Assignment 4 - Helper Functions

Apriori

We begin by including the functions to generate frequent itemsets (via the Apriori algorithm) and resulting association rules:

```
# (c) 2016 Everaldo Aguiar & Reid Johnson
#
# Modified from:
# Marcel Caraciolo (https://gist.github.com/marcelcaraciolo/1423287)
#
# Functions to compute and extract association rules from a given
frequent itemset
# generated by the Apriori algorithm.
#
# The Apriori algorithm is defined by Agrawal and Srikant in:
# Fast algorithms for mining association rules
# Proc. 20th int. conf. very large data bases, VLDB. Vol. 1215. 1994
import csv
import numpy as np

def load_dataset(filename):
    '''Loads an example of market basket transactions from a provided
csv file.

    Returns: A list (database) of lists (transactions). Each element
of a transaction is
    an item.
    '''

    with open(filename,'r') as dest_f:
        data_iter = csv.reader(dest_f, delimiter = ',', quotechar =
'"')
        #data = [data for data in data_iter]
        data = [transaction for transaction in data_iter if
transaction]  # This ensures empty lines are ignored
        data_array = np.asarray(data)

    return data_array

def apriori(dataset, min_support=0.5, verbose=False):
    """Implements the Apriori algorithm.

    The Apriori algorithm will iteratively generate new candidate
    k-itemsets using the frequent (k-1)-itemsets found in the previous
```

```
    iteration.

    Parameters
    ----------
    dataset : list
        The dataset (a list of transactions) from which to generate
        candidate itemsets.

    min_support : float
        The minimum support threshold. Defaults to 0.5.

    Returns
    -------
    F : list
        The list of frequent itemsets.

    support_data : dict
        The support data for all candidate itemsets.

    References
    ----------
    .. [1] R. Agrawal, R. Srikant, "Fast Algorithms for Mining
Association
           Rules", 1994.

    """
    C1 = create_candidates(dataset)
    D = list(map(set, dataset))
    F1, support_data = support_prune(D, C1, min_support,
verbose=False) # prune candidate 1-itemsets
    F = [F1] # list of frequent itemsets; initialized to frequent 1-
itemsets
    k = 2 # the itemset cardinality
    while (len(F[k - 2]) > 0):
        Ck = apriori_gen(F[k-2], k) # generate candidate itemsets
        Fk, supK = support_prune(D, Ck, min_support) # prune candidate
itemsets
        support_data.update(supK) # update the support counts to
reflect pruning
        F.append(Fk) # add the pruned candidate itemsets to the list
of frequent itemsets
        k += 1

    if verbose:
        # Print a list of all the frequent itemsets.
        for kset in F:
            for item in kset:
                print("" \
                    + "{" \
                    + "".join(str(i) + ", " for i in
```

```
iter(item)).rstrip(', ') \
                    + "}" \
                    + ":  sup = " + str(round(support_data[item], 3)))

    return F, support_data

def create_candidates(dataset, verbose=False):
    """Creates a list of candidate 1-itemsets from a list of
transactions.

    Parameters
    ----------
    dataset : list
        The dataset (a list of transactions) from which to generate
candidate
        itemsets.

    Returns
    -------
    The list of candidate itemsets (c1) passed as a frozenset (a set
that is
    immutable and hashable).
    """
    c1 = [] # list of all items in the database of transactions
    for transaction in dataset:
        for item in transaction:
            if not [item] in c1:
                c1.append([item])
    c1.sort()

    if verbose:
        # Print a list of all the candidate items.
        print("" \
            + "{" \
            + "".join(str(i[0]) + ", " for i in iter(c1)).rstrip(', ')
\
            + "}")

    # Map c1 to a frozenset because it will be the key of a
dictionary.
    return list(map(frozenset, c1))

def support_prune(dataset, candidates, min_support, verbose=False):
    """Returns all candidate itemsets that meet a minimum support
threshold.

    By the apriori principle, if an itemset is frequent, then all of
its
    subsets must also be frequent. As a result, we can perform
support-based
```

```
    pruning to systematically control the exponential growth of
candidate
    itemsets. Thus, itemsets that do not meet the minimum support
level are
    pruned from the input list of itemsets (dataset).

    Parameters
    ----------
    dataset : list
        The dataset (a list of transactions) from which to generate
candidate
        itemsets.

    candidates : frozenset
        The list of candidate itemsets.

    min_support : float
        The minimum support threshold.

    Returns
    -------
    retlist : list
        The list of frequent itemsets.

    support_data : dict
        The support data for all candidate itemsets.
    """
    sscnt = {} # set for support counts
    for tid in dataset:
        for can in candidates:
            if can.issubset(tid):
                sscnt.setdefault(can, 0)
                sscnt[can] += 1

    num_items = float(len(dataset)) # total number of transactions in
the dataset
    retlist = [] # array for unpruned itemsets
    support_data = {} # set for support data for corresponding
itemsets
    for key in sscnt:
        # Calculate the support of itemset key.
        support = sscnt[key] / num_items
        if support >= min_support:
            retlist.insert(0, key)
        support_data[key] = support

    # Print a list of the pruned itemsets.
    if verbose:
        for kset in retlist:
            for item in kset:
```

```
                print("{" + str(item) + "}")
        print("")
        for key in sscnt:
            print("" \
                + "{" \
                + "".join([str(i) + ", " for i in
iter(key)]).rstrip(', ') \
                + "}" \
                + ":  sup = " + str(support_data[key]))

    return retlist, support_data

def apriori_gen(freq_sets, k):
    """Generates candidate itemsets (via the F_k-1 x F_k-1 method).

    This operation generates new candidate k-itemsets based on the
frequent
    (k-1)-itemsets found in the previous iteration. The candidate
generation
    procedure merges a pair of frequent (k-1)-itemsets only if their
first k-2
    items are identical.

    Parameters
    ----------
    freq_sets : list
        The list of frequent (k-1)-itemsets.

    k : integer
        The cardinality of the current itemsets being evaluated.

    Returns
    -------
    retlist : list
        The list of merged frequent itemsets.
    """
    retList = [] # list of merged frequent itemsets
    lenLk = len(freq_sets) # number of frequent itemsets
    for i in range(lenLk):
        for j in range(i+1, lenLk):
            a=list(freq_sets[i])
            b=list(freq_sets[j])
            a.sort()
            b.sort()
            F1 = a[:k-2] # first k-2 items of freq_sets[i]
            F2 = b[:k-2] # first k-2 items of freq_sets[j]

            if F1 == F2: # if the first k-2 items are identical
                # Merge the frequent itemsets.
                retList.append(freq_sets[i] | freq_sets[j])
```

```
    return retList

def rules_from_conseq(freq_set, H, support_data, rules,
min_confidence=0.5, verbose=False):
    """Generates a set of candidate rules.

    Parameters
    ----------
    freq_set : frozenset
        The complete list of frequent itemsets.

    H : list
        A list of frequent itemsets (of a particular length).

    support_data : dict
        The support data for all candidate itemsets.

    rules : list
        A potentially incomplete set of candidate rules above the
minimum
        confidence threshold.

    min_confidence : float
        The minimum confidence threshold. Defaults to 0.5.
    """
    m = len(H[0])
    if m == 1:
        Hmp1 = calc_confidence(freq_set, H, support_data, rules,
min_confidence, verbose)
    if (len(freq_set) > (m+1)):
        Hmp1 = apriori_gen(H, m+1) # generate candidate itemsets
        Hmp1 = calc_confidence(freq_set, Hmp1, support_data, rules,
min_confidence, verbose)
        if len(Hmp1) > 1:
            # If there are candidate rules above the minimum
confidence
            # threshold, recurse on the list of these candidate rules.
            rules_from_conseq(freq_set, Hmp1, support_data, rules,
min_confidence, verbose)

def calc_confidence(freq_set, H, support_data, rules,
min_confidence=0.5, verbose=False):
    """Evaluates the generated rules.

    One measurement for quantifying the goodness of association rules
is
    confidence. The confidence for a rule 'P implies H' (P -> H) is
defined as
    the support for P and H divided by the support for P
```

```
    (support (P|H) / support(P)), where the | symbol denotes the set
union
    (thus P|H means all the items in set P or in set H).

    To calculate the confidence, we iterate through the frequent
itemsets and
    associated support data. For each frequent itemset, we divide the
support
    of the itemset by the support of the antecedent (left-hand-side of
the
    rule).

    Parameters
    ----------
    freq_set : frozenset
        The complete list of frequent itemsets.

    H : list
        A list of frequent itemsets (of a particular length).

    min_support : float
        The minimum support threshold.

    rules : list
        A potentially incomplete set of candidate rules above the
minimum
        confidence threshold.

    min_confidence : float
        The minimum confidence threshold. Defaults to 0.5.

    Returns
    -------
    pruned_H : list
        The list of candidate rules above the minimum confidence
threshold.
    """
    pruned_H = [] # list of candidate rules above the minimum
confidence threshold
    for conseq in H: # iterate over the frequent itemsets
        conf = support_data[freq_set] / support_data[freq_set -
conseq]
        if conf >= min_confidence:
            rules.append((freq_set - conseq, conseq, conf))
            pruned_H.append(conseq)

            if verbose:
                print("" \
                    + "{" \
                    + "".join([str(i) + ", " for i in iter(freq_set-
```

```
conseq)]).rstrip(', ') \
                    + "}" \
                    + " ---> " \
                    + "{" \
                    + "".join([str(i) + ", " for i in
iter(conseq)]).rstrip(', ') \
                    + "}" \
                    + ":  conf = " + str(round(conf, 3)) \
                    + ", sup = " + str(round(support_data[freq_set],
3)))

    return pruned_H

def generate_rules(F, support_data, min_confidence=0.5, verbose=True):
    """Generates a set of candidate rules from a list of frequent
itemsets.

    For each frequent itemset, we calculate the confidence of using a
    particular item as the rule consequent (right-hand-side of the
rule). By
    testing and merging the remaining rules, we recursively create a
list of
    pruned rules.

    Parameters
    ----------
    F : list
        A list of frequent itemsets.

    support_data : dict
        The corresponding support data for the frequent itemsets (L).

    min_confidence : float
        The minimum confidence threshold. Defaults to 0.5.

    Returns
    -------
    rules : list
        The list of candidate rules above the minimum confidence
threshold.
    """
    rules = []
    for i in range(1, len(F)):
        for freq_set in F[i]:
            H1 = [frozenset([itemset]) for itemset in freq_set]
            if (i > 1):
                rules_from_conseq(freq_set, H1, support_data, rules,
min_confidence, verbose)
            else:
                calc_confidence(freq_set, H1, support_data, rules,
```

```
min_confidence, verbose)

    return rules
```

To load our dataset of grocery transactions, use the command below

```
import csv

def load_dataset(filename):
    transactions = []
    with open(filename, 'r', encoding='utf-8') as file:
        data_iter = csv.reader(file, delimiter=',', quotechar='"')
        for row in data_iter:
            # Convert each row to a set to handle variable-length
transactions
            transactions.append(set(row))
    return transactions

# Path to the grocery dataset CSV file
file_path = 'grocery.csv'

# Load the dataset
dataset = load_dataset(file_path)
D = list(map(set, dataset))

type(dataset)

list

dataset[0]

{'citrus fruit', 'margarine', 'ready soups', 'semi-finished bread'}

dataset[1]

{'coffee', 'tropical fruit', 'yogurt'}
```

D Contains that dataset in a set format (which excludes duplicated items and sorts them)

```
type(dataset[0])

set

D[0]

{'citrus fruit', 'margarine', 'ready soups', 'semi-finished bread'}
```

Complete the assignment below by making use of the provided funtions.

