User Guide and Technical Manual

Business Dynamics Statistics - Disclosure Avoidance System

Claudia Molinar[a], Pavel Zhuravlev[a], and Matthew Graham[b]

[a]Econometrica, Inc.[1]
[b]U.S. Census Bureau

July 20, 2018

[1]Under Econometrica, Inc. and U.S. Census Bureau contract No.: GS-00F-101CA Order No.: YA1323-15-NC-0265

Contents

Overview

The Business Dynamics Statistics-Disclosure Avoidance System (BDS-DAS) is software created for the Center for Economic Studies (CES) under the guidance and support of the Center for Disclosure Research (CDAR). The programming used to write and develop BDS-DAS and this document was developed by Econometrica, Inc.

The purpose of the BDS-DAS is to apply disclosure avoidance statistical techniques to Longitudinal Business Database (LBD) data in order for Census Bureau economists to derive from it the BDS data product in a formally privatized form that can be released to the public for consumption. The purpose of this document is to provide a user guide and technical reference manual to the BDS-DAS. The user guide section (Chapter 1) is for anyone looking to understand and use the BDS-DAS. The User Guide includes a background and description of the BDS data, why BDS-DAS was created, the modes of BDS-DAS use, and lastly provides a high-level overview of data flow using BDS-DAS, as well as a walkthrough for protected tables production for release and description of the applied privacy protection schema and mechanisms. The Technical Specifications (Chapter 2) focuses on a technical understanding of BDS-DAS which includes software requirements, installation procedures, BDS-DAS Modules Technical Specifications, BDS-DAS Architecture, and concludes with processes and tools used by the first iteration BDS-DAS developers.

List of Abbreviations

bdsgc	BDS+grower+continuer, raw/intermediate form data to produce BDS
BDS	Business Dynamic Statistics
CDAR	Center for Disclosure Research
CES	Center for Economic Studies
CSV	Comma Separated Values, file format
CSvD	Computer Services Division
DA	Disclosure Avoidance
DAS	Disclosure Avoidance System
DRB	Disclosure Review Board
DSEP	Data Stewardship Executive Policy Committee
LBD	Longitudinal Business Database
SQL	Structured Query Language
UDF	Spark user defined function

Chapter 1

User Guide

1.1 Source Data

The Business Dynamics Statistics (BDS) is an annual, public-use data product first published in 2008, produced by the Center for Economic Studies (CES), within the U.S. Census Bureau The BDS is an aggregate of statistics describing the U.S. business dynamics: establishment openings and closings; firm startups; job creation and destruction by firm size, age, industrial sector, and state.

(see example in Table 1.1). The BDS is unique not only because it is a measure of employment growth in United States, but more importantly because it aims to capture structural changes driving net employment growth in the U.S. economy. Using BDS, researchers have published findings such as "Job Creation, Worker Churning, and Wages at Young Businesses" (Haltiwanger et al. [2012]) and "Anemic Job Creation and Growth in the Aftermath of the Great Recession: Are Home Prices to Blame?" (Haltiwanger et al. [2013]).

The BDS is derived from the Longitudinal Business Database (LBD). The LBD is a longitudinal dataset that covers nearly the entire non-farm private economy as well as some public sector activities and is constructed and maintained by CES (Jarmin and Miranda [2002]). The LBD contains data beginning in 1975 and is constructed by linking survey data for all employer establishments contained in the Census Bureau's business register, known as the Standard Statistical Establishment List (SSEL).

The SSEL is the source data to LBD and is used as the LBD frame. SSEL source data are received from the Internal Revenue Service by the Census Bureau for statistical purposes such as the construction of BDS. The IRS sends the Census Bureau three files: the **Business Master File,** which provides name, address, legal form of organization, and tax filing requirements for each business; the **Payroll Tax Return File,** which includes the data fields from Form 941, or an Employer's Quarterly Federal Tax Return; and the **Annual Business Income Tax Return File**. These three files are linked together through the employer's unique Employer Identification Numbers (EIN). In addition to information from the IRS (SSEL), the LBD is also constructed from administrative records data such as industry classification from the Social Security Administration (SSA) and data gathered from the quinquennial Economic Censuses, annual Company Organization Survey (COS) and other surveys (Parker et al. [2000]).

The BDS public-use data product is composed of a series of Firm Characteristics tables and Establishment Characteristics tables. Table A.1 is a summary list of all the tables the Census

Bureau aims to release for the 2018 iteration of BDS. The left column on Table A.1 shows a unique identifier (UID) for each marginal table. The column in the middle lists the variables identifying the cells within the tables. The column on the right lists the measure variables. As the table illustrates, all the BDS measures are included in each table (there are 24 of them). For details on how each measure is derived please see BDS Data Schema section A.2.

The distinction between Establishments and Firm Tables are that the former shows measures in the cells defined by establishment characteristics (size and age), and the latter are based on cells defined by firm characteristics (size and age). The difference between establishments and firms is that establishments denote the physical address where business is conducted or carried out and a firm is defined by a business organization that can consist of more than one physical address (may include more than one establishment).

Users new to BDS-DAS and unfamiliar with the BDS data product users are encouraged to view the previous iterations of the data product on Census.gov website searching “Business Dynamics Statistics” or “BDS” data product.

1.2 BDS and Privacy

A new user of a BDS release is likely to assume that the published BDS tables are true data. However, the BDS source data are IRS data which are subject to disclosure avoidance procedures (U.S.Code), so the released public data turns out not to be the precise true data. Public-use data products published by the Census Bureau, including BDS, must not disclose identifying data of an establishment or firm. This means that the Census Bureau cannot release establishment data that would allow an external user of the BDS to uniquely identify an establishment’s data, whether by statistical inference or through advancements in computational methods.

In the past releases of BDS, the data protection was based on cell suppression disclosure avoidance methodology. The purpose of cell suppression disclosure avoidance is protecting risky cells. Risky cells were identified as cells in a quantitative table that potentially run a risk of their individual contribution being disclosed using the Bureau’s p% rule. For example, if a table provided a summation of two cell values, the Census Bureau would not be able to publish the summation value of the two cells because the owner of one cell value would be able to infer the value of the other (Hundepool et al. [2012]). If there are three cells, and if the dominant player is known and the the “attacker” is one of the other two. The p% rule is used to determine when the cell must be suppressed. This approach assumes that publishing a dataset or data derived from smaller number of observations will always carry a greater risk than publishing datasets from larger number of observations.

Cell suppression aims to reduce the risk that a firm’s information may be identified, but this protection method does not include a formal measure of protection, or a privacy guarantee, a mathematically calculated upper bound on the data inference by attacker. It also does not protect against attackers in possession of external data. For example, statistics from other companies obtained from public filings or other reports. Given advancements in the field of statistical privacy with provable privacy methods, in 2017, John Abowd, Director of Statistical Research and Methodology, determined that a new disclosure avoidance methodology should be applied to BDS (Dajani et al. [2017]).

Even though BDS is not an interactive database that is searchable by the public and is not a

Table 1.1: An example of BDS released table. The table provides the 24 BDS measures by Industry (sic1) for 2015 (BDS). The data dictionary can be found on BDS website https://www.census.gov/ces/dataproducts/bds/

sic1	7	10	15	20	40	50	52	60	70
firms	122796.0	20951.0	369686.0	225130.0	191919.0	299474.0	883761.0	453331.0	2427917.0
estabs	127874.0	27685.0	377184.0	272627.0	306102.0	422091.0	1412890.0	726890.0	2907158.0
emp	1130721.0	777469.0	4436787.0	12288157.0	6692383.0	6698759.0	25433017.0	8072558.0	51962691.0
denom	1110682.0	781649.0	4369695.0	12223652.0	6590519.0	6630229.0	25351507.0	8004503.0	51325370.0
estabs_entry	15769.0	2952.0	37614.0	16365.0	34483.0	29519.0	89316.0	72458.0	310230.0
estabs_entry_rate	12.5	10.7	10.0	6.0	11.3	7.0	6.3	10.0	10.8
estabs_exit	13060.0	2777.0	36651.0	18474.0	30886.0	32389.0	112218.0	61219.0	262051.0
estabs_exit_rate	10.3	10.1	9.7	6.8	10.1	7.6	7.9	8.5	9.1
job_creation	189486.0	132101.0	745085.0	1169720.0	960054.0	824217.0	2709166.0	1249037.0	7260308.0
job_creation_births	61691.0	50265.0	144411.0	241643.0	295582.0	218369.0	781125.0	467383.0	2345409.0
job_creation_continuers	127795.0	81836.0	600674.0	928077.0	664472.0	605848.0	1928041.0	781654.0	4914899.0
job_creation_rate_births	5.6	6.4	3.3	2.0	4.5	3.3	3.1	5.8	4.6
job_creation_rate	17.1	16.9	17.1	9.6	14.6	12.4	10.7	15.6	14.1
job_destruction	149408.0	140460.0	610901.0	1040709.0	756327.0	687157.0	2546145.0	1112926.0	5985665.0
job_destruction_deaths	55864.0	41876.0	175764.0	282821.0	271858.0	252989.0	900057.0	332824.0	2013068.0
job_destruction_continuers	93544.0	98584.0	435137.0	757888.0	484469.0	434168.0	1646088.0	780102.0	3972597.0
job_destruction_rate_deaths	5.0	5.4	4.0	2.3	4.1	3.8	3.6	4.2	3.9
job_destruction_rate	13.5	18.0	14.0	8.5	11.5	10.4	10.0	13.9	11.7
net_job_creation	40078.0	-8359.0	134184.0	129011.0	203727.0	137060.0	163021.0	136111.0	1274643.0
net_job_creation_rate	3.6	-1.1	3.1	1.1	3.1	2.0	0.7	1.7	2.4
reallocation_rate	27.0	33.8	28.0	17.0	23.0	20.8	20.0	27.8	23.4
firmdeath_firms	7938.0	1959.0	24810.0	13426.0	16767.0	21358.0	79796.0	31858.0	178841.0
firmdeath_estabs	7946.0	1970.0	24824.0	13518.0	17334.0	21567.0	81053.0	32226.0	179861.0
firmdeath_emp	34321.0	23105.0	126614.0	148328.0	110873.0	130992.0	551852.0	137203.0	1173743.0

release of microdata (LBD), it is useful (within the context of privacy) to think of the released tables as the Census Bureau's release of answers to predefined queries to the LBD. This approach allows one to apply the recently developed framework of differential privacy (Dwork et al. [2006], Dwork and Roth [2014]) with various extensions and modifications (Kifer and Machanavajjhala [2014], Haney et al. [2017]) to BDS. These "frameworks" are also called "guaranteed privacy frameworks" or "formal privacy frameworks" since these approaches pioneer a form of privacy where privacy can be formally measured, or guaranteed, as opposed to prior methods based on *ad hoc* approaches such as cell suppression.

1.3 Modes of Use of BDS-DAS Software

BDS-DAS is a Python system developed within a general framework for disclosure avoidance systems for U.S. Census Bureau, which is implemented in **das-framework** Python package (described in Appendix B). BDS-DAS can be used to apply formal (or other) privacy protection mechanisms to the BDS data while constructing the BDS product from LBD. The following section provides the modes of use of BDS-DAS.

1.3.1 Support and Development

BDS-DAS is expected to be an integral part of the BDS program and as such will be supported by the Census Bureau. Future BDS-DAS developers will continue to improve the original system by adding new features and thus creating future iterations of BDS-DAS. It can be expected that future BDS-DAS developers will continue to fix bugs, test, improve test routines, and improve system operation and performance. For most of these support and development use cases, fake and/or small data sets will be sufficient, however to accurately measure performance BDS-DAS will need to be tested with large datasets. Some of the test routines will require true data.

1.3.2 Disclosure Avoidance Experiments

Another mode of operation, the one that is anticipated to be used the most heavily, will be running disclosure avoidance experiments. These experiments are needed to determine the "optimal" protection mechanisms and their parameters to use when releasing BDS. Applying confidential protection methods reduces the accuracy of the resulting tables. The more accurate the released tables are, the more information is leaked, and privacy protection is reduced. Thus, there is an inherent conflict or tradeoff between privacy and accuracy. Differential privacy and the Pufferfish frameworks (Kifer and Machanavajjhala [2014]) provides a quantitative definition of privacy. Mostly the privacy is measured by the parameter ε, ranging from $\varepsilon = 0$ (absolute privacy, meaning the public release did not leak any information to any possible adversary which also means the tables are so inaccurate that no information is provided to the public) and $\varepsilon = \infty$ (meaning, there is no formally guaranteed upper bound on the information an attacker can extract from the release). The purpose of running the disclosure avoidance experiments is to measure the accuracy (via the preferred quality measure of the data product owner, publisher, or user –in our case, Census Bureau economists) and privacy to provide information for policy makers, who must balance these requirements. The values chosen for those parameters are a policy decision to be made by the Census Bureau's leadership, which at the time this document was produced has not been made.

The Census Bureau staff will run disclosure avoidance experiments in BDS-DAS; these experiments will involve thousands of individual runs of BDS-DAS. These runs will use the same data, with different privacy parameters, different privacy algorithms, or use different privacy application schemas (e.g. "add noise then aggregate" vs "aggregate then add noise").

Another reason to run large number of experiments is that formal privacy mechanisms operate by adding random noise to the data, therefore, two runs, even with exactly the same settings for privacy mechanism schemas and parameters, will produce different results, and different errors/accuracy. Thus, multiple runs for each settings are needed to estimate the average error (and even more to estimate its variance and other statistics).

Finally, the experiments will yield data to produce plots of (ε, accuracy) to identify various privacy parameters and relative accuracy tradeoffs. These plots are used by policy makers when they set the value of ε and other parameters.

The experiments are performed using a special "experiment" mode of the **das-framework** software (see section 2.5 and Appendix B.2).

1.3.3 Private Data Production

After the privacy parameters and the privacy application mechanisms are set, the economists can run BDS-DAS software with the final approved parameters and create BDS protected data for public release. These users will only need to run BDS-DAS in this regime once per year, as the BDS is released annually. Users will take the data and a template BDS-DAS configuration file, a file with a set of instructions including the parameter settings, (as agreed upon during the experiments and the Census Bureau Data Stewardship Executive Policy Committee(DSEP) and Disclosure Review Board (DRB) approval), and run the BDS-DAS code and obtain the BDS privatized tables.

1.4 Data Flow Using BDS-DAS Software

The following section provides a high-level view of how data flows through BDS-DAS and produces output data in which privacy is guaranteed.

Figure 1.1 provides a visual overview of the data flow through different BDS-DAS modules. The boxes captioned in bold show modules of BDS-DAS. The light grey boxes are modules prototyped in, and called by the **das-framework** (see Appendix B).

1.4.1 True Data Input

The *estab* files in Figure 1.1 are the true data (LBD data), that go to the Reader module, which then performs trims them, leaving only the variables needed to produce BDS tables. The Reader module saves the trimmed file as `.parquet` for future use (if the file already exists, it renames it with its timestamp, and saves a new version derived from the SAS file).

