Interactive Query Engine for Computing Differentially Private Statistics: System Design

Ally Kaminsky

Harvard University
Mentors: Jack Murtagh, Thomas Steinke, Salil Vadhan

August 14, 2015

1 Introduction

Certain data cannot be made publicly available because of the sensitive nature of the information gathered. After a study is completed, its data is often destroyed or kept private in order to preserve the privacy of the participants. Access to the data often requires going through a lengthy application process. Therefore, other researchers cannot easily validate findings from studies and the data cannot be used to learn anything further. In some of these cases, differentially private aggregate data may be released to researchers so general trends can be learned about the original data without leaking personal information about the individual participants.

The system developed allows researchers to query a data set for specific statistical computations, i.e. the tool supports interactive queries for aggregate data computations and returns real-time differentially private answers. Users explore which statistics can be computed within a certain accuracy while still maintaining a given level of differential privacy to ensure individual level information is kept private. A global privacy budget is tracked so that researchers can explore how many queries for aggregate statistical information they can receive differentially private answers for before personal information can be gathered.

1.1 Background

1.1.1 Differential Privacy

Differential privacy allows the release of global statistics calculated on a private data set while preserving the privacy of individual's data in the data set. This is done by adding a certain amount of noise to released answers of statistical queries in order to mask the effect of any one individual on the aggregate result, while keeping the result as accurate as possible.

Definition. (Differential Privacy). A randomized algorithm $\mathcal{M}$ with domain X^n is (ϵ, δ)-differentially private if for all $\mathcal{S} \subseteq \text{Range}(\mathcal{M})$ and for all x, y $\in X^n$ such that x and y differ on at most one row:

$$Pr[\mathcal{M}(x) \in \mathcal{S}] \leq e^{\epsilon} Pr[\mathcal{M}(y) \in \mathcal{S}] + \delta,$$

where the probability space is over the coin flips of the mechanism $\mathcal{M}$.

Query For the purposes of this report, a query is defined as a request for an aggregate statistical calculation about a data set. A query requires the user of the system to specify certain information such as upper and lower bounds on the variable and privacy parameters specifying how accurate the released answer must be according to differential privacy.

Batch of Queries Queries can be made in batches, or sets. This means that a user can request multiple statistics to be calculated about a data set at once.

Privacy Parameters ϵ_i, δ_i, and accuracy are considered privacy parameters for a query. These parameters determine how much noise should be added to the answer of a calculated aggregate statistic in order to ensure a certain amount of differential privacy.

Privacy parameters are necessary to allow answers for multiple statistical queries to be calculated at one time. The relationship between privacy parameters of individual queries and the overall effect on a data set is explored further in the Privacy Budget definition and Basic Composition Theorem. Smaller values of ϵ_i for a query indicate that the answer will be calculated to a more private level. For smaller ϵ_i values, more noise is added to the released answer, giving less information about the actual data set. In the current system, smaller accuracy levels indicate that the reported answer of the statistical calculation will be closer to the true answer obtained from the data. Accuracy and ϵ_i have an inverse relationship thus, a more accurate answer to a query will take up more privacy budget. A user may indicate accuracy and receive an ϵ_i value as a function of the provided accuracy, or vice versa.

Example. (Query Privacy Parameters.)

For a query requesting a statistical average on ages reported in census data,
Let $\epsilon_i = 0.01$.
Let $\delta_i = 2^{-21}$.
Let the upper bound of the data = 100 years.
Let the lower bound of the data = 0 years.
The released mean will be computed to an accuracy within 0.0002447 of the real answer.

For the same query, increase ϵ_i to 0.05.
This will increase the accuracy of the answer to be within 0.0000489 of the non-private answer.

This demonstrates the tradeoff between accuracy and ϵ_i because as ϵ_i is increased, the statistic is calculated to a more accurate level.

Privacy Budget Privacy budget indicates how much information can be released about a data set over all queries on the data. It is comprised of global parameters about a data set ϵ and δ.