You may use the notebook file attached with lesson 3 as a reference

```
# Print out the rules
def print_rules(rules, support_data):
    for rule in rules:
        print(f"Rule: {rule[0]} -> {rule[1]}")
        print(f"Confidence: {rule[2]}")
        print(f"Support: {support_data[rule[0] | rule[1]]}")
        print("----------")

# Set the minimum support and confidence values as required for your
dataset
min_support = 0.05  # 5 Percent
min_confidence = 0.5  # 50 Percent

# Apply the Apriori algorithm to the dataset with the specified
minimum support
frequent_itemsets, support_data = apriori(dataset,
min_support=min_support, verbose=True)

# Generate the association rules from the frequent itemsets
rules = generate_rules(frequent_itemsets, support_data,
min_confidence=min_confidence, verbose=True)

print_rules(rules, support_data)
```

```
{domestic eggs}:  sup = 0.063
{whipped/sour cream}:  sup = 0.072
{pork}:  sup = 0.058
{napkins}:  sup = 0.052
{shopping bags}:  sup = 0.099
{brown bread}:  sup = 0.065
{sausage}:  sup = 0.094
{canned beer}:  sup = 0.078
{root vegetables}:  sup = 0.109
{pastry}:  sup = 0.089
{newspapers}:  sup = 0.08
{fruit/vegetable juice}:  sup = 0.072
{soda}:  sup = 0.174
{frankfurter}:  sup = 0.059
{beef}:  sup = 0.052
{curd}:  sup = 0.053
{bottled water}:  sup = 0.111
{bottled beer}:  sup = 0.081
```

```
{rolls/buns}:  sup = 0.184
{butter}:  sup = 0.055
{other vegetables}:  sup = 0.193
{pip fruit}:  sup = 0.076
{whole milk}:  sup = 0.256
{yogurt}:  sup = 0.14
{tropical fruit}:  sup = 0.105
{coffee}:  sup = 0.058
{margarine}:  sup = 0.059
{citrus fruit}:  sup = 0.083
{rolls/buns, whole milk}:  sup = 0.057
{yogurt, whole milk}:  sup = 0.056
{other vegetables, whole milk}:  sup = 0.075

# Set the minimum support and confidence values as required for your
dataset
min_support = 0.04  # 4 Percent
min_confidence = 0.5  # 50 Percent

# Apply the Apriori algorithm to the dataset with the specified
minimum support
frequent_itemsets, support_data = apriori(dataset,
min_support=min_support, verbose=True)

# Generate the association rules from the frequent itemsets
rules = generate_rules(frequent_itemsets, support_data,
min_confidence=min_confidence, verbose=True)

print_rules(rules, support_data)

{frozen vegetables}:  sup = 0.048
{domestic eggs}:  sup = 0.063
{whipped/sour cream}:  sup = 0.072
{pork}:  sup = 0.058
{napkins}:  sup = 0.052
{shopping bags}:  sup = 0.099
{brown bread}:  sup = 0.065
{sausage}:  sup = 0.094
{canned beer}:  sup = 0.078
{root vegetables}:  sup = 0.109
{pastry}:  sup = 0.089
{newspapers}:  sup = 0.08
{fruit/vegetable juice}:  sup = 0.072
{chicken}:  sup = 0.043
{soda}:  sup = 0.174
{frankfurter}:  sup = 0.059
{beef}:  sup = 0.052
{curd}:  sup = 0.053
{white bread}:  sup = 0.042
{chocolate}:  sup = 0.05
```

```
{bottled water}:  sup = 0.111
{bottled beer}:  sup = 0.081
{rolls/buns}:  sup = 0.184
{butter}:  sup = 0.055
{other vegetables}:  sup = 0.193
{pip fruit}:  sup = 0.076
{whole milk}:  sup = 0.256
{yogurt}:  sup = 0.14
{tropical fruit}:  sup = 0.105
{coffee}:  sup = 0.058
{margarine}:  sup = 0.059
{citrus fruit}:  sup = 0.083
{yogurt, other vegetables}:  sup = 0.043
{rolls/buns, whole milk}:  sup = 0.057
{soda, whole milk}:  sup = 0.04
{root vegetables, whole milk}:  sup = 0.049
{other vegetables, root vegetables}:  sup = 0.047
{tropical fruit, whole milk}:  sup = 0.042
{rolls/buns, other vegetables}:  sup = 0.043
{yogurt, whole milk}:  sup = 0.056
{other vegetables, whole milk}:  sup = 0.075

# Set the minimum support and confidence values as required for your
dataset
min_support = 0.03  # 3 Percent
min_confidence = 0.5  # 50 Percent

# Apply the Apriori algorithm to the dataset with the specified
minimum support
frequent_itemsets, support_data = apriori(dataset,
min_support=min_support, verbose=True)

# Generate the association rules from the frequent itemsets
rules = generate_rules(frequent_itemsets, support_data,
min_confidence=min_confidence, verbose=True)

print_rules(rules, support_data)

{onions}:  sup = 0.031
{specialty chocolate}:  sup = 0.03
{frozen vegetables}:  sup = 0.048
{domestic eggs}:  sup = 0.063
{dessert}:  sup = 0.037
{whipped/sour cream}:  sup = 0.072
{pork}:  sup = 0.058
{berries}:  sup = 0.033
{napkins}:  sup = 0.052
{hygiene articles}:  sup = 0.033
{hamburger meat}:  sup = 0.033
{shopping bags}:  sup = 0.099
```

```
{brown bread}:  sup = 0.065
{sausage}:  sup = 0.094
{canned beer}:  sup = 0.078
{waffles}:  sup = 0.038
{salty snack}:  sup = 0.038
{root vegetables}:  sup = 0.109
{pastry}:  sup = 0.089
{sugar}:  sup = 0.034
{newspapers}:  sup = 0.08
{fruit/vegetable juice}:  sup = 0.072
{chicken}:  sup = 0.043
{soda}:  sup = 0.174
{frankfurter}:  sup = 0.059
{beef}:  sup = 0.052
{curd}:  sup = 0.053
{white bread}:  sup = 0.042
{chocolate}:  sup = 0.05
{bottled water}:  sup = 0.111
{bottled beer}:  sup = 0.081
{UHT-milk}:  sup = 0.033
{rolls/buns}:  sup = 0.184
{butter}:  sup = 0.055
{other vegetables}:  sup = 0.193
{long life bakery product}:  sup = 0.037
{pip fruit}:  sup = 0.076
{cream cheese}:  sup = 0.04
{whole milk}:  sup = 0.256
{yogurt}:  sup = 0.14
{tropical fruit}:  sup = 0.105
{coffee}:  sup = 0.058
{margarine}:  sup = 0.059
{citrus fruit}:  sup = 0.083
{yogurt, other vegetables}:  sup = 0.043
{whole milk, pip fruit}:  sup = 0.03
{rolls/buns, whole milk}:  sup = 0.057
{rolls/buns, yogurt}:  sup = 0.034
{pastry, whole milk}:  sup = 0.033
{soda, whole milk}:  sup = 0.04
{soda, other vegetables}:  sup = 0.033
{whipped/sour cream, whole milk}:  sup = 0.032
{root vegetables, whole milk}:  sup = 0.049
{rolls/buns, sausage}:  sup = 0.031
{other vegetables, root vegetables}:  sup = 0.047
{rolls/buns, soda}:  sup = 0.038
{citrus fruit, whole milk}:  sup = 0.031
{tropical fruit, whole milk}:  sup = 0.042
{bottled water, whole milk}:  sup = 0.034
{tropical fruit, other vegetables}:  sup = 0.036
{rolls/buns, other vegetables}:  sup = 0.043
```

```
{yogurt, whole milk}:  sup = 0.056
{other vegetables, whole milk}:  sup = 0.075
```

```python
# Set the minimum support and confidence values as required for your
dataset
min_support = 0.02  # 2 Percent
min_confidence = 0.5  # 50 Percent

# Apply the Apriori algorithm to the dataset with the specified
minimum support
frequent_itemsets, support_data = apriori(dataset,
min_support=min_support, verbose=True)

# Generate the association rules from the frequent itemsets
rules = generate_rules(frequent_itemsets, support_data,
min_confidence=min_confidence, verbose=True)

print_rules(rules, support_data)
```

```
{meat}:  sup = 0.026
{sliced cheese}:  sup = 0.025
{onions}:  sup = 0.031
{frozen meals}:  sup = 0.028
{specialty chocolate}:  sup = 0.03
{frozen vegetables}:  sup = 0.048
{ice cream}:  sup = 0.025
{oil}:  sup = 0.028
{chewing gum}:  sup = 0.021
{ham}:  sup = 0.026
{cat food}:  sup = 0.023
{hard cheese}:  sup = 0.025
{misc. beverages}:  sup = 0.028
{domestic eggs}:  sup = 0.063
{dessert}:  sup = 0.037
{grapes}:  sup = 0.022
{whipped/sour cream}:  sup = 0.072
{pork}:  sup = 0.058
{berries}:  sup = 0.033
{napkins}:  sup = 0.052
{hygiene articles}:  sup = 0.033
{hamburger meat}:  sup = 0.033
{beverages}:  sup = 0.026
{shopping bags}:  sup = 0.099
{brown bread}:  sup = 0.065
{sausage}:  sup = 0.094
{canned beer}:  sup = 0.078
{waffles}:  sup = 0.038
{salty snack}:  sup = 0.038
{root vegetables}:  sup = 0.109
{candy}:  sup = 0.03
```

```
{pastry}:  sup = 0.089
{butter milk}:  sup = 0.028
{specialty bar}:  sup = 0.027
{sugar}:  sup = 0.034
{newspapers}:  sup = 0.08
{fruit/vegetable juice}:  sup = 0.072
{chicken}:  sup = 0.043
{soda}:  sup = 0.174
{frankfurter}:  sup = 0.059
{beef}:  sup = 0.052
{curd}:  sup = 0.053
{white bread}:  sup = 0.042
{chocolate}:  sup = 0.05
{bottled water}:  sup = 0.111
{bottled beer}:  sup = 0.081
{UHT-milk}:  sup = 0.033
{rolls/buns}:  sup = 0.184
{butter}:  sup = 0.055
{other vegetables}:  sup = 0.193
{long life bakery product}:  sup = 0.037
{pip fruit}:  sup = 0.076
{cream cheese}:  sup = 0.04
{whole milk}:  sup = 0.256
{yogurt}:  sup = 0.14
{tropical fruit}:  sup = 0.105
{coffee}:  sup = 0.058
{margarine}:  sup = 0.059
{citrus fruit}:  sup = 0.083
{yogurt, whipped/sour cream}:  sup = 0.021
{yogurt, other vegetables}:  sup = 0.043
{other vegetables, pip fruit}:  sup = 0.026
{other vegetables, pastry}:  sup = 0.023
{other vegetables, shopping bags}:  sup = 0.023
{sausage, other vegetables}:  sup = 0.027
{whole milk, bottled beer}:  sup = 0.02
{shopping bags, whole milk}:  sup = 0.025
{citrus fruit, other vegetables}:  sup = 0.029
{fruit/vegetable juice, whole milk}:  sup = 0.027
{frankfurter, whole milk}:  sup = 0.021
{newspapers, whole milk}:  sup = 0.027
{margarine, whole milk}:  sup = 0.024
{tropical fruit, pip fruit}:  sup = 0.02
{whole milk, pip fruit}:  sup = 0.03
{rolls/buns, whole milk}:  sup = 0.057
{beef, whole milk}:  sup = 0.021
{sausage, whole milk}:  sup = 0.03
{frozen vegetables, whole milk}:  sup = 0.02
{rolls/buns, pastry}:  sup = 0.021
{fruit/vegetable juice, other vegetables}:  sup = 0.021
```

```
{domestic eggs, other vegetables}:  sup = 0.022
{other vegetables, butter}:  sup = 0.02
{rolls/buns, yogurt}:  sup = 0.034
{bottled water, soda}:  sup = 0.029
{tropical fruit, soda}:  sup = 0.021
{yogurt, soda}:  sup = 0.027
{pastry, whole milk}:  sup = 0.033
{yogurt, root vegetables}:  sup = 0.026
{brown bread, whole milk}:  sup = 0.025
{domestic eggs, whole milk}:  sup = 0.03
{soda, pastry}:  sup = 0.021
{soda, whole milk}:  sup = 0.04
{soda, other vegetables}:  sup = 0.033
{pork, whole milk}:  sup = 0.022
{other vegetables, pork}:  sup = 0.022
{whipped/sour cream, whole milk}:  sup = 0.032
{other vegetables, whipped/sour cream}:  sup = 0.029
{root vegetables, whole milk}:  sup = 0.049
{rolls/buns, bottled water}:  sup = 0.024
{soda, shopping bags}:  sup = 0.025
{rolls/buns, sausage}:  sup = 0.031
{sausage, soda}:  sup = 0.024
{rolls/buns, tropical fruit}:  sup = 0.025
{tropical fruit, root vegetables}:  sup = 0.021
{other vegetables, root vegetables}:  sup = 0.047
{rolls/buns, root vegetables}:  sup = 0.024
{rolls/buns, soda}:  sup = 0.038
{yogurt, citrus fruit}:  sup = 0.022
{citrus fruit, whole milk}:  sup = 0.031
{tropical fruit, whole milk}:  sup = 0.042
{yogurt, bottled water}:  sup = 0.023
{bottled water, whole milk}:  sup = 0.034
{curd, whole milk}:  sup = 0.026
{tropical fruit, other vegetables}:  sup = 0.036
{bottled water, other vegetables}:  sup = 0.025
{rolls/buns, other vegetables}:  sup = 0.043
{yogurt, whole milk}:  sup = 0.056
{butter, whole milk}:  sup = 0.028
{other vegetables, whole milk}:  sup = 0.075
{tropical fruit, yogurt}:  sup = 0.029
{yogurt, other vegetables, whole milk}:  sup = 0.022
{other vegetables, root vegetables, whole milk}:  sup = 0.023
{yogurt, other vegetables} ---> {whole milk}:  conf = 0.513, sup = 
0.022
Rule: frozenset({'yogurt', 'other vegetables'}) -> frozenset({'whole 
milk'})
Confidence: 0.5128805620608898
Support: 0.02226741230299949
----------
```

```python
# Set the minimum support and confidence values as required for your
dataset
min_support = 0.01  # 1 Percent
min_confidence = 0.5  # 50 Percent

# Apply the Apriori algorithm to the dataset with the specified
minimum support
frequent_itemsets, support_data = apriori(dataset,
min_support=min_support, verbose=True)

# Generate the association rules from the frequent itemsets
rules = generate_rules(frequent_itemsets, support_data,
min_confidence=min_confidence, verbose=True)

print_rules(rules, support_data)
```

```
{roll products}:  sup = 0.01
{liquor}:  sup = 0.011
{mustard}:  sup = 0.012
{meat}:  sup = 0.026
{dish cleaner}:  sup = 0.01
{frozen fish}:  sup = 0.012
{cake bar}:  sup = 0.013
{soft cheese}:  sup = 0.017
{cling film/bags}:  sup = 0.011
{pasta}:  sup = 0.015
{sliced cheese}:  sup = 0.025
{white wine}:  sup = 0.019
{herbs}:  sup = 0.016
{onions}:  sup = 0.031
{canned vegetables}:  sup = 0.011
{frozen meals}:  sup = 0.028
{salt}:  sup = 0.011
{specialty chocolate}:  sup = 0.03
{flower (seeds)}:  sup = 0.01
{red/blush wine}:  sup = 0.019
{seasonal products}:  sup = 0.014
{frozen vegetables}:  sup = 0.048
{canned fish}:  sup = 0.015
{ice cream}:  sup = 0.025
{oil}:  sup = 0.028
{chewing gum}:  sup = 0.021
{pickled vegetables}:  sup = 0.018
{baking powder}:  sup = 0.018
{ham}:  sup = 0.026
{cat food}:  sup = 0.023
{hard cheese}:  sup = 0.025
{misc. beverages}:  sup = 0.028
{spread cheese}:  sup = 0.011
{domestic eggs}:  sup = 0.063
```

```
{dessert}:  sup = 0.037
{grapes}:  sup = 0.022
{whipped/sour cream}:  sup = 0.072
{pork}:  sup = 0.058
{berries}:  sup = 0.033
{napkins}:  sup = 0.052
{hygiene articles}:  sup = 0.033
{hamburger meat}:  sup = 0.033
{beverages}:  sup = 0.026
{shopping bags}:  sup = 0.099
{brown bread}:  sup = 0.065
{sausage}:  sup = 0.094
{canned beer}:  sup = 0.078
{waffles}:  sup = 0.038
{salty snack}:  sup = 0.038
{root vegetables}:  sup = 0.109
{frozen dessert}:  sup = 0.011
{candy}:  sup = 0.03
{processed cheese}:  sup = 0.017
{detergent}:  sup = 0.019
{pastry}:  sup = 0.089
{butter milk}:  sup = 0.028
{specialty bar}:  sup = 0.027
{packaged fruit/vegetables}:  sup = 0.013
{sugar}:  sup = 0.034
{newspapers}:  sup = 0.08
{fruit/vegetable juice}:  sup = 0.072
{chicken}:  sup = 0.043
{soda}:  sup = 0.174
{frankfurter}:  sup = 0.059
{beef}:  sup = 0.052
{flour}:  sup = 0.017
{dishes}:  sup = 0.018
{curd}:  sup = 0.053
{white bread}:  sup = 0.042
{chocolate}:  sup = 0.05
{bottled water}:  sup = 0.111
{pot plants}:  sup = 0.017
{bottled beer}:  sup = 0.081
{UHT-milk}:  sup = 0.033
{rolls/buns}:  sup = 0.184
{butter}:  sup = 0.055
{other vegetables}:  sup = 0.193
{long life bakery product}:  sup = 0.037
{condensed milk}:  sup = 0.01
{pip fruit}:  sup = 0.076
{cream cheese}:  sup = 0.04
{whole milk}:  sup = 0.256
{yogurt}:  sup = 0.14
```

```
{tropical fruit}:  sup = 0.105
{coffee}:  sup = 0.058
{semi-finished bread}:  sup = 0.018
{margarine}:  sup = 0.059
{citrus fruit}:  sup = 0.083
{margarine, soda}:  sup = 0.01
{newspapers, root vegetables}:  sup = 0.011
{white bread, soda}:  sup = 0.01
{rolls/buns, coffee}:  sup = 0.011
{yogurt, beef}:  sup = 0.012
{pastry, pip fruit}:  sup = 0.011
{chicken, whole milk}:  sup = 0.018
{margarine, yogurt}:  sup = 0.014
{soda, pip fruit}:  sup = 0.013
{tropical fruit, napkins}:  sup = 0.01
{onions, whole milk}:  sup = 0.012
{other vegetables, onions}:  sup = 0.014
{soda, citrus fruit}:  sup = 0.013
{other vegetables, newspapers}:  sup = 0.019
{bottled water, pip fruit}:  sup = 0.011
{sausage, bottled water}:  sup = 0.012
{soda, napkins}:  sup = 0.012
{shopping bags, root vegetables}:  sup = 0.013
{beef, other vegetables}:  sup = 0.02
{butter milk, other vegetables}:  sup = 0.01
{pork, root vegetables}:  sup = 0.014
{tropical fruit, shopping bags}:  sup = 0.014
{other vegetables, cream cheese}:  sup = 0.014
{yogurt, frankfurter}:  sup = 0.011
{root vegetables, pip fruit}:  sup = 0.016
{tropical fruit, whipped/sour cream}:  sup = 0.014
{yogurt, whipped/sour cream}:  sup = 0.021
{sliced cheese, whole milk}:  sup = 0.011
{bottled water, shopping bags}:  sup = 0.011
{white bread, whole milk}:  sup = 0.017
{frozen vegetables, yogurt}:  sup = 0.012
{yogurt, other vegetables}:  sup = 0.043
{rolls/buns, citrus fruit}:  sup = 0.017
{other vegetables, pip fruit}:  sup = 0.026
{citrus fruit, pip fruit}:  sup = 0.014
{fruit/vegetable juice, citrus fruit}:  sup = 0.01
{sausage, citrus fruit}:  sup = 0.011
{tropical fruit, sausage}:  sup = 0.014
{sausage, pip fruit}:  sup = 0.011
{hygiene articles, whole milk}:  sup = 0.013
{hard cheese, whole milk}:  sup = 0.01
{coffee, other vegetables}:  sup = 0.013
{other vegetables, pastry}:  sup = 0.023
{other vegetables, shopping bags}:  sup = 0.023
```

```
{shopping bags, pastry}:  sup = 0.012
{whole milk, cream cheese}:  sup = 0.016
{whole milk, hamburger meat}:  sup = 0.015
{margarine, bottled water}:  sup = 0.01
{other vegetables, frankfurter}:  sup = 0.016
{frankfurter, root vegetables}:  sup = 0.01
{sausage, other vegetables}:  sup = 0.027
{sausage, frankfurter}:  sup = 0.01
{beef, root vegetables}:  sup = 0.017
{whole milk, bottled beer}:  sup = 0.02
{fruit/vegetable juice, yogurt}:  sup = 0.019
{yogurt, shopping bags}:  sup = 0.015
{shopping bags, whole milk}:  sup = 0.025
{yogurt, napkins}:  sup = 0.012
{napkins, whole milk}:  sup = 0.02
{rolls/buns, pork}:  sup = 0.011
{citrus fruit, other vegetables}:  sup = 0.029
{other vegetables, chicken}:  sup = 0.018
{citrus fruit, root vegetables}:  sup = 0.018
{chicken, root vegetables}:  sup = 0.011
{fruit/vegetable juice, sausage}:  sup = 0.01
{sausage, brown bread}:  sup = 0.011
{citrus fruit, whipped/sour cream}:  sup = 0.011
{sausage, yogurt}:  sup = 0.02
{sausage, root vegetables}:  sup = 0.015
{rolls/buns, margarine}:  sup = 0.015
{fruit/vegetable juice, whole milk}:  sup = 0.027
{fruit/vegetable juice, rolls/buns}:  sup = 0.015
{margarine, root vegetables}:  sup = 0.011
{fruit/vegetable juice, root vegetables}:  sup = 0.012
{rolls/buns, domestic eggs}:  sup = 0.016
{rolls/buns, frozen vegetables}:  sup = 0.01
{frozen vegetables, root vegetables}:  sup = 0.012
{soda, bottled beer}:  sup = 0.017
{bottled water, bottled beer}:  sup = 0.016
{fruit/vegetable juice, bottled water}:  sup = 0.014
{butter milk, whole milk}:  sup = 0.012
{frankfurter, whole milk}:  sup = 0.021
{newspapers, whole milk}:  sup = 0.027
{domestic eggs, citrus fruit}:  sup = 0.01
{oil, whole milk}:  sup = 0.011
{margarine, whole milk}:  sup = 0.024
{tropical fruit, pip fruit}:  sup = 0.02
{whole milk, pip fruit}:  sup = 0.03
{rolls/buns, whole milk}:  sup = 0.057
{rolls/buns, pip fruit}:  sup = 0.014
{chocolate, whole milk}:  sup = 0.017
{rolls/buns, brown bread}:  sup = 0.013
{beef, whole milk}:  sup = 0.021
```

```
{sausage, whole milk}:  sup = 0.03
{sausage, pastry}:  sup = 0.013
{salty snack, whole milk}:  sup = 0.011
{frozen vegetables, whole milk}:  sup = 0.02
{frozen vegetables, other vegetables}:  sup = 0.018
{rolls/buns, pastry}:  sup = 0.021
{rolls/buns, beef}:  sup = 0.014
{margarine, other vegetables}:  sup = 0.02
{fruit/vegetable juice, tropical fruit}:  sup = 0.014
{fruit/vegetable juice, other vegetables}:  sup = 0.021
{brown bread, other vegetables}:  sup = 0.019
{domestic eggs, other vegetables}:  sup = 0.022
{ham, whole milk}:  sup = 0.011
{other vegetables, butter}:  sup = 0.02
{rolls/buns, butter}:  sup = 0.013
{curd, other vegetables}:  sup = 0.017
{curd, rolls/buns}:  sup = 0.01
{butter, root vegetables}:  sup = 0.013
{bottled water, root vegetables}:  sup = 0.016
{curd, root vegetables}:  sup = 0.011
{whipped/sour cream, butter}:  sup = 0.01
{rolls/buns, whipped/sour cream}:  sup = 0.015
{curd, whipped/sour cream}:  sup = 0.01
{whipped/sour cream, root vegetables}:  sup = 0.017
{rolls/buns, yogurt}:  sup = 0.034
{bottled water, soda}:  sup = 0.029
{yogurt, newspapers}:  sup = 0.015
{rolls/buns, newspapers}:  sup = 0.02
{bottled water, newspapers}:  sup = 0.011
{yogurt, berries}:  sup = 0.011
{coffee, whole milk}:  sup = 0.019
{tropical fruit, soda}:  sup = 0.021
{yogurt, soda}:  sup = 0.027
{tropical fruit, pastry}:  sup = 0.013
{yogurt, pastry}:  sup = 0.018
{pastry, whole milk}:  sup = 0.033
{yogurt, root vegetables}:  sup = 0.026
{soda, root vegetables}:  sup = 0.019
{pastry, root vegetables}:  sup = 0.011
{waffles, whole milk}:  sup = 0.013
{tropical fruit, brown bread}:  sup = 0.011
{yogurt, brown bread}:  sup = 0.015
{brown bread, whole milk}:  sup = 0.025
{brown bread, root vegetables}:  sup = 0.01
{tropical fruit, domestic eggs}:  sup = 0.011
{yogurt, domestic eggs}:  sup = 0.014
{domestic eggs, whole milk}:  sup = 0.03
{domestic eggs, soda}:  sup = 0.012
{domestic eggs, root vegetables}:  sup = 0.014
```

```
{rolls/buns, canned beer}:  sup = 0.011
{rolls/buns, shopping bags}:  sup = 0.02
{sausage, shopping bags}:  sup = 0.016
{dessert, whole milk}:  sup = 0.014
{dessert, other vegetables}:  sup = 0.012
{soda, pastry}:  sup = 0.021
{soda, whole milk}:  sup = 0.04
{soda, other vegetables}:  sup = 0.033
{berries, whole milk}:  sup = 0.012
{other vegetables, berries}:  sup = 0.01
{pork, whole milk}:  sup = 0.022
{other vegetables, pork}:  sup = 0.022
{soda, pork}:  sup = 0.012
{whipped/sour cream, whole milk}:  sup = 0.032
{other vegetables, whipped/sour cream}:  sup = 0.029
{soda, whipped/sour cream}:  sup = 0.012
{sugar, whole milk}:  sup = 0.015
{sugar, other vegetables}:  sup = 0.011
{root vegetables, whole milk}:  sup = 0.049
{rolls/buns, bottled water}:  sup = 0.024
{other vegetables, hamburger meat}:  sup = 0.014
{other vegetables, napkins}:  sup = 0.014
{rolls/buns, napkins}:  sup = 0.012
{fruit/vegetable juice, soda}:  sup = 0.018
{soda, newspapers}:  sup = 0.015
{soda, canned beer}:  sup = 0.014
{brown bread, soda}:  sup = 0.013
{soda, shopping bags}:  sup = 0.025
{fruit/vegetable juice, shopping bags}:  sup = 0.011
{canned beer, shopping bags}:  sup = 0.011
{rolls/buns, chocolate}:  sup = 0.012
{soda, chocolate}:  sup = 0.014
{rolls/buns, sausage}:  sup = 0.031
{sausage, soda}:  sup = 0.024
{rolls/buns, tropical fruit}:  sup = 0.025
{tropical fruit, root vegetables}:  sup = 0.021
{other vegetables, root vegetables}:  sup = 0.047
{rolls/buns, root vegetables}:  sup = 0.024
{salty snack, other vegetables}:  sup = 0.011
{other vegetables, waffles}:  sup = 0.01
{tropical fruit, newspapers}:  sup = 0.012
{rolls/buns, frankfurter}:  sup = 0.019
{rolls/buns, soda}:  sup = 0.038
{soda, frankfurter}:  sup = 0.011
{tropical fruit, citrus fruit}:  sup = 0.02
{yogurt, citrus fruit}:  sup = 0.022
{citrus fruit, whole milk}:  sup = 0.031
{tropical fruit, whole milk}:  sup = 0.042
{bottled water, citrus fruit}:  sup = 0.014
```

```
{yogurt, bottled water}:  sup = 0.023
{bottled water, whole milk}:  sup = 0.034
{curd, tropical fruit}:  sup = 0.01
{curd, yogurt}:  sup = 0.017
{curd, whole milk}:  sup = 0.026
{tropical fruit, other vegetables}:  sup = 0.036
{tropical fruit, bottled water}:  sup = 0.019
{bottled water, other vegetables}:  sup = 0.025
{other vegetables, chocolate}:  sup = 0.013
{white bread, other vegetables}:  sup = 0.014
{rolls/buns, other vegetables}:  sup = 0.043
{other vegetables, bottled beer}:  sup = 0.016
{rolls/buns, bottled beer}:  sup = 0.014
{yogurt, whole milk}:  sup = 0.056
{yogurt, butter}:  sup = 0.015
{butter, whole milk}:  sup = 0.028
{long life bakery product, whole milk}:  sup = 0.014
{other vegetables, whole milk}:  sup = 0.075
{other vegetables, long life bakery product}:  sup = 0.011
{yogurt, cream cheese}:  sup = 0.012
{yogurt, pip fruit}:  sup = 0.018
{tropical fruit, yogurt}:  sup = 0.029
{other vegetables, pastry, whole milk}:  sup = 0.011
{rolls/buns, yogurt, other vegetables}:  sup = 0.011
{bottled water, other vegetables, whole milk}:  sup = 0.011
{rolls/buns, yogurt, whole milk}:  sup = 0.016
{citrus fruit, other vegetables, whole milk}:  sup = 0.013
{sausage, other vegetables, whole milk}:  sup = 0.01
{other vegetables, butter, whole milk}:  sup = 0.011
{tropical fruit, yogurt, other vegetables}:  sup = 0.012
{yogurt, whipped/sour cream, other vegetables}:  sup = 0.01
{yogurt, whipped/sour cream, whole milk}:  sup = 0.011
{tropical fruit, other vegetables, whole milk}:  sup = 0.017
{fruit/vegetable juice, other vegetables, whole milk}:  sup = 0.01
{yogurt, other vegetables, whole milk}:  sup = 0.022
{yogurt, other vegetables, root vegetables}:  sup = 0.013
{whole milk, other vegetables, pip fruit}:  sup = 0.014
{domestic eggs, other vegetables, whole milk}:  sup = 0.012
{citrus fruit, other vegetables, root vegetables}:  sup = 0.01
{rolls/buns, root vegetables, whole milk}:  sup = 0.013
{rolls/buns, other vegetables, whole milk}:  sup = 0.018
{rolls/buns, tropical fruit, whole milk}:  sup = 0.011
{tropical fruit, root vegetables, whole milk}:  sup = 0.012
{yogurt, root vegetables, whole milk}:  sup = 0.015
{yogurt, soda, whole milk}:  sup = 0.01
{other vegetables, whipped/sour cream, whole milk}:  sup = 0.015
{other vegetables, pork, whole milk}:  sup = 0.01
{soda, other vegetables, whole milk}:  sup = 0.014
{other vegetables, root vegetables, whole milk}:  sup = 0.023
```

