DATA MINING

FP Growth algorithm

Exercise number: **Date: 30.07.2024**

Aim: To implement FP growth algorithm

Q1. FP growth algorithm for a given dataset.

```python
import pandas as pd
from mlxtend.frequent_patterns import fpgrowth, association_rules
transactions = [
    ['Bread', 'Milk', 'Beer'],
    ['Bread', 'Diapers', 'Milk'],
    ['Milk', 'Diapers', 'Bread'],
    ['Bread', 'Milk', 'Diapers', 'Beer'],
    ['Diapers', 'Beer']
]

transaction_df = pd.DataFrame(transactions)
one_hot = transaction_df.stack().groupby(level=0).value_counts().unstack().fillna(0).astype(int)
min_support = 0.4
frequent_itemsets = fpgrowth(one_hot, min_support=min_support, use_colnames=True)
print("Frequent Itemsets:")
print(frequent_itemsets)
min_confidence = 0.7
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=min_confidence)
print("\nAssociation Rules:")
print(rules)
```

Output:

```
Frequent Itemsets:
    support                 itemsets
0       0.8                   (Milk)
1       0.8                  (Bread)
2       0.6                   (Beer)
3       0.8                (Diapers)
4       0.8            (Bread, Milk)
5       0.6         (Bread, Diapers)
6       0.6   (Bread, Diapers, Milk)
7       0.4            (Beer, Bread)
8       0.4             (Beer, Milk)
9       0.4          (Beer, Diapers)
10      0.4      (Beer, Bread, Milk)
11      0.6          (Diapers, Milk)
```

```
Association Rules:
        antecedents       consequents  antecedent support  \
0           (Bread)            (Milk)                 0.8
1            (Milk)           (Bread)                 0.8
2           (Bread)         (Diapers)                 0.8
3         (Diapers)           (Bread)                 0.8
4  (Bread, Diapers)            (Milk)                 0.6
5     (Bread, Milk)         (Diapers)                 0.8
6   (Diapers, Milk)           (Bread)                 0.6
7           (Bread)   (Diapers, Milk)                 0.8
8         (Diapers)     (Bread, Milk)                 0.8
9            (Milk)  (Bread, Diapers)                 0.8
10    (Beer, Bread)            (Milk)                 0.4
11     (Beer, Milk)           (Bread)                 0.4
12        (Diapers)            (Milk)                 0.8
13           (Milk)         (Diapers)                 0.8

    consequent support  support  confidence    lift  leverage  conviction
0                  0.8      0.8        1.00  1.2500      0.16         inf
1                  0.8      0.8        1.00  1.2500      0.16         inf
2                  0.8      0.6        0.75  0.9375     -0.04         0.8
3                  0.8      0.6        0.75  0.9375     -0.04         0.8
4                  0.8      0.6        1.00  1.2500      0.12         inf
5                  0.8      0.6        0.75  0.9375     -0.04         0.8
6                  0.8      0.6        1.00  1.2500      0.12         inf
7                  0.6      0.6        0.75  1.2500      0.12         1.6
8                  0.8      0.6        0.75  0.9375     -0.04         0.8
9                  0.6      0.6        0.75  1.2500      0.12         1.6
10                 0.8      0.4        1.00  1.2500      0.08         inf
11                 0.8      0.4        1.00  1.2500      0.08         inf
12                 0.8      0.6        0.75  0.9375     -0.04         0.8
13                 0.8      0.6        0.75  0.9375     -0.04         0.8
```