Privacy parameters ϵ_i and δ_i are specific to each query about a data set. These indicate the effect each query will have on the overall privacy budget

of the data set. This effect is quantified through composition theorems. The interactive query system uses basic composition, i.e. the sum of all ϵ_i and δ_i values must add to the global privacy budgets ϵ and δ respectively.

Thm. Basic Composition.

Let $M_1 : X^n \to y_1$ be (ϵ_1, δ_1)-differentially private.
Let $M_2 : X^n \to y_2$ be (ϵ_2, δ_2)-differentially private.
Let $M(x) = (M_1(x), M_2(x))$.
Then M is $(\epsilon_1 + \epsilon_2, \delta_1 + \delta 2)$-differentially private.

Example. (Basic Composition.)

Let M_1 be $(\epsilon_1 = 0.02, \delta_1 = 2^{-21})$-differentially private.
Let M_2 be $(\epsilon_2 = 0.08, \delta_2 = 2^{-21})$-differentially private.
The composition of M_1 and M_2 would allow us to guarantee an overall privacy budget of $(\epsilon = 0.1, \delta = 2^{-20})$ by basic composition.

These global privacy parameters make up a "budget". As a user of the system calculates individual differentially private statistics, the global privacy parameters decrease, i.e. part of the privacy budget is spent. When answers are calculated and reported, information about the original data set is released. Differential privacy guarantees that the amount of information leaked is limited. Intuitively, as information is released about the data, the amount of information that can still be provided safely decreases. As such, global privacy parameters can be thought of as a budget for how much total information can be revealed about the data.

1.1.2 Previous System

The interactive query engine described in this report was built to extend a similar system by Jack Murtagh. The previous system was designed to allow the uploader of a data set to make one batch of queries. Using this system, a user can view how individual query privacy parameters affect the overall privacy budget for a data set. Users set global privacy budget parameters, specify queries to make, enter metadata, and receive individual query privacy parameters. Query privacy parameters are allocated using a similar method to batch query mode, described in section 2.1.2. This system has a graphical user interface written in JavaScript. (Murtagh 2014)

1.2 Goals

The overall goal of this project was to implement a system that allows researchers to make interactive statistical queries about a data set and receive differentially private answers in real-time. The system should support users making an arbitrary number of batches of queries for statistical information on a data set.

2 System Details

The interactive query system allows users to explore queries for differentially private statistics by uploading or selecting a data set, choosing and managing global differential privacy parameters for the data, specifying queries to make about the data, tracking how these queries affect the privacy budget, and calculating private answers to the statistics.

2.1 Design Choices

2.1.1 Users of the System

The query engine has two main groups of users: original uploaders of data sets and future researchers analyzing the data. Different features had to be implemented to support having two types of users. Data uploaders have access to submit a data set, set global parameters at the beginning of the session, change global parameters at any time during the session, and reserve privacy budget for future users. Their queries are also released to any future researchers who explore the data set. Other analysts are able to choose a previously created metadata file to start exploring the data set. They are given a privacy budget and should not able to edit any global parameters. Their queries and differentially private answers will not be released to future researchers. Each user should have his or her own privacy budget, so that the entire global privacy budget is not spent by one user. This allows more than one user to explore the data, enabling interactive queries.

An assumption is made that users will not collude or create multiple accounts in order to gain further information about the data because each user is given an individual privacy budget (over all users, more information is released than can typically be guaranteed by differential privacy). For certain particularly sensitive data, this may not be sufficient because the incentive to leak private information is high enough. This can be handled by only allowing a certain amount of information about the data set to be released to all users in total, i.e. each user gets a small portion of privacy budget. For this very sensitive data case, once this privacy budget for a data set is spent, no further information information should be released to any user. This would help ensure the given amount of differential privacy regardless of whether users collude or set up multiple accounts to gain access to the data. For less sensitive data, it may suffice to require a researcher to provide an institutional email address to create an account for data access and agree to data use terms, for example.