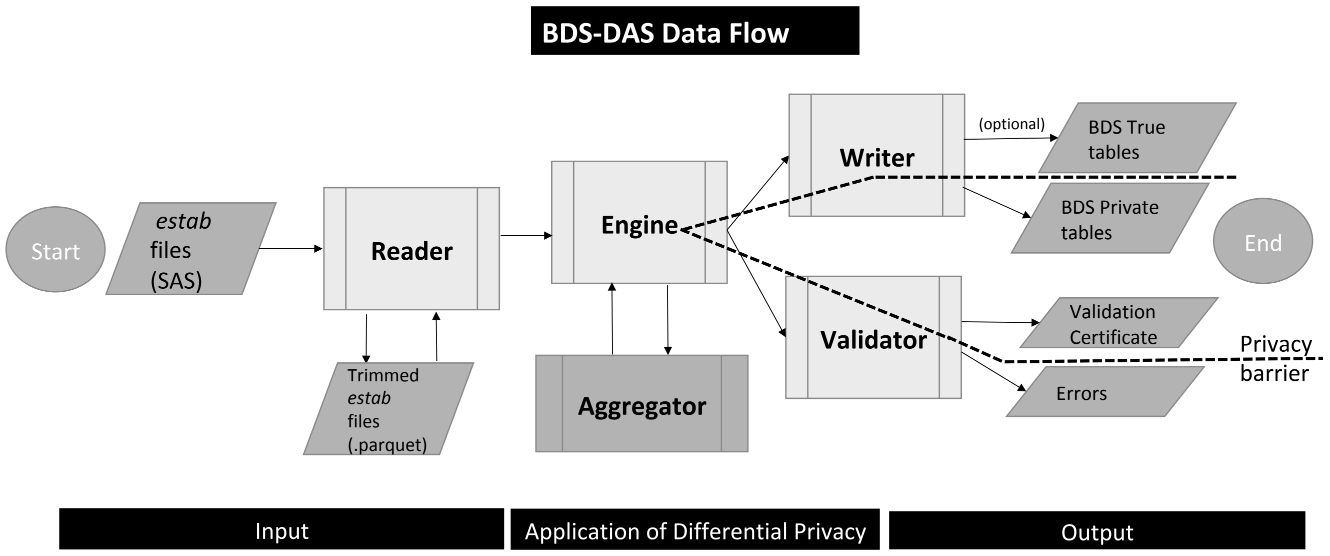

Figure 1.1: Flow of the data through the BDS-DAS modules. The boxes captioned in bold show modules of BDS-DAS. The light grey boxes are modules prototyped in **das-framework**. The Reader module receives an *estab* file in SAS format, trims it and can save as `.parquet` for future use). (It can also read `estab` in CSV or `.parquet` format). The Engine module gets the data (in the form of *estab* file table) and performs from zero to two stages of aggregation, depending on protection schema and mechanisms (see section 1.5.2) with disclosure avoidance noise application before, after or between those stages. The resulting true and protected tables go to Writer module which saves them in CSV format and to Validator module which calculates error/accuracy metrics and prints the privacy certificate.

1.4.2 Application of Differential Privacy

The Engine module takes in the *estab* file tables, directs pre-noise aggregation (if any), and applies the data protection mechanism (adds noise). Then the engine directs after-noise aggregation of both true and noisy data by the Aggregator (both true and noisy datasets are needed to validate the produced tables).

1.4.3 Privatized Output

True and noisy output tables are sent to the Writer and Validator modules. The Writer module outputs the noisy data (and optionally the true data) in CSV format. The Validator module produces a "certificate of disclosure avoidance" affirming that the resulting statistics are now privacy protected. as it compares the true and noisy aggregated data by calculating indicated error measures / quality metrics. The certificate also reflects the history of how the particular noisy data were produced and (possibly) under what privacy parameters.

1.4.4 "Fact-of-Filing" Protection

At the time this document was produced the exact "formal privacy frame" was not decided by Census Bureau's leadership. BDS DA process will not use Pufferfish framework approaches (Kifer and Machanavajjhala [2014]) to protect fact-of-filing. Fact-of-filing is to be protected by other mechanisms distinct from the BDS-DAS. (Establishment counts may still be protected by ε-differential privacy with Laplace mechanism (See Appendix A.3.3))

1.5 BDS Privacy Protection Process

This section provides the user a guide as to how to create BDS tables and how to add noise using BDS-DAS. The first subsection provides information on source data and how to create BDS tables, the second subsection is how to add noise to the created BDS tables, then the final two subsections discuss the production of the final BDS tables and the adjoining certificate which validates the tables are correctly protected.

1.5.1 Getting the Source Data

As described in 1.4.1, the raw data comes in the form of *estab* files in SAS format. These files contain many variables, and the Reader module process the data and leaves only BDS cell variables (see also section A.2 and A.1):

`fage4, fsize, age4, size, ifsize, isize, state, msa, metro, sic1, indetail,`

Note that auxiliary variables `exit` and `entry`, show whether the establishment is new (entry) or dead (exit). Auxiliary variables for employment counts for current and previous year are denoted in *estab* files as `empCUYR` (current employment count) and `empPRYR` (previous year employment count) where `CUYR,PRYR` are placeholders for current and previous year in year format YYYY.

For producing firm counts and firm death firms and establishment counts, the variables firmidCUYR and firmdeath are also kept. Currently, we use firmdeath, exit and entry to define firm death, establishment exit and entry, but they can also be calculated from employment counts. The results do not always coincide.

Source data is extracted from the *estab* files using a statement written in Apache Spark SQL.

The following is an example of a query which can be used to produce a table for establishment and employment counts (which can then be noise-infused and aggregated further to desired level of marginality):

```
SELECT
    fage4, fsize, age4, size, ifsize,
    isize, state, msa, metro, sic1,
    SUM(CAST(empPRYR AS INT)) AS emppy,
    SUM(CAST(empCUYR AS INT)) AS empcy,
    MAX(CAST(empCUYR AS INT)) AS ssmax,
    MAX(CAST(empPRYR AS INT)) AS ssmaxp,
    NOT(CAST(exit as BOOLEAN) OR CAST(entry as BOOLEAN)) as continuer,
    CAST(empCUYR AS INT)>CAST(empPRYR AS INT) as grower,
    COUNT(1) as estabs
FROM estabs
GROUP BY continuer, grower, fage4,
    fsize, age4, size, ifsize, isize,
    state, msa, metro, sic1
```

Thus the establishment level files can aggregated to the level of cells that are defined by interacting of all the cell variables used in BDS release, plus, grower and continuer qualifiers, which are needed to calculate some of the BDS measure variables (see section A.2.2). The continuer is calculated from exit and entry variables rather than from employment counts themselves.

The following is an example of the query for the firm table (by Firm Age). Since the firm counts are not additive, the firm tables have to be produced and infused with noise one by one. The noise infusion happens in the next step.

```
SELECT fage4, firmdeath, COUNT(DISTINCT firmidCUYR) AS firms
FROM estabsfile
GROUP BY firmdeath, fage4
```

1.5.2 Application of Privacy Mechanisms

This section discuses the application of noise highlighted by the dashed line in figure Figure 1.1. In this scenario, the BDS requires BDS noisy tables are to be released. If so, then the following variables in each tables are infused with noise:

- empcy (current year employment), with the specified algorithm with $param1 =$, $param2 =$
- emppy (previous year employment) with the same algorithm and parameters.

- `estabs` maybe with LaplaceDP

Total privacy budget for employment counts (for a release in a given year) is $\varepsilon_{\text{emp}} =$
Total privacy budget for establishment counts (for release in a given year) is $\varepsilon_{\text{est}} =$
Total privacy budget for establishment counts (for release in a given year) is $\varepsilon_{\text{firms}} =$
The noise is applied at the level of tables
...
which are then aggregated into tables
...
In the Econ Census years, the data is corrected and re-released.

Thus, there is $\varepsilon^{\text{emp}}_{\text{program}} = 2\varepsilon^{\text{emp}}_{\text{release}} = 4\varepsilon^{\text{emp}}_{\text{mechanism}} = 2\varepsilon_{\text{emp}}$ for the privacy budget of employment counts, where $\varepsilon^{\text{emp}}_{\text{mechanism}}$ is the privacy budget used within a noise algorithm applied to a single variable. Since, both previous and current year employment are infused with noise, and each is infused and released twice (e.g. 2014 employment count will be used in 2014 release as current year employment and in 2015 release as previous year employment), the counts are treated as non-composable, and privacy budgets spent on each are summed, yielding $\varepsilon^{\text{emp}}_{\text{release}} = 4\varepsilon^{\text{emp}}_{\text{mechanism}}$. Since, all of these will be infused with noise again and re-released in the next Econ Census year, the used privacy budget doubles again, yielding the total budget for the program $\varepsilon^{\text{emp}}_{\text{program}} = 4\varepsilon^{\text{emp}}_{\text{mechanism}}$

The budget for establishment counts is independent (these counts can be thought of as an independent data set release), and there is $\varepsilon^{\text{est}}_{\text{program}} = 2\varepsilon^{\text{est}}_{\text{release}} = 2\varepsilon^{\text{est}}_{\text{mechanism}} = 2\varepsilon_{\text{est}}$ for the privacy budget of establishment counts.

Similarly, for the firm counts $\varepsilon^{\text{firms}}_{\text{program}} = 2\varepsilon^{\text{firms}}_{\text{release}} = \frac{2}{n_t}\varepsilon^{\text{firms}}_{\text{mechanism}} = 2\varepsilon_{\text{firms}}$, where n_t is the number of firm tables that are released, since each table is protected independently.

1.5.3 Production of Final Tables

After that, all the measure variables are calculated from noisy previous and current year counts and establishment counts also using `grower` and `continuer` qualifiers. The measures are then summed over the two latter qualifiers and all the rate measures are calculated. The final tables are used for Census publication to the public.

The true tables are produced in exactly the same way, skipping the noise infusion step.

1.5.4 Validation and Certificate Production

The such-and-such error metrics are calculated from true and noisy data.

1.6 Release Tables Production Walkthrough

The description below is short and general, additional details can be found further in this manual by following references.

1.6.1 Prepare the Software

Make sure the BDS-DAS and all software dependencies and appropriate versions are installed and are working (see section 2.1 for installation procedures)

1. Locate directory with BDS-DAS
2. Change current directory to (`cd /path/to/bds-das/`)
3. Execute `pytest` command
4. Make sure all the tests pass (output is in green saying "PASSED")

If you don't have BDS-DAS, make sure that your system meets all the software requirements listed in section 2.1.1, and the Python that you execute is Python 3. Then install BDS-DAS by cloning the CSvD GitLab repository as described in section 2.1.5. Then run the tests with `pytest` as described above. If there are any errors, they should be instructive of lacking Python modules if there still are any.

1.6.2 Set Up the Output Directory

1. Create a directory where you want protected tables to be written, `/path/to/out/dir`
2. If you want errors to be written in different directory, create it too, `/path/to/err/dir`
3. Copy the BDS-DAS protection template `bds-das-template.ini` file from your BDS-DAS directory to `/path/to/out/dir` and
4. rename the `bds-das-template.ini` file to something more meaningful, reflecting your current purpose

1.6.3 Locate Input Data

The input data for BDS comes in the form of *estab* file in SAS format. However, if BDS-DAS has already been run on the SAS file for the year that you making a release for, it may already be converted to `.parquet` *estab* file. The latest format will work the fastest, *estab* in `.parquet` will also be fast. Reading the SAS file by BDS-DAS will take about 40–50 extra minutes (as tested on app5, and might have been longer due to high load on the network drive, but at least 15 minutes). If you have already run BDS-DAS on this SAS file and have had indicated the directories where to save trimmed `.parquet` *estab* files (default is the current directory), you should use one of latter as your input file (unless your SAS *estabs* file has changed since).

1.6.4 Configure the .ini file

The template file reflects the disclosure avoidance process and settings approved by DSEP/DRB. The only options that you should (or might want to) change are listed below

Input file

You should set the input file name in the `[reader]` section `estab_fname` to the one located previously. You might also want to change relevant options: whether it is SAS, CSV or `.parquet`, and indicate the directory where the trimmed `estab` files are kept. (see also section 2.2.6).

Output directories

The simplest way to change the output directories is to just change the `root` option in the `[DEFAULT]` section to the output directory that you just created `/path/to/out/dir`. In the template, the `output_path` in `[writer]` are `results_path` in `[validator]` are set to `%(root)s` and so will be set to your output path automatically.

Other

You might want to change `name` option in the `[DEFAULT]` section. It is prepended to log file names and to all log messages, so setting it to something corresponding to what you are doing might help to make the logs more organized

For production release, the output of true tables should not be needed, but it can be turned on in the `write_true_data` option in the `[writer]` section.

See also section 2.2 to understand options that are mentioned here and others.

1.6.5 Launch the BDS-DAS

Go to your output directory and run the following command

```
spark-submit /path/to/bds-das-code/bds-das.py configname.ini
```

where `configname.ini` is the name of the config file you edited above. It is possible to launch the program on the background, detaching from the terminal

```
nohup spark-submit /path/to/bds-das-code/bds-das.py configname.ini &
```

The log file should be created, you can follow it with `less` or `tail -f` as the program runs. The errors should be logged in there and/or in `nohup.out` file if you launched with `nohup` or console if you launched without it.

1.6.6 Check the results

Read the log file and make sure it did what expected and used the privacy parameters approved by DRB. Look into *param*_xxxxxxxx.*json* file to see the disclosure avoidance parameters as well. Check the privacy certificate and the output tables.

Chapter 2

BDS-DAS Technical Specifications

This chapter describes software architecture of BDS-DAS for both users and developers. Depending on the reader certain details might not be relevant. It also contains instructions on installation and running of the system, focusing on the U.S. Census Bureau IT environment.

2.1 Installation and Running

This section contains instructions on how to insall BDS-DAS and how to perform production runs

2.1.1 Software Requirements

Under the described modes of use, users should have the following software programs installed into their computers in order to run BDS-DAS:

1. Python v. ≥ 3.5.

2. Python packages as per the contents of `requirements.txt` file for BDS-DAS:

   ```
   numpy==1.13.1
   pyspark==2.2.0
   pyemd==0.4.4
   pytest==3.4.2
   matplotlib==2.0.2
   pandas==0.22.0
   scipy==0.19.1
   ```

   Contact your system administator on the best way to install the packages.

3. **das-framework**. DAS-Framework is a Python package implementing the Census Bureau Disclosure Avoidance Systems Framework, designated to be used in development of all DA systems within the Census Bureau. BDS-DAS is one such system, and so is implemented within the framework. The source code for the framework can be found at

https://adsd-032.ss-inf.nsx1.census.gov/BDS-DP/das-framework

das-framework will be installed automatically as a git submodule of **bds-das**.

4. **dbcompare**. Database (DB) Compare is a Python package developed with the purpose of comparing a true table to a noisy table to calculate errors for validation of the DAS and protected data. DB Compare is built as a separate code from BDS-DAS without reflecting BDS specifics, so it can be used outside BDS-DAS as needed. (see Appendix C). The source code can be found at:

   https://adsd-032.ss-inf.nsx1.census.gov/BDS-DP/dbcompare

   dbcompare will be installed automatically as a git submodule of **bds-das**

5. *git*. For downloading the source codes from the GitLab repository. Not needed if the packages are obtained in some other way.

Python and the supporting packages listed above are needed because Python is the programming language in which BDS-DAS is written. For example, Apache Spark (in the form of `pyspark` package) was used because it is a pre-developed code for large-scale parallel data processing. In addition, other pre-written codes were used such as **das-framework** (see Appendix B), and **dbcompare** (see Appendix C), which is developed in parallel with BDS-DAS and is a package to compare two data tables by a variety of quality metrics. The `pyemd` python package implements the `earth mover distance` measure for comparing two distributions/histograms. The `scipy` package is used to map uniform random number distribution to p.d.f. $\sim \frac{1}{1+z^4}$; `numpy` is used for random number generation and numerical calculations within **dbcompare**; `matplotlib` is for plotting graphs; `pandas` is for operations with data tables before output and for validation and `requests` is for pulling data from the web to produce this manual.

2.1.2 Installation

This section describes how to install the BDS-DAS code (as well as **das-framework** and **dbcompare** required by it) to your system and make sure it works.

