Nate Dauterman
ECE 49595
March 8th, 2022

Gene Data Transaction Frequent List

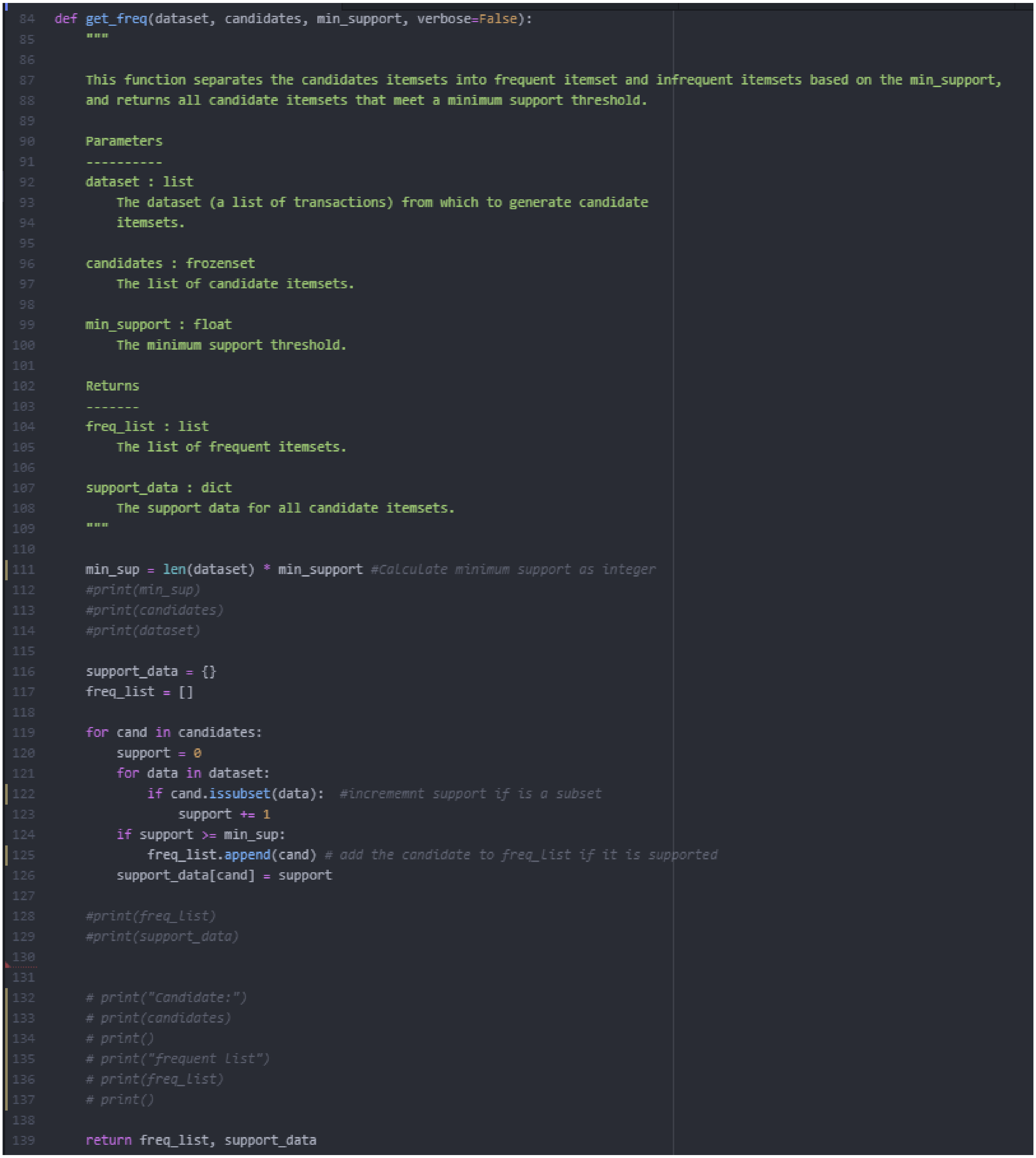
```
{gene_1}:  sup = 83
{gene_12}:  sup = 54
{gene_14}:  sup = 52
{gene_17}:  sup = 55
{gene_21}:  sup = 62
{gene_22}:  sup = 55
{gene_23}:  sup = 54
{gene_25}:  sup = 57
{gene_26}:  sup = 52
{gene_27}:  sup = 51
{gene_3}:  sup = 71
{gene_31}:  sup = 51
{gene_36}:  sup = 61
{gene_37}:  sup = 56
{gene_39}:  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
{gene_1, gene_21}:  sup = 53
{gene_1, gene_3}:  sup = 63
{gene_1, gene_47}:  sup = 59
{gene_5, gene_1}:  sup = 65
{gene_54, gene_1}:  sup = 58
{gene_1, gene_59}:  sup = 62
{gene_6, gene_1}:  sup = 59
{gene_67, gene_1}:  sup = 55
{gene_72, gene_1}:  sup = 61
{gene_8, gene_1}:  sup = 53
{gene_1, gene_81}:  sup = 51
{gene_1, gene_84}:  sup = 50
{gene_87, gene_1}:  sup = 56
{gene_89, gene_1}:  sup = 52
{gene_91, gene_1}:  sup = 55
{gene_94, gene_1}:  sup = 54
{gene_3, gene_47}:  sup = 50
{gene_5, gene_3}:  sup = 59
{gene_3, gene_59}:  sup = 56
{gene_72, gene_3}:  sup = 53
{gene_5, gene_47}:  sup = 53
{gene_5, gene_59}:  sup = 51
{gene_6, gene_5}:  sup = 52
{gene_5, gene_72}:  sup = 51
{gene_5, gene_87}:  sup = 51
{gene_5, gene_91}:  sup = 50
{gene_6, gene_59}:  sup = 51
{gene_72, gene_59}:  sup = 62
{gene_87, gene_59}:  sup = 51
{gene_5, gene_1, gene_3}:  sup = 52
{gene_72, gene_1, gene_59}:  sup = 50
```

