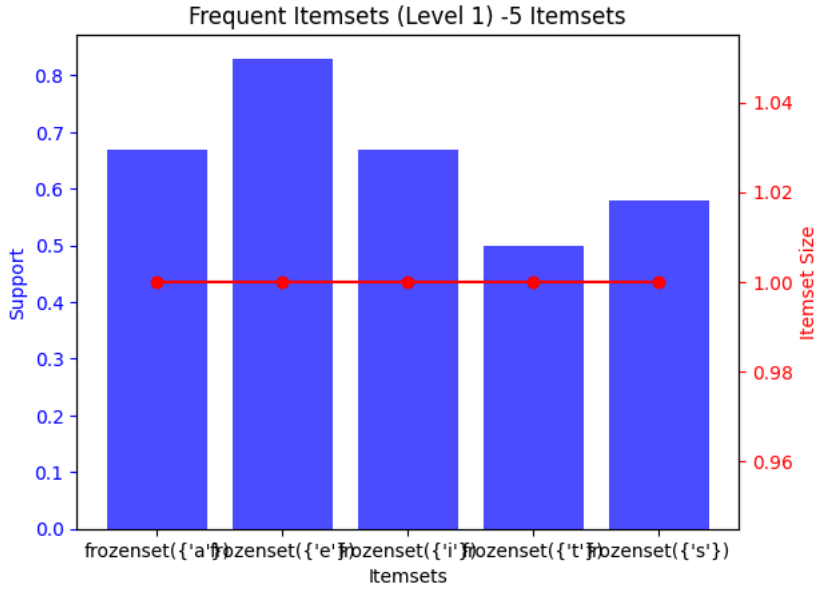
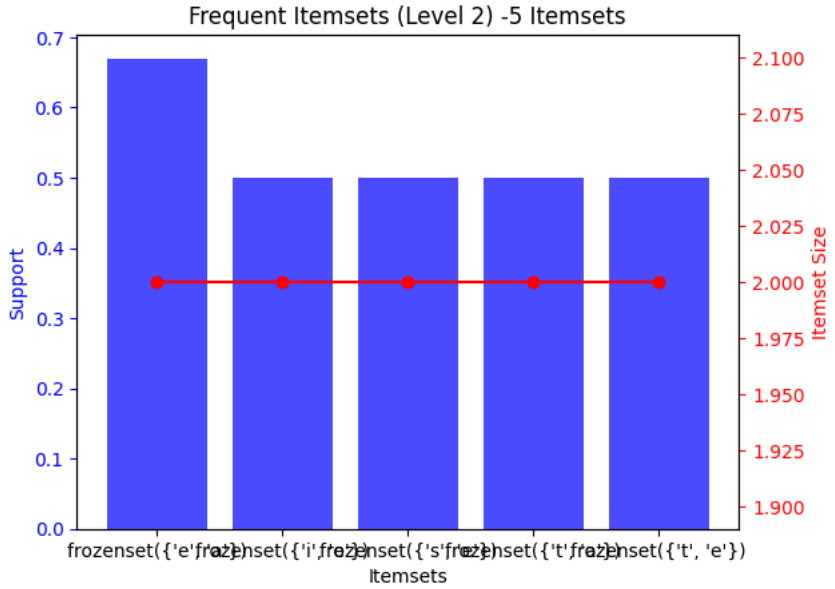
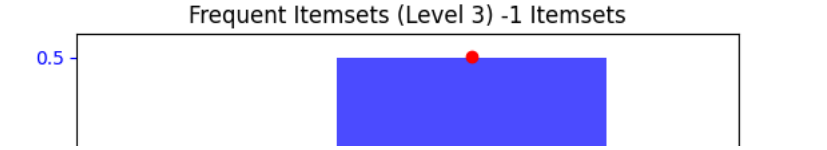
```
import itertools
import matplotlib.pyplot as plt
import networkx as nx
import pandas as pd
def load_data():
    df = pd.read_csv(r'/content/HeartFailure.csv')
    df = df.columns
    matrix = df.values
    data = matrix
    return data
def generate_itemsets(data):
    itemsets = set()
    for transaction in data:
        for item in transaction:
            itemsets.add(frozenset([item]))
    return itemsets
def calculate_support(data, itemsets, min_support):
    support_count = {}
    total_transactions = len(data)
    for transaction in data:
        for itemset in itemsets:
            if itemset.issubset(transaction):
                support_count[itemset] = support_count.get(itemset, 0) + 1
    support = {itemset: count / total_transactions for itemset, count in support_count.items() if count / total_transactions >= min_suppo
    return support
def generate_candidate_itemsets(itemsets, k):
    return set([itemset1.union(itemset2) for itemset1 in itemsets for itemset2 in itemsets if len(itemset1.union(itemset2)) == k])
def apriori(data, min_support):
    frequent_itemsets = []
    all_frequent_itemsets = []
    level_items = [] # Added to store items at each level
    k = 1
    while True:
        if k == 1:
            itemsets = generate_itemsets(data)
        else:
            itemsets = generate_candidate_itemsets(itemsets, k)
        support = calculate_support(data, itemsets, min_support)
        if not support:
            break
        frequent_itemsets.extend([(itemset, round(support[itemset], 2)) for itemset in support])
        all_frequent_itemsets.append(frequent_itemsets.copy())
        level_items.append(set(item for itemset in itemsets for item in itemset))
        frequent_itemsets = []
        k += 1
    return all_frequent_itemsets, level_items
def plot_frequent_itemsets(frequent_itemsets, level, save_path=None):
    labels, support = zip(*list(frequent_itemsets))
    itemset_sizes = [len(itemset) for itemset in labels]
    fig, ax1 = plt.subplots()
    ax1.bar(list(map(str, labels)), support, color='b', alpha=0.7, label='Support')