Thaarugeshwari KS
22i270

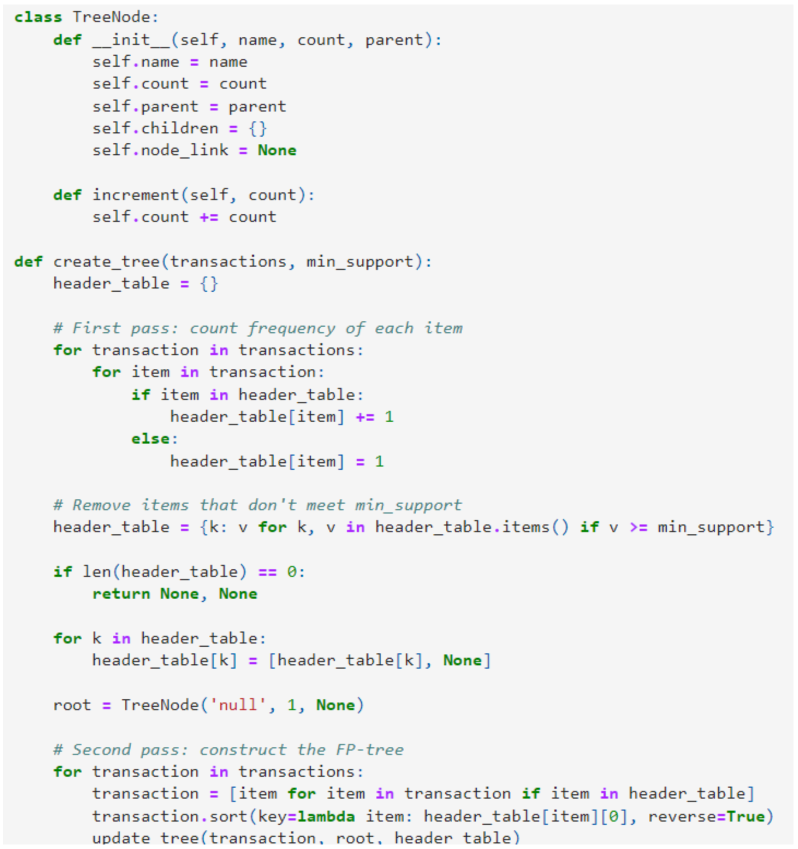
Q2. FP growth algorithm for a given dataset(CSV file)

```python
import pandas as pd
from mlxtend.frequent_patterns import fpgrowth, association_rules
df = pd.read_csv('input.csv')
print("Original Dataset:")
print(df.head())
item_columns = ['Bread', 'Milk', 'Beer', 'Diapers']
item_df = df[item_columns]
print("\nItem Columns:")
print(item_df.head())
min_support = 0.4
frequent_itemsets = fpgrowth(item_df, min_support=min_support, use_colnames=True)
print("\nFrequent Itemsets:")
print(frequent_itemsets)
min_confidence = 0.7
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=min_confidence)

print("\nAssociation Rules:")
print(rules)
```

