SOURCE CODE

```
import numpy as np
from itertools import combinations


# Load a sample dataset
def load_dataset():
    return [
        ['bread', 'butter', 'milk'],
        ['bread', 'butter'],
        ['bread', 'milk'],
        ['bread', 'butter', 'milk', 'eggs'],
        ['bread', 'milk', 'eggs'],
        ['bread', 'butter', 'milk'],
        ['bread', 'butter', 'milk']
    ]
# Generate candidate itemsets of a specific length
def create_candidates(itemset, length):
    return set([i.union(j) for i in itemset for j in itemset if len(i.union(j)) == length])
# Scan the dataset to calculate support for candidates
def scan_dataset(dataset, candidates, min_support):
    counts = {}
    for transaction in dataset:
        for candidate in candidates:
            if candidate.issubset(transaction):
                counts[candidate] = counts.get(candidate, 0) + 1
    num_transactions = float(len(dataset))
    return {item: count / num_transactions for item, count in counts.items() if count / num_transactions >= min_support}
# Apriori algorithm to find frequent itemsets
def apriori(dataset, min_support=0.5):
    # Generate initial candidates
    items = set(item for transaction in dataset for item in transaction)
    candidates = [frozenset([item]) for item in items]
```

```
frequent_itemsets = []
    k = 1
    while candidates:
        # Scan dataset to determine support for candidates
        frequent_candidates = scan_dataset(dataset, candidates, min_support)
        frequent_itemsets.extend(frequent_candidates.keys())
        # Generate new candidates
        candidates = create_candidates(frequent_candidates.keys(), k + 1)
        k += 1
    return frequent_itemsets

# Generate association rules from frequent itemsets
def generate_association_rules(frequent_itemsets, dataset, min_confidence=0.7):
    rules = []
    for itemset in frequent_itemsets:
        if len(itemset) > 1:
            for antecedent in combinations(itemset, len(itemset) - 1):
                antecedent = frozenset(antecedent)
                consequent = itemset - antecedent
                # Calculate support for antecedent, consequent, and rule
                antecedent_support = sum(1 for transaction in dataset if antecedent.issubset(transaction)) / len(dataset)
                consequent_support = sum(1 for transaction in dataset if consequent.issubset(transaction)) / len(dataset)
                rule_support = sum(1 for transaction in dataset if itemset.issubset(transaction)) / len(dataset)

                # Calculate confidence and lift
                confidence = calculate_confidence(antecedent_support, rule_support)
                lift = calculate_lift(rule_support, antecedent_support, consequent_support)

                # Add rule if it meets the minimum confidence threshold
                if confidence >= min_confidence:
                    rules.append((antecedent, consequent, rule_support, confidence, lift))
    return rules

# Calculate confidence for a rule
def calculate_confidence(antecedent_support, rule_support):
```

```
    if antecedent_support == 0:
        return 0
    return rule_support / antecedent_support

# Calculate lift for a rule
def calculate_lift(rule_support, antecedent_support, consequent_support):
    if antecedent_support == 0 or consequent_support == 0:
        return 0
    return rule_support / (antecedent_support * consequent_support)

# Main function
def main():
    dataset = load_dataset()
    min_support = 0.5
    min_confidence = 0.7
    # Find frequent itemsets
    frequent_itemsets = apriori(dataset, min_support)
    # Generate association rules
    rules = generate_association_rules(frequent_itemsets, dataset, min_confidence)
    # Print frequent itemsets
    print("Frequent Itemsets:")
    for itemset in frequent_itemsets:
        print(f"Support: {sum(1 for transaction in dataset if itemset.issubset(transaction)) / len(dataset):.6f}, Itemset:
{itemset}")
    # Print association rules
    print("\nAssociation Rules:")
    print("Antecedents -> Consequents | Support | Confidence | Lift")
    for rule in rules:
        antecedent, consequent, support, confidence, lift = rule
        print(f"{antecedent} -> {consequent} | {support:.6f} | {confidence:.6f} | {lift:.6f}")

if __name__ == "__main__":
    main()
```

OUTPUT

```
[Done] exited with code=0 in 0.165 seconds

[Running] python -u "e:\ALL ACADEMIC\DMW\apriori.py"
Frequent Itemsets:
Support: 1.000000, Itemset: frozenset({'bread'})
Support: 0.714286, Itemset: frozenset({'butter'})
Support: 0.857143, Itemset: frozenset({'milk'})
Support: 0.714286, Itemset: frozenset({'bread', 'butter'})
Support: 0.571429, Itemset: frozenset({'butter', 'milk'})
Support: 0.857143, Itemset: frozenset({'bread', 'milk'})
Support: 0.571429, Itemset: frozenset({'bread', 'butter', 'milk'})

Association Rules:
Antecedents -> Consequents | Support | Confidence | Lift
frozenset({'bread'}) -> frozenset({'butter'}) | 0.714286 | 0.714286 | 1.000000
frozenset({'butter'}) -> frozenset({'bread'}) | 0.714286 | 1.000000 | 1.000000
frozenset({'butter'}) -> frozenset({'milk'}) | 0.571429 | 0.800000 | 0.933333
frozenset({'bread'}) -> frozenset({'milk'}) | 0.857143 | 0.857143 | 1.000000
frozenset({'milk'}) -> frozenset({'bread'}) | 0.857143 | 1.000000 | 1.000000
frozenset({'bread', 'butter'}) -> frozenset({'milk'}) | 0.571429 | 0.800000 | 0.933333
frozenset({'butter', 'milk'}) -> frozenset({'bread'}) | 0.571429 | 1.000000 | 1.000000

[Done] exited with code=0 in 0.755 seconds
```