    ax1.set_xlabel('Itemsets')
    ax1.set_ylabel('Support', color='b')
    ax1.tick_params('y', colors='b')
    ax2 = ax1.twinx()
    ax2.plot(list(map(str, labels)), itemset_sizes, color='r', marker='o', label='Itemset Size')
    ax2.set_ylabel('Itemset Size', color='r')
    ax2.tick_params('y', colors='r')
    plt.title(f'Frequent Itemsets (Level {level}) -{len(frequent_itemsets)} Itemsets')
    plt.xticks(rotation=45, ha='right')
    if save_path:
        plt.savefig(save_path)
    else:
        plt.show()
def plot_all_frequent_itemsets(all_frequent_itemsets):
    for level, frequent_itemsets in enumerate(all_frequent_itemsets, start=1):
        plot_frequent_itemsets(frequent_itemsets, level, save_path=f'frequent_itemsets_level_{level}.png')
def generate_association_rules(frequent_itemsets, min_confidence):
    association_rules = []
    for itemset, support in frequent_itemsets:
        if len(itemset) > 1:
            for subset in itertools.combinations(itemset, len(itemset) - 1):
                antecedent = frozenset(subset)
                consequent = itemset - antecedent
                confidence = support / frequent_itemsets_dict.get(antecedent, 1) # Fix here, use get method
                if confidence >= min_confidence:
                    association_rules.append((set(antecedent), set(consequent), round(confidence, 2)))
    return association_rules
def print_and_save_association_rules(association_rules):
    print("Association Rules:")
    for antecedent, consequent, confidence in association_rules:
```

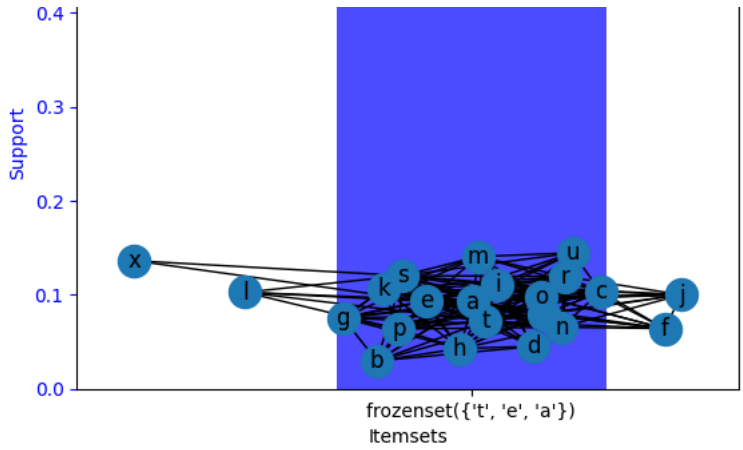
```
            print(f"{antecedent} => {consequent} (Confidence: {confidence})")
        with open("association_rules.txt", "w") as file:
            file.write("Association Rules:\n")
            for antecedent, consequent, confidence in association_rules:
                file.write(f"{antecedent} => {consequent} (Confidence: {confidence})\n")
    def generate_strong_association_rules(association_rules, min_confidence):
        strong_association_rules = [rule for rule in association_rules if rule[2] >= min_confidence]
        return strong_association_rules
    def plot_association_rules(association_rules, title, save_path=None):