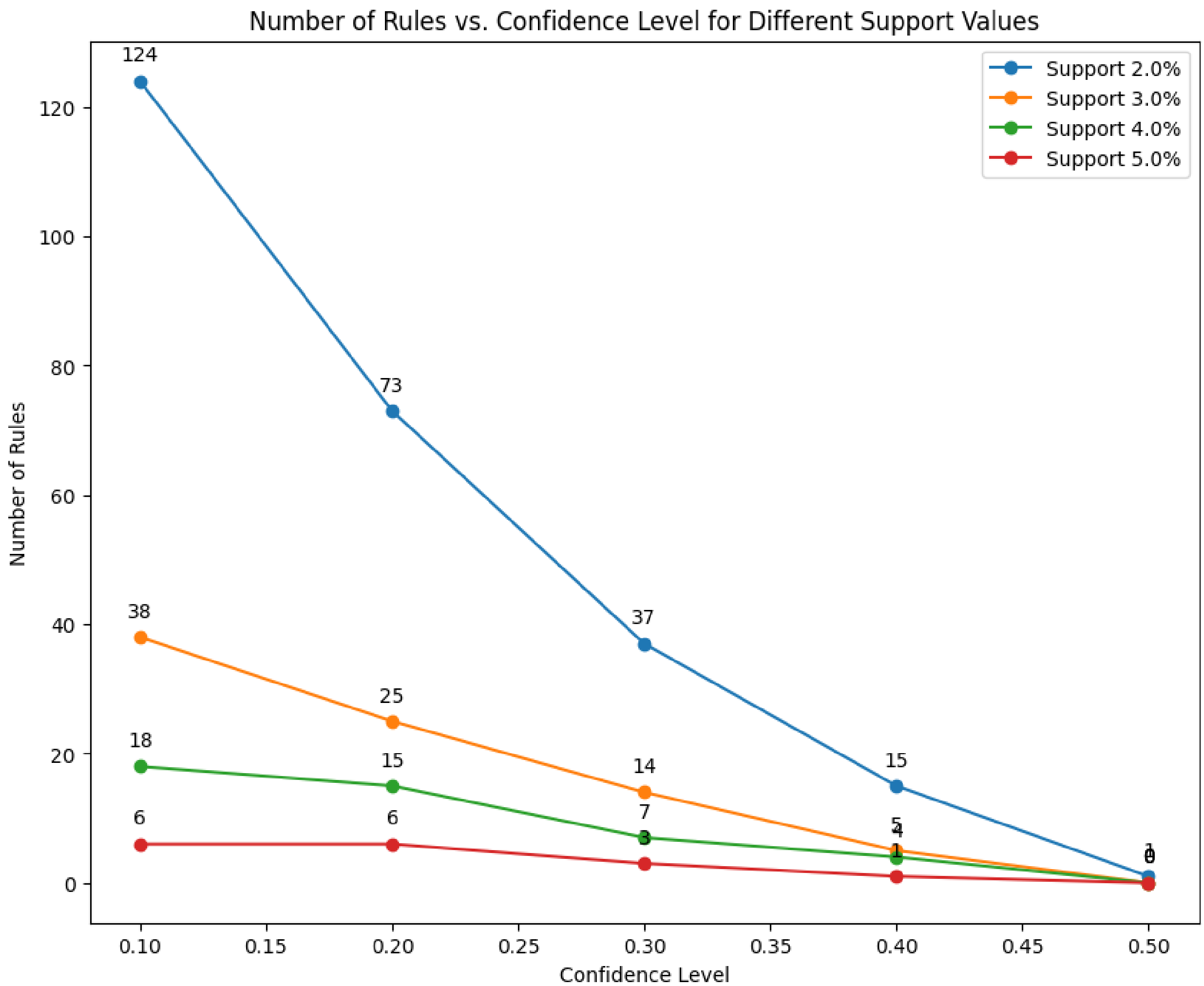
```
{rolls/buns, other vegetables, root vegetables}:  sup = 0.012
{tropical fruit, other vegetables, root vegetables}:  sup = 0.012
{curd, yogurt, whole milk}:  sup = 0.01
{tropical fruit, yogurt, whole milk}:  sup = 0.015
{yogurt, citrus fruit, whole milk}:  sup = 0.01
{other vegetables, butter} ---> {whole milk}:  conf = 0.574, sup =
0.011
{yogurt, whipped/sour cream} ---> {whole milk}:  conf = 0.525, sup =
0.011
{yogurt, other vegetables} ---> {whole milk}:  conf = 0.513, sup =
0.022
{yogurt, root vegetables} ---> {other vegetables}:  conf = 0.5, sup =
0.013
{other vegetables, pip fruit} ---> {whole milk}:  conf = 0.518, sup =
0.014
{domestic eggs, other vegetables} ---> {whole milk}:  conf = 0.553,
sup = 0.012
{citrus fruit, root vegetables} ---> {other vegetables}:  conf =
0.586, sup = 0.01
{rolls/buns, root vegetables} ---> {whole milk}:  conf = 0.523, sup =
0.013
{tropical fruit, root vegetables} ---> {whole milk}:  conf = 0.57, sup
= 0.012
{yogurt, root vegetables} ---> {whole milk}:  conf = 0.563, sup =
0.015
{whipped/sour cream, other vegetables} ---> {whole milk}:  conf =
0.507, sup = 0.015
{rolls/buns, root vegetables} ---> {other vegetables}:  conf = 0.502,
sup = 0.012
{tropical fruit, root vegetables} ---> {other vegetables}:  conf =
0.585, sup = 0.012
{curd, yogurt} ---> {whole milk}:  conf = 0.582, sup = 0.01
{tropical fruit, yogurt} ---> {whole milk}:  conf = 0.517, sup = 0.015
Rule: frozenset({'other vegetables', 'butter'}) -> frozenset({'whole
milk'})
Confidence: 0.5736040609137055
Support: 0.011489578037620742
----------
Rule: frozenset({'yogurt', 'whipped/sour cream'}) -> frozenset({'whole
milk'})
Confidence: 0.5245098039215685
Support: 0.010879511947127605
----------
Rule: frozenset({'yogurt', 'other vegetables'}) -> frozenset({'whole
milk'})
Confidence: 0.5128805620608898
Support: 0.022267412302999490
----------
Rule: frozenset({'yogurt', 'root vegetables'}) -> frozenset({'other
```

```
vegetables'})
Confidence: 0.5
Support: 0.012913065582104729
----------
Rule: frozenset({'other vegetables', 'pip fruit'}) ->
frozenset({'whole milk'})
Confidence: 0.5175097276264592
Support: 0.013523131672597865
----------
Rule: frozenset({'domestic eggs', 'other vegetables'}) ->
frozenset({'whole milk'})
Confidence: 0.5525114155251142
Support: 0.012302999491611592
----------
Rule: frozenset({'citrus fruit', 'root vegetables'}) ->
frozenset({'other vegetables'})
Confidence: 0.5862068965517241
Support: 0.010371123538383325
----------
Rule: frozenset({'rolls/buns', 'root vegetables'}) ->
frozenset({'whole milk'})
Confidence: 0.5230125523012552
Support: 0.012709710218607015
----------
Rule: frozenset({'tropical fruit', 'root vegetables'}) ->
frozenset({'whole milk'})
Confidence: 0.570048309178744
Support: 0.011997966446365024
----------
Rule: frozenset({'yogurt', 'root vegetables'}) -> frozenset({'whole
milk'})
Confidence: 0.562992125984252
Support: 0.014539908490086425
----------
Rule: frozenset({'whipped/sour cream', 'other vegetables'}) ->
frozenset({'whole milk'})
Confidence: 0.5070422535211268
Support: 0.014641586171835282
----------
Rule: frozenset({'rolls/buns', 'root vegetables'}) ->
frozenset({'other vegetables'})
Confidence: 0.502092050209205
Support: 0.012201321809862735
----------
Rule: frozenset({'tropical fruit', 'root vegetables'}) ->
frozenset({'other vegetables'})
Confidence: 0.5845410628019324
Support: 0.012302999491611592
----------
```

```
Rule: frozenset({'curd', 'yogurt'}) -> frozenset({'whole milk'})
Confidence: 0.5823529411764706
Support: 0.010066090493136757
----------
Rule: frozenset({'tropical fruit', 'yogurt'}) -> frozenset({'whole
milk'})
Confidence: 0.5173611111111111
Support: 0.015149974580579562
----------
```

```
import matplotlib.pyplot as plt

# Load the dataset
dataset = load_dataset('grocery.csv')

# Define the support and confidence levels to test
support_levels = [0.02, 0.03, 0.04, 0.05]
confidence_levels = [0.1, 0.2, 0.3, 0.4, 0.5]

# Prepare the figure
plt.figure(figsize=(10, 8))
plt.title('Number of Rules vs. Confidence Level for Different Support
Values')
plt.xlabel('Confidence Level')
plt.ylabel('Number of Rules')

# For each support level, generate rules and count them for each
confidence level
for support in support_levels:
    rule_counts = []  # to hold the count of rules for each confidence
level

    # Get the frequent itemsets and support data for the current
support level
    frequent_itemsets, support_data = apriori(dataset,
min_support=support, verbose=False)

    for confidence in confidence_levels:
        # Generate rules using the frequent itemsets and support data
for the current support level
        rules = generate_rules(frequent_itemsets, support_data,
min_confidence=confidence, verbose=False)
        rule_count = len(rules)
        rule_counts.append(len(rules))

        # Annotate the data points with the exact number of rules
        plt.annotate(f'{rule_count}', (confidence, rule_count),
textcoords="offset points", xytext=(0,10), ha='center')

    # Plot the results
```

```
    plt.plot(confidence_levels, rule_counts, marker='o',
label=f'Support {support*100}%')

# Add legend
plt.legend()

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



The outputs of the Apriori algorithm show the variation of minimum support values on the number of frequent itemsets. A lower minimum support value results in more number of itemsets being considered as frequent. This means that more patterns could be found, but it also refers to less common ones that are to be included.

Keeping the minimum confidence constant at 0.5 helped us in focusing on reliable rules. This is because if the value of 0.5 confidence says that half of the time one item is bought, the other is too. It helps in knowing if there are strong links among items without obtaining too many weak associations.

We also observed that 'whole milk', 'rolls/buns' and 'other vegetables' seem to occur again and again. It does show that maybe they are all daily staples in many shopping baskets. Rules involving these items show their importance in customers' purchases.

Lowering min_support also extracted specific rules, such as `{yogurt, whipped/sour cream} -> {whole milk}`. Such insights would assist the store to plan promotions or re-stacking to encourage buying of these items together.

All the items and rules we found show differences in eating habits and preferences. Items such as 'whole milk' and 'yogurt' being common suggest the consumption of dairy products in the region of this dataset.

This exercise is just an example showing how crucial tuning the parameters of the Apriori algorithm is in order to retrieve some useful patterns. It involves some balance between too many rules and really interesting patterns omitted - something that is quite often of high importance in practical analysis.

In summary, shopping behavior analysis with the Apriori algorithm availed an enlightening view of patterns and associations. This demonstrates how data mining reveals hidden relations between the items in the data, helping the business strategies in responding better to the need of the customer.

FPgrowth