2.1.3 Clone the Source Code from CSvD GitLab

In order to be able to download the code from GitLab you need to have *git* installed on your machine and have access rights to repositories for BDS-DAS, **das-framework**, **dbcompare** and **dfxml** with SSH keys (since HTTPS access is prevented in the U.S. Census network via certificate substitution). The repositories addresses are:
https://adsd-032.ss-inf.nsx1.census.gov/BDS-DP/bds-das
https://adsd-032.ss-inf.nsx1.census.gov/BDS-DP/das-framework
https://adsd-032.ss-inf.nsx1.census.gov/BDS-DP/dbcompare

1. Make sure you have *git* and read access rights to the repositories. Below we use the SSH access to the repository, so an SSH key must be configured

2. Change the directory to the parent directory of the location where you want BDS-DAS to reside

3. Clone the repository with the following command:

   `git clone -recursive git@adsd-032.ss-inf.nsx1.census.gov:BDS-DP/bds-das.git`

4. If there is error "Clone of 'https://github.ti.census.gov/CB-DAS/dfxml' into submodule path 'dfxml' failed Failed to recurse into submodule path 'dasframework'", then

   (a) `cd dasframework`

   (b) Edit file `.gitmodules`, replacing url parameter under `[submodule "dfxml"]` with git@adsd-032.ss-inf.nsx1.census.gov:BDS-DP/dfxml.git

   (c) Edit file `.git/config`, replacing url parameter under `[submodule "dfxml"]` with git@adsd-032.ss-inf.nsx1.census.gov:BDS-DP/dfxml.git

   (d) Update the dfxml submodule: `git submodule update`

   (e) Return to the parent directory: `cd ../../`

5. Enter into the BDS-DAS directory:

   `cd bds-das` (or `cd bdsdasdirectoryname`)

2.1.4 Run Tests

From the directory where you cloned **bds-das** repository, run the PyTest tests by executing

```
pytest
```

Tests are small pieces of code that check various parts of the system ensuring that the system can run properly. For example, they will fail is some package is missing on your system, and it is a fast and reliable way to find out about this kind of problems, far in advance before setting up the production run. Make sure that the Python that is run by PyTest is Python 3 (it will be in the first line of PyTest output). If it is not, the following way command can be run instead of `pytest` to use the python executable that you specify

```
python3 -m pytest
```

(where `python3` is an executable running Python 3, which may have a different name on your system)

From the output, make sure that the all tests pass (in this case, it will end printing in green and indicated number of passed tests and will not contain the word “failed”).

Otherwise, the output will end in red, and list how many and which tests have failed and how many and which have passed.

If only one test fails, run the tests once again (one test uses random numbers, so can fail in rare cases).

If tests keep failing, it may be helpful to increase PyTest verbosity by adding `-v` or `-vv` option to the command (`pytest -vv` or `python3 -m pytest -vv`). The output should be helpful to identify the problems (most likely lacking python modules or wrong versions).

2.1.5 Running

BDS-DAS has a main executable *bds-das.py*. This file must be called with one positional argument, the path to the config file. For example, it can be run as

```
python3 /path/to/bds-das.py /path/to/config.ini
```

(`python3` may be omitted, if `bds-das.py` has permission for the user to execute). Also, on some systems, executable with name *python3* may not exist, in which case an executable that runs Python 3 (version ≥ 3.5) should be used).

The above command will run BDS-DAS in the current directory (i.e., if "`./`" is used in the config file, it will correspond to the directory from which you have run that command).

The BDS-DAS has several command-line options (which are actually **das-framework** options) that can be seen by adding the `--help` option. However, most of its operation and behavior are set up via the config file. The one exception is the `--experiment` option, which turns on the experiment regime. See the BDS-DAS Architecture section 2.5 of this manual for more details on single run vs experiment regimes of running the software.

The config file specifies the path to the input file, directories where to write protected data and errors data. Also, it specifies, which tables to produce from *estab* file table before application of noise, to which variables in those pre-aggregated tables apply noise and how (algorithm + parameters). Finally, it also specifies which error metrics to calculate in the Validator module. For a more detailed description of the config file see section 2.2, the BDS-DAS Architecture section 2.5 and Config File Appendix A.1.

2.1.6 Running through spark-submit

Spark jobs should be launched by *spark-submit*. The simplest way is to use the `spark-submit` instead of `python3` executable. There are also various options, for instance to designate specific amount of processors to the task. For example:

```
spark-submit --master local[4] bds-das.py config.ini
```

will submit the Spark job using 4 processors on the local machine.

The documentation on *spark-submit* options is found at https://spark.apache.org/docs/latest/submitting-applications.html

2.2 Config File

Most of the behavior of BDS-DAS is set by the config file, which uses standard *.ini* configuration files syntax: it is split into sections, under which option names and values are listed:

```
[section_name_1]
option1 = option1value
option2 : option2value
```

```
[section_name_2]
option3 = 123
option4 = apple
```

The option names are separated from values by a colon (:) or equal sign (=). For options that are boolean, the values of “1”, “true”, and “on” correspond to the option being on, and “0”, “false”, and “off” to the option being off.

The following is the list of all config file options used by BDS-DAS by section, with explanations. It may be useful to read the BDS-DAS Architecture section 2.5, where the options are also explained next to example pieces of config file.

2.2.1 [DEFAULT]

The options below are all **das-framework** options, not specific to BDS-DAS.

- `name`. The name of the program instance, used in the log files
- `root`. (Optional) A path to the directory can be set and referred to as `%(root)s` later in the file, for instance to write output and validation results to. In experiment mode, it is set automatically to the value of `--experiment` command line option

The `root` option is an example of config file variable. Other config file variables can also be set here and referred to later.

2.2.2 [ENVIRONMENT]

This section allows the user to set up the OS environment variables for the program runtime (**das-framework** option)

2.2.3 [experiment]

In experiment section, the user of BDS-DAS can set looping for various parameters, over which experiment is conducted. The program will run for each combination of parameters defined in the loops. The options below are all **das-framework** options, not specific to BDS-DAS.

- `loop1`. The syntax is one of the following:
  1. `FOR variable_name = start_value TO end_value STEP step`. The variable will loop from sstart_value toend_value, adding `step` each time.
  2. `FOR variable_name = start_value TO end_value MULSTEP step`. The variable will loop from `start_value` to `end_value`, multiplying `step` each time.
  3. `FOR variable_name IN value1, value2, ...valueN`. The variable will loop iterating over the list of values.

The syntax of variable names is `section_name.variable_name`. For example, if variable `epsilon` is set in `[engine]` section, where it is used for disclosure avoidance, use `engine.epsilon` when specifying loops in this section. The `[engine]` section must have `epsilon` set to some value.

- `loop2`. Next loop. The user may specify any number of loops
- . . . more loops . . .
- `Scaffolding`. This option tells **das-framework** the name of the file and class implementing the Scaffolding module. For BDS-DAS it should always be set to `Scaffolding.Scaffold`

2.2.4 [spark]

- `local_dir`. This sets the Spark "spark.local.dir" option, which is the place where Spark stores temporary files, essentially a scratch disk. It should be on a local disk with fast access, and have enough space. The default is `/tmp`, which is usually the best. However, sometimes on Census Bureau's servers `/tmp` directory is not maintained and runs out of space. The program will crash if there is not enough space in Spark local directory, which is why this option was implemented in BDS-DAS. On the other hand, setting it to be on a "storage" drive where data are kept can result in unsatisfactory performance, since many of those drives are accessed via network or other means, which are slower than the local bus, or just too heavily used by other users, resulting in very long access time, so that the program will spend most of the time waiting to access the disk. For the current use of BDS-DAS the best is to not set this option (i.e. use the default `/tmp`), as it is extremely unlikely to run out of space with the current BDS DA process.

2.2.5 [setup]

- `setup`. This option tells **das-framework** the name of the file and class implementing the **setup** module. For BDS-DAS, it should always be set to `bds_das_spark_setup.setup`

2.2.6 [reader]

- `reader`. This option tells **das-framework** the name of the file and class implementing the **reader** module. For BDS-DAS, it should always be set to `bds_das_spark_reader.reader`
- `estab_fname`. The input *estab* file name (including path, using full path will be the least error-prone). Can be in `.sas7bdat, .csv` or `.parquet` format.
- `trimmed_estab_dir`. When reading SAS files, BDS-DAS drops all unneeded variables and sets types of the rest and saves the result in `.parquet` format. Default is current directory (`./`). If the file already exists in this directory, it will be renamed with the timestamp, and a new file will be created.

2.2.7 [engine]

- `engine`. This option tells **das-framework** the name of the file and class implementing the **engine** module. There are two engine modules implemented in BDS-DAS (with possibility of more). One is a general Spark SQL DAS Engine, which has no BDS specifics in it, and operates via SQL queries supplied in the config file. It is implemented in `spark_sql_das_engine.py` file. The other is a subclass of `spark_sql_das_engine.engine` implemented in bds_das_spark_engine. file. It implements an extra step of automatic generation of SQL queries for the set of tables to release, according to a particular DA schema and algorithms. (It does not perform this step if the set of released tables is not indicated in config file and works identically to `spark_sql_das_engine.engine`). More subclasses can be created for different schemas, or this one can be modified to take in more options. Thus, for BDS-DAS, the `engine` option of the `[engine]` section should always be set to `bds_das_spark_engine.engine`.

- `seed`. Random seed to generate random numbers used in differential privacy disclosure avoidance algorithms. Should be set to `urandom` (which is the default), or not set at all, for all production (i.e., making tables for release) runs of BDS-DAS. If set to an integer value, the results will be reproducible, i.e. another launch of the program with the same data and the same config file will produce identical output. This can be useful for testing and debugging, but should NEVER be used in production mode, since knowing the seed and seeing the BDS-DAS source code allows an adversary to reverse-engineer the true values of the data. The `urandom` regime is also generally faster, since it does not involve joining the random numbers with data across Spark partitions.

- `epsilon`. This is the total ε used for all variables and all tables, also known as "release" ε. If any differential privacy algorithms are used, this option has to be set. To change ε in an experimental loop for more than one variable simultaneously, the ε (spent on all those variables) has to be set here, by this option, and referred to as `engine.epsilon` in the loop specifying syntax of the `[experiment]` section. In addition to `epsilon`, in the `[engine]` section, the user can set any parameters used by any of the privacy algorithms specified in following sections.

- `alpha`. (Optional) As noted above, there can be any number of parameters used by disclosure avoidance algorithms to be specified here, not just ε. For instance if α is to be the same for all, or almost all, variables to which noise is applied, it can be set here rather than in the sections for individual variables.

- `run`. (Optional) One can also set any variable in any section. For instance, setting `run` here is a convenient way to have multiple runs with the same set of disclosure avoidance parameters, looping over `engine.run` in the `[experiment]` section (However, `run` can really be set in any other section, including the `[DEFAULT]`, but it might make bookkeeping sense to keep all the changing parameters in one place.)

- `input_table_name`. In BDS-DAS, the input data comes as a single table. The engine sets up the table name in the SQL environment (i.e. table name to be used in SQL queries to refer to the input data table) equal to the value of this option. Can be a comma separated list, if you need several different tables supplied to the engine. Default value is `estabsfile` reflecting the fact that input tables in the current BDS disclosure avoidance process are the *estab* files. Make sure it does not coincide or is fully included into any of the variable names in the queries, since it can interfere with changing queries to apply to noisy tables. (e.g., in BDS we use "estabs" to denote Number of Establishments, so it should not be named "estabs" or "estab", but can be named "estabsfile")

- `ignore_privacy_guarantee_check`. Some of the implemented privacy algorithms (from Haney et al. [2017]) only guarantee privacy (i.e. produce data that complies with their formal privacy definition) if the privacy parameters (such as ε, α and δ) satisfy certain conditions. However, for *SmoothLaplace* algorithm from Haney et al. [2017], the δ parameter only enters in this condition, but does not actually influence the output private data. Thus, the experimental exploration of parameters can be made just once for all δ, and then some of the (ε,α) combinations should be dropped, depending on δ. In such case, this flag can be used, to perform the calculations and output results even if they don't satisfy formal privacy guarantee condition for these particular $(\varepsilon_i, \alpha_i, \delta_i)$, because the results will satisfy it for the same ε_i and α_i, but some other δ_j and so can be used. The default value is `False`.

The options above can be used with either `spark_sql_das_engine.engine`
or `bds_das_spark_engine.engine`.
The options below are only used by `bds_das_spark_engine.engine`.

- `tables2release`. Specifies which BDS tables to release. Each table is defined by a list of variables, separated by comma (,). The lists (corresponding to tables) are separated by semicolon(;). See example in Section 2.5

- `basetable`. list of variables setting the base table, to which the noise is applied, and all the tables2realease are marginal with respect to it and produced from it. See example in Section 2.5

2.2.8 [prenoise_tables]

This section sets up which tables are to be produced from the input data (which is a single table, in case of BDS, and has a name in Spark SQL environment (see the [engine] section above)). The tables, to which the noise will be added, are produced from the input table as results of SQL queries applied to it. Every option in this section is a table name, and its value is the SQL query, returning this table.

- `tablename1`. SQL Query producing the table named "tablename1"
- `tablename2`. SQL Query producing the table named "tablename2"
- ...

```
tablename1 = SELECT var1, SUM(var2) as var3 FROM estabsfile GROUP BY var 1
tablename2 = SELECT SUM(var3) FROM tablename1
```

The queries are processed in the order listed, so tables produced in the preceding queries can be used in the following queries (see more details on using SQL and examples below in section 2.2.17, BDS-DAS Architecture description in section 2.5 and Appendix A.1)

2.2.9 [NoNoiseDA]

This purpose of this section is to support disclosure avoidance protocols other than differential-privacy-like noise infusion, such as cell suppression. Therefore, this section is not to be used in the current BDS disclosure avoidance process. Like [prenoise_tables] it also produces tables from the input data table, and those tables can be also subsequently subjected to noise infusion if needed. Every option in this section is a table name, and its value is the SQL query, returning this table, like in previous section (see more details on using SQL and examples below in section 2.2.17, BDS-DAS Architecture description in section 2.5 and Appendix A.1)

2.2.10 [out_tables]

This section sets up which tables are to be in the output.

They are usually produced from prenoise tables (specified in [prenoise_section]): directly, for the true data output, and after noise infusion for the noisy data. Every option in this section is a table name, and its value is the SQL query, returning this table, like in the previous sections. The same query produces the true table and the corresponding noisy table, the only difference is the table name (which is taken care of inside the program, by prepending "noisy_" to the name indicated here, for noisy tables). See more details on using SQL and examples below in section 2.2.17, BDS-DAS Architecture description in section 2.5 and Appendix A.1

2.2.11 Pandas manipulation sections

After the noise infusion and production of output tables the user might still want to perform some operations on the output tables, such as remove some columns or add columns corresponding to new variables, calculated from existing variables and/or perform more aggregation.

In principle, it is possible to do all these things within the SQL queries in [out_tables] section, but that may make the queries too long and cumbersome, and so these operations are also implemented in BDS-DAS with the use of Pandas package.

They proceed in four stages: adding variables, aggregation, adding variables to aggregated, dropping variables.

In BDS, for example, *job_creation_continuers* and *job_creation_continuers_rate* variables are calculated from previous and current year employment (conventionally denoted as *emppy* and *empcy* in BDS-DAS), as well as auxiliary variable *denom*, which is just the average of *emppy* and *empcy*. To calculate *job_creation_continuers*, the difference between *empcy* and *emppy* is summed over the cells with both *grower* and *continuer* qualifiers being `True` (see section A.2). These two qualifiers do not enter into final output tables, and all the variables which are linear combinations of *empcy, emppy, grower* and *continuer*) can calculated individually for the cells with different values of *grower* and *continuer* and then summed (aggregated) over different values of *grower* and *continuer*. The rates, however, such as *job_creation_continuers_rate* which is *job_creation_continuers* divided by *denom*, are not linear, and have to be calculated for the final cells, being divided by the final, total (i.e. summed over *grower* and *continuer* values) *denom*.

Hence, the need for the three staged add-variables – aggregate – add-more-variables process.

[out_tables_add_vars_pandas]

In this section the option names are table names, and values are lists of variable names to add. The variables are separated by comma (",") and semicolon (";") separates pre-aggregation and post-aggregation list.

- `tablename1`. variable1, variable2, variable3; variable4, variable5
- `tablename2`. variable1, variable2, variable3; variable4, variable5
- ...

For the BDS example described above:

```
[out_tables_add_vars_pandas]
tablename = denom, job_creation_continuers; job_creation_continuers_rate
```

The names of variables to add must correspond to the names of existing functions in the *variable_calulations.py* file of the DAS source code.

