Assignment 4
Exercise 2
Question 1
Implementation

1. First I converted the file to a data list. The goal was to read the file line by line and insert it into the data. I used get_data_from_file method to get the data from the file. I considered support as 0.2.

   Challenges

- First challenge was to read the data from the file. I was not able to read the file directly as it was a .gz extension. So I used gzip to open the file and read it.
- Read line was in binary and I wanted to convert the line into a list of numbers. So while opening I opened it in “rb”(read binary) mode. I first added “ascii” decoding to decode each line and then removed the space before and after by strip method. Then I split the line with space using split method and then mapped each element to integer field. This way I got the list of items on each line.

2. Once I got the data, I took only the sample of data. For this I used get_random_sample_data which returns the sample of data by percentage sample size.
   The function get_random_sample_data takes in two arguments- first is sample size percentage and second is the whole data.

3. Now once we get sampled data. We will start applying **apriori algorithm**.
   - Get the unique items in all the data. For this, I used get_items_list function which takes the sample data and get only the unique items.
     Challenges
     For getting the items list, I tried two approaches first
     Approach 1 was to loop through the sample data and then take a set of the list directly which will give the unique elements in each basket.
     Then add it to a temporary list and then again take the set to remove duplicate items.
     This approach was applying two sets which I thought was unnecessary.
     So I went with Approach 2 discussed below.
     In approach 2, I first looped through the data. Now each iteration is a list of items. Now I again looped through the items inside each iteration and then checked if the item is in the temporary list or not and if not then added the item. This way I got the unique items list without using set function.
   - Then I sorted this items list.
   - First task was to get the candidates list and true frequent itemset list as C_k and I_k. Here I filtered data only 2 times. So k=2
   - For finding the candidates list, I wanted to get the count of each item in the whole data. So I went through the items list and then went through each basket of data and checked if the item in the basket matches or not, If it matches, I increased the count for each item.
   - For finding the frequent itemset of size 1. I went through the candidate's dictionary and compared the count of each item is greater than equal to support or not.

The support was in float but we wanted it in comparison with the items count so I calculated support = int(support * number of items in items list)
Challenge: While looping through the data I wanted to go through items that has the highest count first. So I used Counter() function which stores the key according to the count of items.

- For candidate 2, I took two pointers while looping through the frequent itemset list 1 calculated above. This is because C1 is passed to L1 then I1 is passed to C2 and then C2 is passed to L2. Where C is candidates items list and L is the frequent itemset.
  Challenge: How to get two items as one and store it and get the count as a whole.
  The first thing I did was I made a pair of first item in itemset and second item in itemset and then moved forward. Now if the items length is two then I added it directly into a temporary list of items which will be used to get the candidates list.
- The temporary items list created for finding candidates dictionary is then looped and we compare if the doubleton item is a subset of the basket or not.
  Challenge: How to check if both the items in doubleton is in each basket or not.
  Option 1: Looping through each item in doubleton and finding if that item is in the basket or not then if only the first item in the basket, the second item was checked in the basket and then the count of the doubleton was increased.
  Option 2: A little searching and I found out about **issubset** method. This method checks if both the elements is in the basket or not with a simple function.
  So I used issubset method.
- Now again the same concept to create frequent itemset 2 was used but this time candidates 2 list is passed.
- Now once the final frequent itemset is generated, I applied association rules concept to get the rules.
- For this I looped through the second frequent itemset dict.
- I first calculated the count of first item in basket. Then I calculated count of second item in basket. Then I calculated the count for both items in the basket. Through this I calculated confidence using the ratio of count of both items to count of each item.
- Then I made a rule that is (first item, second item) and store it in the rules list.
- This is how the apriori algorithm was implemented.

I ran the code with
python simple_randomized_algorithm.py
This algorithm runs on the full dataset of chess.dat.gz file.
The output of the full dataset of chess.dat.gz is in the Exercise 2 folder and the path is Output/Output SRA/output.txt.