```
# (c) 2014 Reid Johnson
#
# Modified from:
# Eric Naeseth <eric@naeseth.com>
# (https://github.com/enaeseth/python-fp-growth/blob/master/fp_growth.py)
#
# A Python implementation of the FP-growth algorithm.

from collections import defaultdict, namedtuple
#from itertools import imap

__author__ = 'Eric Naeseth <eric@naeseth.com>'
__copyright__ = 'Copyright © 2009 Eric Naeseth'
__license__ = 'MIT License'

def fpgrowth(dataset, min_support=0.5, include_support=True, verbose=False):
    """Implements the FP-growth algorithm.

    The `dataset` parameter can be any iterable of iterables of items.
    `min_support` should be an integer specifying the minimum number of
    occurrences of an itemset for it to be accepted.
```

```
    Each item must be hashable (i.e., it must be valid as a member of
a
    dictionary or a set).

    If `include_support` is true, yield (itemset, support) pairs
instead of
    just the itemsets.

    Parameters
    ----------
    dataset : list
        The dataset (a list of transactions) from which to generate
        candidate itemsets.

    min_support : float
        The minimum support threshold. Defaults to 0.5.

    include_support : bool
        Include support in output (default=False).

    References
    ----------
    .. [1] J. Han, J. Pei, Y. Yin, "Mining Frequent Patterns without
Candidate
           Generation," 2000.

    """

    F = []
    support_data = {}
    for k,v in find_frequent_itemsets(dataset,
min_support=min_support, include_support=include_support,
verbose=verbose):
        F.append(frozenset(k))
        support_data[frozenset(k)] = v

    # Create one array with subarrays that hold all transactions of
equal length.
    def bucket_list(nested_list, sort=True):
        bucket = defaultdict(list)
        for sublist in nested_list:
            bucket[len(sublist)].append(sublist)
        return [v for k,v in sorted(bucket.items())] if sort else
bucket.values()

    F = bucket_list(F)

    return F, support_data
```

```
def find_frequent_itemsets(dataset, min_support,
include_support=False, verbose=False):
    """
    Find frequent itemsets in the given transactions using FP-growth.
This
    function returns a generator instead of an eagerly-populated list
of items.

    The `dataset` parameter can be any iterable of iterables of items.
    `min_support` should be an integer specifying the minimum number
of
    occurrences of an itemset for it to be accepted.

    Each item must be hashable (i.e., it must be valid as a member of
a
    dictionary or a set).

    If `include_support` is true, yield (itemset, support) pairs
instead of
    just the itemsets.

    Parameters
    ----------
    dataset : list
        The dataset (a list of transactions) from which to generate
        candidate itemsets.

    min_support : float
        The minimum support threshold. Defaults to 0.5.

    include_support : bool
        Include support in output (default=False).

    """
    items = defaultdict(lambda: 0) # mapping from items to their
supports
    processed_transactions = []

    # Load the passed-in transactions and count the support that
individual
    # items have.
    for transaction in dataset:
        processed = []
        for item in transaction:
            items[item] += 1
            processed.append(item)
        processed_transactions.append(processed)

    # Remove infrequent items from the item support dictionary.
    items = dict((item, support) for item, support in items.items()
```

```
        if support >= min_support)

    # Build our FP-tree. Before any transactions can be added to the
tree, they
    # must be stripped of infrequent items and their surviving items
must be
    # sorted in decreasing order of frequency.
    def clean_transaction(transaction):
        #transaction = filter(lambda v: v in items, transaction)
        transaction.sort(key=lambda v: items[v], reverse=True)
        return transaction

    master = FPTree()
    for transaction in map(clean_transaction, processed_transactions):
        master.add(transaction)

    support_data = {}
    def find_with_suffix(tree, suffix):
        for item, nodes in tree.items():
            support = float(sum(n.count for n in nodes)) /
len(dataset)
            if support >= min_support and item not in suffix:
                # New winner!
                found_set = [item] + suffix
                support_data[frozenset(found_set)] = support
                yield (found_set, support) if include_support else
found_set

                # Build a conditional tree and recursively search for
frequent
                # itemsets within it.
                cond_tree =
conditional_tree_from_paths(tree.prefix_paths(item),
                    min_support)
                for s in find_with_suffix(cond_tree, found_set):
                    yield s # pass along the good news to our caller

    if verbose:
        # Print a list of all the frequent itemsets.
        for itemset, support in find_with_suffix(master, []):
            print("" \
                + "{" \
                + "".join(str(i) + ", " for i in
iter(itemset)).rstrip(', ') \
                + "}" \
                + ":  sup = " +
str(round(support_data[frozenset(itemset)], 3)))

    # Search for frequent itemsets, and yield the results we find.
    for itemset in find_with_suffix(master, []):
```

```
        yield itemset

class FPTree(object):
    """
    An FP tree.

    This object may only store transaction items that are hashable
(i.e., all
    items must be valid as dictionary keys or set members).
    """

    Route = namedtuple('Route', 'head tail')

    def __init__(self):
        # The root node of the tree.
        self._root = FPNode(self, None, None)

        # A dictionary mapping items to the head and tail of a path of
        # "neighbors" that will hit every node containing that item.
        self._routes = {}

    @property
    def root(self):
        """The root node of the tree."""
        return self._root

    def add(self, transaction):
        """
        Adds a transaction to the tree.
        """

        point = self._root

        for item in transaction:
            next_point = point.search(item)
            if next_point:
                # There is already a node in this tree for the current
                # transaction item; reuse it.
                next_point.increment()
            else:
                # Create a new point and add it as a child of the
point we're
                # currently looking at.
                next_point = FPNode(self, item)
                point.add(next_point)

                # Update the route of nodes that contain this item to
include
                # our new node.
                self._update_route(next_point)
```

```
            point = next_point

    def _update_route(self, point):
        """Add the given node to the route through all nodes for its
item."""
        assert self is point.tree

        try:
            route = self._routes[point.item]
            route[1].neighbor = point # route[1] is the tail
            self._routes[point.item] = self.Route(route[0], point)
        except KeyError:
            # First node for this item; start a new route.
            self._routes[point.item] = self.Route(point, point)

    def items(self):
        """
        Generate one 2-tuples for each item represented in the tree.
The first
        element of the tuple is the item itself, and the second
element is a
        generator that will yield the nodes in the tree that belong to
the item.
        """
        for item in self._routes.keys():
            yield (item, self.nodes(item))


    def nodes(self, item):
        """
        Generates the sequence of nodes that contain the given item.
        """

        try:
            node = self._routes[item][0]
        except KeyError:
            return

        while node:
            yield node
            node = node.neighbor

    def prefix_paths(self, item):
        """Generates the prefix paths that end with the given item."""

        def collect_path(node):
            path = []
            while node and not node.root:
                path.append(node)
```

```
                node = node.parent
            path.reverse()
            return path

        return (collect_path(node) for node in self.nodes(item))

    def inspect(self):
        print("Tree:")
        self.root.inspect(1)

        print("")
        print("Routes:")
        for item, nodes in self.items():
            print("  %r" % item)
            for node in nodes:
                print("    %r" % node)

    def _removed(self, node):
        """Called when `node` is removed from the tree; performs
cleanup."""

        head, tail = self._routes[node.item]
        if node is head:
            if node is tail or not node.neighbor:
                # It was the sole node.
                del self._routes[node.item]
            else:
                self._routes[node.item] = self.Route(node.neighbor,
tail)
        else:
            for n in self.nodes(node.item):
                if n.neighbor is node:
                    n.neighbor = node.neighbor # skip over
                    if node is tail:
                        self._routes[node.item] = self.Route(head, n)
                    break

def conditional_tree_from_paths(paths, min_support):
    """Builds a conditional FP-tree from the given prefix paths."""
    tree = FPTree()
    condition_item = None
    items = set()

    # Import the nodes in the paths into the new tree. Only the counts
of the
    # leaf notes matter; the remaining counts will be reconstructed
from the
    # leaf counts.
    for path in paths:
        if condition_item is None:
```

```
            condition_item = path[-1].item

        point = tree.root
        for node in path:
            next_point = point.search(node.item)
            if not next_point:
                # Add a new node to the tree.
                items.add(node.item)
                count = node.count if node.item == condition_item else
0
                next_point = FPNode(tree, node.item, count)
                point.add(next_point)
                tree._update_route(next_point)
            point = next_point

    assert condition_item is not None

    # Calculate the counts of the non-leaf nodes.
    for path in tree.prefix_paths(condition_item):
        count = path[-1].count
        for node in reversed(path[:-1]):
            node._count += count

    # Eliminate the nodes for any items that are no longer frequent.
    for item in items:
        support = sum(n.count for n in tree.nodes(item))
        if support < min_support:
            # Doesn't make the cut anymore
            for node in tree.nodes(item):
                if node.parent is not None:
                    node.parent.remove(node)

    # Finally, remove the nodes corresponding to the item for which
this
    # conditional tree was generated.
    for node in tree.nodes(condition_item):
        if node.parent is not None: # the node might already be an
orphan
            node.parent.remove(node)

    return tree

class FPNode(object):
    """A node in an FP tree."""

    def __init__(self, tree, item, count=1):
        self._tree = tree
        self._item = item
        self._count = count
        self._parent = None
```

```
        self._children = {}
        self._neighbor = None

    def add(self, child):
        """Adds the given FPNode `child` as a child of this node."""

        if not isinstance(child, FPNode):
            raise TypeError("Can only add other FPNodes as children")

        if not child.item in self._children:
            self._children[child.item] = child
            child.parent = self

    def search(self, item):
        """
        Checks to see if this node contains a child node for the given item.
        If so, that node is returned; otherwise, `None` is returned.
        """

        try:
            return self._children[item]
        except KeyError:
            return None

    def remove(self, child):
        try:
            if self._children[child.item] is child:
                del self._children[child.item]
                child.parent = None
                self._tree._removed(child)
                for sub_child in child.children:
                    try:
                        # Merger case: we already have a child for that item, so
                        # add the sub-child's count to our child's count.
                        self._children[sub_child.item]._count += sub_child.count
                        sub_child.parent = None # it's an orphan now
                    except KeyError:
                        # Turns out we don't actually have a child, so just add
                        # the sub-child as our own child.
                        self.add(sub_child)
                child._children = {}
            else:
                raise ValueError("that node is not a child of this node")
        except KeyError:
```

```
            raise ValueError("that node is not a child of this node")

    def __contains__(self, item):
        return item in self._children

    @property
    def tree(self):
        """The tree in which this node appears."""
        return self._tree

    @property
    def item(self):
        """The item contained in this node."""
        return self._item

    @property
    def count(self):
        """The count associated with this node's item."""
        return self._count

    def increment(self):
        """Increments the count associated with this node's item."""
        if self._count is None:
            raise ValueError("Root nodes have no associated count.")
        self._count += 1

    @property
    def root(self):
        """True if this node is the root of a tree; false if
otherwise."""
        return self._item is None and self._count is None

    @property
    def leaf(self):
        """True if this node is a leaf in the tree; false if
otherwise."""
        return len(self._children) == 0

    def parent():
        doc = "The node's parent."
        def fget(self):
            return self._parent
        def fset(self, value):
            if value is not None and not isinstance(value, FPNode):
                raise TypeError("A node must have an FPNode as a
parent.")
            if value and value.tree is not self.tree:
                raise ValueError("Cannot have a parent from another
tree.")
            self._parent = value
```

```
        return locals()
    parent = property(**parent())

    def neighbor():
        doc = """
        The node's neighbor; the one with the same value that is "to
the right"
        of it in the tree.
        """
        def fget(self):
            return self._neighbor
        def fset(self, value):
            if value is not None and not isinstance(value, FPNode):
                raise TypeError("A node must have an FPNode as a
neighbor.")
            if value and value.tree is not self.tree:
                raise ValueError("Cannot have a neighbor from another
tree.")
            self._neighbor = value
        return locals()
    neighbor = property(**neighbor())

    @property
    def children(self):
        """The nodes that are children of this node."""
        return tuple(self._children.values())

    def inspect(self, depth=0):
        print(('  ' * depth) + repr(self))
        for child in self.children:
            child.inspect(depth + 1)

    def __repr__(self):
        if self.root:
            return "<%s (root)>" % type(self).__name__
        return "<%s %r (%r)>" % (type(self).__name__, self.item,
self.count)

def rules_from_conseq(freq_set, H, support_data, rules,
min_confidence=0.5, verbose=False):
    """Generates a set of candidate rules.

    Parameters
    ----------
    freq_set : frozenset
        The complete list of frequent itemsets.

    H : list
        A list of frequent itemsets (of a particular length).
```

```
    support_data : dict
        The support data for all candidate itemsets.

    rules : list
        A potentially incomplete set of candidate rules above the
minimum
        confidence threshold.

    min_confidence : float
        The minimum confidence threshold. Defaults to 0.5.
    """
    m = len(H[0])
    if m == 1:
        Hmp1 = calc_confidence(freq_set, H, support_data, rules,
min_confidence, verbose)
    if (len(freq_set) > (m+1)):
        Hmp1 = apriori_gen(H, m+1) # generate candidate itemsets
        Hmp1 = calc_confidence(freq_set, Hmp1,  support_data, rules,
min_confidence, verbose)
        if len(Hmp1) > 1:
            # If there are candidate rules above the minimum
confidence
            # threshold, recurse on the list of these candidate rules.
            rules_from_conseq(freq_set, Hmp1, support_data, rules,
min_confidence, verbose)

def calc_confidence(freq_set, H, support_data, rules,
min_confidence=0.5, verbose=False):
    """Evaluates the generated rules.

    One measurement for quantifying the goodness of association rules
is
    confidence. The confidence for a rule 'P implies H' (P -> H) is
defined as
    the support for P and H divided by the support for P
    (support (P|H) / support(P)), where the | symbol denotes the set
union
    (thus P|H means all the items in set P or in set H).