        labels = [f"{set(antecedent)} => {set(consequent)}\n(Confidence: {confidence})" for antecedent, consequent, confidence in association
        confidences = [confidence for antecedent, consequent, confidence in association_rules]
        fig, ax = plt.subplots()
        ax.barh(labels, confidences, color='green', alpha=0.7)
        ax.set_xlabel('Confidence')
        ax.set_title(title)
        if save_path:
            plt.savefig(save_path)
        else:
            plt.show()
    def evaluate_efficiency(all_frequent_itemsets):
        for level, frequent_itemsets in enumerate(all_frequent_itemsets, start=1):
            print(f"Level {level} - Number of Itemsets: {len(frequent_itemsets)}")
    def generate_graph(data):
        G = nx.Graph()
        for transaction in data:
            G.add_nodes_from(transaction)
            G.add_edges_from(itertools.combinations(transaction, 2))
        return G
    def plot_graph(graph, save_path=None):
        nx.draw(graph, with_labels=True)
        if save_path:
            plt.savefig(save_path)
        else:
            plt.show()
    if __name__ == "__main__":
        data = load_data()
        min_support = 0.5
        min_confidence = 0.6
        all_frequent_itemsets, level_items = apriori(data, min_support)
        evaluate_efficiency(all_frequent_itemsets)
        plot_all_frequent_itemsets(all_frequent_itemsets)
        graph_data = generate_graph(data)
        plot_graph(graph_data, save_path='graph.png')
        frequent_itemsets_dict = dict(itertools.chain.from_iterable(all_frequent_itemsets))
        association_rules = generate_association_rules(frequent_itemsets_dict.items(), min_confidence)
        print_and_save_association_rules(association_rules)
        strong_association_rules = generate_strong_association_rules(association_rules, min_confidence)
        print("Strong Association Rules:")
        for rule in strong_association_rules:
            print(f"{rule[0]} => {rule[1]} (Confidence: {rule[2]})")
        plot_association_rules(association_rules, 'Association Rules', save_path='association_rules.png')
        plot_association_rules(strong_association_rules, 'Strong Association Rules', save_path='strong_association_rules.png')