Len of rules	Time for execution
4474	14.904957294464111 seconds

Question 2

Implementation of Son Algorithm

First pass

First I got the data from file which is the same function as implemented in simple randomized algorithm.

Then I converted the whole data into chunk of data in process of 3. For this I used **get_chunk_data** function. This takes the chunk of data and returns the list of chunk data.

Now in each process, we get chunk of data and if it is not empty then we apply apriori algorithm on this chunk of data.

This is similar to applying simple randomized algorithm on only a sample of data.

For applying algorithm on the chunk of data I used **apriori** function which will return the list of rules.

Once we get the rules for all the processes. We combine all the processes rules using **combining_rules_from_each_chunk** method.

Challenge:

How to get rules for all the chunks?

So I made another temp list where I first looped through the processes and for each process the rules returned was appended on to the temp list.

This way I can compare all the rules for the whole dataset.

Second Pass

Once I get the data of the combined rules, I again appled the randomized algorithm but this time on combined data that I got.

The same process of creating chunk data depending on the processes and getting the rules back happens.

The rules that I get for second pass in chunks is similarly appended to a temp list. This way the temp list will store all the rules of the whole data.

Now once we get the final temp list of rules, we again combine the rules that we got in the chunk.

This last combining rules that we get are the final rules.

Run the code using the command

python son_algorithm.py

This will choose the default file as chess.dat.gz and apply son algorithm on it.

The output of the file is in the Exercise 2 folder and the path is Output/Output SON/output.txt.

Length of rules	Time for execution
1346	49.307217836380005 seconds

Question 3

Comparison between the two algorithms on all the dataset

Dataset file name: T10I4D100K.dat.gz

Simple Randomized algorithm

I changed the output path in code to Output/Output SRA/ouput_t1.txt
Command to run the python code for simple randomized algorithm
python simple_randomized_algorithm.py T10I4D100K.dat.gz

The output will be stored in Exercise 2 and path Output/Output SRA/ouput_t1.txt

Length of rules	Time of execution on whole dataset
2373	16.233786821365356 seconds

SON Algorithm

I changed the output path in code to Output/Output SON/ouput_t1.txt
Command to run the python code for son algorithm
python son_algorithm.py T10I4D100K.dat.txt

The output will be stored in Exercise 2 and path Output/Output SON/ouput_t1.txt

Length of rules	Time of execution on whole dataset

Dataset file name: T40I10D100K.dat.gz

Simple Randomized algorithm

I changed the output path in code to Output/Output SRA/ouput_t4.txt
Command to run the python code for simple randomized algorithm
python simple_randomized_algorithm.py T40I10D100K.dat.gz

The output will be stored in Exercise 2 and path Output/Output SRA/ouput_t4.txt

Length of rules	Time of execution on whole dataset
2373	16.118103981018066 seconds

SON Algorithm

I changed the output path in code to Output/Output SON/ouput_t4.txt
Command to run the python code for son algorithm
python son_algorithm.py T40I10D100K.dat.txt

The output will be stored in Exercise 2 and path Output/Output SON/ouput_t4.txt

Length of rules	Time of execution on whole dataset

Dataset file name: chess.dat.gz

Simple Randomized algorithm

I changed the output path in code to Output/Output SRA/ouput_chess.txt
Command to run the python code for simple randomized algorithm
python simple_randomized_algorithm.py chess.dat.gz

The output will be stored in Exercise 2 and path Output/Output SRA/ouput_chess.txt

Length of rules	Time of execution on whole dataset
2373	16.09648895263672 seconds

SON Algorithm

I changed the output path in code to Output/Output SON/ouput_chess.txt
Command to run the python code for son algorithm
python son_algorithm.py chess.dat.gz

The output will be stored in Exercise 2 and path Output/Output SON/ouput_chess.txt

Length of rules	Time of execution on whole dataset
1346	49.59644794464111 seconds