Gene Data Transaction Length-3 Candidates

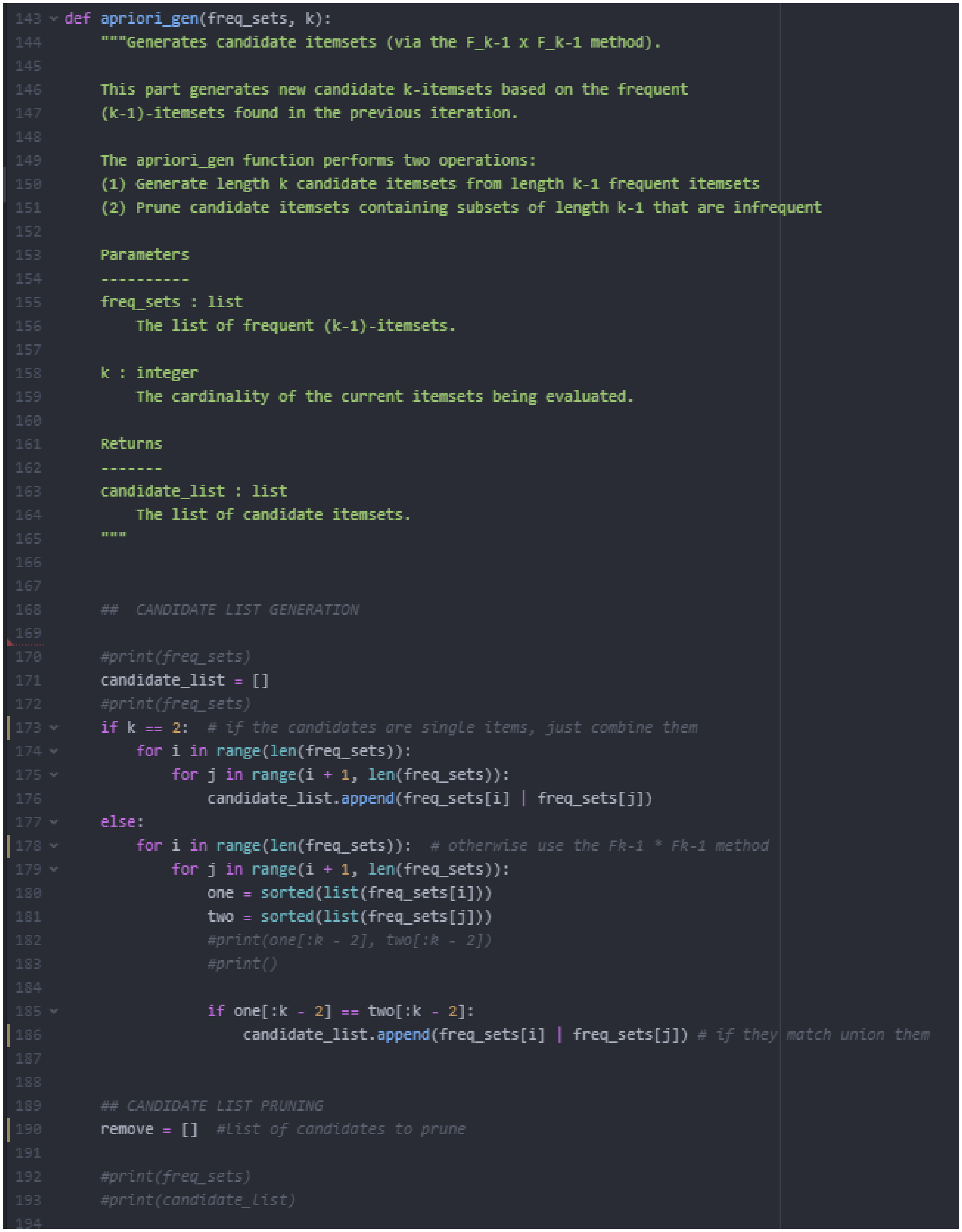
```
[frozenset({'gene_1', 'gene_3', 'gene_47'}), frozenset({'gene_5',
   'gene_1', 'gene_3'}), frozenset({'gene_1', 'gene_3', 'gene_59'}),

 frozenset({'gene_72', 'gene_1', 'gene_3'}), frozenset({'gene_5',
   'gene_1', 'gene_47'}), frozenset({'gene_5', 'gene_1', 'gene_59'}),

 frozenset({'gene_6', 'gene_5', 'gene_1'}), frozenset({'gene_5', 'gene_1',
   'gene_72'}), frozenset({'gene_5', 'gene_1', 'gene_87'}),

frozenset({'gene_5', 'gene_91', 'gene_1'}), frozenset({'gene_6', 'gene_1',
   'gene_59'}), frozenset({'gene_72', 'gene_1', 'gene_59'}),

 frozenset({'gene_87', 'gene_1', 'gene_59'}), frozenset({'gene_5',
   'gene_3', 'gene_47'}), frozenset({'gene_5', 'gene_3', 'gene_59'}),

 frozenset({'gene_5', 'gene_72', 'gene_3'}), frozenset({'gene_72',
   'gene_3', 'gene_59'}), frozenset({'gene_6', 'gene_5', 'gene_59'}),

 frozenset({'gene_5', 'gene_72', 'gene_59'}), frozenset({'gene_5',
   'gene_87', 'gene_59'})]
```