2.1.2 Modes of Allocating Privacy Budget to Queries

The system currently supports two modes of allocating privacy parameters to queries submitted by data analysts.

Individual Query Mode The first mode asks users to specify how accurate an answer should be and allocates an ϵ_i value accordingly. This mode allows analysts to directly indicate the accuracy level required for released answers, i.e. how much noise can be added to the answer of a query. For example, if an answer is calculated to a very accurate level, little noise is added to the differentially private computation. In this case, more information about the true data is released, so more privacy budget is consumed. Individual query mode allows multiple queries to be explored in a batch, but the privacy parameters of one query do not depend on another. This mode gives researchers direct control over how accurate calculations are. However, it does not allow a query to be added or an accuracy value to be improved if it will exceed the overall privacy budget for the data set.

Batch Query Mode In the second mode, queries are assigned privacy parameters in batches. The user specifies what percentage of the remaining privacy budget they want to use on the current set of queries. The indicated amount of privacy budget is spread over the requested queries and accuracy values are assigned based on epsilon values. As the user adds more queries, the individual epsilons for previously entered queries are reduced to allow for more statistics to be calculated. As more queries are added, the accuracy of all queries decrease proportionally to ensure that the same amount of privacy budget is spent for the batch. Thus, the system spreads the batch privacy budget equally over all queries in the batch except in the case that some privacy parameters have already been allocated individually. In the latter situation, the ratio of any previously assigned privacy parameters should be preserved. Queries can also be held constant, i.e. batch query mode does not change the privacy parameters for the held query when new queries are added. Batch query mode gives a user more direct control over the privacy budget used by a batch of queries than individual query mode.

2.1.3 Integrating the Tool with TwoRavens

TwoRavens currently supports the visualization of statistics on data that are not private. In the future, it will also allow differentially private calculations for protected data sets. The data collected and computed by the interactive query system is exported in a way to allow easy integration with TwoRavens. Metadata and other information about the interactive query engine session is exported and saved in a JSON(JavaScript Object Notation) file to easily interface with TwoRavens. The system computes confidence intervals for certain statistics to allow visualization of the error created by the noise added to differentially private computations. (Bu 2015)

2.1.4 Composition Theorems and Tracking Privacy Budgets

The tool uses a modular design for composition theorems. The current version of the system is implemented using basic composition, but any composition theorem can easily be interchanged. Other composition theorems improve the use of the global privacy budget from basic composition, which uses simple summation. If the sum of individual privacy budgets do not equal the amount of global privacy budget spent, it should be explained to the user that individuals epsilon values are upper bounds on how much the query will cost to calculate.

When a researcher requests a query, its `submit` value in the metadata table described in 2.2 is set to 1. A `submit` value of 1 for a query indicates that when the current batch of queries is calculated, the query will be included. If a query's `submit` value is set to 0, it will not be calculated when the batch of queries is calculated. The researcher can choose not to calculate the answer for any query, and only enter the metadata for it. The only queries whose privacy parameters reduce the remaining privacy budget are queries whose `submit` values are set to 1 when the batch is calculated. This is because queries with a `submit` value of 1 are the only ones that will have answers calculated for them. All queries that have a `submit` value of 1 are computed and differentially private answers are released when the user chooses to calculate a batch of queries. Users can change how much privacy budget they want to spend on each query,

how accurate the results should be, and which queries they want to submit in order to explore how batches of queries affect the global privacy budget before making any calculations. This allows the user to track the global privacy budget in a useful way. Once answers to queries are calculated, the budget is reduced because information about the data has been released. The user explores how these calculations will affect the global privacy budget before actually spending it. After a batch of queries is computed and answers are released, the user can then start exploring a new batch of queries to be calculated to use the remaining global privacy budget.

2.1.5 Counting Queries and Transformations

The system allows users to supply a function that takes in the original data provided by the uploader and create a new data column based on the data. Any transformations such as comparison and logical operations are permitted, as long as they can be coded in R. The transformations should only occur on one data row at a time to guarantee differential privacy, i.e. any information about a single observation can be used to create a new data point about itself, but not about another observation. The interactive query engine then appends the new transformed column to the original data. The researcher can then make any query about the new transformed variable as if it was an existing `attribute` of the data. Differentially private answers can be computed for these queries, just as for any other query. Privacy budget is also tracked in the same way.