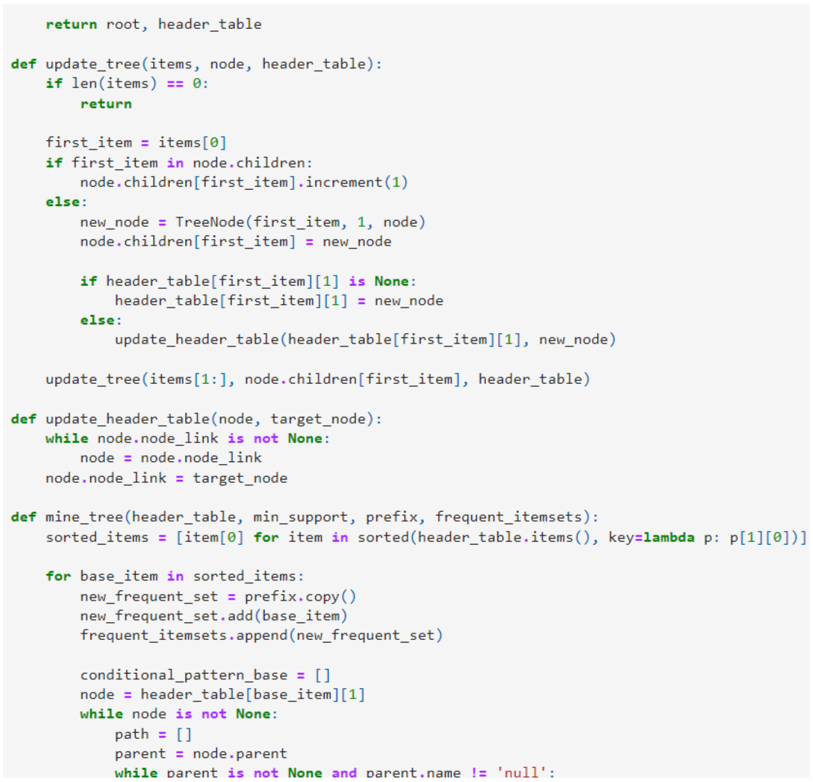
Output:

```
Frequent Itemsets:
    support                 itemsets
0       0.8                  (Bread)
1       0.6                   (Beer)
2       0.6                   (Milk)
3       0.8                (Diapers)
4       0.6         (Bread, Diapers)
5       0.4            (Beer, Bread)
6       0.4          (Beer, Diapers)
7       0.6            (Bread, Milk)
8       0.4             (Beer, Milk)
9       0.4          (Diapers, Milk)
10      0.4      (Beer, Bread, Milk)
11      0.4   (Bread, Diapers, Milk)

Association Rules:
        antecedents  consequents  antecedent support  consequent support  \
0           (Bread)    (Diapers)                 0.8                 0.8
1         (Diapers)      (Bread)                 0.8                 0.8
2           (Bread)       (Milk)                 0.8                 0.6
3            (Milk)      (Bread)                 0.6                 0.8
4     (Beer, Bread)       (Milk)                 0.4                 0.6
5      (Beer, Milk)      (Bread)                 0.4                 0.8
6   (Diapers, Milk)      (Bread)                 0.4                 0.8

   support  confidence      lift  leverage  conviction  zhangs_metric
0      0.6        0.75  0.937500     -0.04         0.8      -0.250000
1      0.6        0.75  0.937500     -0.04         0.8      -0.250000
2      0.6        0.75  1.250000      0.12         1.6       1.000000
3      0.6        1.00  1.250000      0.12         inf       0.500000
4      0.4        1.00  1.666667      0.16         inf       0.666667
5      0.4        1.00  1.250000      0.08         inf       0.333333
6      0.4        1.00  1.250000      0.08         inf       0.333333
```