Get Frequent List Function

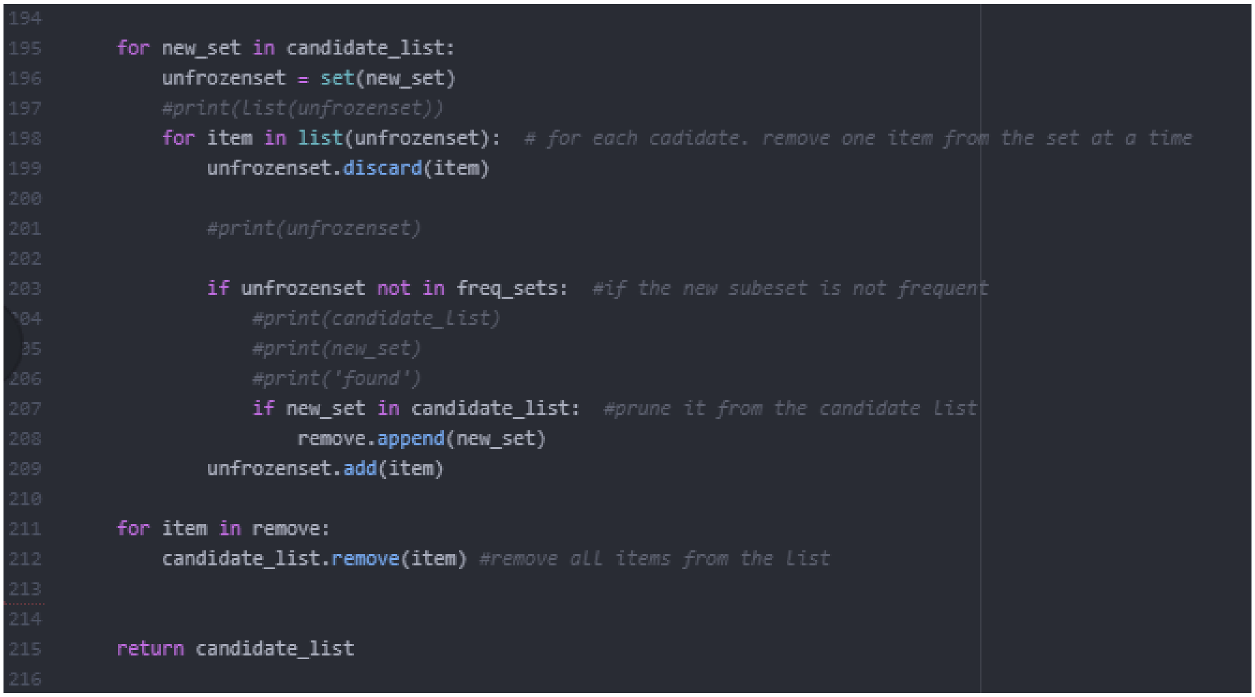


```python
def get_freq(dataset, candidates, min_support, verbose=False):
    """

    This function separates the candidates itemsets into frequent itemset and infrequent 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.
    """

    min_sup = len(dataset) * min_support #Calculate minimum support as integer
    #print(min_sup)
    #print(candidates)
    #print(dataset)

    support_data = {}
    freq_list = []

    for cand in candidates:
        support = 0
        for data in dataset:
            if cand.issubset(data):  #incrememnt support if is a subset
                support += 1
        if support >= min_sup:
            freq_list.append(cand) # add the candidate to freq_list if it is supported
        support_data[cand] = support

    #print(freq_list)
    #print(support_data)


    # print("Candidate:")
    # print(candidates)
    # print()
    # print("frequent list")
    # print(freq_list)
    # print()

    return freq_list, support_data
```

Apriori Generation Function



```python
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.
    """


    ##  CANDIDATE LIST GENERATION

    #print(freq_sets)
    candidate_list = []
    #print(freq_sets)
    if k == 2:  # if the candidates are single items, just combine them
        for i in range(len(freq_sets)):
            for j in range(i + 1, len(freq_sets)):
                candidate_list.append(freq_sets[i] | freq_sets[j])
    else:
        for i in range(len(freq_sets)):  # otherwise use the Fk-1 * Fk-1 method
            for j in range(i + 1, len(freq_sets)):
                one = sorted(list(freq_sets[i]))
                two = sorted(list(freq_sets[j]))
                #print(one[:k - 2], two[:k - 2])
                #print()

                if one[:k - 2] == two[:k - 2]:
                    candidate_list.append(freq_sets[i] | freq_sets[j]) # if they match union them


    ## CANDIDATE LIST PRUNING
    remove = []  #list of candidates to prune

    #print(freq_sets)
    #print(candidate_list)

```



```

    for new_set in candidate_list:
        unfrozenset = set(new_set)
        #print(list(unfrozenset))
        for item in list(unfrozenset):  # for each cadidate. remove one item from the set at a time
            unfrozenset.discard(item)

            #print(unfrozenset)

            if unfrozenset not in freq_sets:  #if the new subeset is not frequent
                #print(candidate_list)
                #print(new_set)
                #print('found')
                if new_set in candidate_list:  #prune it from the candidate list
                    remove.append(new_set)
            unfrozenset.add(item)

    for item in remove:
        candidate_list.remove(item) #remove all items from the list


    return candidate_list

```