    To calculate the confidence, we iterate through the frequent
itemsets and
    associated support data. For each frequent itemset, we divide the
support
    of the itemset by the support of the antecedent (left-hand-side of
the
    rule).

    Parameters
    ----------
    freq_set : frozenset
```

```
        The complete list of frequent itemsets.

    H : list
        A list of frequent itemsets (of a particular length).

    min_support : float
        The minimum support threshold.

    rules : list
        A potentially incomplete set of candidate rules above the
minimum
        confidence threshold.

    min_confidence : float
        The minimum confidence threshold. Defaults to 0.5.

    Returns
    -------
    pruned_H : list
        The list of candidate rules above the minimum confidence
threshold.
    """
    pruned_H = [] # list of candidate rules above the minimum
confidence threshold
    for conseq in H: # iterate over the frequent itemsets
        conf = support_data[freq_set] / support_data[freq_set -
conseq]
        if conf >= min_confidence:
            rules.append((freq_set - conseq, conseq, conf))
            pruned_H.append(conseq)

            if verbose:
                print("" \
                    + "{" \
                    + "".join([str(i) + ", " for i in iter(freq_set-
conseq)]).rstrip(', ') \
                    + "}" \
                    + " ---> " \
                    + "{" \
                    + "".join([str(i) + ", " for i in
iter(conseq)]).rstrip(', ') \
                    + "}" \
                    + ":  conf = " + str(round(conf, 3)) \
                    + ", sup = " + str(round(support_data[freq_set],
3)))

    return pruned_H

def generate_rules(F, support_data, min_confidence=0.5, verbose=True):
    """Generates a set of candidate rules from a list of frequent
```

```
itemsets.

    For each frequent itemset, we calculate the confidence of using a
    particular item as the rule consequent (right-hand-side of the
rule). By
    testing and merging the remaining rules, we recursively create a
list of
    pruned rules.

    Parameters
    ----------
    F : list
        A list of frequent itemsets.

    support_data : dict
        The corresponding support data for the frequent itemsets (L).

    min_confidence : float
        The minimum confidence threshold. Defaults to 0.5.

    Returns
    -------
    rules : list
        The list of candidate rules above the minimum confidence
threshold.
    """
    rules = []
    for i in range(1, len(F)):
        for freq_set in F[i]:
            H1 = [frozenset([item]) for item in freq_set]
            if (i > 1):
                rules_from_conseq(freq_set, H1, support_data, rules,
min_confidence, verbose)
            else:
                calc_confidence(freq_set, H1, support_data, rules,
min_confidence, verbose)

    return rules

import csv

def load_dataset(filename):
    transactions = []
    with open(filename, 'r', encoding='utf-8') as file:
        data_iter = csv.reader(file, delimiter=',', quotechar='"')
        for row in data_iter:
            # Convert each row to a set to handle variable-length
transactions
            transactions.append(set(row))
    return transactions
```

```python
# Path to the grocery dataset CSV file
file_path = 'grocery.csv'

# Load the dataset
dataset = load_dataset(file_path)
D = list(map(set, dataset))

# Define your support threshold
min_support = 0.05  # 5% support
min_confidence = 0.5  # minimum confidence for the rules

# Generate frequent itemsets using FP-growth
frequent_itemsets, support_data = fpgrowth(dataset,
min_support=min_support, include_support=True, verbose=True)

# Generate association rules from the frequent itemsets
rules = generate_rules(frequent_itemsets, support_data,
min_confidence=min_confidence, verbose=True)

# Print out the rules
for rule in rules:
    print(rule)
```

```
{citrus fruit}:  sup = 0.083
{margarine}:  sup = 0.059
{yogurt}:  sup = 0.14
{whole milk, yogurt}:  sup = 0.056
{tropical fruit}:  sup = 0.105
{coffee}:  sup = 0.058
{whole milk}:  sup = 0.256
{pip fruit}:  sup = 0.076
{other vegetables}:  sup = 0.193
{whole milk, other vegetables}:  sup = 0.075
{butter}:  sup = 0.055
{rolls/buns}:  sup = 0.184
{whole milk, rolls/buns}:  sup = 0.057
{bottled beer}:  sup = 0.081
{bottled water}:  sup = 0.111
{curd}:  sup = 0.053
{beef}:  sup = 0.052
{soda}:  sup = 0.174
{frankfurter}:  sup = 0.059
{newspapers}:  sup = 0.08
{fruit/vegetable juice}:  sup = 0.072
{pastry}:  sup = 0.089
{root vegetables}:  sup = 0.109
{canned beer}:  sup = 0.078
{sausage}:  sup = 0.094
{shopping bags}:  sup = 0.099
```

```
{brown bread}:  sup = 0.065
{napkins}:  sup = 0.052
{whipped/sour cream}:  sup = 0.072
{pork}:  sup = 0.058
{domestic eggs}:  sup = 0.063
```

```python
# Define your support threshold
min_support = 0.04  # 4% support
min_confidence = 0.5  # minimum confidence for the rules

# Generate frequent itemsets using FP-growth
frequent_itemsets, support_data = fpgrowth(dataset,
min_support=min_support, include_support=True, verbose=True)

# Generate association rules from the frequent itemsets
rules = generate_rules(frequent_itemsets, support_data,
min_confidence=min_confidence, verbose=True)

# Print out the rules
for rule in rules:
    print(rule)
```

```
{citrus fruit}:  sup = 0.083
{margarine}:  sup = 0.059
{yogurt}:  sup = 0.14
{whole milk, yogurt}:  sup = 0.056
{other vegetables, yogurt}:  sup = 0.043
{tropical fruit}:  sup = 0.105
{whole milk, tropical fruit}:  sup = 0.042
{coffee}:  sup = 0.058
{whole milk}:  sup = 0.256
{pip fruit}:  sup = 0.076
{other vegetables}:  sup = 0.193
{whole milk, other vegetables}:  sup = 0.075
{butter}:  sup = 0.055
{rolls/buns}:  sup = 0.184
{other vegetables, rolls/buns}:  sup = 0.043
{whole milk, rolls/buns}:  sup = 0.057
{bottled beer}:  sup = 0.081
{bottled water}:  sup = 0.111
{chocolate}:  sup = 0.05
{white bread}:  sup = 0.042
{curd}:  sup = 0.053
{beef}:  sup = 0.052
{soda}:  sup = 0.174
{whole milk, soda}:  sup = 0.04
{frankfurter}:  sup = 0.059
{chicken}:  sup = 0.043
{newspapers}:  sup = 0.08
{fruit/vegetable juice}:  sup = 0.072
```

```
{pastry}:  sup = 0.089
{root vegetables}:  sup = 0.109
{other vegetables, root vegetables}:  sup = 0.047
{whole milk, root vegetables}:  sup = 0.049
{canned beer}:  sup = 0.078
{sausage}:  sup = 0.094
{shopping bags}:  sup = 0.099
{brown bread}:  sup = 0.065
{napkins}:  sup = 0.052
{whipped/sour cream}:  sup = 0.072
{pork}:  sup = 0.058
{domestic eggs}:  sup = 0.063
{frozen vegetables}:  sup = 0.048

# Define your support threshold
min_support = 0.03  # 3% support
min_confidence = 0.5  # minimum confidence for the rules

# Generate frequent itemsets using FP-growth
frequent_itemsets, support_data = fpgrowth(dataset,
min_support=min_support, include_support=True, verbose=True)

# Generate association rules from the frequent itemsets
rules = generate_rules(frequent_itemsets, support_data,
min_confidence=min_confidence, verbose=True)

# Print out the rules
for rule in rules:
    print(rule)

{citrus fruit}:  sup = 0.083
{whole milk, citrus fruit}:  sup = 0.031
{margarine}:  sup = 0.059
{yogurt}:  sup = 0.14
{whole milk, yogurt}:  sup = 0.056
{rolls/buns, yogurt}:  sup = 0.034
{other vegetables, yogurt}:  sup = 0.043
{tropical fruit}:  sup = 0.105
{other vegetables, tropical fruit}:  sup = 0.036
{whole milk, tropical fruit}:  sup = 0.042
{coffee}:  sup = 0.058
{whole milk}:  sup = 0.256
{pip fruit}:  sup = 0.076
{whole milk, pip fruit}:  sup = 0.03
{cream cheese}:  sup = 0.04
{other vegetables}:  sup = 0.193
{whole milk, other vegetables}:  sup = 0.075
{long life bakery product}:  sup = 0.037
{butter}:  sup = 0.055
{rolls/buns}:  sup = 0.184
```

```
{other vegetables, rolls/buns}:  sup = 0.043
{whole milk, rolls/buns}:  sup = 0.057
{bottled beer}:  sup = 0.081
{UHT-milk}:  sup = 0.033
{bottled water}:  sup = 0.111
{whole milk, bottled water}:  sup = 0.034
{chocolate}:  sup = 0.05
{white bread}:  sup = 0.042
{curd}:  sup = 0.053
{beef}:  sup = 0.052
{soda}:  sup = 0.174
{rolls/buns, soda}:  sup = 0.038
{whole milk, soda}:  sup = 0.04
{other vegetables, soda}:  sup = 0.033
{frankfurter}:  sup = 0.059
{chicken}:  sup = 0.043
{newspapers}:  sup = 0.08
{fruit/vegetable juice}:  sup = 0.072
{sugar}:  sup = 0.034
{pastry}:  sup = 0.089
{whole milk, pastry}:  sup = 0.033
{root vegetables}:  sup = 0.109
{other vegetables, root vegetables}:  sup = 0.047
{whole milk, root vegetables}:  sup = 0.049
{waffles}:  sup = 0.038
{salty snack}:  sup = 0.038
{canned beer}:  sup = 0.078
{sausage}:  sup = 0.094
{rolls/buns, sausage}:  sup = 0.031
{shopping bags}:  sup = 0.099
{brown bread}:  sup = 0.065
{napkins}:  sup = 0.052
{hamburger meat}:  sup = 0.033
{hygiene articles}:  sup = 0.033
{whipped/sour cream}:  sup = 0.072
{whole milk, whipped/sour cream}:  sup = 0.032
{pork}:  sup = 0.058
{berries}:  sup = 0.033
{dessert}:  sup = 0.037
{domestic eggs}:  sup = 0.063
{frozen vegetables}:  sup = 0.048
{specialty chocolate}:  sup = 0.03
{onions}:  sup = 0.031

# Define your support threshold
min_support = 0.02  # 2% support
min_confidence = 0.5  # minimum confidence for the rules

# Generate frequent itemsets using FP-growth
frequent_itemsets, support_data = fpgrowth(dataset,
```

```python
min_support=min_support, include_support=True, verbose=True)

# Generate association rules from the frequent itemsets
rules = generate_rules(frequent_itemsets, support_data,
min_confidence=min_confidence, verbose=True)

# Print out the rules
for rule in rules:
    print(rule)
```

```
{citrus fruit}:  sup = 0.083
{whole milk, citrus fruit}:  sup = 0.031
{yogurt, citrus fruit}:  sup = 0.022
{other vegetables, citrus fruit}:  sup = 0.029
{margarine}:  sup = 0.059
{whole milk, margarine}:  sup = 0.024
{yogurt}:  sup = 0.14
{whole milk, yogurt}:  sup = 0.056
{soda, yogurt}:  sup = 0.027
{rolls/buns, yogurt}:  sup = 0.034
{other vegetables, yogurt}:  sup = 0.043
{whole milk, other vegetables, yogurt}:  sup = 0.022
{tropical fruit}:  sup = 0.105
{yogurt, tropical fruit}:  sup = 0.029
{other vegetables, tropical fruit}:  sup = 0.036
{whole milk, tropical fruit}:  sup = 0.042
{rolls/buns, tropical fruit}:  sup = 0.025
{root vegetables, tropical fruit}:  sup = 0.021
{soda, tropical fruit}:  sup = 0.021
{coffee}:  sup = 0.058
{whole milk}:  sup = 0.256
{pip fruit}:  sup = 0.076
{whole milk, pip fruit}:  sup = 0.03
{tropical fruit, pip fruit}:  sup = 0.02
{other vegetables, pip fruit}:  sup = 0.026
{cream cheese}:  sup = 0.04
{other vegetables}:  sup = 0.193
{whole milk, other vegetables}:  sup = 0.075
{long life bakery product}:  sup = 0.037
{butter}:  sup = 0.055
{whole milk, butter}:  sup = 0.028
{other vegetables, butter}:  sup = 0.02
{rolls/buns}:  sup = 0.184
{other vegetables, rolls/buns}:  sup = 0.043
{whole milk, rolls/buns}:  sup = 0.057
{bottled beer}:  sup = 0.081
{whole milk, bottled beer}:  sup = 0.02
{UHT-milk}:  sup = 0.033
{bottled water}:  sup = 0.111
{other vegetables, bottled water}:  sup = 0.025
```

```
{whole milk, bottled water}:  sup = 0.034
{yogurt, bottled water}:  sup = 0.023
{rolls/buns, bottled water}:  sup = 0.024
{soda, bottled water}:  sup = 0.029
{chocolate}:  sup = 0.05
{white bread}:  sup = 0.042
{curd}:  sup = 0.053
{whole milk, curd}:  sup = 0.026
{beef}:  sup = 0.052
{whole milk, beef}:  sup = 0.021
{soda}:  sup = 0.174
{rolls/buns, soda}:  sup = 0.038
{whole milk, soda}:  sup = 0.04
{other vegetables, soda}:  sup = 0.033
{frankfurter}:  sup = 0.059
{whole milk, frankfurter}:  sup = 0.021
{chicken}:  sup = 0.043
{newspapers}:  sup = 0.08
{whole milk, newspapers}:  sup = 0.027
{fruit/vegetable juice}:  sup = 0.072
{other vegetables, fruit/vegetable juice}:  sup = 0.021
{whole milk, fruit/vegetable juice}:  sup = 0.027
{sugar}:  sup = 0.034
{specialty bar}:  sup = 0.027
{pastry}:  sup = 0.089
{soda, pastry}:  sup = 0.021
{whole milk, pastry}:  sup = 0.033
{rolls/buns, pastry}:  sup = 0.021
{other vegetables, pastry}:  sup = 0.023
{butter milk}:  sup = 0.028
{root vegetables}:  sup = 0.109
{other vegetables, root vegetables}:  sup = 0.047
{whole milk, other vegetables, root vegetables}:  sup = 0.023
{rolls/buns, root vegetables}:  sup = 0.024
{whole milk, root vegetables}:  sup = 0.049
{yogurt, root vegetables}:  sup = 0.026
{waffles}:  sup = 0.038
{salty snack}:  sup = 0.038
{candy}:  sup = 0.03
{canned beer}:  sup = 0.078
{sausage}:  sup = 0.094
{rolls/buns, sausage}:  sup = 0.031
{soda, sausage}:  sup = 0.024
{whole milk, sausage}:  sup = 0.03
{other vegetables, sausage}:  sup = 0.027
{shopping bags}:  sup = 0.099
{soda, shopping bags}:  sup = 0.025
{whole milk, shopping bags}:  sup = 0.025
{other vegetables, shopping bags}:  sup = 0.023
```

```
{brown bread}:  sup = 0.065
{whole milk, brown bread}:  sup = 0.025
{beverages}:  sup = 0.026
{napkins}:  sup = 0.052
{hamburger meat}:  sup = 0.033
{hygiene articles}:  sup = 0.033
{whipped/sour cream}:  sup = 0.072
{whole milk, whipped/sour cream}:  sup = 0.032
{other vegetables, whipped/sour cream}:  sup = 0.029
{yogurt, whipped/sour cream}:  sup = 0.021
{pork}:  sup = 0.058
{whole milk, pork}:  sup = 0.022
{other vegetables, pork}:  sup = 0.022
{berries}:  sup = 0.033
{grapes}:  sup = 0.022
{dessert}:  sup = 0.037
{domestic eggs}:  sup = 0.063
{whole milk, domestic eggs}:  sup = 0.03
{other vegetables, domestic eggs}:  sup = 0.022
{misc. beverages}:  sup = 0.028
{hard cheese}:  sup = 0.025
{cat food}:  sup = 0.023
{ham}:  sup = 0.026
{oil}:  sup = 0.028
{chewing gum}:  sup = 0.021
{ice cream}:  sup = 0.025
{frozen vegetables}:  sup = 0.048
{whole milk, frozen vegetables}:  sup = 0.02
{specialty chocolate}:  sup = 0.03
{frozen meals}:  sup = 0.028
{onions}:  sup = 0.031
{sliced cheese}:  sup = 0.025
{meat}:  sup = 0.026
{yogurt, other vegetables} ---> {whole milk}:  conf = 0.513, sup =
0.022
(frozenset({'yogurt', 'other vegetables'}), frozenset({'whole milk'}),
0.5128805620608898)

# Define your support threshold
min_support = 0.01  # 1% support
min_confidence = 0.5  # minimum confidence for the rules

# Generate frequent itemsets using FP-growth
frequent_itemsets, support_data = fpgrowth(dataset,
min_support=min_support, include_support=True, verbose=True)