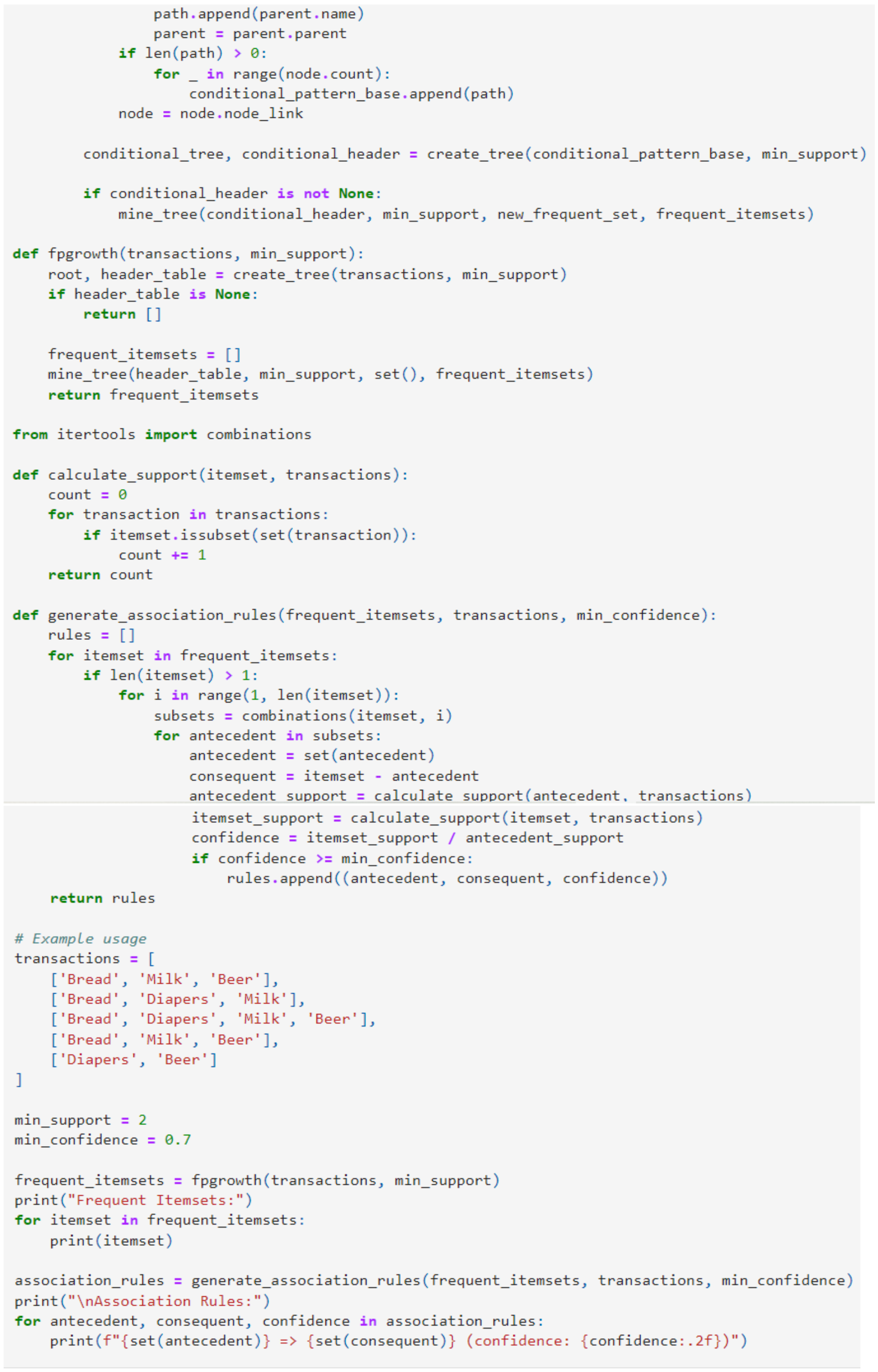
Q3. FP growth algorithm for a given dataset without built-in functions.



```python
class TreeNode:
    def __init__(self, name, count, parent):
        self.name = name
        self.count = count
        self.parent = parent
        self.children = {}
        self.node_link = None

    def increment(self, count):
        self.count += count

def create_tree(transactions, min_support):
    header_table = {}

    # First pass: count frequency of each item
    for transaction in transactions:
        for item in transaction:
            if item in header_table:
                header_table[item] += 1
            else:
                header_table[item] = 1

    # Remove items that don't meet min_support
    header_table = {k: v for k, v in header_table.items() if v >= min_support}

    if len(header_table) == 0:
        return None, None

    for k in header_table:
        header_table[k] = [header_table[k], None]

    root = TreeNode('null', 1, None)

    # Second pass: construct the FP-tree
    for transaction in transactions:
        transaction = [item for item in transaction if item in header_table]
        transaction.sort(key=lambda item: header_table[item][0], reverse=True)
        update_tree(transaction, root, header_table)
```



```python
    return root, header_table

def update_tree(items, node, header_table):
    if len(items) == 0:
        return

    first_item = items[0]
    if first_item in node.children:
        node.children[first_item].increment(1)
    else:
        new_node = TreeNode(first_item, 1, node)
        node.children[first_item] = new_node

        if header_table[first_item][1] is None:
            header_table[first_item][1] = new_node
        else:
            update_header_table(header_table[first_item][1], new_node)

    update_tree(items[1:], node.children[first_item], header_table)

def update_header_table(node, target_node):
    while node.node_link is not None:
        node = node.node_link
    node.node_link = target_node

def mine_tree(header_table, min_support, prefix, frequent_itemsets):
    sorted_items = [item[0] for item in sorted(header_table.items(), key=lambda p: p[1][0])]

    for base_item in sorted_items:
        new_frequent_set = prefix.copy()
        new_frequent_set.add(base_item)
        frequent_itemsets.append(new_frequent_set)

        conditional_pattern_base = []
        node = header_table[base_item][1]
        while node is not None:
            path = []
            parent = node.parent
            while parent is not None and parent.name != 'null':
```



```
                path.append(parent.name)
                parent = parent.parent
            if len(path) > 0:
                for _ in range(node.count):
                    conditional_pattern_base.append(path)
            node = node.node_link

        conditional_tree, conditional_header = create_tree(conditional_pattern_base, min_support)

        if conditional_header is not None:
            mine_tree(conditional_header, min_support, new_frequent_set, frequent_itemsets)

def fpgrowth(transactions, min_support):
    root, header_table = create_tree(transactions, min_support)
    if header_table is None:
        return []

    frequent_itemsets = []
    mine_tree(header_table, min_support, set(), frequent_itemsets)
    return frequent_itemsets

from itertools import combinations

def calculate_support(itemset, transactions):
    count = 0
    for transaction in transactions:
        if itemset.issubset(set(transaction)):
            count += 1
    return count

def generate_association_rules(frequent_itemsets, transactions, min_confidence):
    rules = []
    for itemset in frequent_itemsets:
        if len(itemset) > 1:
            for i in range(1, len(itemset)):
                subsets = combinations(itemset, i)
                for antecedent in subsets:
                    antecedent = set(antecedent)
                    consequent = itemset - antecedent
                    antecedent_support = calculate_support(antecedent, transactions)
                    itemset_support = calculate_support(itemset, transactions)
                    confidence = itemset_support / antecedent_support
                    if confidence >= min_confidence:
                        rules.append((antecedent, consequent, confidence))
    return rules

# Example usage
transactions = [
    ['Bread', 'Milk', 'Beer'],
    ['Bread', 'Diapers', 'Milk'],
    ['Bread', 'Diapers', 'Milk', 'Beer'],
    ['Bread', 'Milk', 'Beer'],
    ['Diapers', 'Beer']
]

min_support = 2
min_confidence = 0.7

frequent_itemsets = fpgrowth(transactions, min_support)
print("Frequent Itemsets:")
for itemset in frequent_itemsets:
    print(itemset)

association_rules = generate_association_rules(frequent_itemsets, transactions, min_confidence)
print("\nAssociation Rules:")
for antecedent, consequent, confidence in association_rules:
    print(f"{set(antecedent)} => {set(consequent)} (confidence: {confidence:.2f})")
```