In BDS-DAS, all the BDS measure variables are implemented, namely

- *denom*
- *net_job_creation*
- *estabs_exit*
- *estabs_entry*
- *firm_death_estabs*
- *firm_death_firms*
- *job_creation_continuers*
- *job_creation_births*
- *job_destruction_continuers*
- *job_destruction_deaths*
- *job_creation*
- *job_destruction*
- *estabs_exit_rate*
- *estabs_entry_rate*
- *net_job_creation_rate*
- *job_creation_rate_births*

- *job_destruction_rate_deaths*
- *job_creation_rate*
- *job_destruction_rate*
- *reallocation_rate*

The order of the variables in the list in [out_tables_add_vars_pandas] section matters.

They are calculated in that order, and some of them use the previously calculated ones. For instance, *net_job_creation* only needs *empcy* and *emppy*, but all the other job creation and job destruction variables use *net_job_creation*.

The order above is a correct one.

[out_tables_aggregate_pandas]

The option names are table names, and the values are lists of variables over which to aggregate. The aggregation operation follows after a semi-colon (";"). Aggregation operation are those supported by Pandas *groupby*, i.e. *sum, mean, max, min* etc.

(See also http://pandas.pydata.org/pandas-docs/stable/groupby.html)

- `tablename1`. variable1, variable2; aggregation_operation
- `tablename2`. variable1, variable2; aggregation_operation
- ...

For the BDS example described above:

```
[out_tables_aggregate_pandas]
tablename = grower, continuer; sum
```

[out_tables_drop_vars_pandas]

The section has the same structure as the sections above, with options being table names and the values being lists of variables to drop from final output tables. Some variables are auxiliary and only needed to apply disclosure avoidance or calculate other variables. Examples in BDS include *denom, grower, continuer* and local/smooth sensitivity columns, conventionally denoted as *ssmax* and *ssmaxp* in BDS-DAS. Note, that the tables are not strictly private if they contain local sensitivities, as they leak information about the data set, so they MUST be dropped from the final output tables.

It is fine to list variables, that don't exist in the table, it won't produce an error. For the BDS example above:

```
[out_tables_drop_vars_pandas]
tablename = denom, grower, continuer, ssmax, ssmaxp
```

2.2.12 [variables2addnoise]

This section sets up to which particular variables in the pre-noise tables the noise will be added. The structure is similar to the above section: the option name is table name and the option value is a list of variables in that table, which are to be masked by noise.

In the current BDS disclosure avoidance process only previous and current year employment will be subjected to noise addition.

The list has two types of separators: the semi-colon (";") divides groups of composable variables, and comma divides variables within a composable group.

Operationally, "composable variables" means that they share an ε bugdet, i.e. if each one of them uses ε_1 while adding noise to it, the total ε spent on this composable group will be ε_1, rather than the sum of all ε used for each variable, as will be the case for non-composable variables.

By mathematical formal privacy sence, "composable" is roughly equivalent to being independent, but one has to be a formal privacy expert to be able to really figure out whether particular variables are composable in a particular disclosure avoidance setting.

As an example, in BDS, previous and current year employment are independent, and therefore, composable (at least, if only a single year of BDS were to be released). In contrast, if one first calculated, say, *denom*, *job_ creation_ births* and *job_ destruction_ continuers* and only then added noise to these variables – they would not have been composable, so if ε was used in privacy algorithm for each of the three, the total privacy budget used would have been 3ε (maybe, less, but definitely more than 1ε as in composable case, and not more than 3ε). The current BDS DA process only protects employment counts. If one also wanted to protect establishment counts (conventionally denoted as *estabs* BDS and BDS-DAS), and if formal privacy expert determined that establishment count should be treated as not composable with employment counts, and there was only single year release of BDS, the settings would look as follows:

```
[variables2addnoise]
tablename = empcy, emppy; estabs
```

The privacy budget allocation could be, for example like this: if *empcy* uses 0.5ε, *emppy* uses 0.5ε and *estabs* uses 0.5ε, then the total will be ε, since composable *empcy, emppy* only use 0.5ε total, while still using 0.5ε each.

If several tables are listed in this section, then variables within each one will be infused with noise, and then privacy budgets of each table will be added to obtain the total ε used.

In the current approach to BDS DA, *empcy* and *emppy* are treated as not composable, since each count is released twice: as current year employment, and then, next year, as previous year employment, so each release gets its own budget; and *estabs* is not protected so the most likely option to be used (it is set in the template config file in 2.2) is the following:

```
[variables2addnoise]
estabsfile = empcy; emppy
```

Examples with multiple tables are in section 2.2.17.

2.2.13 Privacy Algorithms Application to Variables Sections

Each of the variable of each table listed in [variables2addnoise] section must have a corresponding section in the config file, where the details of noise application to that variable are set.

The names of the section are formed as [tablename_variablename], and options depend on particular noise algorithm used.

- `algorithm`. The name of the noise algorithm. NoNoise, LaplaceDP, LogLaplace, SmoothLaplace, SmoothGamma and Pub1075 are currently implemented (See Appendix 2.4 for algorithm details).
- `epsilon_fraction`. Fraction of total privacy budget (equal to the `epsilon` option set in `[engine]` section to spend on this variable.)
- `sensitivity_field`. For smooth sensitivity algorithms, the name of the variable (column) of the table, that stores the local/smooth sensitivity. For BDS-DAS and employment counts it is conventionally named `ssmax` and `ssmaxp`
- `alpha`. The α parameter for Pufferfish framework algorithms that mask counts up to multiplicative factor.
- `delta`. For SmoothLaplace algorithm, δ parameter.
- `p`. For Pub1075 algorithm, p-parameter.

Parameters, specific to algorithms, i.e. `alpha`, `delta`, `p` and `sensitivity_field` can be set up in the [engine] section, which sets them for all the variables that are infused with noise. If the value is set in both places, the value set in this section overrides the value set in [engine] section. For the BDS example above (protecting only employment counts), for SmoothLaplace algorithm, for example, the settings could be as follows:

```
[engine]
epsilon = 2.0
alpha = 0.05
delta = 0.05
engine = bds_das_spark_engine.engine

[estabsfile_empcy]
algotithm = SmoothLaplace
epsilon_fraction = 0.5
sensitivity_field = ssmax

[estabsfile_emppy]
algotithm = SmoothLaplace
epsilon_fraction = 0.5
sensitivity_field = ssmaxp
```

2.2.14 [writer]

This section sets options related to writing the private and true tables to the hard disk.

- `writer`. This option tells **das-framework** the name of the file and class implementing the **writer** module. For BDS-DAS it should always be set to `bds_das_spark_writer.writer`.
- `output_path`. Path to the directory where noisy and (possibly) true tables files are to be written
- `write_true_data`. Boolean option, whether to output true tables. Default is `False`.

2.2.15 [validator]

This section sets options related to calculating errors and writing them to disk along with the certificate.

- `validator`. This option tells **das-framework** the name of the file and class implementing the **validator** module. For BDS-DAS it should always be set to `bds_das_spark_validator.validator`.
- `cell_id_vars`. List of variables (column names) in the tables that define the cell (such as state, msa, firm age in BDS) as opposed to measure variables, such as employment or job creation
- `results_path`. Path to the directory where the files with calculated errors are to be written
- `euclid_norm_dim`. One type of error that **validator** calculates are L^p measures, which are p-norms of data vectors, defined for difference between a true and noisy data column as

$$L^p = \left(\sum_i |x_i^{\text{true}} - x_i^{\text{noisy}}|^p \right)^{1/p} \tag{2.1}$$

  and equal to Euclidean norm at $p = 2$. This value of this option is list of values of p to calculate L^p. For instance, if user wants L^1 and L^2 norms to be calculated, the value of the option should be `1,2`
- `error_metrics`. Which other error metrics to calculate among the metrics implemented in **dbcompare** (See Appendix C).

```
[validator]
validator = bds_das_validator.validator
cell_id_vars = state,msa,sic1,age4,size,isize,fage4,fsize,ifsize,metro,continuer,grower
euclid_norm_dim = 1,2,5,10
error_metrics = chisq,spearman_rank,jsd
results_path: %(root)s
```

2.2.16 [takedown]

This section sets options for **takedown** module. In BDS-DAS **takedown** module does not do anything, so there are no options except for the one required by **das-framework**

- `takedown`. This option tells **das-framework** the name of the file and class implementing the **takedown** module. For BDS-DAS it should always be set to `bds_das_spark_takedown.takedown`.

2.2.17 SQL in Config File

BDS-DAS runs on Apache Spark v.2+ and uses Spark SQL (Structured Query Language), allowing operations on the data within Spark using SQL queries. Pre-noise and post-noise aggregation of data in BDS-DAS are managed by SQL queries in the config file, the `[prenoise_tables]` and `[out_tables]` sections, respectively.

Sample Usage

Imagine an economist wants to release two tables: T1.1 (Firm Age) and T512.1 (Industry) from Table A.1. The economist starts with the *estabsfile* table, which has default name *estabsfile* in BDS-DAS Spark SQL space (it can be changed in config file option `[engine]/input_table_name`). The economist can go about it in two ways:

1. Aggregate *estabsfile* to T513.2 (Firm Age × Industry), apply noise to it, and then aggregate T513.2 to T1.1 and to T512.1.

2. Aggregate *estabsfile* to T1.1 and apply noise to it; and aggregate *estabsfile* to T512.1 and apply noise to it.

For the first case, put the following query into the `[prenoise_tables]` section to aggregate *estabsfile* to T513.2 (for employment counts only):

```
[prenoise_tables]
t513 = SELECT fage4, sic1, SUM(empcy) AS empcy, SUM(emppy) AS emppy FROM estabsfile GROUP BY fage4,sic1
```

where t513 is an arbitrary table name to be used in following queries. Then put the following to the `[out_tables]`:

```
[out_tables]
byind = SELECT sic1, SUM(empcy) AS empcy, SUM(emppy) AS emppy FROM t513 GROUP BY sic1
byfage = SELECT fage4 SUM(empcy) AS empcy, SUM(emppy) AS emppy FROM t513 GROUP BY fage4
```

And the noise is infused only into the base table

```
[variables2addnoise]
t513 = empcy; emppy
```

In the second case, put

```
[prenoise_tables]
byind = SELECT sic1, SUM(empcy) AS empcy, SUM(emppy) AS emppy FROM estabsfile GROUP BY sic1
byfage = SELECT fage4 SUM(empcy) AS empcy, SUM(emppy) AS emppy FROM estabsfile GROUP BY fage4

[out_tables]
byind = SELECT * FROM byind
byfage = SELECT * FROM byfage
```

The noise is infused into both tables, spending twice the privacy budget.

```
[variables2addnoise]
byind = empcy; emppy
byfage = empcy; emppy
```

Of course, many more options of order of aggregation and noise application are possible if more tables are to be released, and SQL queries provide a flexible way to do so. One can even use SQL to apply non-formal disclosure avoidance methods like cell suppression. If suppression depends on a parameter that changes in experimental loop, it can be included into the query via config file variable functionality (`%(varname)s`) and should be put into the `[NoNoiseDA]` section of the config file, which is recalculated on every experimental run, unlike the values specified in the `[prenoise_tables]` section.

2.3 True Data Input

The data from which BDS (both true and protected) can be produces comes in the form of *estab* files (currently produced by CES) which are then aggregated trimmed by BDS-DAS, to reduce size and only keep data needed to produce the protected BDS tables.

2.3.1 *estab* files.

These are the form in which LBD true data are received (*estab* is short for Establishment files). These files are in SAS format (.sas7bdat). BDS-DAS supports this format as the input format. After reading a SAS input file, it saves a trimmed copy of it in .parquet format, if the corresponding option is set in config file. For subsequent run, this .parquet file may be indicated as input file to save time. Table 2.1 shows a listing of all *estab* files including size of the data. The table also shows that each file contains data of the associated year and the year before, which is necessary, as creators of BDS data product will need to calculate year-over-year changes or net variables that make up BDS data product.

2.3.2 Trimmed *estab* files.

The BDS-DAS software Reader module trims the *estab* files, leaving only the indicators needed to produce the protected BDS tables and saves the result an `.parquet` format (for possible future use).

Table 2.2 shows the filenames and their sizes. Note: At the time this document was produced, not all the *estab* files had been trimmed)

Table 2.1: *estab* files containing LBD data to produce private BDS data files

Filename	Size
estabs19761977.sas7bdat	3.0G
estabs19771978.sas7bdat	3.2G
estabs19781979.sas7bdat	3.3G
estabs19791980.sas7bdat	3.4G
estabs19801981.sas7bdat	3.4G
estabs19811982.sas7bdat	3.4G
estabs19821983.sas7bdat	3.5G
estabs19831984.sas7bdat	3.5G
estabs19841985.sas7bdat	3.7G
estabs19851986.sas7bdat	3.8G
estabs19861987.sas7bdat	3.9G
estabs19871988.sas7bdat	4.0G
estabs19881989.sas7bdat	4.0G
estabs19891990.sas7bdat	4.1G
estabs19901991.sas7bdat	4.1G
estabs19911992.sas7bdat	4.2G
estabs19921993.sas7bdat	4.2G
estabs19931994.sas7bdat	4.3G
estabs19941995.sas7bdat	4.4G
estabs19951996.sas7bdat	4.5G
estabs19961997.sas7bdat	4.6G
estabs19971998.sas7bdat	4.7G
estabs19981999.sas7bdat	4.8G
estabs19992000.sas7bdat	4.8G
estabs20002001.sas7bdat	4.9G
estabs20012002.sas7bdat	4.9G
estabs20022003.sas7bdat	4.7G
estabs20032004.sas7bdat	4.7G
estabs20042005.sas7bdat	4.8G
estabs20052006.sas7bdat	4.8G
estabs20062007.sas7bdat	4.8G
estabs20072008.sas7bdat	4.8G
estabs20082009.sas7bdat	4.7G
estabs20092010.sas7bdat	4.6G
estabs20102011.sas7bdat	4.6G
estabs20112012.sas7bdat	4.6G
estabs20122013.sas7bdat	4.6G
estabs20132014.sas7bdat	4.7G
estabs20142015.sas7bdat	4.5G
Total	165.7G

Table 2.2: *estab* files containing LBD data to produce private BDS data files

Filename	Size
estabs19881989.parquet	69.0M
estabs19891990.parquet	69.8M
estabs19901991.parquet	71.4M
estabs19911992.parquet	72.6M
estabs19921993.parquet	72.2M
estabs19931994.parquet	73.5M
estabs19941995.parquet	74.9M
estabs19951996.parquet	76.4M
estabs19961997.parquet	78.4M
estabs19971998.parquet	78.8M
estabs19981999.parquet	79.5M
estabs19992000.parquet	80.0M
estabs20002001.parquet	81.0M
estabs20012002.parquet	82.9M
estabs20022003.parquet	79.8M
estabs20032004.parquet	81.3M
estabs20042005.parquet	86.3M
estabs20052006.parquet	88.7M
estabs20062007.parquet	90.5M
estabs20072008.parquet	89.9M
estabs20082009.parquet	88.5M
estabs20092010.parquet	86.4M
estabs20102011.parquet	85.9M
estabs20112012.parquet	86.8M
estabs20122013.parquet	86.7M
estabs20132014.parquet	86.1M
estabs20142015.parquet	86.1M
fakeaggestabs1.parquet	0.0M
Total	2183.5M

2.4 Privacy Algorithms Implemented

The supported algorithms are the following:

- `NoNoise` This is a pass-through with no noise applied;
- `LaplaceDP` This is the "usual" differential privacy algorithm which adds noise that follows the Laplace distribution which uses the epsilon parameter for sensitivity (see also Appendices A.3.3, A.3.4 for algorithm description and how it can be used with a protection mechanism for BDS);
- `SmoothLaplace`, `SmoothGamma` and `LogLaplace` from Haney et al. [2017] (see also Appendix A.3.5 for algorithm description and how it can be used with a protection mechanism for BDS);
- `Pub1075` A suppression method of certain cells with no complimentary suppression based on rules from IRS Pub. 1075 (see also Appendix A.3.1 for algorithm description. It was implemented as for testing purposes and should not be used in production. It is not a formal privacy method, and results in extremely poor accuracy).