Differentially private counting queries of binary transformed data can be made by asking the system for a differentially private mean. The mean statistic will return the percentage of data values that are 1 with some amount of noise added to make it differentially private. This value can be post-processed by multiplying it by the total number of observations in the data set to obtain a differentially private count of how many rows meet the conditions tested for in the transformation function.

Allowing users to create their own transformation functions is not the most secure way to handle data transformations. Users could write functions that violate differential privacy in a number of ways (eg, through side channels such as timing). Security measures should be taken to ensure that users maintain differential privacy in their transformations.

Another option for implementing transformations is to only allow researchers to use system provided functions. For example, the query engine could give users the option to choose from a list of transformations such as logical operations and comparisons to transform the data. Some of these transformations have been implemented as examples but are not included in the current version of the interactive query engine. If this approach is taken in the future, user testing should be conducted to determine a useful set of data transformations to implement and make available for users.

2.2 Data Structures

Separate data structures for metadata, answers, and confidence intervals are used in the system. This separation is necessary to allow for answers and confidence intervals of different sizes and types from different statistical computations. The metadata table is implemented as a dataframe in R and contains

information about each query the researcher has made. The metadata dataframe stores:

`Attribute`: variable of the data to make a calculation on

`Type`: the type of the data, ex. "integer"

`Statistic`: the statistical computation to calculate, ex. "mean"

`Upper Bound`: the upper bound of the data

`Lower Bound`: the lower bound of the data

`Granularity`: the difference between possible data values, if required for the statistic

`Number of Bins`: the number of bins for a categorical variable, if required for the statistic

`Accuracy`: the accuracy level the statistic will be computed to, corresponds to how much noise will be added to the calculation

`Epsilon`: the individual statistic level epsilon value, i.e. the upper bound on how much privacy budget will be spent when the query is calculated

`Delta`: the individual statistic level delta value

`Beta`: the individual statistic level beta value

`Submit`: a boolean value indicating if the query is set to submit as described in section 2.1.4

`Hold`: a boolean value indicating if the privacy parameters for the statistic should be held constant when exploring privacy budgets and queries in batch query mode

`Calculated`: a boolean value indicating if the statistic has already been calculated. If this value is set to true, the metadata for the query cannot be modified.

`Answer`: the index of the differentially private answer in an answer list

2.3 User Interface

The current user interface for the system is a command line system. The user first uploads data and sets global parameters to start a new session or chooses a saved metadata file to restore a previous session. Lettered commands are entered for the main functions of the system. The commands are:

2.3.1 A: Add a new query

This command allows a researcher to add a new query on a data `attribute`. The user is asked to specify which `attribute` he or she would like to query from a list of all possible choices and what `statistic` to compute on the specified `attribute` (currently the choices are mean, quantile, and histogram). The user is then asked to provide metadata for the query. If any metadata has already been specified for an `attribute`, the information automatically populates for the new query so that the user is not asked to provide it again.

If the system is set to allocate privacy budgets in individual query mode, the researcher must provide an `accuracy` for the query. `Epsilon` for the query is then calculated from this value. For batch query mode, the batch epsilon is spread over all queries in the metadata table and `accuracy` values are assigned accordingly.

Some metadata values are automatically set. `Type` is set to the class of the data column from the original data provided by the uploader. `Submit` is set to

1, meaning that queries are initialized to be included when the current batch of queries is calculated. `Hold` is set to 0, indicating that in batch query mode, the parameters will change as new queries are added. `Calculated` is also set to 0 because the answer for the query has not been computed yet.
t Answer is set to the index where the answer will be stored in the answer list and the confidence intervals will be stored in the confidence interval lists, which is simply the next available index in the lists.