Output:

```
Frequent Itemsets:
{'Diapers'}
{'Diapers', 'Milk'}
{'Bread', 'Diapers'}
{'Bread', 'Diapers', 'Milk'}
{'Beer', 'Diapers'}
{'Bread'}
{'Milk'}
{'Bread', 'Milk'}
{'Beer'}
{'Beer', 'Milk'}
{'Beer', 'Bread'}
{'Beer', 'Bread', 'Milk'}

Association Rules:
{'Bread', 'Diapers'} => {'Milk'} (confidence: 1.00)
{'Diapers', 'Milk'} => {'Bread'} (confidence: 1.00)
{'Bread'} => {'Milk'} (confidence: 1.00)
{'Milk'} => {'Bread'} (confidence: 1.00)
{'Beer'} => {'Milk'} (confidence: 0.75)
{'Milk'} => {'Beer'} (confidence: 0.75)
{'Beer'} => {'Bread'} (confidence: 0.75)
{'Bread'} => {'Beer'} (confidence: 0.75)
{'Beer'} => {'Bread', 'Milk'} (confidence: 0.75)
{'Bread'} => {'Beer', 'Milk'} (confidence: 0.75)
{'Milk'} => {'Beer', 'Bread'} (confidence: 0.75)
{'Beer', 'Bread'} => {'Milk'} (confidence: 1.00)
{'Beer', 'Milk'} => {'Bread'} (confidence: 1.00)
{'Bread', 'Milk'} => {'Beer'} (confidence: 0.75)
```

Result:

All code executed successfully.