Assignment 5

```
#!/usr/bin/env python3
"""
Self-contained Market Basket Apriori script.
Usage:
    1. Place 'Market_Basket_Optimisation.csv' (no header) in the same folder as this script.
    2. Run: python3 market_basket_apriori.py
It will output association_rules.csv and some basic plots (requires matplotlib and pandas).
If you prefer no external packages, the script will still run but plotting requires matplotlib.
"""

import csv
from collections import defaultdict, Counter
import itertools
import math
import os
import sys
import pandas as pd

def load_transactions(csv_file):
    transactions = []
    with open(csv_file, newline='', encoding='utf-8') as f:
        reader = csv.reader(f)
        for row in reader:
            items = [cell.strip() for cell in row if cell and cell.strip()!='']
            if items:
                transactions.append(items)
    return transactions

def get_frequent_itemsets(transactions, min_support):
    """
    Apriori (simple implementation).
    Returns dict mapping itemset (frozenset) -> support (fraction).
    """
    n = len(transactions)
    # count single items
    item_counts = Counter()
    for t in transactions:
```

```
        for item in set(t):
            item_counts[item] += 1
    L1 = {frozenset([item]): count / n for item, count in item_counts.items() if count / n
>= min_support}
    freq_itemsets = dict(L1)
    k = 2
    Lk_prev = set(L1.keys())
    while Lk_prev:
        # generate candidate itemsets Ck by joining Lk_prev
        candidates = set()
        prev_list = list(Lk_prev)
        for i in range(len(prev_list)):
            for j in range(i+1, len(prev_list)):
                union = prev_list[i] | prev_list[j]
                if len(union) == k:
                    candidates.add(union)
        # prune candidates if any (k-1)-subset not frequent (Apriori property)
        pruned = set()
        for c in candidates:
            all_subs_frequent = True
            for subset in itertools.combinations(c, k-1):
                if frozenset(subset) not in Lk_prev:
                    all_subs_frequent = False
                    break
            if all_subs_frequent:
                pruned.add(c)
        # count support for candidates
        counts = Counter()
        for t in transactions:
            tset = set(t)
            for c in pruned:
                if c.issubset(tset):
                    counts[c] += 1
        Lk = {c: counts[c] / n for c in counts if counts[c] / n >= min_support}
        if not Lk:
            break
        freq_itemsets.update(Lk)
        Lk_prev = set(Lk.keys())
        k += 1
    return freq_itemsets

def generate_rules(freq_itemsets, transactions, min_confidence=0.3):
    n = len(transactions)
    support_count = {iset: int(freq_itemsets[iset] * n) for iset in freq_itemsets}
```

```
    rules = []
    for itemset in [iset for iset in freq_itemsets if len(iset) >= 2]:
        itemset_support = freq_itemsets[itemset]
        # all non-empty proper subsets as antecedents
        for r in range(1, len(itemset)):
            for antecedent in itertools.combinations(list(itemset), r):
                antecedent = frozenset(antecedent)
                consequent = itemset - antecedent
                antecedent_support = freq_itemsets.get(antecedent, 0)
                if antecedent_support > 0:
                    confidence = itemset_support / antecedent_support
                    # compute lift = confidence / support(consequent)
                    consequent_support = freq_itemsets.get(consequent, 0)
                    lift = confidence / consequent_support if consequent_support>0 else 0
                    if confidence >= min_confidence:
                        rules.append({
                            'antecedent': ', '.join(sorted(antecedent)),
                            'consequent': ', '.join(sorted(consequent)),
                            'support': itemset_support,
                            'confidence': confidence,
                            'lift': lift
                        })
    # sort rules
    rules.sort(key=lambda x: (x['lift'], x['confidence']), reverse=True)
    return rules

def main():
    csv_file = r"C:\Users\rohit\OneDrive\Desktop\Sem
5\ML_LAB\Market_Basket_Optimisation.csv"
    if not os.path.exists(csv_file):
        print(f'ERROR: {csv_file} not found in {os.getcwd()}')
        return
    transactions = load_transactions(csv_file)
    print('Transactions loaded:', len(transactions))
    min_support = 0.01
    freq_itemsets = get_frequent_itemsets(transactions, min_support)
    print('Frequent itemsets found:', len(freq_itemsets))
    min_confidence = 0.3
    rules = generate_rules(freq_itemsets, transactions, min_confidence)
    print('Rules generated:', len(rules))
    # Save to CSV via pandas if available, else plain csv
    try:
        import pandas as pd
        df = pd.DataFrame(rules)
```

```
        df.to_csv('association_rules.csv', index=False)
        print('Saved rules to association_rules.csv')
    except Exception as e:
        import csv
        with open('association_rules.csv','w', newline='', encoding='utf-8') as f:
            writer = csv.DictWriter(f,
fieldnames=['antecedent','consequent','support','confidence','lift'])
            writer.writeheader()
            for r in rules:
                writer.writerow(r)
        print('Saved rules to association_rules.csv (csv module)')

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

Output :