2.5 BDS-DAS Architecture

This architecture description is meant to provide a high level structure of BDS-DAS as well as provide the software specifications of each module.

In the Architecure description below the following notation is used:

- [section_name] – in square brackets, the name of config file section
- [section_name]/*option_name* – config file option under config file section
- **module_name**, **ClassName** – bold font denotes names of the modules and classes
- *variable_name* – variables and config file options are shown in italics

BDS-DAS consists of a collection of modules that perform a single Disclosure Avoidance (DA) "run" or a single application of DA to the data, which is driven by the **das-framework**. There are two regimes (relevant to this context of module architecture description) that BDS-DAS is designed to run:

- Single run, called using:

```
bds-das.py configfilename.ini
```

- Experiment, which performs multiple single runs with different sets of parameters and collects data on accuracy of private (noisy) tables, called using:

```
bds-das.py configfilename.ini --experiment=/path/to/experiment/directory
```

Note: A BDS-DAS user should launch BDS-DAS via *spark-submit* (like with any other Apache Spark application). For the current way of BDS-DAS use, memory allocated for Spark driver and executors by default (1 Gb) should be enough, but it may be needed to allocate more to prevent the program from crashing (throwing Java OutOfMemory error, Java heap space or similar). For example, one might use the following:

```
spark-submit --master local[12] --driver-memory 16G --executor-memory 4G bds-das.py
    /path/to/config.ini --experiment=/path/to/exp/dir/
```

which directs Spark to run locally on the machine using 12 processes, and setting memory allocation for driver to 16 Gb, and 4 Gb to each executor. For reference, see the *spark-submit* documentation found here:

https://spark.apache.org/docs/latest/submitting-applications.html

In the experiment regime, the **experiment runner** module of the **das-framework** takes care of creating "single run" config files for single runs from the "master" config file given to the main executable (*bds-das.py* here), which calls the **das-framework** driver, and setting the root directory for the output to the value of the

```
--experiment
```

command line option. Modules are largely agnostic about whether the regime is "single run" or "experiment", treating everything as a single run (except for caching some data). Each single run is performed with a set of parameters defined in the loops in the [experiment] section of the "master" config file. Typically, one of these loops is for the "run" variable, which does not change any DA parameters, but makes each set of parameters run multiple times; another is "master", "release", or "effective", ε (if you're testing/using a differentially private algorithm), which is combined from individual ε-s, spent by individual variables in individual tables-to-be-released. Yet another loop may be for an α parameter in one of the Pufferfish framework algorithms that conceal exact counts down to multiplicative intervals.

The modules described below run with a single set of these parameters and output noisy tables corresponding to that set and (parameters, Errors) data; and also print a certificate.

2.5.1 BDS-DAS Config File

The config file specifies a series of options and parameters for running the program. Each item below is an option or group of options. Each module uses a combination of these options or group of options. Example excerpts of the config file, along with more detailed descriptions of the options, are given in the module descriptions below, as well as "Config File" section (2.2) of the Technical Manual. A commented template config file *bds-das-template.ini* is in the repository and also is in "Config File Example" Appendix (A.1) of the Manual.

- Root directory. If there is an

  ```
  --experiment
  ```

  command line option, then the root directory is set to that option.

- Loops for variables/parameters changing within the experiment.
- The names and module (python file) names for all the modules: **setup, reader, engine, writer, validator, takedown,** in the corresponding section (of the same name, i.e. [setup], [reader] etc.).
- Name and module name for **scaffold** module, in the [experiment] section.
- Which file contains the raw data to produce the true tables to apply disclosure avoidance to and in which format. For BDS, the data come as so called *estab* files (see description in the Technical Manual section "True Data Input" (1.4.1)), in .csv or .sas7bat formats.
- Aggregation to perform before applying noise in the form of SQL queries for SparkSQL, including for true tables.
- Disclosure Avoidance aggregation to perform before or instead of applying noise (the same form), but not for true tables.
- Aggregation to perform after applying noise (the same form), including for true tables.
- A list of variables (fieldnames in data) to add noise to, with corresponding algorithms and parameters, and additional data needed for algorithms, e.g. smooth sensitivity (as field name in data).
- Lists of variables for each output table to add in Pandas (names of functions in aggregator).
- Lists of variables for each output table over which to aggregate/summarize in Pandas with summarizing operation (names of variables; operation)
- Lists of variables for each output table to drop in Pandas (names of variables).
- Random seed (the seed, if reproducible results are desired, or a special value "urandom" to take it from /dev/urandom, if production privacy and faster operation is desired).
- Where to write noisy (and true, if wanted) tables, directory as a string.
- Where to write errors+parameters, directory as a string.
- Variable names that define cells (such as State, MSA, Firm Age, Firm Size, as opposed to variable names defining the measures, such as Employment, Number of Establishments, or Net Job Creation).
- Which error metrics to calculate (L1, L2... Jensen-Shannon divergence, Spearman rank etc.).

Config File Examples

- https://adsd-032.ss-inf.nsx1.census.gov/BDS-DP/bds-das/blob/master/bds-das-template.ini.
- https://adsd-032.ss-inf.nsx1.census.gov/BDS-DP/bds-das/blob/master/tests/et_config.ini.
- https://adsd-032.ss-inf.nsx1.census.gov/BDS-DP/bds-das/blob/master/tests/gentestconfig.ini.

2.5.2 Scaffolding Module

Scaffolding is a class that keeps/caches the information needed between the runs of the experiment. Implemented as a Singleton with one field, "val", which is a dictionary where everything is kept.

2.5.3 Setup Module

Performs actions needed to set up a single DAS run.

Setup Module Operation

- Checks the Complete Flag in the folder, terminates if True (should be implemented in **AbstractDASSetup**).
- Sets Spark configuration parameters, such as *spark.local.dir.*
- Sets the random seed, if it has not been set before (that is, in a previous run in experiment regime; the **setup** module checks the Scaffolding to find out), and if indicated in config.

Setup Module Inputs

- No external inputs, only internal fields, *self.config*, a **ConfigParser** object, a field within the **AbstractDASModule** class.

Setup Module Config Options

1. [engine]/*seed*, default=urandom
2. [spark] /*local_dir*, default=/tmp/

This following example sets a particular random seed in the config file. The alternatives are to set a different number, set seed = urandom, or not set the seed (the same as using "urandom"). When producing the tables for release, the seed should be obtained from /dev/urandom, not set to a particular value.

```
[setup]
setup = bds_das_spark_setup.setup

[engine]
seed = 100
engine = bds_das_spark_engine.engine
```

The *local_dir* option in [spark] section sets Spark "spark.local.dir" option, which is the place where Spark stores temporary files, essentially a scratch disk. It should be on a local disk with fast access, and have enough space.

Outputs of the Setup Module

- *setup_ data*, object containing references to other objects needed for operation, in this case (BDS-DAS), just returns *self*, which has access to config.

2.5.4 Reader Module

Reads input data from the specified file (*estab* file().

Reader Module Operation

- Checks whether the input file has been already read (that is, in a previous run in experiment regime; the **reader** module checks the Scaffolding to find out)
- Reads the *estab* file (if it is in SAS format, trims it, by only keeping variables needed to produce private BDS tables, and saves it as .parquet)

Reader Module Inputs

- *config*, a **ConfigParser** object
- *setup_ data* the output of **setup** module

Reader Module Config options

1. [reader]/*input_fname*
2. [reader]/*trimmed_ storage_ dir*, directory where trimmed estab files converted from .sas7bdat are saved and kept as .parquet

```
[reader]
reader = bds_das_spark_reader.reader
input_fname = /path/to/bdgsc.csv
```

or

```
[reader]
reader = bds_das_spark_reader.reader
input_fname = /path/to/estabPRYRCUYR.csv
```

or

```
[reader]
reader = bds_das_spark_reader.reader
input_fname = /path/to/estabPRYRCUYR.sas7bdat
trimmed_storage_dir = /path/to/trimmed/estab/files/in/parquet/format/dir
```

For *estab* file trimming, the reader module will look for two previous and current year values in YYYY format in the *input_fname* option (denoted as PRYR and CUYR in the example above).

If the *estab* file is in SAS format, the reader module will read it and save as

```
/path/to/trimmed/estab/files/in/parquet/format/dir/estabPRYRCUYR.parquet
```

If file with that name already exists, it will rename it with a timestamp. If the program is stopped before SAS reading and writing as .parquet is finished, the directory will have .parquet.partial extension. If the file

```
/path/to/trimmed/estab/files/in/parquet/format/dir/estabPRYRCUYR.parquet.partial
```

exists, it will be deleted first.

Reader Module Outputs

- *original_data*, dictionary with “original_data” field referencing the Spark DataFrame obtained from input file.

2.5.5 Engine Module

This module applies actual disclosure avoidance to the tables supplied to it. **das-framework** includes an engine module implemented in *spark_sql_das_engine.py* which does not have any BDS specifics and just performs disclosure avoidance based on SQL queries and [table_variable] sections set up in the config file.

This module has an empty prototype function to modify config file, *elaborate_config()*, which can be used in the subclasses to automate generation of SQL queries and other disclosure avoidance parameter setup for a particular project / set of experiments.

Subclass implemented in *bds_das_spark_engine.py* is one example that uses this function, to produce SQL queries and [table_variable] sections based on the list of BDS tables to release and sets of variables defining the pre-noise(“base”) table(s), i.e. the most fully crossed table (noise is applied at that level).

Engine Module Operation

- Creates sections in the config, that are needed for **spark_sql_das_engine** operation, based on list of pre-noise(base) and output tables provided in the config file
- The sections include [prenoise_tables], [out_tables], [variables2addnoise], and set of [table_variable] sections (see SparkSQLDASEngine documentation in **das-framework**, also in~B)
- Creates noise algorithms that are used and the set of pre-noise tables
- Constructs true tables from pre-noise tables
- Applied noise to pre-noise tables

- Constructs noisy tables with the exact same queries (except for table names) as for true tables
- Creates *privitized_data*, list of Spark or Pandas DataFrames, and changes *original_data*, adding *original_data["true_tables"]*, *original_data["noisy_tables"]* from the **Aggregator** output, *original_data["params"]*, containing parameters of noise application, such as algorithms, values of ε-s and other parameters and *original_data["exvars"]*, containing names and current values of variables in experimental loops.

Engine Module Inputs

- *original_data*, a dictionary including a Spark DataFrame under original_data["original_data"]
- *config*, a **ConfigParser** object
- *setup_data* the output of **setup** module

Engine Module Config Options

1. [engine]/*epsilon*, desired total/effective/master (do we now call it "release"?) ε
2. [engine]/*input_table_name* (name of SQL table of the input file before any aggregation, default = estabsfile). Make sure it does not coinside or is fully included into any of the variable names in the queries (e.g., in BDS we use "estabs" to denote Number of Establishments, so it should not be named "estabs" or "estab", but can be named "estabsfile")
3. Variables in experimental loops and their current values
4. [engine]/*seed*, default=urandom
5. "Master" parameters for all privacy algorithms
6. [engine]/*ignore_privacy_guarantee_check*, default=False. For SmoothLaplace, calculate and output the noisy results even if the smooth sensitivity is unbounded / there is no privacy guarantee (the results don't depend on δ, so could be used with another δ)
7. [engine]/*tables2release* - list of lists of variables. Each sublist defines a BDS table to release. Sublist items divided by comma (,), lists divided by semicolon(;)
8. [engine]/*basetables* - list of lists of variables. Each sublist defines a prenoise/base table to which the noise is applied, and all the tables2realease are marginal with respect to one of them and produced from it. Sublist items divided by comma (,), lists divided by semicolon(;)

Example:

```
[engine]
epsilon = 8.0
alpha = 0.05
delta = 0.05
```

```
engine = bds_das_spark_engine.engine
basetables = fage4,fsize,naicssector,msa
tables2release =
    fage4,fsize,msa;
    fage4,fsize,naicssector;
    fage4,fsize;fage4,naicssector;
    fage4,msa;
    msa;
    fage4;
    naicssector;
    fsize;
    total
```

Engine Module Outputs

- *privitized_ data*, dictionary with "true tables," "noisy tables," parameters of performed DA, and values of experimental loop variables.
- *true_ data*, constructed before applying noise.

2.5.6 Validator Module

Receives data from the **engine** module, pulls out true and noisy tables; parameters and other metadata; calculates relative errors for each cell; sends the true and noisy tables to **dbcompare** to calculate error measures; saves the results of error calculations together with metadata.

Validator Module Inputs

- *original_ data*, dictionary with "true tables" constructed by **engine** before applying noise, and "noisy tables." Each is a dictionary of Spark or Pandas dataframes. *original_ data["expvars"]* contains experimental loop parameter names and values.
- *written_ data*, filename of the noisy data written (not used in BDS-DAS validator).

Validator Module Config options

1. [validator]/*cell_ id_ vars* (list of field names in the data that define the cell as opposed to being a measure/value)
2. [validator]/*error_ metrics* (list of error metrics to calculate)
3. [validator]/*euclid_ norm_ dim* (list of L1, L2, etc. errors to calculate)
4. [validator]/*results_path* (where to write results)

```
[validator]
validator = bds_das_validator.validator
cell_id_vars = state,msa,sic1,age4,size,isize,fage4,fsize,ifsize,metro,continuer,grower
```

```
euclid_norm_dim = 1,2,5,10
error_metrics = chisq,spearman_rank,jsd
results_path: %(testdir)s
```

Validator Module Outputs

- PDF certificate
- Errors in the form of dictionary with "relative," "metrics," and "stat." Saved to the hard drive as individual .csv files (each table) and as a single pickle file. The file names contain experimental parameters and their values which are taken from the dictionary item *original_data*["expvars"], which is filled by **engine** module by reading loops in [experiment] section of the config file, even in the single run regime.

Validator Module Operation

- From true and noisy tables in the object calculates Errors using specified Error Functions
- Prints certificate

2.5.7 Writer Module

The purpose of this module is to export the final dataset after disclosure avoidance.

Writer Module Inputs

- *data_to_write*, tuple of dictionary of Pandas DataFrames with noisy data, dictionary of metadata and true tables

Writer Module Config Options

1. [writer]/*output_path*, path to output the tables
2. [writer]/*write_true_data*, whether to output the true tables
3. [writer]/*add_param_cols_to_data*, whether to include columns corresponding to the experimental parameters (such as number of run, values of ε, α) directly in the output dataframe

```
[writer]
writer = bds_das_writer.writer
output_path = /path/to/output/dir
write_true_data = 1
```

Writer Module Outputs

- CSV files with noisy (and optionally true) data, JSON file with parameters/metadata

Writer Module Operation

- Forms the output file name for noisy tables as "tablename_parameters_values.csv". The parameter values are taken from the dictionary with metadata and true tables item *data_to_write*[1] ["expvars"], which is filled by **engine** module by reading loops in [experiment] section of the config file, even in the sigle run regime
- Looping over tables, saves each as CSV
- Checks if the *write_true_data* is on, and saves true data as CSV
- Saves parameters to JSON file, "params_parameters_values.json", with the same parameter values as above
- Returns a list of filenames with noisy tables

2.6 Processes and Tools Used to Develop BDS-DAS

The following section describes the processes and tools used to develop the first iteration of BDS-DAS.

2.6.1 Unit Tests

PyTest. The original founding BDS-DAS developers used **pytest** (v. ≥ 3.4) to build and run unit test and other tests of **bds-das** and **dbcombare**. The tests reside in the *tests* subfolder of **bds-das**. The original developers aimed to have close to 100 percent coverage of the code by unit tests. There are also a few tests that run the whole **bds-das** on a complete config file. One such test is a DAS0 test, where path-through algorithms that do not apply noise are used on all variables, and the results are compared with true data, to which both of these data sets should be theoretically identical.