After the user enters all required information about the data and remaining information is automatically populated, a new row with the query information is added to the metadata table. A new empty row is also added to the answer list and confidence interval lists at the index stored in the answer field. The user may repeat the A command as many times as he or she would like to build up a metadata table of queries to explore.

2.3.2 E: Edit an existing query

This command allows a user to edit any information about a query. The user must specify the `attribute` and `statistic` for the query so that it can be identified. Then the system asks the researcher to indicate which value to change about the query and what the new value should be. Any queries that have already been calculated cannot have their metadata edited.

When some values are edited, further actions must be taken. If `accuracy` or `epsilon` is changed, the other is recalculated. These two values generally have an inverse relationship so when one is edited, the other must be updated. Also, if the `submit` value is changed, the remaining global privacy budget must change accordingly by running the composition theorem on the new set of queries with `submit` values set to 1.

2.3.3 D: Delete a query

To delete a query, the user specifies the `attribute` and `statistic` to identify the query. The system then asks the researcher to verify that the metadata row containing the query should actually be deleted. Once the query is identified and the request to delete a row is confirmed, the metadata, answer list, and confidence interval lists rows corresponding to the query are deleted. This action cannot be undone, but the researcher can add the query to the metadata table again as a new query using the A command. If a statistic has already been calculated for a query, it cannot be deleted.

When a query is deleted, the composition theorem should be called again to calculate the new remaining privacy budget being explored in the current batch of queries. This case is similar to the case presented in the E command when a `submit` value for a query is changed because the set of queries set to be calculated may change.

2.3.4 S: Submit statistics for calculation

When the researcher is satisfied with the current batch of queries with `submit` values set to 1 and how calculating these statistics will affect the global privacy budget, this command is used to call differentially private algorithms and release query answers. The user is asked to verify this action because it permanently

decreases the remaining global privacy budget. Each statistic is then calculated and the answer is released. The remaining global privacy budget is reported after each statistic is computed, so that the researcher can see how each individual query affected the privacy budget. The `calculated` value for each statistic that was computed is set to 1, the `submit` value is set to 0 so that it is not recalculated in the next batch of queries, and the differentially private answer and confidence intervals are stored in the respective lists.

If the mode of the system is set to batch query mode, the user is asked what percentage of the remaining privacy budget to spend on the next batch of queries. This is the privacy budget that will be spread over the next batch of queries to submit.

2.3.5 V: View the metadata table

When this command is entered, the system prints out the metadata dataframe, answer list, and confidence intervals lists.

2.3.6 P: Edit global parameters

This command should be used by the original uploader to edit global parameters after they are initially set. The parameters that can be edited should include global epsilon, delta, and beta, and the amount of privacy budget to save for future users. Data analysts other than the original uploader should not be allowed to use this command to edit most global parameters, as they are given fixed global privacy parameters. The only exception is that they should be allowed to edit the β value.

2.3.7 B: Backup metadata to a file

Data should be backed up between sessions and when interactive query engine session information is sent to TwoRavens or another JavaScript interface. When this command is run, the user specifies a file to save the data in. The data is then written in the file in JSON format. Differentially private query metadata table rows and global session information are stored.

2.3.8 T: Make a transformation on the data

This command asks users to supply a path to a file containing a transformation function. The transformation is then run and the new data column is appended to the old data so that the user can submit queries on the new `attribute`. In the future, the user should be able to specify the transformation at the command line. This could be achieved by writing command line input to a file and then running the transformation the way it is currently implemented.

2.3.9 R: View remaining privacy budget

When this command is run, the system prints out the user's total remaining privacy budget and the remaining privacy budget if the current batch were calculated. When the query engine is set to batch query mode, the batch privacy budget is also displayed to the user. This allows the user to track how much total privacy budget he or she has remaining as well as how much privacy budget

would be left if the user were to calculate answers for the queries with `submit` values set to 1. From these values, the user is able to get a sense of how the current batch will affect the global privacy budget after it is computed.