Dataset file name: connect.dat.gz

Simple Randomized algorithm

I changed the output path in code to Output/Output SRA/ouput_connect.txt
Command to run the python code for simple randomized algorithm
python simple_randomized_algorithm.py connect.dat.gz

The output will be stored in Exercise 2 and path Output/Output SRA/ouput_connect.txt

Length of rules	Time of execution on whole dataset
2373	16.10885190963745 seconds

SON Algorithm

I changed the output path in code to Output/Output SON/ouput_connect.txt
Command to run the python code for son algorithm
python son_algorithm.py connect.dat.gz

The output will be stored in Exercise 2 and path Output/Output SON/ouput_connect.txt

Length of rules	Time of execution on whole dataset
260	234.596448694464346 seconds

Dataset file name: mushroom.dat.gz

Simple Randomized algorithm

I changed the output path in code to Output/Output SRA/ouput_mushroom.txt
Command to run the python code for simple randomized algorithm
python simple_randomized_algorithm.py mushroom.dat.gz

The output will be stored in Exercise 2 and path Output/Output SRA/ouput_mushroom.txt

Length of rules	Time of execution on whole dataset
2332	15.38296914100647 seconds

SON Algorithm

I changed the output path in code to Output/Output SON/ouput_mushroom.txt
Command to run the python code for son algorithm
python son_algorithm.py output_mushroom.dat.gz

The output will be stored in Exercise 2 and path Output/Output SON/ouput_mushroom.txt

Length of rules	Time of execution on whole dataset
194	184.85034489631653

Dataset file name: pumsb.dat.gz

Simple Randomized algorithm

I changed the output path in code to Output/Output SRA/ouput_pumsb.txt
Command to run the python code for simple randomized algorithm
python simple_randomized_algorithm.py pumsb.dat.gz

The output will be stored in Exercise 2 and path Output/Output SRA/ouput_pumsb.txt

Length of rules	Time of execution on whole dataset
2182	14.581166982650757

SON Algorithm

I changed the output path in code to Output/Output SON/ouput_pumsb.txt
Command to run the python code for son algorithm
python son_algorithm.py pumsb.dat.gz

The output will be stored in Exercise 2 and path Output/Output SON/ouput_pumsb.txt

Length of rules	Time of execution on whole dataset

Dataset file name: pumsb_start.dat.gz

Simple Randomized algorithm

I changed the output path in code to Output/Output SRA/ouput_pumsb_star.txt
Command to run the python code for simple randomized algorithm
python simple_randomized_algorithm.py pumsb_star.dat.gz

The output will be stored in Exercise 2 and path Output/Output SRA/ouput_pumsb_star.txt

Length of rules	Time of execution on whole dataset
2234	14.93704104423523 seconds

SON Algorithm

I changed the output path in code to Output/Output SON/ouput_pumsb_star.txt
Command to run the python code for son algorithm
python son_algorithm.py pumsb_star.dat.gz

The output will be stored in Exercise 2 and path Output/Output SON/ouput_pumsb_star.txt

Length of rules	Time of execution on whole dataset

Question 4
Change the output path for each file according to the output filename given in the table below

Dataset file name: T10I4D100K.dat.gz

Simple Randomized algorithm
Code to run
python simple_randomized_algorithm.py filename sample_size_percentage.
For eg. python simple_randomized_algorithm.py T10I4D100K.dat.gz 1

Length of rules	Sample Size Percentage	Time of execution on whole dataset	Output file path .txt file
2182	1 %	14.654384851455688 seconds	output_t1_one
2291	2 %	15.140046119689941 seconds	output_t1_two
2332	5%	15.446140766143799 seconds	output_t1_five
2380	10%	16.45607614517212 seconds	output_t1_ten

Dataset file name: T40I10D100K.dat.gz

Simple Randomized algorithm
Code to run
python simple_randomized_algorithm.py filename sample_size_percentage.
For eg. python simple_randomized_algorithm.py T10I4D100K.dat.gz 1