# Generate association rules from the frequent itemsets
rules = generate_rules(frequent_itemsets, support_data,
min_confidence=min_confidence, verbose=True)
```

```python
# Print out the rules
for rule in rules:
    print(rule)
```

```
{citrus fruit}:  sup = 0.083
{whole milk, citrus fruit}:  sup = 0.031
{yogurt, citrus fruit}:  sup = 0.022
{whole milk, yogurt, citrus fruit}:  sup = 0.01
{bottled water, citrus fruit}:  sup = 0.014
{tropical fruit, citrus fruit}:  sup = 0.02
{other vegetables, citrus fruit}:  sup = 0.029
{whole milk, other vegetables, citrus fruit}:  sup = 0.013
{root vegetables, citrus fruit}:  sup = 0.018
{other vegetables, root vegetables, citrus fruit}:  sup = 0.01
{sausage, citrus fruit}:  sup = 0.011
{rolls/buns, citrus fruit}:  sup = 0.017
{soda, citrus fruit}:  sup = 0.013
{margarine}:  sup = 0.059
{other vegetables, margarine}:  sup = 0.02
{whole milk, margarine}:  sup = 0.024
{rolls/buns, margarine}:  sup = 0.015
{root vegetables, margarine}:  sup = 0.011
{bottled water, margarine}:  sup = 0.01
{yogurt, margarine}:  sup = 0.014
{soda, margarine}:  sup = 0.01
{semi-finished bread}:  sup = 0.018
{yogurt}:  sup = 0.14
{whole milk, yogurt}:  sup = 0.056
{soda, yogurt}:  sup = 0.027
{whole milk, soda, yogurt}:  sup = 0.01
{rolls/buns, yogurt}:  sup = 0.034
{whole milk, rolls/buns, yogurt}:  sup = 0.016
{other vegetables, rolls/buns, yogurt}:  sup = 0.011
{other vegetables, yogurt}:  sup = 0.043
{whole milk, other vegetables, yogurt}:  sup = 0.022
{tropical fruit}:  sup = 0.105
{yogurt, tropical fruit}:  sup = 0.029
{whole milk, yogurt, tropical fruit}:  sup = 0.015
{other vegetables, yogurt, tropical fruit}:  sup = 0.012
{other vegetables, tropical fruit}:  sup = 0.036
{whole milk, other vegetables, tropical fruit}:  sup = 0.017
{bottled water, tropical fruit}:  sup = 0.019
{whole milk, tropical fruit}:  sup = 0.042
{rolls/buns, tropical fruit}:  sup = 0.025
{whole milk, rolls/buns, tropical fruit}:  sup = 0.011
{root vegetables, tropical fruit}:  sup = 0.021
{other vegetables, root vegetables, tropical fruit}:  sup = 0.012
{whole milk, root vegetables, tropical fruit}:  sup = 0.012
{soda, tropical fruit}:  sup = 0.021
{coffee}:  sup = 0.058
```

```
{whole milk, coffee}:  sup = 0.019
{other vegetables, coffee}:  sup = 0.013
{rolls/buns, coffee}:  sup = 0.011
{whole milk}:  sup = 0.256
{pip fruit}:  sup = 0.076
{yogurt, pip fruit}:  sup = 0.018
{whole milk, pip fruit}:  sup = 0.03
{rolls/buns, pip fruit}:  sup = 0.014
{tropical fruit, pip fruit}:  sup = 0.02
{sausage, pip fruit}:  sup = 0.011
{citrus fruit, pip fruit}:  sup = 0.014
{other vegetables, pip fruit}:  sup = 0.026
{whole milk, other vegetables, pip fruit}:  sup = 0.014
{root vegetables, pip fruit}:  sup = 0.016
{bottled water, pip fruit}:  sup = 0.011
{soda, pip fruit}:  sup = 0.013
{pastry, pip fruit}:  sup = 0.011
{cream cheese}:  sup = 0.04
{yogurt, cream cheese}:  sup = 0.012
{whole milk, cream cheese}:  sup = 0.016
{other vegetables, cream cheese}:  sup = 0.014
{other vegetables}:  sup = 0.193
{whole milk, other vegetables}:  sup = 0.075
{long life bakery product}:  sup = 0.037
{whole milk, long life bakery product}:  sup = 0.014
{other vegetables, long life bakery product}:  sup = 0.011
{condensed milk}:  sup = 0.01
{butter}:  sup = 0.055
{whole milk, butter}:  sup = 0.028
{yogurt, butter}:  sup = 0.015
{other vegetables, butter}:  sup = 0.02
{whole milk, other vegetables, butter}:  sup = 0.011
{rolls/buns, butter}:  sup = 0.013
{root vegetables, butter}:  sup = 0.013
{whipped/sour cream, butter}:  sup = 0.01
{rolls/buns}:  sup = 0.184
{other vegetables, rolls/buns}:  sup = 0.043
{whole milk, other vegetables, rolls/buns}:  sup = 0.018
{whole milk, rolls/buns}:  sup = 0.057
{bottled beer}:  sup = 0.081
{other vegetables, bottled beer}:  sup = 0.016
{rolls/buns, bottled beer}:  sup = 0.014
{bottled water, bottled beer}:  sup = 0.016
{soda, bottled beer}:  sup = 0.017
{whole milk, bottled beer}:  sup = 0.02
{UHT-milk}:  sup = 0.033
{pot plants}:  sup = 0.017
{bottled water}:  sup = 0.111
{other vegetables, bottled water}:  sup = 0.025
```

```
{whole milk, other vegetables, bottled water}:  sup = 0.011
{whole milk, bottled water}:  sup = 0.034
{yogurt, bottled water}:  sup = 0.023
{rolls/buns, bottled water}:  sup = 0.024
{soda, bottled water}:  sup = 0.029
{chocolate}:  sup = 0.05
{other vegetables, chocolate}:  sup = 0.013
{rolls/buns, chocolate}:  sup = 0.012
{soda, chocolate}:  sup = 0.014
{whole milk, chocolate}:  sup = 0.017
{white bread}:  sup = 0.042
{other vegetables, white bread}:  sup = 0.014
{whole milk, white bread}:  sup = 0.017
{soda, white bread}:  sup = 0.01
{curd}:  sup = 0.053
{whole milk, curd}:  sup = 0.026
{yogurt, curd}:  sup = 0.017
{whole milk, yogurt, curd}:  sup = 0.01
{tropical fruit, curd}:  sup = 0.01
{other vegetables, curd}:  sup = 0.017
{rolls/buns, curd}:  sup = 0.01
{root vegetables, curd}:  sup = 0.011
{whipped/sour cream, curd}:  sup = 0.01
{dishes}:  sup = 0.018
{flour}:  sup = 0.017
{beef}:  sup = 0.052
{rolls/buns, beef}:  sup = 0.014
{whole milk, beef}:  sup = 0.021
{root vegetables, beef}:  sup = 0.017
{other vegetables, beef}:  sup = 0.02
{yogurt, beef}:  sup = 0.012
{soda}:  sup = 0.174
{rolls/buns, soda}:  sup = 0.038
{whole milk, soda}:  sup = 0.04
{other vegetables, soda}:  sup = 0.033
{whole milk, other vegetables, soda}:  sup = 0.014
{frankfurter}:  sup = 0.059
{rolls/buns, frankfurter}:  sup = 0.019
{soda, frankfurter}:  sup = 0.011
{whole milk, frankfurter}:  sup = 0.021
{other vegetables, frankfurter}:  sup = 0.016
{root vegetables, frankfurter}:  sup = 0.01
{sausage, frankfurter}:  sup = 0.01
{yogurt, frankfurter}:  sup = 0.011
{chicken}:  sup = 0.043
{other vegetables, chicken}:  sup = 0.018
{root vegetables, chicken}:  sup = 0.011
{whole milk, chicken}:  sup = 0.018
{newspapers}:  sup = 0.08
```

```
{tropical fruit, newspapers}:  sup = 0.012
{soda, newspapers}:  sup = 0.015
{rolls/buns, newspapers}:  sup = 0.02
{yogurt, newspapers}:  sup = 0.015
{bottled water, newspapers}:  sup = 0.011
{whole milk, newspapers}:  sup = 0.027
{other vegetables, newspapers}:  sup = 0.019
{root vegetables, newspapers}:  sup = 0.011
{fruit/vegetable juice}:  sup = 0.072
{soda, fruit/vegetable juice}:  sup = 0.018
{shopping bags, fruit/vegetable juice}:  sup = 0.011
{other vegetables, fruit/vegetable juice}:  sup = 0.021
{whole milk, other vegetables, fruit/vegetable juice}:  sup = 0.01
{tropical fruit, fruit/vegetable juice}:  sup = 0.014
{bottled water, fruit/vegetable juice}:  sup = 0.014
{whole milk, fruit/vegetable juice}:  sup = 0.027
{rolls/buns, fruit/vegetable juice}:  sup = 0.015
{root vegetables, fruit/vegetable juice}:  sup = 0.012
{sausage, fruit/vegetable juice}:  sup = 0.01
{yogurt, fruit/vegetable juice}:  sup = 0.019
{citrus fruit, fruit/vegetable juice}:  sup = 0.01
{sugar}:  sup = 0.034
{whole milk, sugar}:  sup = 0.015
{other vegetables, sugar}:  sup = 0.011
{packaged fruit/vegetables}:  sup = 0.013
{specialty bar}:  sup = 0.027
{pastry}:  sup = 0.089
{soda, pastry}:  sup = 0.021
{whole milk, pastry}:  sup = 0.033
{yogurt, pastry}:  sup = 0.018
{root vegetables, pastry}:  sup = 0.011
{tropical fruit, pastry}:  sup = 0.013
{rolls/buns, pastry}:  sup = 0.021
{sausage, pastry}:  sup = 0.013
{other vegetables, pastry}:  sup = 0.023
{whole milk, other vegetables, pastry}:  sup = 0.011
{shopping bags, pastry}:  sup = 0.012
{butter milk}:  sup = 0.028
{whole milk, butter milk}:  sup = 0.012
{other vegetables, butter milk}:  sup = 0.01
{detergent}:  sup = 0.019
{processed cheese}:  sup = 0.017
{root vegetables}:  sup = 0.109
{other vegetables, root vegetables}:  sup = 0.047
{whole milk, other vegetables, root vegetables}:  sup = 0.023
{rolls/buns, root vegetables}:  sup = 0.024
{other vegetables, rolls/buns, root vegetables}:  sup = 0.012
{whole milk, rolls/buns, root vegetables}:  sup = 0.013
{whole milk, root vegetables}:  sup = 0.049
```

```
{soda, root vegetables}:  sup = 0.019
{yogurt, root vegetables}:  sup = 0.026
{whole milk, yogurt, root vegetables}:  sup = 0.015
{other vegetables, yogurt, root vegetables}:  sup = 0.013
{bottled water, root vegetables}:  sup = 0.016
{waffles}:  sup = 0.038
{other vegetables, waffles}:  sup = 0.01
{whole milk, waffles}:  sup = 0.013
{salty snack}:  sup = 0.038
{other vegetables, salty snack}:  sup = 0.011
{whole milk, salty snack}:  sup = 0.011
{candy}:  sup = 0.03
{frozen dessert}:  sup = 0.011
{canned beer}:  sup = 0.078
{soda, canned beer}:  sup = 0.014
{shopping bags, canned beer}:  sup = 0.011
{rolls/buns, canned beer}:  sup = 0.011
{sausage}:  sup = 0.094
{rolls/buns, sausage}:  sup = 0.031
{soda, sausage}:  sup = 0.024
{shopping bags, sausage}:  sup = 0.016
{whole milk, sausage}:  sup = 0.03
{yogurt, sausage}:  sup = 0.02
{root vegetables, sausage}:  sup = 0.015
{other vegetables, sausage}:  sup = 0.027
{whole milk, other vegetables, sausage}:  sup = 0.01
{tropical fruit, sausage}:  sup = 0.014
{bottled water, sausage}:  sup = 0.012
{shopping bags}:  sup = 0.099
{soda, shopping bags}:  sup = 0.025
{rolls/buns, shopping bags}:  sup = 0.02
{whole milk, shopping bags}:  sup = 0.025
{yogurt, shopping bags}:  sup = 0.015
{other vegetables, shopping bags}:  sup = 0.023
{bottled water, shopping bags}:  sup = 0.011
{tropical fruit, shopping bags}:  sup = 0.014
{root vegetables, shopping bags}:  sup = 0.013
{brown bread}:  sup = 0.065
{soda, brown bread}:  sup = 0.013
{whole milk, brown bread}:  sup = 0.025
{yogurt, brown bread}:  sup = 0.015
{root vegetables, brown bread}:  sup = 0.01
{tropical fruit, brown bread}:  sup = 0.011
{other vegetables, brown bread}:  sup = 0.019
{rolls/buns, brown bread}:  sup = 0.013
{sausage, brown bread}:  sup = 0.011
{beverages}:  sup = 0.026
{napkins}:  sup = 0.052
{other vegetables, napkins}:  sup = 0.014
```

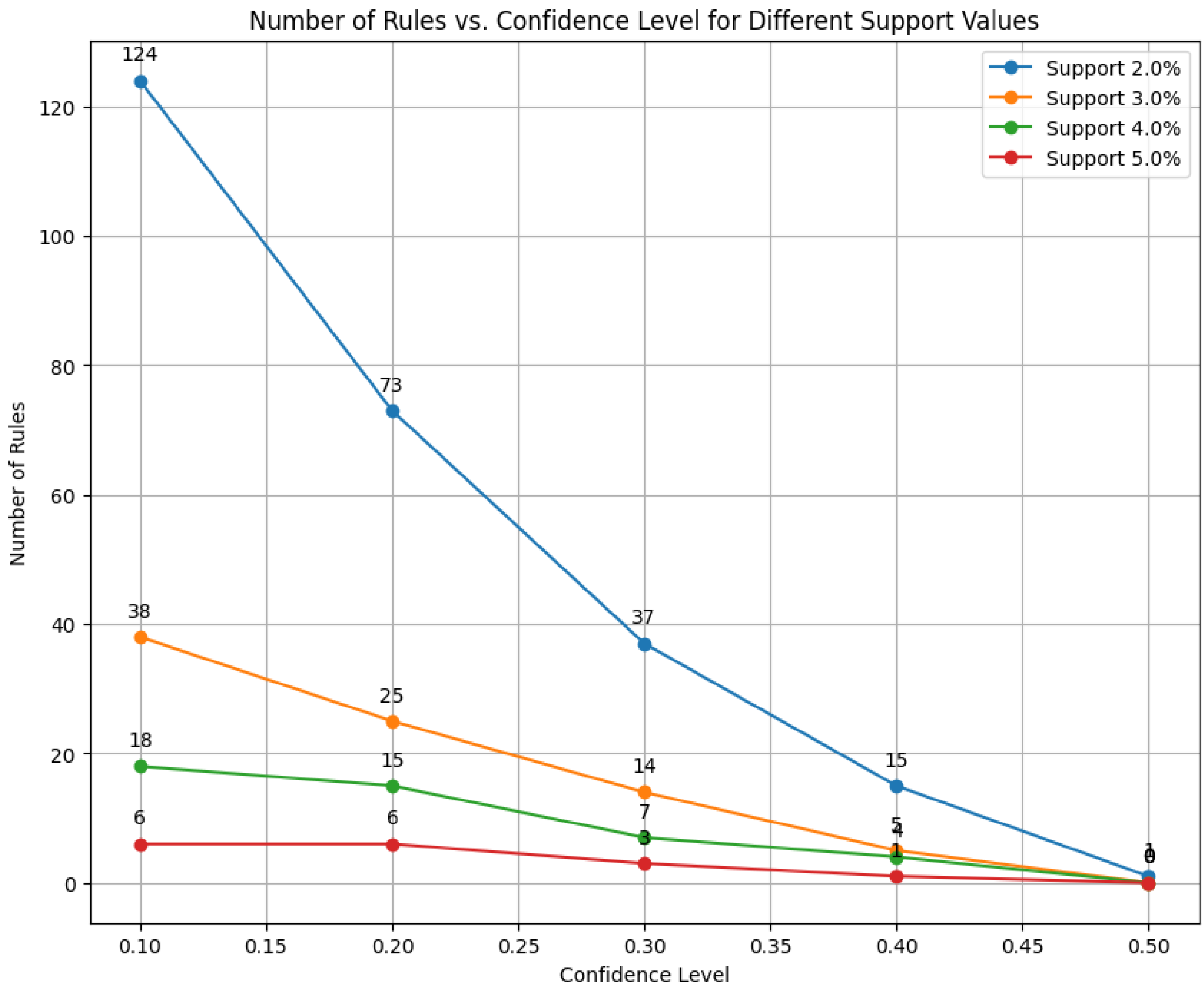
```
{rolls/buns, napkins}:  sup = 0.012
{whole milk, napkins}:  sup = 0.02
{yogurt, napkins}:  sup = 0.012
{soda, napkins}:  sup = 0.012
{tropical fruit, napkins}:  sup = 0.01
{hamburger meat}:  sup = 0.033
{other vegetables, hamburger meat}:  sup = 0.014
{whole milk, hamburger meat}:  sup = 0.015
{hygiene articles}:  sup = 0.033
{whole milk, hygiene articles}:  sup = 0.013
{whipped/sour cream}:  sup = 0.072
{whole milk, whipped/sour cream}:  sup = 0.032
{other vegetables, whipped/sour cream}:  sup = 0.029
{whole milk, other vegetables, whipped/sour cream}:  sup = 0.015
{soda, whipped/sour cream}:  sup = 0.012
{rolls/buns, whipped/sour cream}:  sup = 0.015
{root vegetables, whipped/sour cream}:  sup = 0.017
{citrus fruit, whipped/sour cream}:  sup = 0.011
{yogurt, whipped/sour cream}:  sup = 0.021
{whole milk, yogurt, whipped/sour cream}:  sup = 0.011
{other vegetables, yogurt, whipped/sour cream}:  sup = 0.01
{tropical fruit, whipped/sour cream}:  sup = 0.014
{pork}:  sup = 0.058
{whole milk, pork}:  sup = 0.022
{other vegetables, pork}:  sup = 0.022
{whole milk, other vegetables, pork}:  sup = 0.01
{soda, pork}:  sup = 0.012
{rolls/buns, pork}:  sup = 0.011
{root vegetables, pork}:  sup = 0.014
{berries}:  sup = 0.033
{whole milk, berries}:  sup = 0.012
{other vegetables, berries}:  sup = 0.01
{yogurt, berries}:  sup = 0.011
{grapes}:  sup = 0.022
{dessert}:  sup = 0.037
{whole milk, dessert}:  sup = 0.014
{other vegetables, dessert}:  sup = 0.012
{domestic eggs}:  sup = 0.063
{whole milk, domestic eggs}:  sup = 0.03
{soda, domestic eggs}:  sup = 0.012
{yogurt, domestic eggs}:  sup = 0.014
{root vegetables, domestic eggs}:  sup = 0.014
{tropical fruit, domestic eggs}:  sup = 0.011
{other vegetables, domestic eggs}:  sup = 0.022
{whole milk, other vegetables, domestic eggs}:  sup = 0.012
{citrus fruit, domestic eggs}:  sup = 0.01
{rolls/buns, domestic eggs}:  sup = 0.016
{spread cheese}:  sup = 0.011
{misc. beverages}:  sup = 0.028
```