2.3.10 M: Toggle the mode for allocating privacy budgets over queries

This command toggles the mode of the system from individual query mode to batch query mode, as described in 2.1.2.

2.3.11 Q: Quit the system

This command allows the user to close the command line interactive query engine. No data is preserved when this happens in this prototype. In the future, this should not be an option if differentially private statistics have already been calculated.

2.4 Algorithm Overview

All of the functions described in this section are outlined in the Appendix.

The system is implemented be to modular. The code is separated into components–the user interface system, logic code, and data structures management.

2.4.1 User Interface

The system is implemented such that any interface can be used without major changes. This will allow the system to be accessed through a graphical user interface so it can be integrated with Dataverse and TwoRavens in the future. The user interface code manages the command line user interactions and calls the API created to access the remaining code.

2.4.2 API

The API consists of functions to create and manage data structures such as the metadata table, answer list, and confidence interval lists. The API also contains methods to call logic functions to get privacy parameters, constrain the data to given `upper bounds`, `lower bounds`, and `granularity` values, spread `epsilon` values over queries for batch query mode, manage the remaining privacy budget (i.e. call composition theorems), calculate differentially private statistics by calling various algorithms, and call functions to compute confidence intervals for certain statistics.

3 Future Work

Future work could include improving the code by adding unit tests and fixing other bugs. Also, a user interface in JavaScript could be developed to allow the system to be incorporated with Dataverse and/or TwoRavens.

Another system improvement would be to assign each query a unique idea. This could allow statistics to be recomputed with increased accuracy and would

benefit the current system because users would not have to specify `attribute` and `statistic` values to identify a query.

Work could be done with data transformations. A catalog of transformations for users to choose from could be developed. The specific transformations provided could be determined from user testing.

The security of the entire system should be guaranteed to make sure no data is leaked. This should include verifying that the R function call mechanism prevents data from being accessed by anyone outside the system.

4 Appendix

4.1 Design Document

4.1.1 UI_Interactive.R

This is the user interface for the interactive query system. It asks the user a series of questions to allow him or her to choose queries to explore. The system allows the user to select which queries to compute answers for and "price" them within a global privacy budget before submitting. The user interface supports batch queries, meaning that multiple statistics can be computed at once. Remaining privacy budgets are tracked here. The UI uses a while loop and reads input from the command line to explore queries. The specific commands are detailed in section 2.3. The user interface does not include any functions, but is simply a script designed to handle user input.

4.1.2 DPUtilities_Interactive.R

This file contains the API functions that can be used to call all data structure management and logic code, allowing the system to modular so that a different UI can be substituted for the command line one.

1. new_answer_list()
   This function creates a new empty answer list
   Args: none
   Returns: empty answer list

2. new_answer_row(answer_list)
   This function adds a row to the answer list
   Args:
   answer_list: an answer list to add a new row onto
   Returns: updated answer_list with the new empty row

3. delete_answer_row(df, var, stat, answer_list)
   This function deletes a specific row in the answer_list
   Args:
   df: a dataframe containing the current metadata
   var: the variable of the query identifying the row to delete
   stat: the statistic of the query identifying the row to delete
   answer_list: an answer list to delete a row from
   Returns: updated answer_list without the deleted row

4. set_answer_value(answer_list, index, answer)
   This function stores a computed answer in the answer_list

Args:
answer_list: an answer list to store the answer in
index: the index of the answer_list to store the answer in
answer: the computed statistic to store
Returns: updated answer_list with the answer set

5. get_answer_value(answer_list, index)
This function returns an answer stored in the answer_list
Args:
answer_list: an answer list to store the answer in
index: the index of the answer_list get the answer from
Returns: the answer stored at the indicated index in the answer_list

6. new_df()
This function creates a new empty metadata dataframe
Args: none
Returns: empty dataframe

7. add_query(df, var, stat)
This function adds a new statistic to the metadata dataframe
Args:
df: dataframe, contains the current metadata
var: string, specifies which variable compute a statistic on
stat: string, specifies which statistic to compute on a given variable
Returns: an updated metadata dataframe with the new query added

