Important Note: We are using the unbounded DP definition, in which a single user is either added or removed from the dataset (rather than replaced).

High level description of algorithm:

The data synthesizer has the following main steps:

1. **Preprocess**: transform the dataset using supplied domain information as well as domain information manually gathered from the IPUMS website regarding the set of possible values.
2. **Setup:** Select a carefully chosen set of linear queries to make on the data. This is done manually by analyzing the provisional dataset.
3. **Measure:** Use the Gaussian mechanism to answer those queries with differential privacy.
4. **Postprocess:** Find a synthetic dataset that matches the true data with respect to the measured queries.

These steps are explained further below:

Preprocess:

By looking at the codebook, the IPUMS website, and the provided specs file, we identify the set of possible values each column. We do not look at the true data for this step. For the state-dependent columns SEA, METAREA, COUNTY, CITY, METAREAD, we were given permission to identify the set of possible values by looking at the true data. We thus use this in place of the values from the codebook/specs file.

We then load the true data and transform each column to contain values 0, 1, …, k. Specifically, if the set of possible values for a column are v0, v1, …, vk, then we map those to the values 0, 1, …, k. We do this for every column except INCWAGE and VALUEH, which we do something slightly different for.

For the VALUEH column we map each value v to floor(v / 5) if it is less than 25000 and otherwise we map it 5000. We map 9999998 to 5001 and 9999999 to 5002.

For the INCWAGE columns, we break it up into two columns: INCWAGE_A, and INCWAGE_B. If the true INCWAGE is v, then INCWAGE_A is given value floor(v / 100) if v < 5000 and 50 if v >= 5000, and 51 if v = 999998. INCWAGE_B is set to have value v % 100. We then drop the INCWAGE column from the dataset, and only measure things about it through the derived columns.

Additional preprocessing is done. Namely, we load a “cities.json” file which contains the set of valid city codes and their corresponding states. We use this later to infer (from the privatized measurements, not the true data), what the set of valid cities are.

All pre-processing code is done in Mechanism.__init__ and Match3.__init__. Domain information from the codebook and the IPUMS website was done offline, and the relevant information was saved in the domain.json file.

Setup:

In this step, we list out the set of queries to take about the data. The queries were chosen offline, and are hard-coded into the script. The queries mainly consist of 1,2, and 3 way marginals, with a couple caveats described later. Queries are broken up into two groups: round 1 and round2. The round1 queries contain the set of all 1-way marginals. I.e., for each column C and each possible value v we count the number of records in the database that have value v for column C. The round2 queries contain a carefully chosen set of 2 and 3-way marginals. I.e., for each pair of columns C1 and C2 and each possible value (v1, v2) we count the number of records in the database that have value v1 for column C1 and value v2 for column C2. 3-way marginals are defined similarly.

We do not attempt to measure these queries until the next step.

All setup is done in Match3.setup.

Measure:

In step 1, we use the Gaussian mechanism to answer the round1 queries in the previous step (1 way marginals). The amount of noise we add will be discussed later in the section. For each measured marginal, we try to identify the support of the distribution by finding the elements of the noisy vector that exceed some threshold (which is based on the standard deviation of the noise added). From these support values, we compress the domain further. I.e., If v0, …, vt are the values whose counts are above the threshold, we map those values to 0, …, t and we map all other values to t+1 in the true data. We do the above procedure for all columns except for INCWAGE_A and CITY. For INCWAGE_A, we do not try to compress the domain, and keep it as is (0, …, 51). For CITY, we identify the support by using the cities.json file loaded earlier and the noisy marginal.

After compressing the domain of the true data, we take the round2 measurements over the new data by again applying the Gaussian mechanism.

We further weight the queries in round1 and round2 so that both have L2 sensitivity of 1. Specifically, for each query in round1 (resp. round2), we assign it a weight wi. When we answer a query, we return wi*count rather than count. The L2 sensitivity of the collection of queries is

sqrt(sum (wi^2)), because a the addition or removal of a single individual will change one entry of each output marginal by at most wi. The weights wi are explicitly normalized in the code so that sqrt(sum(wi^2)) = 1.

Now we'll discuss how much noise we add to obtain differential privacy. Note that the above procedure is 2-fold *adaptive composition*, since the exact measurements in round2 depend on the output from the noisy measurements in round1. Thus, we use the moments accountant from [1], which allows you to track the privacy loss under k-fold adaptive composition. In the publicly available code [2] for the moments accountant, there are two functions: compute_rdp and get_privacy_spent. As input, you can provide the standard deviation of the gaussian noise (as a ratio with the L2 sensitivity) for the round1 and round2 measurements, and as output you get the (eps, delta)-privacy guarantee for fixed delta. To figure out the minimum amount of noise necessary for a given epsilon, we invert the function by performing a binary search (i.e., we adjust sigma until the epsilon given by the moments account is lower than the epsilon we want by a small amount). Once we figure out the amount of noise we need to add, we actually take the round1 and round2 measurements by using the Gaussian mechanism (as described above in first 2 paragraphs).

One side note: for the (SEX, INCWAGE_A) and (SEX, CITY, INCWAGE_A) measurements, we don't measure the marginal directly. Instead, we carefully discretize INCWAGE_A in the following bins: [0, 1-9, 10-19, 20-50, 51], and measure those bins as a whole. We do this to improve accuracy for score3 by avoiding small counts. This does not affect the sensitivity analysis above: one individual can still only affect one cell of the output by at most wi.

This step is implemented in Match3.measure, which calls functions defined in utility.py and moments_accountant.py.

Postprocess:

In this step, we take all of the measured queries and feed it into an inference engine, which finds a distribution consistent with the measured information. Specifically, we use the technique described in [3], which finds a graphical model and then samples from it. After synthetic data is generated, it is transformed back into the original domain, by reversing the mappings done during the domain compression phase (of the **measure** step) and the domain transformation phase (of the **preprocess** step). This step does not have access to the true underlying data, or any statistic derived from it, other than the privatized measurements taken in the previous step.

This step is implemented in Match3.postprocess, which makes calls to the files in the mbi folder.

[1] Abadi, Martin, et al. "Deep learning with differential privacy." *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016.

[2] https://github.com/tensorflow/privacy/blob/master/privacy/analysis/rdp_accountant.py

[3] McKenna, Ryan, Daniel Sheldon, and Gerome Miklau. "Graphical-model based estimation and inference for differential privacy." *arXiv preprint arXiv:1901.09136* (2019).