```
{hard cheese}:  sup = 0.025
{whole milk, hard cheese}:  sup = 0.01
{cat food}:  sup = 0.023
{ham}:  sup = 0.026
{whole milk, ham}:  sup = 0.011
{baking powder}:  sup = 0.018
{pickled vegetables}:  sup = 0.018
{oil}:  sup = 0.028
{whole milk, oil}:  sup = 0.011
{chewing gum}:  sup = 0.021
{ice cream}:  sup = 0.025
{frozen vegetables}:  sup = 0.048
{whole milk, frozen vegetables}:  sup = 0.02
{other vegetables, frozen vegetables}:  sup = 0.018
{rolls/buns, frozen vegetables}:  sup = 0.01
{root vegetables, frozen vegetables}:  sup = 0.012
{yogurt, frozen vegetables}:  sup = 0.012
{canned fish}:  sup = 0.015
{seasonal products}:  sup = 0.014
{red/blush wine}:  sup = 0.019
{specialty chocolate}:  sup = 0.03
{flower (seeds)}:  sup = 0.01
{salt}:  sup = 0.011
{frozen meals}:  sup = 0.028
{canned vegetables}:  sup = 0.011
{onions}:  sup = 0.031
{whole milk, onions}:  sup = 0.012
{other vegetables, onions}:  sup = 0.014
{white wine}:  sup = 0.019
{herbs}:  sup = 0.016
{sliced cheese}:  sup = 0.025
{whole milk, sliced cheese}:  sup = 0.011
{pasta}:  sup = 0.015
{cling film/bags}:  sup = 0.011
{soft cheese}:  sup = 0.017
{cake bar}:  sup = 0.013
{frozen fish}:  sup = 0.012
{dish cleaner}:  sup = 0.01
{meat}:  sup = 0.026
{mustard}:  sup = 0.012
{liquor}:  sup = 0.011
{roll products}:  sup = 0.01
{citrus fruit, root vegetables} ---> {other vegetables}:  conf =
0.586, sup = 0.01
{yogurt, other vegetables} ---> {whole milk}:  conf = 0.513, sup =
0.022
{tropical fruit, yogurt} ---> {whole milk}:  conf = 0.517, sup = 0.015
{tropical fruit, root vegetables} ---> {other vegetables}:  conf =
0.585, sup = 0.012
```

```
{tropical fruit, root vegetables} ---> {whole milk}:  conf = 0.57, sup
= 0.012
{other vegetables, pip fruit} ---> {whole milk}:  conf = 0.518, sup =
0.014
{other vegetables, butter} ---> {whole milk}:  conf = 0.574, sup =
0.011
{curd, yogurt} ---> {whole milk}:  conf = 0.582, sup = 0.01
{rolls/buns, root vegetables} ---> {other vegetables}:  conf = 0.502,
sup = 0.012
{rolls/buns, root vegetables} ---> {whole milk}:  conf = 0.523, sup =
0.013
{yogurt, root vegetables} ---> {whole milk}:  conf = 0.563, sup =
0.015
{yogurt, root vegetables} ---> {other vegetables}:  conf = 0.5, sup =
0.013
{other vegetables, whipped/sour cream} ---> {whole milk}:  conf =
0.507, sup = 0.015
{yogurt, whipped/sour cream} ---> {whole milk}:  conf = 0.525, sup =
0.011
{domestic eggs, other vegetables} ---> {whole milk}:  conf = 0.553,
sup = 0.012
(frozenset({'citrus fruit', 'root vegetables'}), frozenset({'other
vegetables'}), 0.5862068965517241)
(frozenset({'yogurt', 'other vegetables'}), frozenset({'whole milk'}),
0.5128805620608898)
(frozenset({'tropical fruit', 'yogurt'}), frozenset({'whole milk'}),
0.5173611111111111)
(frozenset({'tropical fruit', 'root vegetables'}), frozenset({'other
vegetables'}), 0.5845410628019324)
(frozenset({'tropical fruit', 'root vegetables'}), frozenset({'whole
milk'}), 0.570048309178744)
(frozenset({'other vegetables', 'pip fruit'}), frozenset({'whole
milk'}), 0.5175097276264592)
(frozenset({'other vegetables', 'butter'}), frozenset({'whole milk'}),
0.5736040609137055)
(frozenset({'curd', 'yogurt'}), frozenset({'whole milk'}),
0.5823529411764706)
(frozenset({'rolls/buns', 'root vegetables'}), frozenset({'other
vegetables'}), 0.502092050209205)
(frozenset({'rolls/buns', 'root vegetables'}), frozenset({'whole
milk'}), 0.5230125523012552)
(frozenset({'yogurt', 'root vegetables'}), frozenset({'whole milk'}),
0.562992125984252)
(frozenset({'yogurt', 'root vegetables'}), frozenset({'other
vegetables'}), 0.5)
(frozenset({'other vegetables', 'whipped/sour cream'}),
frozenset({'whole milk'}), 0.5070422535211268)
(frozenset({'yogurt', 'whipped/sour cream'}), frozenset({'whole
milk'}), 0.5245098039215685)
```

```
(frozenset({'domestic eggs', 'other vegetables'}), frozenset({'whole
milk'}), 0.5525114155251142)
```

```python
import matplotlib.pyplot as plt

# Define the support and confidence levels to test
support_levels = [0.02, 0.03, 0.04, 0.05]
confidence_levels = [0.1, 0.2, 0.3, 0.4, 0.5]

# Prepare the figure
plt.figure(figsize=(10, 8))
plt.title('Number of Rules vs. Confidence Level for Different Support
Values')
plt.xlabel('Confidence Level')
plt.ylabel('Number of Rules')

# For each support level, generate rules and count them for each
confidence level
for support in support_levels:
    rule_counts = []  # to hold the count of rules for each confidence
level

    # Get the frequent itemsets and support data for the current
support level
    frequent_itemsets, support_data = fpgrowth(dataset,
min_support=support, verbose=False)

    for confidence in confidence_levels:
        # Generate rules using the frequent itemsets and support data
for the current support level
        rules = generate_rules(frequent_itemsets, support_data,
min_confidence=confidence, verbose=False)
        rule_count = len(rules)
        rule_counts.append(rule_count)

        # Annotate the data points with the exact number of rules
        plt.annotate(f'{rule_count}', (confidence, rule_count),
textcoords="offset points", xytext=(0,10), ha='center')

    # Plot the results
    plt.plot(confidence_levels, rule_counts, marker='o',
label=f'Support {support*100}%')

# Add legend
plt.legend()

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



By lowering the minimum support value in the FP-growth algorithm, more item combinations are considered frequent, revealing a wider range of purchasing patterns among consumers. This adjustment allows for the identification of both common and less frequent item sets that are significant for understanding customer behavior.

Keeping the minimum confidence constant at 0.5 highlights only the strongest associations between items, such as yogurt and vegetables often leading to the purchase of whole milk. These findings provide insights into the most reliable buying patterns observed in the data.

The results showcase common shopping habits, indicating that items like whole milk, vegetables, and yogurt are often bought together. Such information can guide retailers in creating effective cross-selling strategies and optimizing store layouts to enhance customer experiences.