8. edit_metadata(df, var, stat, value_to_set, value)
This function sets metadata values in the dataframe one cell at a time
Args:
df: dataframe, contains the current metadata
var: string, specifies which variable to set a value for
stat: string, specifies which statistic to set a value for
value_to_set: string, specifies which value to set
value: the actual value to set value_to_set in the table
Returns: an updated metadata dataframe with the new information set

9. get_metadata_value(df, var, stat, value_to_get)
This function returns metadata values from the dataframe
Args:
df: dataframe, contains the current metadata
var: string, specifies which variable to get a value for
stat: string, specifies which statistic to get a value for
value_to_set: string, specifies which value to get
Returns: the value of value_to_get in the dataframe

10. deleterow(var, stat, df)
This function deletes a specified row from the metadata dataframe
Args:
var: string, specifies which variable to delete a query for
stat: string, specifies which statistic to delete a query for
df: dataframe, contains the current metadata
Returns: the updated dataframe without the deleted row

11. get_privacy_param(param, df_row, n)

This function gets accuracy or epsilon for a query
Args:
param: string, indicates whether accuracy or epsilon requested
df_row: the metadata df row for the statistic
n: the number of observations in the dataset
Returns: the requested accuracy or epsilon

12. constrain_data(orig_data, df_row)
This function enforces metadata parameters on the data
Args:
orig_data: column of data to process
df_row: the metadata df row for the statistic
Returns: column of data with the modified values

13. calculate_stats(data, df_row)
This function calculates differentially private statistics
Args:
data: column of data to compute the statistic on
df_row: the metadata df row for the statistic
Returns: the calculated statistic

14. manage_remaining_eps(oldeps, df)
This function calls a composition theorem to compute new privacy budgets
Args:
oldeps: the current remaining privacy budget
df: the current metadata dataframe
Returns: global privacy budget if queries set to submit in the df were calculated

15. checkmetadata(var, stat, df)
Checks if metadata params have previously been entered for an attribute
Args:
var: the variable of the data to check for params
stat: the requested statistic for the query
df: the current metadata dataframe
Returns: a list of metadata params for the requested stat

16. get_numstats(df)
This function returns the number of queries in the metadata table
Args:
df: the current metadata dataframe
Returns: the number of stats in the df

17. spread_epsilon(df, eps, n)
This function spreads the privacy over over statistics for batch query mode
Args:
df: the current metadata dataframe
eps: the batch epsilon to spread over the stats
n: the number of observations in the dataset
Returns: the updated df with new epsilon and accuracy values set for queries

18. getCI(df_row, answer, n)
This function computes confidence intervals for means and histograms
Args:

df_row: the metadata row for the statistic
answer: the differentially private answer
n: the number of observations in the dataset

Returns: a list of confidence intervals, i.e. list(lower bound, upper bound)

4.2 Other Code

This is the underlying code called by the API functions in DPUtilities_Interactive.R.

4.2.1 Calculate_stats_Interactive.R

This code enforces the metadata constraints given by the user and then calls the relevant differentially private algorithms to calculate statistics.

4.2.2 CreateJSON.R

This code takes in interactive query engine session information and returns it in JSON format that can be written to a file.

4.2.3 GetFunctions_Interactive.R

This code handles getting accuracy and epsilon values for queries from the differentially private algorithms.

4.2.4 Metadata_Interactive.R

This file contains functions to handle the dataframe used to store metadata information and the lists to store answers and confidence intervals.

4.2.5 T.R

This is an example transformation that can be used for testing the T command in the interactive query engine.

4.2.6 Transform.R

This file contains some code that serves as a potential example of how to code a catalog of transformations for data in R. This currently contains an example that takes in two comparisons and outputs a new data column accordingly.

References

Bu, Jessica (2015). "Visualization of Uncertainty in Differential Privacy".
Murtagh, Jack (2014). "Releasing Differentially Private Summary Statistics: Overall System Architecture".