```

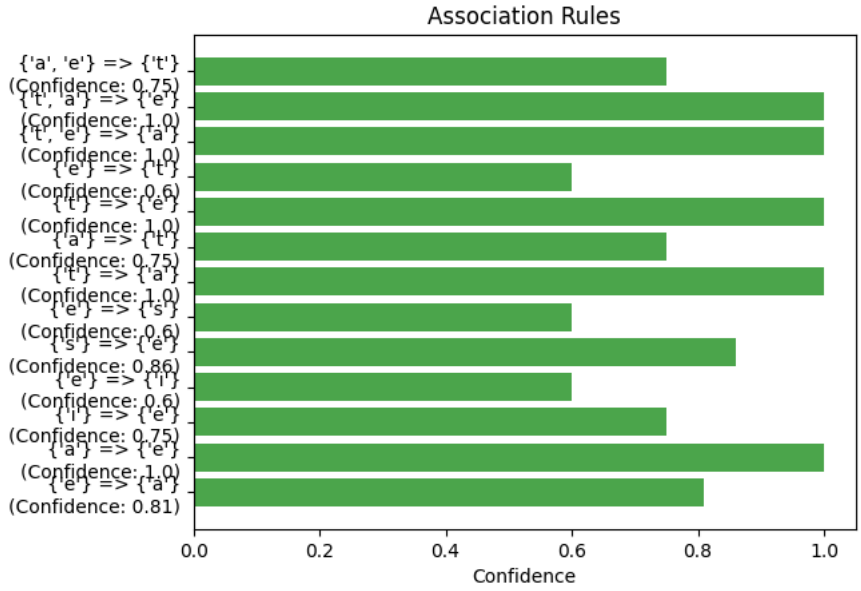
```
Level 1 - Number of Itemsets: 5
Level 2 - Number of Itemsets: 5
Level 3 - Number of Itemsets: 1
Association Rules:
{'e'} => {'a'} (Confidence: 0.81)
{'a'} => {'e'} (Confidence: 1.0)
{'i'} => {'e'} (Confidence: 0.75)
{'e'} => {'i'} (Confidence: 0.6)
{'s'} => {'e'} (Confidence: 0.86)
{'e'} => {'s'} (Confidence: 0.6)
{'t'} => {'a'} (Confidence: 1.0)
{'a'} => {'t'} (Confidence: 0.75)
{'t'} => {'e'} (Confidence: 1.0)
{'e'} => {'t'} (Confidence: 0.6)
{'t', 'e'} => {'a'} (Confidence: 1.0)
{'t', 'a'} => {'e'} (Confidence: 1.0)
{'a', 'e'} => {'t'} (Confidence: 0.75)
Strong Association Rules:
{'e'} => {'a'} (Confidence: 0.81)
{'a'} => {'e'} (Confidence: 1.0)
{'i'} => {'e'} (Confidence: 0.75)
{'e'} => {'i'} (Confidence: 0.6)
{'s'} => {'e'} (Confidence: 0.86)
{'e'} => {'s'} (Confidence: 0.6)
{'t'} => {'a'} (Confidence: 1.0)
{'a'} => {'t'} (Confidence: 0.75)
{'t'} => {'e'} (Confidence: 1.0)
{'e'} => {'t'} (Confidence: 0.6)
{'t', 'e'} => {'a'} (Confidence: 1.0)
{'t', 'a'} => {'e'} (Confidence: 1.0)
{'a', 'e'} => {'t'} (Confidence: 0.75)
```

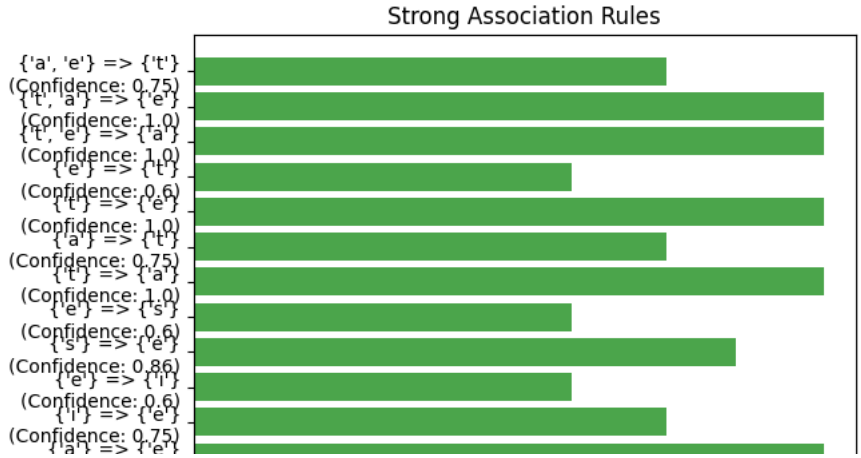







Support
0.4
0.3
0.2
0.1
0.0
x
l
m
u
s
k
i
r
c
j
e
a
o
g
t
p
n
f
h
d
b
frozenset({'t', 'e', 'a'})
Itemsets


Association Rules
{'a', 'e'} => {'t'}
(Confidence: 0.75)
{'t', 'a'} => {'e'}
(Confidence: 1.0)
{'t', 'e'} => {'a'}
(Confidence: 1.0)
{'e'} => {'t'}
(Confidence: 0.6)
{'t'} => {'e'}
(Confidence: 1.0)
{'a'} => {'t'}
(Confidence: 0.75)
{'t'} => {'a'}
(Confidence: 1.0)
{'e'} => {'s'}
(Confidence: 0.6)
{'s'} => {'e'}
(Confidence: 0.86)
{'e'} => {'i'}
(Confidence: 0.6)
{'i'} => {'e'}
(Confidence: 0.75)
{'a'} => {'e'}
(Confidence: 1.0)
{'e'} => {'a'}
(Confidence: 0.81)
0.0
0.2
0.4
0.6
0.8
1.0
Confidence


Strong Association Rules
{'a', 'e'} => {'t'}
(Confidence: 0.75)
{'t', 'a'} => {'e'}
(Confidence: 1.0)
{'t', 'e'} => {'a'}
(Confidence: 1.0)
{'e'} => {'t'}
(Confidence: 0.6)
{'t'} => {'e'}
(Confidence: 1.0)
{'a'} => {'t'}
(Confidence: 0.75)
{'t'} => {'a'}
(Confidence: 1.0)
{'e'} => {'s'}
(Confidence: 0.6)
{'s'} => {'e'}
(Confidence: 0.86)
{'e'} => {'i'}
(Confidence: 0.6)
{'i'} => {'e'}
(Confidence: 0.75)