Adjusting the minimum support threshold uncovers a spectrum of consumer behaviors, from widely observed to niche patterns. This knowledge is crucial for retailers aiming to implement targeted marketing strategies, improve product placement, and ultimately, cater to diverse customer preferences.

```python
# Define your support threshold
min_support = 0.02  # 2% support
min_confidence = 0.3  # minimum confidence for the rules

# Generate frequent itemsets using FP-growth
frequent_itemsets, support_data = fpgrowth(dataset,
min_support=min_support, include_support=True, verbose=True)

# Generate association rules from the frequent itemsets
rules = generate_rules(frequent_itemsets, support_data,
min_confidence=min_confidence, verbose=True)

# Print out the rules
for rule in rules:
    print(rule)

#print(support_data)
```

```
{citrus fruit}:  sup = 0.083
{whole milk, citrus fruit}:  sup = 0.031
{yogurt, citrus fruit}:  sup = 0.022
{other vegetables, citrus fruit}:  sup = 0.029
{margarine}:  sup = 0.059
{whole milk, margarine}:  sup = 0.024
{yogurt}:  sup = 0.14
{whole milk, yogurt}:  sup = 0.056
{soda, yogurt}:  sup = 0.027
{rolls/buns, yogurt}:  sup = 0.034
{other vegetables, yogurt}:  sup = 0.043
{whole milk, other vegetables, yogurt}:  sup = 0.022
{tropical fruit}:  sup = 0.105
{yogurt, tropical fruit}:  sup = 0.029
{other vegetables, tropical fruit}:  sup = 0.036
{whole milk, tropical fruit}:  sup = 0.042
{rolls/buns, tropical fruit}:  sup = 0.025
{root vegetables, tropical fruit}:  sup = 0.021
{soda, tropical fruit}:  sup = 0.021
{coffee}:  sup = 0.058
{whole milk}:  sup = 0.256
{pip fruit}:  sup = 0.076
{whole milk, pip fruit}:  sup = 0.03
{tropical fruit, pip fruit}:  sup = 0.02
{other vegetables, pip fruit}:  sup = 0.026
{cream cheese}:  sup = 0.04
{other vegetables}:  sup = 0.193
{whole milk, other vegetables}:  sup = 0.075
{long life bakery product}:  sup = 0.037
{butter}:  sup = 0.055
{whole milk, butter}:  sup = 0.028
{other vegetables, butter}:  sup = 0.02
```

```
{rolls/buns}:  sup = 0.184
{other vegetables, rolls/buns}:  sup = 0.043
{whole milk, rolls/buns}:  sup = 0.057
{bottled beer}:  sup = 0.081
{whole milk, bottled beer}:  sup = 0.02
{UHT-milk}:  sup = 0.033
{bottled water}:  sup = 0.111
{other vegetables, bottled water}:  sup = 0.025
{whole milk, bottled water}:  sup = 0.034
{yogurt, bottled water}:  sup = 0.023
{rolls/buns, bottled water}:  sup = 0.024
{soda, bottled water}:  sup = 0.029
{chocolate}:  sup = 0.05
{white bread}:  sup = 0.042
{curd}:  sup = 0.053
{whole milk, curd}:  sup = 0.026
{beef}:  sup = 0.052
{whole milk, beef}:  sup = 0.021
{soda}:  sup = 0.174
{rolls/buns, soda}:  sup = 0.038
{whole milk, soda}:  sup = 0.04
{other vegetables, soda}:  sup = 0.033
{frankfurter}:  sup = 0.059
{whole milk, frankfurter}:  sup = 0.021
{chicken}:  sup = 0.043
{newspapers}:  sup = 0.08
{whole milk, newspapers}:  sup = 0.027
{fruit/vegetable juice}:  sup = 0.072
{other vegetables, fruit/vegetable juice}:  sup = 0.021
{whole milk, fruit/vegetable juice}:  sup = 0.027
{sugar}:  sup = 0.034
{specialty bar}:  sup = 0.027
{pastry}:  sup = 0.089
{soda, pastry}:  sup = 0.021
{whole milk, pastry}:  sup = 0.033
{rolls/buns, pastry}:  sup = 0.021
{other vegetables, pastry}:  sup = 0.023
{butter milk}:  sup = 0.028
{root vegetables}:  sup = 0.109
{other vegetables, root vegetables}:  sup = 0.047
{whole milk, other vegetables, root vegetables}:  sup = 0.023
{rolls/buns, root vegetables}:  sup = 0.024
{whole milk, root vegetables}:  sup = 0.049
{yogurt, root vegetables}:  sup = 0.026
{waffles}:  sup = 0.038
{salty snack}:  sup = 0.038
{candy}:  sup = 0.03
{canned beer}:  sup = 0.078
{sausage}:  sup = 0.094
```

```
{rolls/buns, sausage}:  sup = 0.031
{soda, sausage}:  sup = 0.024
{whole milk, sausage}:  sup = 0.03
{other vegetables, sausage}:  sup = 0.027
{shopping bags}:  sup = 0.099
{soda, shopping bags}:  sup = 0.025
{whole milk, shopping bags}:  sup = 0.025
{other vegetables, shopping bags}:  sup = 0.023
{brown bread}:  sup = 0.065
{whole milk, brown bread}:  sup = 0.025
{beverages}:  sup = 0.026
{napkins}:  sup = 0.052
{hamburger meat}:  sup = 0.033
{hygiene articles}:  sup = 0.033
{whipped/sour cream}:  sup = 0.072
{whole milk, whipped/sour cream}:  sup = 0.032
{other vegetables, whipped/sour cream}:  sup = 0.029
{yogurt, whipped/sour cream}:  sup = 0.021
{pork}:  sup = 0.058
{whole milk, pork}:  sup = 0.022
{other vegetables, pork}:  sup = 0.022
{berries}:  sup = 0.033
{grapes}:  sup = 0.022
{dessert}:  sup = 0.037
{domestic eggs}:  sup = 0.063
{whole milk, domestic eggs}:  sup = 0.03
{other vegetables, domestic eggs}:  sup = 0.022
{misc. beverages}:  sup = 0.028
{hard cheese}:  sup = 0.025
{cat food}:  sup = 0.023
{ham}:  sup = 0.026
{oil}:  sup = 0.028
{chewing gum}:  sup = 0.021
{ice cream}:  sup = 0.025
{frozen vegetables}:  sup = 0.048
{whole milk, frozen vegetables}:  sup = 0.02
{specialty chocolate}:  sup = 0.03
{frozen meals}:  sup = 0.028
{onions}:  sup = 0.031
{sliced cheese}:  sup = 0.025
{meat}:  sup = 0.026
{citrus fruit} ---> {whole milk}:  conf = 0.369, sup = 0.031
{citrus fruit} ---> {other vegetables}:  conf = 0.349, sup = 0.029
{margarine} ---> {whole milk}:  conf = 0.413, sup = 0.024
{yogurt} ---> {whole milk}:  conf = 0.402, sup = 0.056
{yogurt} ---> {other vegetables}:  conf = 0.311, sup = 0.043
{tropical fruit} ---> {other vegetables}:  conf = 0.342, sup = 0.036
{tropical fruit} ---> {whole milk}:  conf = 0.403, sup = 0.042
{pip fruit} ---> {whole milk}:  conf = 0.398, sup = 0.03
```

```
{pip fruit} ---> {other vegetables}:  conf = 0.345, sup = 0.026
{other vegetables} ---> {whole milk}:  conf = 0.387, sup = 0.075
{butter} ---> {whole milk}:  conf = 0.497, sup = 0.028
{butter} ---> {other vegetables}:  conf = 0.361, sup = 0.02
{rolls/buns} ---> {whole milk}:  conf = 0.308, sup = 0.057
{bottled water} ---> {whole milk}:  conf = 0.311, sup = 0.034
{curd} ---> {whole milk}:  conf = 0.49, sup = 0.026
{beef} ---> {whole milk}:  conf = 0.405, sup = 0.021
{frankfurter} ---> {whole milk}:  conf = 0.348, sup = 0.021
{newspapers} ---> {whole milk}:  conf = 0.343, sup = 0.027
{fruit/vegetable juice} ---> {whole milk}:  conf = 0.368, sup = 0.027
{pastry} ---> {whole milk}:  conf = 0.374, sup = 0.033
{root vegetables} ---> {other vegetables}:  conf = 0.435, sup = 0.047
{root vegetables} ---> {whole milk}:  conf = 0.449, sup = 0.049
{sausage} ---> {rolls/buns}:  conf = 0.326, sup = 0.031
{sausage} ---> {whole milk}:  conf = 0.318, sup = 0.03
{brown bread} ---> {whole milk}:  conf = 0.389, sup = 0.025
{whipped/sour cream} ---> {whole milk}:  conf = 0.45, sup = 0.032
{whipped/sour cream} ---> {other vegetables}:  conf = 0.403, sup =
0.029
{pork} ---> {whole milk}:  conf = 0.384, sup = 0.022
{pork} ---> {other vegetables}:  conf = 0.376, sup = 0.022
{domestic eggs} ---> {whole milk}:  conf = 0.473, sup = 0.03
{domestic eggs} ---> {other vegetables}:  conf = 0.351, sup = 0.022
{frozen vegetables} ---> {whole milk}:  conf = 0.425, sup = 0.02
{yogurt, whole milk} ---> {other vegetables}:  conf = 0.397, sup =
0.022
{yogurt, other vegetables} ---> {whole milk}:  conf = 0.513, sup =
0.022
{root vegetables, whole milk} ---> {other vegetables}:  conf = 0.474,
sup = 0.023
{other vegetables, whole milk} ---> {root vegetables}:  conf = 0.31,
sup = 0.023
{other vegetables, root vegetables} ---> {whole milk}:  conf = 0.489,
sup = 0.023
(frozenset({'citrus fruit'}), frozenset({'whole milk'}),
0.36855036855036855)
(frozenset({'citrus fruit'}), frozenset({'other vegetables'}),
0.34889434889434895)
(frozenset({'margarine'}), frozenset({'whole milk'}),
0.41319444444444444)
(frozenset({'yogurt'}), frozenset({'whole milk'}),
0.40160349854227406)
(frozenset({'yogurt'}), frozenset({'other vegetables'}),
0.3112244897959184)
(frozenset({'tropical fruit'}), frozenset({'other vegetables'}),
0.34205426356589147)
(frozenset({'tropical fruit'}), frozenset({'whole milk'}),
0.40310077519379844)
```

```
(frozenset({'pip fruit'}), frozenset({'whole milk'}),
0.3978494623655914)
(frozenset({'pip fruit'}), frozenset({'other vegetables'}),
0.3454301075268817)
(frozenset({'other vegetables'}), frozenset({'whole milk'}),
0.38675775091960063)
(frozenset({'butter'}), frozenset({'whole milk'}), 0.4972477064220184)
(frozenset({'butter'}), frozenset({'other vegetables'}),
0.3614678899082569)
(frozenset({'rolls/buns'}), frozenset({'whole milk'}),
0.30790491984521834)
(frozenset({'bottled water'}), frozenset({'whole milk'}),
0.31094756209751606)
(frozenset({'curd'}), frozenset({'whole milk'}), 0.4904580152671756)
(frozenset({'beef'}), frozenset({'whole milk'}), 0.4050387596899225)
(frozenset({'frankfurter'}), frozenset({'whole milk'}),
0.3482758620689655)
(frozenset({'newspapers'}), frozenset({'whole milk'}),
0.34267515923566877)
(frozenset({'fruit/vegetable juice'}), frozenset({'whole milk'}),
0.36849507735583686)
(frozenset({'pastry'}), frozenset({'whole milk'}), 0.3737142857142857)
(frozenset({'root vegetables'}), frozenset({'other vegetables'}),
0.43470149253731344)
(frozenset({'root vegetables'}), frozenset({'whole milk'}),
0.44869402985074625)
(frozenset({'sausage'}), frozenset({'rolls/buns'}),
0.3257575757575758)
(frozenset({'sausage'}), frozenset({'whole milk'}),
0.3181818181818182)
(frozenset({'brown bread'}), frozenset({'whole milk'}),
0.3887147335423197)
(frozenset({'whipped/sour cream'}), frozenset({'whole milk'}),
0.449645390070922)
(frozenset({'whipped/sour cream'}), frozenset({'other vegetables'}),
0.40283687943262414)
(frozenset({'pork'}), frozenset({'whole milk'}), 0.3844797178130512)
(frozenset({'pork'}), frozenset({'other vegetables'}),
0.37566137566137564)
(frozenset({'domestic eggs'}), frozenset({'whole milk'}),
0.47275641025641024)
(frozenset({'domestic eggs'}), frozenset({'other vegetables'}),
0.35096153846153844)
(frozenset({'frozen vegetables'}), frozenset({'whole milk'}),
0.4249471458773784)
(frozenset({'yogurt', 'whole milk'}), frozenset({'other vegetables'}),
0.39745916515426494)
(frozenset({'yogurt', 'other vegetables'}), frozenset({'whole milk'}),
0.5128805620608898)
```

```
(frozenset({'root vegetables', 'whole milk'}), frozenset({'other vegetables'}), 0.47401247401247404)
(frozenset({'other vegetables', 'whole milk'}), frozenset({'root vegetables'}), 0.30978260869565216)
(frozenset({'other vegetables', 'root vegetables'}), frozenset({'whole milk'}), 0.4892703862660944)
```

```
# Define a function to calculate the interest factor
def calculate_interest_factor(rule, support_data):
    """
    Interest Factor is defined as |confidence - expected_confidence|.
    Expected confidence is the support of the consequent divided by the total number of transactions.
    """
    support_antecedent = support_data[rule[0]]
    support_consequent = support_data[rule[1]]
    support_both = support_data[rule[0] | rule[1]]
    confidence = support_both / support_antecedent
    expected_confidence = support_consequent
    interest_factor = abs(confidence - expected_confidence)
    return interest_factor


# Update each rule tuple with the interest factor
rules_with_interest = []
for rule in rules:
    # Calculate the interest factor for the rule
    interest_factor = calculate_interest_factor(rule, support_data)
    # Append the interest factor to the rule's tuple
    updated_rule = rule + (interest_factor,)
    rules_with_interest.append(updated_rule)

# Print out the rules
for rule in rules_with_interest:
    print(rule)
```

```
(frozenset({'citrus fruit'}), frozenset({'whole milk'}), 0.36855036855036855, 0.11303435431549308)
(frozenset({'citrus fruit'}), frozenset({'other vegetables'}), 0.34889434889434895, 0.15540172052627574)
(frozenset({'margarine'}), frozenset({'whole milk'}), 0.4131944444444444, 0.15767843020956895)
(frozenset({'yogurt'}), frozenset({'whole milk'}), 0.40160349854227406, 0.1460874843073986)
(frozenset({'yogurt'}), frozenset({'other vegetables'}), 0.3112244897959184, 0.11773186142784517)
(frozenset({'tropical fruit'}), frozenset({'other vegetables'}), 0.34205426356589147, 0.14856163519781826)
(frozenset({'tropical fruit'}), frozenset({'whole milk'}),
```

```
0.40310077519379844, 0.14758476095892298)
(frozenset({'pip fruit'}), frozenset({'whole milk'}),
0.3978494623655914, 0.1423334481307159)
(frozenset({'pip fruit'}), frozenset({'other vegetables'}),
0.3454301075268817, 0.15193747915880848)
(frozenset({'other vegetables'}), frozenset({'whole milk'}),
0.38675775091960063, 0.13124173668472516)
(frozenset({'butter'}), frozenset({'whole milk'}), 0.4972477064220184,
0.2417316921871429)
(frozenset({'butter'}), frozenset({'other vegetables'}),
0.3614678899082569, 0.1679752615401837)
(frozenset({'rolls/buns'}), frozenset({'whole milk'}),
0.30790491984521834, 0.05238890561034287)
(frozenset({'bottled water'}), frozenset({'whole milk'}),
0.31094756209751606, 0.055431547862640596)
(frozenset({'curd'}), frozenset({'whole milk'}), 0.4904580152671756,
0.23494200103230012)
(frozenset({'beef'}), frozenset({'whole milk'}), 0.4050387596899225,
0.149522745455047)
(frozenset({'frankfurter'}), frozenset({'whole milk'}),
0.3482758620689655, 0.09275984783409003)
(frozenset({'newspapers'}), frozenset({'whole milk'}),
0.34267515923566877, 0.0871591450007933)
(frozenset({'fruit/vegetable juice'}), frozenset({'whole milk'}),
0.36849507735583686, 0.1129790631209614)
(frozenset({'pastry'}), frozenset({'whole milk'}), 0.3737142857142857,
0.11819827147941026)
(frozenset({'root vegetables'}), frozenset({'other vegetables'}),
0.43470149253731344, 0.24120886416924023)
(frozenset({'root vegetables'}), frozenset({'whole milk'}),
0.44869402985074625, 0.19317801561587078)
(frozenset({'sausage'}), frozenset({'rolls/buns'}),
0.3257575757575758, 0.14182264947389506)
(frozenset({'sausage'}), frozenset({'whole milk'}),
0.3181818181818182, 0.06266580394694271)
(frozenset({'brown bread'}), frozenset({'whole milk'}),
0.3887147335423197, 0.13319871930744426)
(frozenset({'whipped/sour cream'}), frozenset({'whole milk'}),
0.449645390070922, 0.19412937583604656)
(frozenset({'whipped/sour cream'}), frozenset({'other vegetables'}),
0.40283687943262414, 0.20934425106455093)
(frozenset({'pork'}), frozenset({'whole milk'}), 0.3844797178130512,
0.12896370357817571)
(frozenset({'pork'}), frozenset({'other vegetables'}),
0.37566137566137564, 0.18216874729330243)
(frozenset({'domestic eggs'}), frozenset({'whole milk'}),
0.47275641025641024, 0.21724039602153478)
(frozenset({'domestic eggs'}), frozenset({'other vegetables'}),
0.35096153846153844, 0.15746891009346523)
```

```
(frozenset({'frozen vegetables'}), frozenset({'whole milk'}),
0.4249471458773784, 0.16943113164250295)
(frozenset({'yogurt', 'whole milk'}), frozenset({'other vegetables'}),
0.39745916515426494, 0.20396653678619173)
(frozenset({'yogurt', 'other vegetables'}), frozenset({'whole milk'}),
0.5128805620608898, 0.25736454782601437)
(frozenset({'root vegetables', 'whole milk'}), frozenset({'other
vegetables'}), 0.47401247401247404, 0.2805198456444008)
(frozenset({'other vegetables', 'whole milk'}), frozenset({'root
vegetables'}), 0.30978260869565216, 0.2007841338608784)
(frozenset({'other vegetables', 'root vegetables'}), frozenset({'whole
milk'}), 0.4892703862660944, 0.23375437203121896)

# Sort the rules by support, confidence, and interest factor,
respectively
# Here we need to ensure we're using the correct indices for support
and confidence
rules_by_support = sorted(rules_with_interest, key=lambda r:
support_data[r[0] | r[1]], reverse=True)
rules_by_confidence = sorted(rules_with_interest, key=lambda r: r[2],
reverse=True)
rules_by_interest_factor = sorted(rules_with_interest, key=lambda r:
r[3], reverse=True)

# Print the top-5 rules for each sorted list
print("Top 5 Rules by Support:")
for rule in rules_by_support[:5]:
    print(rule)

print("\nTop 5 Rules by Confidence:")
for rule in rules_by_confidence[:5]:
    print(rule)

print("\nTop 5 Rules by Interest Factor:")
for rule in rules_by_interest_factor[:5]:
    print(rule)

Top 5 Rules by Support:
(frozenset({'other vegetables'}), frozenset({'whole milk'}),
0.38675775091960063, 0.13124173668472516)
(frozenset({'rolls/buns'}), frozenset({'whole milk'}),
0.30790491984521834, 0.05238890561034287)
(frozenset({'yogurt'}), frozenset({'whole milk'}),
0.40160349854227406, 0.1460874843073986)
(frozenset({'root vegetables'}), frozenset({'whole milk'}),
0.44869402985074625, 0.19317801561587078)
(frozenset({'root vegetables'}), frozenset({'other vegetables'}),
0.43470149253731344, 0.24120886416924023)

Top 5 Rules by Confidence:
```

```
(frozenset({'yogurt', 'other vegetables'}), frozenset({'whole milk'}),
0.5128805620608898, 0.25736454782601437)
(frozenset({'butter'}), frozenset({'whole milk'}), 0.4972477064220184,
0.2417316921871429)
(frozenset({'curd'}), frozenset({'whole milk'}), 0.4904580152671756,
0.23494200103230012)
(frozenset({'other vegetables', 'root vegetables'}), frozenset({'whole
milk'}), 0.4892703862660944, 0.23375437203121896)
(frozenset({'root vegetables', 'whole milk'}), frozenset({'other
vegetables'}), 0.47401247401247404, 0.2805198456444008)

Top 5 Rules by Interest Factor:
(frozenset({'root vegetables', 'whole milk'}), frozenset({'other
vegetables'}), 0.47401247401247404, 0.2805198456444008)
(frozenset({'yogurt', 'other vegetables'}), frozenset({'whole milk'}),
0.5128805620608898, 0.25736454782601437)
(frozenset({'butter'}), frozenset({'whole milk'}), 0.4972477064220184,
0.2417316921871429)
(frozenset({'root vegetables'}), frozenset({'other vegetables'}),
0.43470149253731344, 0.24120886416924023)
(frozenset({'curd'}), frozenset({'whole milk'}), 0.4904580152671756,
0.23494200103230012)
```

```python
# Extract just the antecedent and consequent parts of the rules for
comparison
top5_support_rules = {(rule[0], rule[1]) for rule in
rules_by_support[:5]}
top5_confidence_rules = {(rule[0], rule[1]) for rule in
rules_by_confidence[:5]}
top5_interest_factor_rules = {(rule[0], rule[1]) for rule in
rules_by_interest_factor[:5]}

# Find common rules based on antecedent and consequent
common_rule_bases = top5_support_rules & top5_interest_factor_rules

# Now find the full rule info (with support and interest factor) for
the common rules
common_rules_full_info = [rule for rule in rules_with_interest if
(rule[0], rule[1]) in common_rule_bases]

# Print the common rules with full information
print("\nCommon Rules in Top 5 of support and interest factor:")
for rule in common_rules_full_info:
    print(rule)
```

```
Common Rules in Top 5 of support and interest factor:
(frozenset({'root vegetables'}), frozenset({'other vegetables'}),
0.43470149253731344, 0.24120886416924023)
```

```python
# Find common rules based on antecedent and consequent
common_rule_bases = top5_confidence_rules & top5_interest_factor_rules

# Now find the full rule info (with confidence and interest factor)
for the common rules
common_rules_full_info = [rule for rule in rules_with_interest if
(rule[0], rule[1]) in common_rule_bases]

# Print the common rules with full information
print("\nCommon Rules in Top 5 of confidence and interest factor:")
for rule in common_rules_full_info:
    print(rule)
```

```
Common Rules in Top 5 of confidence and interest factor:
(frozenset({'butter'}), frozenset({'whole milk'}), 0.4972477064220184,
0.2417316921871429)
(frozenset({'curd'}), frozenset({'whole milk'}), 0.4904580152671756,
0.23494200103230012)
(frozenset({'yogurt', 'other vegetables'}), frozenset({'whole milk'}),
0.5128805620608898, 0.25736454782601437)
(frozenset({'root vegetables', 'whole milk'}), frozenset({'other
vegetables'}), 0.47401247401247404, 0.2805198456444008)
```

```python
# Find common rules based on antecedent and consequent
common_rule_bases = top5_support_rules & top5_confidence_rules

# Now find the full rule info (with support and confidence) for the
common rules
common_rules_full_info = [rule for rule in rules_with_interest if
(rule[0], rule[1]) in common_rule_bases]

# Print the common rules with full information
print("\nCommon Rules in Top 5 of support and confidence:")
for rule in common_rules_full_info:
    print(rule)
```

```
Common Rules in Top 5 of support and confidence:
```

```python
# Find common rules based on antecedent and consequent
common_rule_bases = top5_support_rules & top5_confidence_rules &
top5_interest_factor_rules

# Now find the full rule info (with support, confidence, and interest
factor) for the common rules
common_rules_full_info = [rule for rule in rules_with_interest if
(rule[0], rule[1]) in common_rule_bases]

# Print the common rules with full information
print("\nCommon Rules in Top 5:")
```

```python
for rule in common_rules_full_info:
    print(rule)
```

```
Common Rules in Top 5:
```

There are no common rules across all three top 5 lists, but there are four common rules between top 5 confidence and interest factor and 1 common rule between top 5 support and interest factor