2.6.2 Coding Standards

The original founding BDS-DAS developers largely follow PEP 8 (van Rossum et al.) and Google Python Style Guide (Patel et al.), with the following exceptions:

- Line length is 120 as per PEP8, but not strictly followed.
- Class names for **das-framework** modules are all-lowercase rather than CamelCase.
- Some function names have capital letters, e.g. when it's a proper name, like uniform2Laplace.

The original founding BDS-DAS developers ran all code through the PyCharm IDEA code inspector and TODO: pylint Even though the Python virtual environment used by the original development team was based on Python 3.6.3, no Python 3.6 specific (i.e., later than 3.5) features were used in the code.

2.6.3 Version Control

The original founding BDS-DAS developers used **git** for version control and Computer Services Division (CSvD) GitLab to host the code repositories for **bds-das** and **dbcompare**. Each team developer committed and then pushed their changes to those repositories at the end of the working day (at a minimum). Branching has not been used so far (no need), everyone on the team worked on the master branch. There are client-side git hooks preventing the controlled unclassified information (CUI) data to be pushed. (Each developer was responsible for installing the hooks to their checkout directory. The hooks, however, reside in the **das-framework** repository, so this can be done via symlink)

Git hooks can be found here: https://adsd-032.ss-inf.nsx1.census.gov/BDS-DP/das-framework/tree/master/hooks.

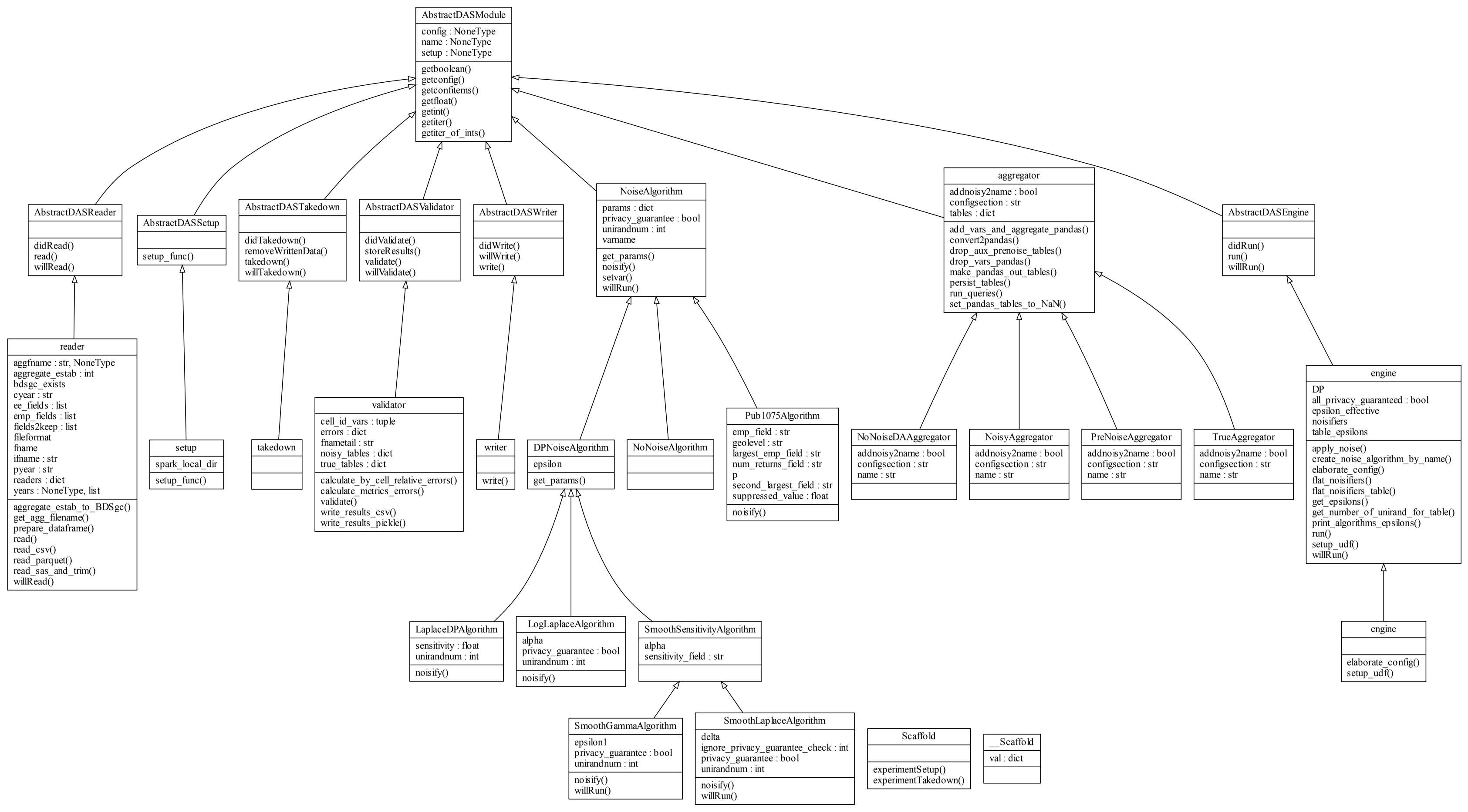

Figure 2.1: BDS-DAS and das-framework class hierarchy.

Bibliography

Bds business dynamis statistics. http://www2.census.gov/ces/bds/2015/firm/bds_f_sic_release.csv.

Aref N. Dajani, Amy D. Lauger, Phyllis E. Singer, Daniel Kifer, Jerome P. Reiter, Ashwin Machanavajjhala, Simson L. Garfinkel, Scot A. Dahl, Matthew Graham, Vishesh Karwa, Hang Kim, Philip Leclerc, Ian M. Schmutte, William N. Sexton, Lars Vilhuber, and John M. Abowd. The modernization of statistical disclosure limitation at the u.s. census bureau. Technical report, U.S. Census Bureau Scientific Advisory Commitee, 2017.

Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, (3-4):211–407, 2014. doi: 10.1561/0400000042.

Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Proceedings of the Third Conference on Theory of Cryptography*, TCC'06, pages 265–284, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 3-540-32731-2, 978-3-540-32731-8. doi: 10.1007/11681878_14. URL http://dx.doi.org/10.1007/11681878_14.

John Haltiwanger, Henry Hyatt, Erika McEntarfer, and Liliana Sousa. Job creation, worker churning, and wages at young businesses. Technical report, Ewing Marion Kauffman Foundation, 2012.

John Haltiwanger, Javier Miranda, and Ron Jarmin. Anemic job creation and growth in the aftermath of the great recession: Are home prices to blame. Technical report, Ewing Marion Kauffman Foundation, 2013.

Samuel Haney, Ashwin Machanavajjhala, John M. Abowd, Matthew Graham, Mark Kutzbach, and Lars Vilhuber. Utility cost of formal privacy for releasing national employer-employee statistics. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, pages 1339–1354, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4197-4. doi: 10.1145/3035918.3035940. URL http://doi.acm.org/10.1145/3035918.3035940.

Anco Hundepool, Josep Domingo-Ferrer, Luisa Franconi, Sarah Giessing, Eric Nordholt, Schulte, Keith Spicer, and Peter-Paul de Wolf. *Statistical Disclosure Control*. John Wiley & Sons, Ltd, 2012.

Internal Revenue Service. Tax Information Security Guidelines for Federal, State and Local Agencies: Safeguards for Protecting Federal Tax Returns and Return Information. Publication 1075, Internal Revenue Service, Washington, DC, USA, September 2016. URL https://www.irs.gov/pub/irs-pdf/p1075.pdf. Rev. 11-2016.

Ron S Jarmin and Javier Miranda. The Longitudinal Business Database. Working Papers 02-17, Center for Economic Studies, U.S. Census Bureau, July 2002. URL https://ideas.repec.org/p/cen/wpaper/02-17.html.

Daniel Kifer and Ashwin Machanavajjhala. Pufferfish: A framework for mathematical privacy definitions. *ACM Trans. Database Syst.*, 39(1):3:1–3:36, 2014. doi: 10.1145/2514689.

Robert P. Parker, James R. Spletzer, and Michael Searson. The business establishment list - standard statistical establishment list comparison project. Technical report, Federal Economic Statistics Advisory Committee, June 2000. URL https://www.bls.gov/ore/pdf/st010030.pdf.

Amit Patel, Antoine Picard, Eugene Jhong, Jeremy Hylton, Matt Smart, and Mike Shields. Pep 8 – google python style guide. https://google.github.io/styleguide/pyguide.html.

Colleen M. Sullivan. An Overview of Disclosure Principles. Statistical Research Division Research Report Series RR-92/09, U.S. Census Bureau, Washington, DC, USA, September 1992. URL https://www.census.gov/srd/papers/pdf/rr92-09.pdf.

U.S.Code. *26 U.S. Code §6103.* URL https://www.law.cornell.edu/uscode/text/26/6103.

Guido van Rossum, Barry Warsaw, and Nick Coghlan. Pep 8 – style guide for python code. https://www.python.org/dev/peps/pep-0008/.

Appendix A

Technical Details

A.1 Config File Example

```
[DEFAULT]
# This name is prepended to log files and log messages
name = bdstemplaterun
# %(root)s variable is convenient to use to indicate all outputs
root = ./
loglevel = INFO

# This text variable is used for variable manipulations in Pandas
# to create all BDS measures from employment and establishment counts
addvars = denom, net_job_creation, job_creation, job_destruction, job_destruction_deaths,
    job_destruction_continuers, job_creation_births,job_creation_continuers,estabs_entry,estabs_exit;
          net_job_creation_rate, estabs_entry_rate, estabs_exit_rate, job_creation_rate_births,
          job_creation_rate,job_destruction_rate_deaths,job_destruction_rate,reallocation_rate

# This is a common part in SQL queries that create the output tables
sqlpart = grower,continuer, SUM(empcy) AS empcy, SUM(emppy) AS emppy,
    SUM(estabs) as estabs FROM byfage4byfsizebysic1bymsa GROUP BY grower, continuer

# This section sets up OS environment variables
[ENVIRONMENT]
das_framework_version = 0.0.1

[experiment]
# Loops are only used in experiment regime, and will be ignored in single run
# (like when producing tables for release)
# However, values set here will become a part of output file names
# loop1 : FOR engine.run = 1 TO 5
# loop3 : FOR engine.epsilon IN 8
# loop4 : FOR engine.alpha IN 0.05
# scaffold = Scaffolding.Scaffold

[spark]
# Spark temporary directory. Should be on fast access drive and have enough space.
# Default is /tmp, better use the default
#local_dir = /tmp/
```

```
# This is on a slow access drive and may result in extremely slow performance
#local_dir = /cesprod/bdstt/data/experiments/tmp/

[setup]
setup = bds_das_spark_setup.setup

[reader]
reader = bds_das_spark_reader.reader

# The name of the input file
input_fname = /cesprod/bdstt/data/bughunt/estabs20142015.csv

# If running the file not for the first time, change it to the trimmed .parquet file,
# created in the first run
# input_fname = /cesprod/bdstt/data/estabsparquet/estabs20142015.parquet

# Directory to store trimmed estab files in .parquet format. The default is current directory, ./
trimmed_estab_dir = /cesprod/bdstt/data/estabsparquet/

# If your input file is estab file, set to yes. If it is bdsgc, set to no (it is default)
aggregate_estab = yes

# Directory to store bdsgc files. Will not save bdsgc files if this option is not set.
# If your input file is estab, but bdsgc already exists in the directory indicated in this option,
# then this bdsgc will be used for input. If your estab file changes, and you want to run with
# the changed data, make sure to delete the bdsgc
aggregated_bdsgc_dir = /cesprod/bdstt/data/aggregated/

[engine]
# Release / total / effective / master epsilon
epsilon = 8.0

# alpha and delta for SmoothLaplace algorithm
alpha = 0.05
delta = 0.05
engine = bds_das_spark_engine.engine

[prenoise_tables]
# Format: tablename = query
# The name "tablename" can be used in the subsequent queries,
# because tables are created in the order listed here
# (it will be registered as a view in SparkContext)
# The name of noisy table will be "noisy_tablename"

# Firm Age X Firm Size X Industry X MSA ( X grower X continuer) table
# Includes smooth sensitivities for previous year
# and current year employment counts (ssmaxp and ssmax),
# equal to the largest employer in the cell
byfage4byfsizebysic1bymsa =
    SELECT fage4,fsize,sic1,msa,
        grower,continuer,
        SUM(empcy) AS empcy,
        SUM(emppy) AS emppy,
        SUM(estabs) as estabs,
        MAX(empcy) AS ssmax,
        MAX(emppy) AS ssmaxp
    FROM bdsgc
    GROUP BY grower,continuer,
        fage4,fsize,sic1,msa

[out_tables]
# Format: tablename = query
# The name "tablename" can be used in the subsequent queries,
# because tables are created in the order listed here
# (it will be registered as a view in SparkContext)

# Firm Age X Firm Size X MSA ( X grower X continuer) table. Showing actual full query, for illustration
byfage4byfsizebymsa =
    SELECT fage4,fsize,msa,
        grower,continuer,
        SUM(empcy) AS empcy,
        SUM(emppy) AS emppy,
        SUM(estabs) as estabs
    FROM byfage4byfsizebysic1bymsa
    GROUP BY grower,continuer,
        fage4,fsize,msa

# We have set the %(sqlpart)s variable in the [DEFAULT] section, so can use it here in order to
# avoid duplicating SQL code and be more concise. It should have been used for above query too,
# but was not for illustration purposes

# Firm Age X Firm Size X Industry table
byfage4byfsizebysic1 = SELECT fage4,fsize,sic1, %(sqlpart)s, fage4,fsize,sic1

# Firm Age X Firm Size table
byfage4byfsize = SELECT fage4,fsize, %(sqlpart)s, fage4,fsize

# Firm Age  X Industry table
byfage4bysic1 = SELECT fage4,sic1,%(sqlpart)s, fage4,sic1

# Firm Age X MSA table
byfage4bymsa = SELECT fage4,msa,%(sqlpart)s, fage4,msa

# MSA table
bymsa = SELECT msa, %(sqlpart)s, msa

# Firm Age table
byfage4 = SELECT fage4, %(sqlpart)s, fage4

# Industry table
bysic1 = SELECT sic1, %(sqlpart)s, sic1

# Firm Size table
byfsize = SELECT fsize, %(sqlpart)s, fsize

# National Economy Wide table
total = SELECT %(sqlpart)s
```

```
[out_tables_add_vars_pandas]
# Format:
# tablename = list of variables over which to aggregate Pandas dataframe before aggregation;
#    list of variables over which to aggregate Pandas dataframe after aggregation;

# We have created the lists in [DEFAULT] section %(addvars)s variable, so use it here
byfage4byfsizebymsa = %(addvars)s
byfage4byfsizebysic1 = %(addvars)s
byfage4byfsize = %(addvars)s
byfage4bysic1 = %(addvars)s
byfage4bymsa = %(addvars)s
bymsa = %(addvars)s
byfage4 = %(addvars)s
bysic1 = %(addvars)s
byfsize = %(addvars)s
total = %(addvars)s

[out_tables_aggregate_pandas]
# Format: tablename = list of variables over which to aggregate Pandas dataframe; aggregate operation

# We want to sum over grower and continuer qualifiers
byfage4byfsizebymsa = grower,continuer;sum
byfage4byfsizebysic1 = grower,continuer;sum
byfage4byfsize = grower,continuer;sum
byfage4bysic1 = grower,continuer;sum
bymsa = grower,continuer;sum
byfage4 = grower,continuer;sum
bysic1 = grower,continuer;sum
byfsize = grower,continuer;sum
total= grower,continuer;sum

[out_tables_drop_vars_pandas]
# Format: tablename = list of variables to drop from Pandas dataframe

# Sensitivities MUST be removed, because they leak privacy. Also, remove any unwanted columns
byfage4byfsizebymsa = emppy,ssmax,ssmaxp
byfage4byfsizebysic1 = emppy,ssmax,ssmaxp
byfage4byfsize = emppy,ssmax,ssmaxp
byfage4bysic1 = emppy,ssmax,ssmaxp
byfage4bymsa = emppy,ssmax,ssmaxp
bymsa = emppy,ssmax,ssmaxp
byfage4 = emppy,ssmax,ssmaxp
bysic1 = emppy,ssmax,ssmaxp
byfsize = emppy,ssmax,ssmaxp
total = emppy,ssmax,ssmaxp

[variables2addnoise]
# Format:
# tablename = list of variables to add noise to
# (separate by comma within group of composable vars, and by semicolon between groups)

# Here, epsilons for empcy and emppcy will be added to obtain the table epsilon.
# If they were separated by comma, the epsilon would not double
byfage4byfsizebysic1bymsa = empcy; emppy
```

```
# The following sections direct application of noise algortithms to the variables of the set up in
# [variables2addnoise] section. The section should be titles as [tablename_varname] and then list the
# chosen privacy algorithm and its parameters. The supported algorithms are NoNoise (a pass-through, no
# noise applied); LaplaceDP (the "usual" differential privacy adding Laplace noise with
# sensitivity/epsilon as the parameter); SmoothLaplace, SmoothGamma and LogLaplace# from Haney et al.
# 2017; Pub1075 (suppression of certain cells based on rules from IRS Publication 1075}, no
# complementary suppression). The docstrings of the algorithm functions have more detail.
# "Sensitivity", "alpha", "delta" and "p" parameters can be set up in the [engine] section for all the
# variables simultaneously. Setting them in a section below will override that for that particular
# variable

[byfage4byfsizebysic1bymsa_empcy]
algorithm = SmoothLaplace

# Name of the column that contains smooth sensitivities for cells
sensitivity_field = ssmax

# What fraction of total epsilon (indicated in [engine]) to use on this variable
epsilon_fraction = .5

[byfage4byfsizebysic1bymsa_emppy]
algorithm = SmoothLaplace
sensitivity_field = ssmaxp
# The values for alpha and delta can set in [engine] can be overriden here if needed
# delta = 0.05
epsilon_fraction = .5

[writer]
writer = bds_das_writer.writer

# Where to write output private tables. Using the %(root)s variable set in [DEFAULT] section
output_path = %(root)s

# Whether you want to output true tables as well. Default = False
# write_true_data = True

[validator]
validator = bds_das_validator.validator
cell_id_vars = state,msa,sic1,age4,size,isize,fage4,fsize,ifsize,continuer,grower

# These are L1, L2 etc (Ln) errors:
euclid_norm_dim = 1,2,5,10
# Which error measures to calculate.
error_metrics = chisq
# Where to write calculated errors. Using the %(root)s variable set in [DEFAULT] section
results_path = %(root)s

[takedown]
takedown = bds_das_spark_takedown.takedown
```

