NistDp3 – Differential Privacy 3
Code Guide

John Gardner (gardn999)
jmg@john-gardner.net

Pre-processing:

Field.java defines a Field object with basic information about the field such as name, type, maxval and valid values.

in Main.java...)

Line 9) This is the Laplace function used for adding noise.

Line 14) readSpecInfo returns a map of field name to Field object with information about the field.
  Lines 21 – 50) Using the specs file, add all fields to the map and set Field name, type and maxval.
  Lines 53 – 57) Adds valid values to the Field object using codebook.cbk.
  Lines 70 – 75) Makes sure maxval is included as a valid value.

Line 80) runCommandLine: method to accept command line arguments and run the solution
  Lines 83 - 95) Generates output for epsilon = 8.0, 1.0 and 0.3
  Lines 99 – 107) Generates output for a single epsilon. Saves output to args[2] using epsilon = args[3]

in Solution.java...)

Line 23) Begin running the solution.

Line 32) Initialize random number generator to use a different seed each time.

Lines 42-46) Read the input data set header and create a map of field name to data column.

Line 53) Add each data line to the data list.

Lines 61 – 84) This is where the distinct values for the 5 state-dependent fields are determined. The valid values in the corresponding Field objects are updated. This is skipped if boolean findDistinctStateValues is set to false.

Lines 86 – 123) Set valid values for several fields not included in the codebook.
SUPDIST) 10 to 630 in steps of 10
MIGSEA5) 1 to 502, 990, 991, 997
INCWAGE) 0 to 5000, 999998
VALUEH) 0 to 30000, 9999998, 9999999
EDSCOR50) 0 to 1000, 9999
MIGMET5) 0 to 9600 in steps of 20, 9999
MIGCOUNTY) 0 to 8000 in steps of 10, 9997, 9999
NPBOSS50) 0 to 1000, 9999
IND) 0 to 231, 995, 998, 999

Lines 126 – 136) The number of bins for each field is either the number of valid values defined in the Field object or maxval+1 if undefined.

Lines 140 to 168) Define field groups for highly correlated fields using the field names as parameters.

Lines 232 - 258) Each call to addGroup adds a FieldGroup object to the fieldGroups list. The fields corresponding to the names in the parameter list are added if they exist.

Lines 261 - 320) FieldGroup class is defined. It includes a list of the columns of the fields in the group and a counting histogram named counts. The number of bins in counts is the product of the number of bins in each field in the FieldGroup.

Lines 172 – 172) Any fields not already in a group are placed in a group by themselves.

Lines 177 - 181) Fill the counts histograms using the input data. For each data row, one bin is incremented in the counts array for every FieldGroup.

Lines 273 - 279) FieldGroup add method increments the correct bin.

Privatization:

Line 192) For each group, FieldGroup.privatize(scale) is called using scale = nGroups/epsilon.

Lines 287 – 289) Laplacian noise is added to every bin.

Post-processing:

Line 286) For each FieldGroup, define a noise threshold proportional to scale * log10(number of bins). This will result in a large threshold if there are lots of bins to prevent noisy bins from overwhelming the true counts.

Line 290) If bin counts after adding noise is less than the threshold, set to 0. Otherwise, round bin counts to the nearest integer.

Line 291) To prepare for the synthetic data generation step, add bin counts from the previous bin.

Line 204) This is the start of the loop to generate 800,000 rows of synthetic data using counts weighted random bin values.

Lines 207 - 212) For each FieldGroup, get random values for the fields in the group and and fill the colValues array with these values.

Lines 297 - 320) Get random values for each field in the FieldGroup.

Lines 215 - 217) Use colValues array to create a row of synthetic data.

Line 226) Write the row to the output file.