Length of rules	Sample Size Percentage	Time of execution on whole dataset	Output file path .txt file
2177	1 %	14.468321084976196 seconds	output_t4_one
2332	2 %	15.728078126907349 seconds	output_t4_two
2291	5%	14.991923809051514 seconds	output_t4_five
2380	10%	16.133537769317627 seconds	output_t4_ten

Dataset file name: chess.dat.gz

Simple Randomized algorithm
Code to run
python simple_randomized_algorithm.py filename sample_size_percentage.
For eg. python simple_randomized_algorithm.py chess.dat.gz 1

Length of rules	Sample Size Percentage	Time of execution on whole dataset	Output file path .txt file
2233	1 %	15.066042184829712 seconds	output_chess_one
2239	2 %	15.026381969451904	output_chess_two
2380	5%	15.96620512008667 seconds	output_ches_five
2366	10%	16.077756881713867 seconds	output_chess_ten

Dataset file name: connect.dat.gz

Simple Randomized algorithm
Code to run
python simple_randomized_algorithm.py filename sample_size_percentage.
For eg. python simple_randomized_algorithm.py chess.dat.gz 1

Length of rules	Sample Size Percentage	Time of execution on whole dataset	Output file path .txt file
2123	1 %	14.078474998474121 seconds	output_connect_one
2325	2 %	15.52085018157959 seconds	output_connect_two
2346	5%	15.747562885284424 seconds	output_connect_five
2380	10%	16.443382024765015 seconds	output_connect_ten

Dataset file name: mushroom.dat.gz

Simple Randomized algorithm
Code to run
python simple_randomized_algorithm.py filename sample_size_percentage.
For eg. python simple_randomized_algorithm.py chess.dat.gz 1

Length of rules	Sample Size Percentage	Time of execution on whole dataset	Output file path .txt file
2297	1 %	15.723680973052979	output_mushroom_one
2312	2 %	15.487175941467285 seconds	output_mushroom_two
2366	5%	15.797584772109985 seconds	output_mushroom_five
2366	10%	16.026835918426514 seconds	output_mushroom_ten

Dataset file name: pumsb.dat.gz

Simple Randomized algorithm
Code to run
python simple_randomized_algorithm.py filename sample_size_percentage.
For eg. python simple_randomized_algorithm.py chess.dat.gz 1

Length of rules	Sample Size Percentage	Time of execution on whole dataset	Output file path .txt file
2125	1 %	14.196056127548218 seconds	output_pumsb_one
2265	2 %	15.298069953918457 seconds	output_pumsb_two
2332	5%	15.718727111816406 seconds	output_pumsb_five
2380	10%	16.116619110107422	output_pumsb_ten

Dataset file name: pumsb_star.dat.gz

Simple Randomized algorithm
Code to run
python simple_randomized_algorithm.py filename sample_size_percentage.
For eg. python simple_randomized_algorithm.py chess.dat.gz 1

Length of rules	Sample Size Percentage	Time of execution on whole dataset	Output file path .txt file
2204	1 %	14.793367147445679 seconds	output_pumsb_star_one
2325	2 %	15.562747955322266 seconds	output_pumsb_star_two
2366	5%	15.898927211761475 seconds	output_pumsb_star_five
2380	10%	16.271042108535767	output_pumsb_star_ten

While comparing with the simple randomized algorithm vs SON algorithm it came to notice that simple algorithm overall took less time. And when the percentage of sample data increased the frequent itemsets increased for few and the time of execution increased.
If you notice the chess.dat.gz on randomized algorithm takes less time but for SON it takes 46 seconds which more than 3 times.
For connect.dat.gz SON algorithm takes 234 seconds which is way too high but the frequent itemsets are too less.
For mushroom.dat.gz the SON algorithm took more 184 seconds and the frequent itemsets was too less.
SON algorithm for pumsb, T1 dataset T4 dataset and pumsb_star dataset It took a lot of time more than 10 minutes and the CPU usage was at the maximum. Could not generate output for these files for SON algorithm.

The comparison of sample sizes with 1%, 2%, 10% is given above.