A.2 BDS Data Schema

A.2.1 Precursor Definitions

The following concepts are used to construct the variables in the next section.

Establishment Level Employment $E_{i,t}$ is the employment at establishment, i, in time, t.

Establishment Level Employment Change $\Delta_{i,t} = E_{i,t} - E_{i,t-1}$

Establishment Existence $\varepsilon_{i,t} = 0$ if $E_{i,t} = 0$ and $\varepsilon_{i,t} = 1$ otherwise.

Establishment Level Growth $\gamma_{i,t} = 0$ if $\Delta_{i,t} < 0$ and $\gamma_{i,t} = 1$ otherwise. This indicator identifies whether estblishment, i, grew from time, $t-1$ to t.

Establishment Level Continuance $c_{i,t} = \varepsilon_{i,t} \cdot \varepsilon_{i,t-1}$. This indicator identifies whether the establishment existed in both this quarter and last quarter.

Firm Existence $\zeta_{j,t} = 0$ if $E_{i,t} = 0, \forall i \in j$ where j is an index on all firms. $\zeta_{j,t} = 1$ otherwise. A firm, ζ, exists if any of its establishments, i, had positive employment in time, t.

Firm Death $FD_{j,t} = 1$ if $\gamma_{i,t} = 0$ and $c_{i,t} = 0, \forall i \in j$. $FD_{j,t} = 0$ otherwise. A firm death, FD, exists in time, t, if all its establishments, i, exit (shrink and do not continue).

A.2.2 Variables

The following are intended to be the variables on the protected, published tables:

Employment Counts

Employment $E_{s_t,t} = \sum_{i \in s_t} E_{i,t}$ is the employment at the establishments in marginal, s_t, at time, t.

Denominator $X_{s_t,t} = \frac{1}{2}(E_{s_t,t} + E_{s_t,t-1})$ is the average employment for establishments in marginal, s_t over time periods, t and $t-1$. This variable is also used to construct rates.

Job Destruction $JD_{s_t,t} = -\sum_{i \in s_t} \Delta_{i,t}$ where $s = \{\gamma = 0, c = (0,1), \cdots\}$ is the total change in employment for all establishments in marginal, s_t, at time, t, where marginal, s_t, only includes establishments that did not grow.

Job Creation $JC_{s_t,t} = \sum_{i \in s_t} \Delta_{i,t}$ where $s = \{\gamma = 1, c = (0,1), \cdots\}$ is the total change in employment for all establishments in marginal, s_t, at time, t, where marginal, s_t, only includes establishments that grew or remained the same size.

Net Job Creation $NET_{s_t,t} = \sum_{i \in s_t} \Delta_{i,t}$ where $s = \{\gamma = (0,1), c = (0,1), \cdots\}$ is the total change in employment for all establishments in marginal, s_t, at time, t, where marginal, s_t, does not discriminate among establishments by growth or continuance indicators.

Job Destruction Deaths $JDD_{s_t,t} = -\sum_{i \in s_t} \Delta_{i,t}$ where $s = \{\gamma = 0, c = 0, \cdots\}$ is the total change in employment for all establishments in marginal, s_t, at time, t, where marginal, s_t, only includes establishments that exited (did not grow and did not continue).

Job Destruction Continuers $JDC_{s_t,t} = -\sum_{i \in s_t} \Delta_{i,t}$ where $s = \{\gamma = 0, c = 1, \cdots\}$ is the total change in employment for all establishments in marginal, s_t, at time, t, where marginal, s_t, only includes establishments that shrank but continued.

Job Creation Births $JCB_{s_t,t} = \sum_{i \in s_t} \Delta_{i,t}$ where $s = \{\gamma = 1, c = 0, \cdots\}$ is the total change in employment for all establishments in marginal, s_t, at time, t, where marginal, s_t, only includes establishments that entered (grew and did not continue).

Job Creation Continuers $JCC_{s_t,t} = \sum_{i \in s_t} \Delta_{i,t}$ where $s = \{\gamma = 1, c = 1, \cdots\}$ is the total change in employment for all establishments in marginal, s_t, at time, t, where marginal, s_t, only includes establishments that grew and continued.

Creation/Destruction Rates The rates of five of the count variables are created by dividing the counts by the denominator for the appropriate margnal, $X_{s_t,t}$. Briefly, these are:

Job Destruction Rate $JDR_{s_t,t} = \frac{JD_{s_t,t}}{X_{s_t,t}}$

Job Creation Rate $JCR_{s_t,t} = \frac{JC_{s_t,t}}{X_{s_t,t}}$

Net Job Creation Rate $NETR_{s_t,t} = \frac{JD_{s_t,t}}{X_{s_t,t}}$

Job Destruction Rate Deaths $JDRD_{s_t,t} = \frac{JDD_{s_t,t}}{X_{s_t,t}}$

Job Destruction Rate Births $JCRB_{s_t,t} = \frac{JCB_{s_t,t}}{X_{s_t,t}}$

Reallocation Rate $RAR_{s_t,t} = JCR_{s_t,t} + JDR_{s_t,t} - |NET_{s_t,t}|$

Firm Death Employment $FDE_{s_t,t} = -\sum_{j \in s_t} \sum_{i \in j} \Delta_{i,t}$ where $s = \{FD_{j,t} = 1, \cdots\}$ is the count of job destruction in firm deaths in marginal, s_t, at time, t.

Establishment/Firm Counts

These counts may not need protections.

Establishments $\varepsilon_{s_t,t} = \sum_{i \in s_t} \varepsilon_{i,t}$ is the count of establishments in marginal, s_t, at time, t.

Establishment Entries $EN_{s_t,t} = \sum_{i \in s_t} \varepsilon_{i,t}$ where $s = \{\gamma = 1, c = 0, \cdots\}$ is the count of establishments in marginal, s_t, at time, t where marginal, s_t, only includes establishments that entered.

Establishment Exits $EX_{s_t,t} = \sum_{i \in s_t} \varepsilon_{i,t}$ where $s = \{\gamma = 0, c = 0, \cdots\}$ is the count of establishments in marginal, s_t, at time, t where marginal, s_t, only includes establishments that exited.

Establishment Entry Rate $ENR_{s_t,t} = \frac{EN_{s_t,t}}{\varepsilon_{s_t,t}}$

Establishment Exit Rate $EXR_{s_t,t} = \frac{EX_{s_t,t}}{\varepsilon_{s_t,t}}$

Firms $\zeta_{s_t,t} = \sum_{j \ni i \in s_t} \zeta_{j,t}$ is the count of firms in marginal, s_t, at time, t. Words: *Firms* is the count of firms, $\zeta_{j,t}$, that have at least one establishment, i, in the marginal, s_t. TODO: WARNING: *Firms* is not additive.

Firm Death Firms $FDF_{s_t,t} = \sum_{j \ni i \in s_t} FD_{j,t}$ is the count of firm deaths in marginal, s_t, at time, t. Words: *Firm Death Firms* is the count of firm deaths, $FD_{j,t}$, for which the firm had at least one establishment, i, in the marginal, s_t, in time, $t-1$. TODO: WARNING: *Firm Death Firms* is not additive.

Firm Death Estabs $FD\varepsilon_{s_t,t} = \sum_{i \in s_t} \varepsilon_{i,t}$ where $s = \{FD_{j,t} = 1, \cdots\}$ is the count of establishments that participate in firm deaths in marginal, s_t, at time, t.

A.2.3 Proposed Tables

The following marginal tables constitute the preferred release of BDS. All variables in the previous section appear in each table. NOTE: Currently, tables with "industry" as part of the margin use SIC codes. However, it is planned that this will switch to NAICS codes this year. Changes in the output specification – if they occur – have not been discussed.

A.3 Privacy Alrorithms

A.3.1 IRS Pub. 1075 Rules, Employment and Establishment Counts

Overview

This method implements the exact suppressions outlined for business data in IRS Pub. 1075. In addition, it uses the $p\%$ rule (see the Implementation Detail below) to determine whether a cell would "identify a particular taxpayer, either directly or indirectly." Currently our simplistic implementation does not implement complementary suppression, which has sigificant consequences for disclosure avoidance. However, implementing complementary suppression would require additional SME resources.

An alternative implementation of this could be run in cooredination with other methods. In such a case each cell in a table would be flagged as to whether it met the IRS Pub. 1075 rules. Then alternative forms of protection could be applied to those cells that do not meet the requirements. This could allow a compromise between maintaining the rules from IRS Pub. 1075 and considering other methods for cells that would have been otherwise suppressed.

Disclosure Avoidance

This method does not provide provable privacy and the resulting dataset would likely be vulnerable to a database reconstruction attack and inferential disclosure. Additionally, a proper implementation of the rules outlined in IRS Pub. 1075 would need to implement complementary suppression, which would require further development and additional resources for ensuring the complementary suppression is performed appropriately.

Table A.1: **BDS marginal tables desired to be released.** The UID is based on the cell variables used (definining the marginal), summing the following values: Firm Age – 1, Firm Size – 2, Establishment Age – 4, Establishment Size – 8, Initial Firm Size – 16, Initial Establishment Size – 32, State – 64, Metropolitan Statistical Areas – 128, Metro or Nonmetro – 256, Industry – 512. The number of cell variables is added after the dot. Note that tables T0.1, T512.1, T64.1, T128.1 appear in both Firms and Establishments lists.

UID	Cell Variables	Measure Variables
	Firm Tables (distinguished by the use of Firm Age/Size)	
T0.1	Total (Economy Wide)	All
T512.1	Industry	All
T16.1	Initial Firm Size	All
T2.1	Firm Size	All
T1.1	Firm Age	All
T64.1	State	All
T256.1	Metro or Nonmetro	All
T128.1	Metropolitan Statistical Areas	All
T528.2	Initial Firm Size × Industry	All
T514.2	Firm Size × Industry	All
T513.2	Firm Age × Industry	All
T17.2	Firm Age × Initial Firm Size	All
T3.2	Firm Age × Firm Size	All
T80.2	Initial Firm Size × State	All
T66.2	Firm Size × State	All
T65.2	Firm Age × State	All
T272.2	Initial Firm Size × Metro or Nonmetro	All
T258.2	Firm Size × Metro or Nonmetro	All
T257.2	Firm Age × Metro or Nonmetro	All
T130.2	Firm Size × Metropolitan Statistical Areas	All
T129.2	Firm Age × Metropolitan Statistical Areas	All
T323.4	Firm Age × Firm Size × State × Metro or Nonmetro	All
T337.4	Firm Age × Initial Firm Size × State × Metro or Nonmetro	All
T529.3	Firm Age × Initial Firm Size × Industry	All
T515.3	Firm Age × Firm Size × Industry	All
T81.3	Firm Age × Initial Firm Size × State	All
T67.3	Firm Age × Firm Size × State	All
T273.3	Firm Age × Initial Firm Size × Metro or Nonmetro	All
T259.3	Firm Age × Firm Size × Metro or Nonmetro	All
T131.3	Firm Age × Firm Size × Metropolitan Statistical Areas	All
	Establishment Tables	
T0.1	Total (Economy Wide)	All
T512.1	Industry	All
T32.1	Initial Establishment Size	All
T8.1	Establishment Size	All
T4.1	Establishment Age	All
T64.1	State	All
T128.1	Metropolitan Statistical Areas	All
T544.2	Initial Establishment Size × Industry	All
T520.2	Establishment Size × Industry	All
T516.2	Establishment Age × Industry	All
T36.2	Establishment Age × Initial Establishment Size	All
T12.2	Establishment Age × Establishment Size	All
T96.2	Initial Establishment Size × State	All
T72.2	Establishment Size × State	All
T68.2	Establishment Age × State	All
T548.3	Establishment Age × Initial Establishment Size × Industry	All
T524.3	Establishment Age × Establishment Size × Industry	All
T100.3	Establishment Age × Initial Establishment Size × State	All
T76.3	Establishment Age × Establishment Size × State	All

Implementation Detail

For each requested table, suppress a cell if:

1. If the geography is national *and* the cell has less than 5 establishments.[1]
2. If the geography is state *and* the cell has less than 10 establishments.
3. If the geography is Metro/Nonmetro or CBSA *and* the cell has less than 20 establishments.
4. If the employment distribution among the establishments in the cell fails the $p\%$ rule, where $p = 90\%$[2]: Suppress if $R < p * L$ where L is the employment of the largest establishment and R is the employment of the cell when the employment for the top *two* establishments has been removed. In this case, we use "suppress" to mean replace the cell count with a "(D)" to indicate that the value is withheld for disclosure avoidance purposes. These values are interpreted as a zero (usually) when the error calculations are performed. (In the current implementation, the counts are replaced with actual zeros).

A.3.2 ε-Differential Privacy, Global Sensitivity with Large Employer Suppression, Employment Counts

Overview

The general challenge in protecting fact-of-filing is the node sensitivity for businesses. In principle the sensistivity is as large as the largest *imaginable* business. This method avoids this problem—to some extent—by suppressing any businesses with employment greater than or equal to θ, which is a design parameter.[3] Then the sensitivity is set at θ and we can add Laplace noise scaled to θ rather than the much larger "as big as we can imagine."

