater than 0.195 (or 19.5%).
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell`
  and should_run_async(code)
```

```
filtered_rules
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipykernel.py:283: DeprecationWarning: `
  and should_run_async(code)
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	le
66	(Ice cream)	(Butter)	0.410410	0.420420	0.207207	0.504878	1.200889	0.
67	(Butter)	(Ice cream)	0.420420	0.410410	0.207207	0.492857	1.200889	0.
68	(Kidney Beans)	(Butter)	0.408408	0.420420	0.202202	0.495098	1.177626	0.
69	(Butter)	(Kidney Beans)	0.420420	0.408408	0.202202	0.480952	1.177626	0.
70	(Milk)	(Butter)	0.405405	0.420420	0.198198	0.488889	1.162857	0.
71	(Butter)	(Milk)	0.420420	0.405405	0.198198	0.471429	1.162857	0.
72	(Nutmeg)	(Butter)	0.401401	0.420420	0.198198	0.493766	1.174457	0.
73	(Butter)	(Nutmeg)	0.420420	0.401401	0.198198	0.471429	1.174457	0.
74	(Onion)	(Butter)	0.403403	0.420420	0.197197	0.488834	1.162726	0.
75	(Butter)	(Onion)	0.420420	0.403403	0.197197	0.469048	1.162726	0.
76	(Sugar)	(Butter)	0.409409	0.420420	0.196196	0.479218	1.139853	0.
77	(Butter)	(Sugar)	0.420420	0.409409	0.196196	0.466667	1.139853	0.
82	(chocolate)	(Butter)	0.421421	0.420420	0.202202	0.479810	1.141262	0.
83	(Butter)	(chocolate)	0.420420	0.421421	0.202202	0.480952	1.141262	0.
92	(Kidney Beans)	(Cheese)	0.408408	0.404404	0.200200	0.490196	1.212143	0.
93	(Cheese)	(Kidney Beans)	0.404404	0.408408	0.200200	0.495050	1.212143	0.
114	(Corn)	(Kidney Beans)	0.407407	0.408408	0.195195	0.479115	1.173128	0.
209	(chocolate)	(Milk)	0.421421	0.405405	0.211211	0.501188	1.236263	0.

Next steps: Generate code with filtered_rules View recommended plots

```
def rules_to_coordinates(rules): #rules_to_coordinates is a function that takes a DataFrame rules as input.
    rules['antecedent'] = rules['antecedents'].apply(lambda antecedent:list(antecedent)[0])
    #rules['antecedent']: This creates a new column in the DataFrame called 'antecedent'.
    #rules['antecedents']: This column contains sets of antecedents for each rule, such as {('milk',)}.
    #.apply(lambda antecedent: list(antecedent)[0]): This lambda function takes each set from the 'antecedents' column, converts it to a lis

    rules['consequent'] = rules['consequents'].apply(lambda consequent:list(consequent)[0])
    #rules['consequent']: This creates a new column in the DataFrame called 'consequent'.
    #rules['consequents']: This column contains sets of consequents for each rule, such as {('bread',)}.
    #.apply(lambda consequent: list(consequent)[0]): This lambda function extracts the first item from each consequent set, converting it tc

    rules['rule'] = rules.index
    #rules['rule']: This creates a new column in the DataFrame called 'rule' that contains the index of each row.
    #rules.index: This refers to the index of the DataFrame, providing a unique identifier for each rule.

    return rules[['antecedent','consequent','rule']]
    #rules[['antecedent', 'consequent', 'rule']]: This returns a DataFrame with only the columns 'antecedent', 'consequent', and 'rule'.
    #This filtered DataFrame contains the essential information for each rule and is now ready for visualization or further processing.
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipykernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell`
  and should_run_async(code)
```

```
from pandas.plotting import parallel_coordinates
# Convert rules into coordinates suitable for use in a parallel coordinates plot
coords = rules_to_coordinates(filtered_rules)
# Generate parallel coordinates plot
plt.figure(figsize=(3,6))
parallel_coordinates(coords, 'rule',colormap = 'ocean')
plt.legend([])
plt.show()
```