Justin Figueroa
ECE 59500 Data Mining
Assignment 2
October 16, 2021

1.)
Frequent itemsets
L1
{gene_1}: sup = 83
{gene_12}: sup = 54
{gene_14}: sup = 52
{gene_17}: sup = 55
{gene_21}: sup = 62
{gene_22}: sup = 55
{gene_23}: sup = 54
{gene_25}: sup = 57
{gene_26}: sup = 52
{gene_27}: sup = 51
{gene_3}: sup = 71
{gene_31}: sup = 51
{gene_36}: sup = 61
{gene_37}: sup = 56
{gene_39}: sup = 51
{gene_4}: sup = 50
{gene_43}: sup = 50
{gene_45}: sup = 58
{gene_47}: sup = 66
{gene_48}: sup = 57
{gene_5}: sup = 73
{gene_50}: sup = 50
{gene_53}: sup = 50
{gene_54}: sup = 67
{gene_55}: sup = 55
{gene_56}: sup = 51
{gene_59}: sup = 76
{gene_6}: sup = 66
{gene_60}: sup = 54
{gene_63}: sup = 50
{gene_64}: sup = 50
{gene_66}: sup = 59
{gene_67}: sup = 62
{gene_71}: sup = 58
{gene_72}: sup = 74
{gene_75}: sup = 57
{gene_77}: sup = 58
{gene_78}: sup = 59
{gene_8}: sup = 66
{gene_81}: sup = 58

{gene_83}: sup = 50
{gene_84}: sup = 54
{gene_87}: sup = 67
{gene_89}: sup = 59
{gene_9}: sup = 50
{gene_90}: sup = 52
{gene_91}: sup = 65
{gene_93}: sup = 53
{gene_94}: sup = 62
{gene_98}: sup = 51
{gene_99}: sup = 56
L2
{gene_1, gene_21}: sup = 53
{gene_3, gene_1}: sup = 63
{gene_47, gene_1}: sup = 59
{gene_5, gene_1}: sup = 65
{gene_1, gene_54}: sup = 58
{gene_59, gene_1}: sup = 62
{gene_1, gene_6}: sup = 59
{gene_67, gene_1}: sup = 55
{gene_1, gene_72}: sup = 61
{gene_8, gene_1}: sup = 53
{gene_81, gene_1}: sup = 51
{gene_1, gene_84}: sup = 50
{gene_1, gene_87}: sup = 56
{gene_1, gene_89}: sup = 52
{gene_1, gene_91}: sup = 55
{gene_1, gene_94}: sup = 54
{gene_3, gene_47}: sup = 50
{gene_3, gene_5}: sup = 59
{gene_3, gene_59}: sup = 56
{gene_3, gene_72}: sup = 53
{gene_5, gene_47}: sup = 53
{gene_5, gene_59}: sup = 51
{gene_5, gene_6}: sup = 52
{gene_5, gene_72}: sup = 51
{gene_5, gene_87}: sup = 51
{gene_5, gene_91}: sup = 50
{gene_59, gene_6}: sup = 51
{gene_59, gene_72}: sup = 62
{gene_59, gene_87}: sup = 51
L3
{gene_3, gene_5, gene_1}: sup = 52
{gene_59, gene_1, gene_72}: sup = 50

2.)
Length-3 candidate itemsets
[frozenset({'gene_47', 'gene_3', 'gene_1'}),
frozenset({'gene_3', 'gene_5', 'gene_1'}),
frozenset({'gene_3', 'gene_59', 'gene_1'}),
frozenset({'gene_72', 'gene_3', 'gene_1'}),
frozenset({'gene_47', 'gene_5', 'gene_1'}),
frozenset({'gene_59', 'gene_5', 'gene_1'}),
frozenset({'gene_6', 'gene_5', 'gene_1'}),
frozenset({'gene_72', 'gene_5', 'gene_1'}),
frozenset({'gene_87', 'gene_5', 'gene_1'}),
frozenset({'gene_5', 'gene_91', 'gene_1'}),
frozenset({'gene_6', 'gene_59', 'gene_1'}),
frozenset({'gene_72', 'gene_59', 'gene_1'}),
frozenset({'gene_87', 'gene_59', 'gene_1'}),
frozenset({'gene_47', 'gene_3', 'gene_5'}),
frozenset({'gene_3', 'gene_59', 'gene_5'}),
frozenset({'gene_72', 'gene_3', 'gene_5'}),
frozenset({'gene_72', 'gene_3', 'gene_59'}),
frozenset({'gene_6', 'gene_59', 'gene_5'}),
frozenset({'gene_72', 'gene_59', 'gene_5'}),
frozenset({'gene_87', 'gene_59', 'gene_5'})]

3.) Codes of implemented Apriori algorithm.

```
def get_freq(dataset, candidates, min_support, verbose=False):
    """

    This function separates the candidates itemsets into frequent itemset and inf
requent itemsets based on the min_support,
    and returns all candidate itemsets that meet a minimum support threshold.

    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
    -------
    freq_list : list
```

```
        The list of frequent itemsets.

    support_data : dict
        The support data for all candidate itemsets.
    """
    freq_list = []
    support_data = dict()

    # return if you have no candidates
    if candidates == []:
        return freq_list, support_data

    # support count
    for item in candidates:
        support_data[item] = 0
        for transaction in dataset:
            if (item.issubset(transaction)):
                support_data[item] += 1

    # generate list of frequent items that meet min_support threshold
    for key, value in support_data.items():
        if((float(value/len(dataset))) >= min_support):
            freq_list.append(key)

    return freq_list, support_data
```

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

    This part generates new candidate k-itemsets based on the frequent
    (k-1)-itemsets found in the previous iteration.

    The apriori_gen function performs two operations:
    (1) Generate length k candidate itemsets from length k-1 frequent itemsets
    (2) Prune candidate itemsets containing subsets of length k-
1 that are infrequent

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

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

```
    Returns
    -------
    candidate_list : list
        The list of candidate itemsets.
    """
    # operation 1
    candidate_list = []

    if k == 2:
        for a in freq_sets:
            for b in freq_sets:
                union = a | b  # for set iteration
                if len(union) == k and a != b and not union in candidate_list:
                    candidate_list.append(union)
    else:
        # Apply Fk-1 x Fk-1 candidate generation method for 3-
itemsets or higher.
        # Start by transforming frozenset to lists so order can be accessed
        freq_lists = ([sorted(list(x)) for x in freq_sets])
        for a in freq_lists:
            for b in freq_lists:
                union = sorted(a + [d for d in b if not d in a])
                if list(a)[0:k-2] == list(b)[0:k-
2] and len(union) == k and a != b and not union in candidate_list:
                    candidate_list.append(union)
        removedCand = []
        # generate subsets of cancidates and verify that all are from frequent Fk
-1-itemsets. If not, prune.
        for itemset in candidate_list:
            subsets = []
            x = len(itemset)
            for i in range(1 << x):
                subsets.append(sorted([itemset[j] for j in range(x) if (i & (1 <<
 j))]))
            for subset in subsets:
                if len(subset) == k-1 and not subset in freq_lists:
                    removedCand.append(itemset)
                    continue
        for removal in removedCand:
            candidate_list.remove(removal)
        candidate_list = ([frozenset(y) for y in candidate_list]) # transform pru
ned candidates back into frozenset

    return candidate_list
```