Description:

1. SON Implementation:

This implementation is done using Python Language and using Pyspark library for incorporating spark functions.

Versions used:

Spark- 2.2.1
Python- 2.7

SON algorithm makes use of Map-Reduce paradigm in order to find frequent item sets of different size for each chunk and these chunks will be processed in parallel.
Frequent item sets from each chunk will be combined to form the candidate sets and they will be distributed to many processors and each processor inturn will count the support for each candidate in a subset of the baskets, and finally sum those supports to get the support for each candidate itemset in the whole dataset.

2 Phases of SON:

Phase-1: All possible candidate itemsets are determined.
Phase 2: Counts of each candidate itemsets from each partition are summed and those itemsets having support less than the original given support are discarded, remaining itemsets are declared as frequent .

Apriori algorithm is used to generate candidate itemsets, here Monotonocity of itemsets is used to generate candidate itemsize of size+1,i.e we consider singletons which are frequent to generate pairs of frequent itemsets, similarly to generate triplets we use frequent pairs.

First we create baskets of users and movies for two cases. The Algorithm is run on each chunk (obtained using MapPartitions) in the first Map Phase. Support threshold is calculated for each chunk using chunk size and also the total size of dataset.
These candidate itemsets from each chunk are collected from all the partitions and sent to each partition by calling "MapPartitions" to obtain their count in baskets in (key,value) pair format
With (Candidate_itemsets(key), its count(value)).
Phase 2 reduce function --'counts' of each key is summed using "reduceByKey(add)" and they itemsets which have count (support) greater or equal to support are emitted as frequent itemsets.

1. Commands to execute the program:

```
bin\spark-submit Prashanth_Manja_SON.py  <case_num>  <path including Small1.csv> <support>
```

A. Problem 1 : (small dataset)

```
bin\spark-submit Prashanth_Manja_SON.py  <case_num>  <path including Small1.csv> <support>
```

Example:
Case 1:

```
bin\spark-submit Prashanth_Manja_SON.py  1  \mov\Small1.csv 3
```

Case 2:

```
bin\spark-submit Prashanth_Manja_SON.py  2  \mov\Small2.csv 5
```

B.Problem 2 : (ml-latest-small)

```
bin\spark-submit Prashanth_Manja_SON.py  <case_num>  <path including Small1.csv> <support>
```

Example:
Case 1:

```
bin\spark-submit Prashanth_Manja_SON.py  1  \ml-latest-small\ ratings.csv 120
```

```
bin\spark-submit Prashanth_Manja_SON.py  1  \ml-latest-small\ ratings.csv 150
```

Case 2:

```
bin\spark-submit Prashanth_Manja_SON.py  2  \ml-latest-small\ratings.csv 180
```

```
bin\spark-submit Prashanth_Manja_SON.py  2  \ml-latest-small\ratings.csv 200
```

Case 1		**Case 2**	
Support	**Execution_time**	**Support**	**Execution_time**
120	**41 secs**	**180**	**322 secs**
150	**33.64 secs**	**200**	**195 secs**

C.Problem 3 : (ml-20m)

```
bin\spark-submit Prashanth_Manja_SON.py  <case_num>  <path including Small1.csv> <support>
```

Example:
Case 1:

```
bin\spark-submit Prashanth_Manja_SON.py  1  \ml-20m\ratings.csv 30000
```

```
bin\spark-submit Prashanth_Manja_SON.py  1  \ml-20m\ratings.csv 35000
```

Case 2:

```
bin\spark-submit Prashanth_Manja_SON.py  2  \ml-20m \ratings.csv 2800
```

```
bin\spark-submit Prashanth_Manja_SON.py  2  \ml-20m \ratings.csv 3000
```

Case 1		Case 2	
Support	Execution_time	Support	Execution_time
30000	530 secs	2800	595.65 secs
35000	403 secs	3000	510.35 secs