The tradeoff here, though, is stark: We must exclude large businesses from the published data. If we choose to set this bar high and only exclude *very large* businesses, then we must add lots of noise (on average). Whereas if we are willing to exclude more businesses by lowering the threshold, then we can add less noise (on average) to all the cells.

Disclosure Avoidance

This method is node-differentially private, which means it protects fact-of-filing while still allowing the publication of employment counts.

Quality and Accuracy

This method has the potential to be biased for cells that include businesses with employment greater than or equal to θ. Additionally, if θ is set too high, then the overall quality of the data will likely be poor. However, for tables with Firm/Establishment Size included in the margin definition, any size groups that only include businesses less than θ will not have any additional

[1] We refer to establishments here instead of EINs because we need to protect against the worst case: a large EIN that is a single-establishment firm.

[2] The level for the $p\%$ rule was borrowed from Sullivan [1992]

[3] This is the same in principle as the "Truncated Laplace" method that was tested in Haney et al. [2017].

quality issues from the large firm suppression. The DA experiments using this method have shown extremely poor quality (as measure by various L1 errors).

Implementation Detail

Modify the database $D \rightarrow D'$ by removing establishments with employment counts $> \theta$. Then construct marginal tables from D'. For each requested employment table:

1. n is the true employment for each cell of a marginal table constructed from D'. ε is a privacy parameter. θ is the establishment suppression parameter. $\tilde{n}$ is the noisy employment count.

2. Sample $\eta \sim \mathit{Laplace}\,(\theta/\varepsilon)$

3. $\tilde{n} \leftarrow n + \eta$

A.3.3 ε-Differential Privacy, Establishment Counts

Overview

In the case of tables of establishment count queries, the sensitivity is 1 since the outcome of a query can change by at most 1 when an establishment is added or removed from the database. As a result fact-of-filing is easier to protect than for employment count queries.

Disclosure Avoidance

This method is differentially private for queries on establishment counts. Since employment and establishment counts are not independent, the recommendation is to treat the tables as not composable and to require *adding* the values of ε from establishment tables to the values of ε for employment tables.

Quality and Accuracy

This method should produce relatively good quality data.

Implementation Detail

Given the database D, construct marginal tables. For each requested establishment table:

1. n is the true establishment count for each cell of a marginal table constructed from D. ε is a privacy parameter. $\tilde{n}$ is the noisy employment count.

2. Sample $\eta \sim \mathit{Laplace}\,(1/\varepsilon)$

3. $\tilde{n} \leftarrow n + \eta$

A.3.4 ε-Differential Privacy, with Large Employer Suppression, Establishment Counts

Overview

This method is similar to the previous one, but it also accounts for Large Employer Suppression described in Section A.3.2. In this case the same employers should be suppressed prior to generating the establishment tables.

Disclosure Avoidance

This method is differentially private for queries on establishment counts. Since employment and establishment counts are not independent, the recommendation is to treat the tables as not composable and to require *adding* the values of ε from establishment tables to the values of ε for employment tables.

Quality and Accuracy

This method has the potential to be biased for cells that include businesses with employment greater than or equal to θ. However, the bias generally should be smaller than in the employment tables. Additionally, the noise being added to cells does not scale with θ so its only effect on quality is from the suppression of large employers. For tables with Firm/Establishment Size included in the margin definition, any size groups that only include businesses less than θ will not have any additional quality issues from the large employer suppression.

Implementation Detail

Modify the database $D \rightarrow D'$ by removing establishments with employment counts $> \theta$. Then construct marginal tables from D'. For each requested employment table:

1. n is the true employment for each cell of a marginal table constructed from D'. ε is a privacy parameter. θ is the establishment suppression parameter. $\tilde{n}$ is the noisy employment count.
2. Sample $\eta \sim \mathit{Laplace}\,(1/\varepsilon)$
3. $\tilde{n} \leftarrow n + \eta$

A.3.5 (ε, α)-EE-ER Privacy, Various Algorithms, Employment Counts

Overview

This method uses the privacy definition from Haney et al. [2017] to protect employment counts within a pre-determined multiplicative factor, α. This approach only protects the employment tables.

Disclosure Avoidance

This method is provably private for protecting exact employment counts. However, this general approach does not explicitly protect fact-of-filing because the existence of a large establishment in the data (and thus fact-of-filing) can be inferred from the noisy employment when we only protect the employment counts within a multiplicative factor of α.

Quality and Accuracy

The algorithms that work under (ε, α)-EE-ER Privacy provide generally good quality measures for very aggregated tables and can provide acceptable quality on more disaggregated tables. The methods do differ in their quality outcomes. The Log-Laplace algorithm is biased and Haney et al. [2017] found that the Smooth Laplace mechanism had marginally better quality for L1 error on a different dataset and publication tables.

Implementation Detail

Three algorithms are outlined below—all of which meet the (ε, α)-EE-ER Privacy guarantee.

Log Laplace Algorithm For each requested employment table:

1. n is the true employment for each cell of a marginal table constructed from the database, D. ε and α are the privacy parameters, $\tilde{n}$ is the noisy employment count.
2. $\gamma \leftarrow 1/\alpha$
3. $\ell \leftarrow \ln(n + \gamma)$
4. Sample $\eta \sim Laplace(2 \ln(1 + \alpha)/\varepsilon)$
5. $\tilde{n} \leftarrow e^{\ell+\eta} - \gamma$

Smooth "Gamma" Algorithm For each requested employment table:

1. n is the true employment for each cell of a marginal table constructed from the database, D. ε and α are the privacy parameters: $\alpha + 1 \leq e^{\varepsilon/5}$ (in practice we only show results for combinations of parameters when this inequality is true). $\tilde{n}$ is the noisy employment count.
2. $x_v \leftarrow$ size of largest establishment in cell
3. $S^*_{v,\varepsilon_2/5}(x) \leftarrow \max(x_v \cdot \alpha, 1)$
4. Sample $\eta \sim \frac{1}{(1+|z|^4)}$
5. $\varepsilon_2 \leftarrow 5 \cdot \ln(\alpha + 1)$
6. $\varepsilon_1 \leftarrow \varepsilon - \varepsilon_2$ [Note that $\varepsilon_1 > 0$ by the condition in #1.]
7. $\tilde{n} \leftarrow n + \frac{S^*_{v,\varepsilon_2/5}(x)}{\varepsilon_1/5}\eta$

Smooth Laplace Algorithm For each requested employment table:

1. n is the true employment for each cell of a marginal table constructed from the database, D. ε and α are the privacy parameters: $\alpha + 1 \leq e^{\frac{\varepsilon}{2\ln(1/\delta)}}$ (in practice we only show results for combinations of parameters when this inequality is true). $\tilde{n}$ is the noisy employment count.

2. $x_v \leftarrow$ size of largest establishment in cell

3. $S^*_{v,\frac{\varepsilon}{2\ln(1/\delta)}}(x) \leftarrow \max\left(x_v \cdot \alpha, 1\right)$

4. Sample $\eta \sim \mathit{Laplace}(1)$

5. $\tilde{n} \leftarrow n + \frac{S^*_{v,\frac{\varepsilon}{2\ln(1/\delta)}}(x)}{\varepsilon/2}\eta$

Appendix B

das-framework

B.1 Overview

The software product **das-framework**, written in Python 3, is a disclosure avoidance system framework which is supposed to be used for developing and then running and evaluating disclosure avoidance systems. The **driver** module of the framework implements generic flow, common to all disclosure avoidance procedures in a modular structure.

- **reader** module reads the true data (or data from which true data can be produced)
- **engine** module performs disclosure avoidance
- **writer** module outputs the protected data
- **validator** module calculates accuracy of the protected data by comparing it to true data

The **setup**, **takedown** and **Scaffolding** modules perform "servicing" functions needed to run the other modules and to keep data between the runs.

B.2 Running

The DA systems using **das-framework** software can run in two regimes: single run, and experiment. The latter performs multiple runs of the system, allowing for exploration of parameter space and/or multiple runs with the same parameters for collecting statistics when disclosure avoidance involves random noise infusion. The experiment regime is turned on via `--experiment` command line option. Otherwise, the behavior is managed via config file (parsed by Python **ConfigParser** module). The path to config file is the only mandatory, positional command line option when running **das-framework** based disclosure avoidance systems.

B.3 Installation

das-framework can be installed either as Python module to the Python environment used when developing and/or running a DAS, or by copying files *driver.py* and *experiment.py* into the DAS project. The former case will make it easier to update and test **das-framework**.

When using Spark in the DAS, one has to supply the *driver.py* and *experiment.py* to the Spark Context with **pyspark** *addPyFile* function or with `--py-files` command line option of *spark-submit*.

Installing **das-framework** as python module would make updating and testing it easier (as it comes with a set of **pytest** tests), but harder to use the *addPyFile* function. Just copying *driver.py* and *experiment.py* is the opposite.

Arguably, the most convenient way is to install **das-framework** as a submodule to the DAS you are developing or using. This can be achieved by creating a subdirectory under your DAS directory that will contain **das-framework** or is a symlink to the directory with **das-framework**. If you are using **pytest** with your DAS, you will have to configure the tests locations in the *pytest.ini* file, so that **das-framework** tests don't get in the way.

B.4 Developing your DAS with das-framework

The modules performing the disclosure avoidance flow directed by **driver** module of **das-framework** are prototyped in **das-framework** as AbstractDASWriter, AbstractDASReader, AbstractDASEngine etc. classes. These modules are all inherited from AbstractDASModule which mainly has a config file parsing functionality implemented. Developing a new DAS, based on **das-framework**, the modules should be inherited from these AbstractDASxxxxxx classes.

The excerpts from the **das-framework** code below provide the flow, from which follow the requirements to module implementations:

```
setup_data = setup_obj.setup_func()
self.reader = getattr(reader_module, reader_class_name)(config=config, setup=setup_data, name=READER)
self.engine = getattr(engine_module, engine_class_name)(config=config, setup=setup_data, name=ENGINE)
self.writer = getattr(writer_module, writer_class_name)(config=config, setup=setup_data, name=WRITER)
self.validator= getattr(validator_module, validator_class_name)(config=config, setup=setup_data, name=VALIDATOR)
self.takedown = getattr(takedown_module, takedown_class_name)(config=config, setup=setup_data, name=TAKEDOWN)
...
original_data = self.reader.read()
...
privitized_data = self.engine.run( original_data )
...
written_data = self.writer.write( privitized_data)
...
valid = self.validator.validate(original_data, written_data)
if not valid:
    logging.info("self.validator.validate() returned false")
    raise RuntimeError("Did not validate.")
...
self.takedown.takedown()
```

Thus, the requirements are as follows:

- The **setup** module must have *setup_func()* function, which returns the setup data needed for initialization of all the other modules (may return the setup module object itself).

- The **reader** module must have a *read()* function, which returns the original data

- The **engine** module must have a *run(original_ data)* function that will take it in and return privitized data

- The **writer** module must have a *write(privitized_ data)* function that writes the data and returns a link to it (such as filename)

- The **validator** module must have a *validate(original_ data, written_ data)* function and return a boolean, which, when *False* will interrupt the program

The AbstractDASxxxx prototypes have all these functions and also, *willRead(), willRun(), willWrite(), willValidate(), willTakedown()* which will raise Runtime Errors when returning *False*. Also there are *didRead(), didRun(), didWrite(), didValidate(), didTakedown()*, whose returns are not processed by **das-framework**. These functions can be optionally implemented if the developer finds them useful.

The only requirement for **Scaffolding** class is that it has *experimentSetup(config)* and *experimentTakedown(config)* functions, which will be run once at the start and the conclusion of the experiment. No prototype scaffold class is implemented in **das-framework.**

Appendix C

dbcompare

C.1 Overview

The **dbcompare** repository holds scripts for the comparison of two dataframes using a predefined set of functions. These functions where chosen based on the request put out by CDAR in Possible Error Measures for BDS. They include L1 Error, L2 Error, Earth Mover's Distance, Jensen Shannon Divergence, and Spearman Ranking Correlation. In the final version we included a method for any Ln error and did not include Earth Mover's Distance as the results were found to be too dependent on user defined sample selection and did not provide useful information.

There are three main components of the dbcompare:

- a method for comparing python pandas dataframes,
- a method for comparing spark dataframes,
- testing of the dbcompare scripts.

The script for panda dataframes also includes a method to allow it to be used on two csv files in which case it will read the csv into panda dataframes before continuing the analysis.

The dbcompare code differs from that of the **validator** module of BDS-DAS as it is attempting to be as general as possible, with no BDS specifics. The BDS-DAS **validator** uses **dbcompare** to calculated error measures implemented here, but the BDS-DAS **validator** also includes additional measures such as relative cell error.

The main use method for the script would be to call the *run* function, which takes in two Pandas DataFrames for comparison and parameters and returns Pandas DataFrame with error measures. There are options to allow this output to be written to a csv or latex document but these are turned off by default. Details on each individual function found in the dbcompare module are listed below.

C.2 Installation as a Part of a DAS

dbcompare can be installed either as Python module to the Python environment used when developing and/or running a DAS, or as a submodule to the DAS.

When using Spark functionality of **dbcompare**, one has to supply the *spark_db_comp.py* to the Spark Context with **pyspark** *addPyFile* function or with

```
--py-files
```

command line option of *spark-submit*

To install **dbcompare** as a submodule to the DAS you are developing or using.This can be achieved by creating a subdirectory under your DAS directory that will contain **dbcompare** or is a symlink to the directory with **dbcompare**. If you are using **pytest** with your DAS, you will have to configure the tests locations in the *pytest.ini* file, so that **dbcompare** tests don't get in the way.

C.3 db_comp

- Database comparison using panda dataframes

C.3.1 Function Descriptions

l_error

- inputs: array1, array2, euclidean norm dimension, variable name
- array1: numpy array for first dataset
- array2: numpy array for second dataset
- euclidean norm dimension: dimension for norm
- variable name: name of variable
- output: ln_data
- ln_data: panda dataframe of calculated norm

jsd

- inputs: array1, array2, variable name
- array1: numpy array for first dataset
- array2: numpy array for second dataset
- variable name: name of variable
- output: jsd_data
- jsd_data: panda dataframe of calculated jensen shannon divergence

chisq

- inputs: array1, array2, variable name
- array1: numpy array for first dataset
- array2: numpy array for second dataset
- variable name: name of variable
- output: chisq_data
- jsd_data: panda dataframe of calculated chisq

spearman_rank

- inputs: array1, array2, variable name
- array1: numpy array for first dataset
- array2: numpy array for second dataset
- variable name: name of variable
- output: spear_data
- spear_data: panda dataframe of calculated spearman rank correlation

latex_writer

- inputs: panda dataframe
- ouput: latex text of panda dataframe

csv_writer

- inputs: panda dataframe
- ouput: csv for panda dataframe written in current directory

run

- inputs: data1, data2, euclidean norm dimensions, error metrics, variable names
- data1: data as spark dataframe or panda dataframe
- data2: data as a spark dataframe or panda dataframe
- euclidean norm dimensions: list of dimensions for norms
- variable names: list of variable names
- output: outdata
- outdata: panda data frame with error metrics for requested variables

Independent Call

- arguements
- raw: dataframe 1 as csv
- syn: dataframe 2 as csv
- config: configiruation ini that will request the eucilidean norm dimensions, eroor metrics, and set variable names

C.4 spark_db_comp

- Database comparison using spark dataframes

C.4.1 Function Descriptions

sparkjoin

- inputs: data1, data2
- data1: spark dataframe
- data2: spark dataframe
- ouput: joined spark dataframe on id variabble

chisq

- purpose: for storing and running the sql query that will obtain the results for chisq
- inputs: variable name, modified variable name

jsd

- purpose: for obtaining jensen shannon divergence using scipy.stats.entropy
- inputs: _p, _q
- _p: gaussian kde for data1
- _q: gaussan kde for data2

analyze

- inputs: joined_data, euclidean norm dimension, variable names

run

- inputs: data1, data2, euclidean norm dimension, error metrics, variable names