WP. 29
ENGLISH ONLY

UNITED NATIONS
ECONOMIC COMMISSION FOR EUROPE

CONFERENCE OF EUROPEAN STATISTICIANS

Work Session on Statistical Data Editing
(Oslo, Norway, 24-26 September 2012)

Topic (v): Software & tools for data editing and imputation

TEA FOR SURVEY PROCESSING

Submitted by United States Census Bureau[1]

I. OVERVIEW

1. TEA is a generalized system designed to unify and streamline demographic survey processing, from raw data to editing to imputation to dissemination of output.[2] It is available as an unofficial R package, and has been used by the U.S. Census Bureau for the processing of group quarters data for the American Community Survey and 2010 Census.

2. Users provide a specification file for each survey, including edit rules, a list of models for use in imputation, and a specification for disclosure avoidance tests. The syntax for the specification file is intended to be simple enough that it can be written by analysts with no programming experience.

3. TEA first finds fields that fail the edit rules and marks them as blank. The disclosure avoidance step does summary tabulations as specified by the user, and marks as blank those fields in records that may allow the record to be identified via the tabulations.

4. The imputation component allows users to specify any model for imputation of blank fields, due to failed edits, disclosure risk, or nonresponse. Common models including Hot Deck, OLS, Probit, Lognormal, Beta/Binomial are provided, and users with R or C experience can write new models. All imputations are set in a multiple-imputation framework, so each imputed value can be assigned a variance. Imputed values are checked against the edit rules, so all imputed values are guaranteed to pass the edit requirements.

5. Once the imputation models are fit against survey data, TEA also provides a mechanism to impute from entirely blank data—that is, use the fit models to generate fully synthetic data.

6. In support of these core edit, disclosure-checking, and imputation functions, the package includes tools for reading in data, simple database management, basic reporting, and other typical needs of the survey processor.

---

[1]Prepared by Ben Klemens
U.S. Census Bureau. Thanks to María Garcia and Rolando Rodríguez. This report is released to inform interested parties of research and to encourage discussion. The views expressed are those of the author and not necessarily those of the U.S. Census Bureau.
[2]TEA is named for a popular beverage, imbibed in great quantities while writing the system.

A. Separate components

7. TEA is implemented as a set of distinct components for editing (i.e., identifying those fields that do not meet a set of constraints), inference control, and imputation of missing data. This is very different from other systems where the editing process and imputation method are closely tied. Maintaining segregation between the components provides several benefits.

8. The first benefit is that probabilistic models for imputation are easier to describe and implement. Imputation from a probabilistic model involves fitting a model and then making random draws from the model. One can do such fitting and drawing from a simple distribution such as a Multivariate Normal or a Lognormal distribution, from an ordinary least squares (OLS) regression, or from an empirical distribution consisting of the non-missing values in some subset of the data set.

9. As a corollary to the simplicity of using off-the-shelf models for imputation, we have found that separating consistency checks from imputation models can simplify the specification of the process immensely, because consistency checks are written down in isolation, without an accompanying flowchart of imputation routines, and imputations are written down by simply stating which off-the-shelf model to use.

10. Several examples of the specification will be shown below, so the reader may evaluate whether we have succeeded in developing something user-friendly and readable.

11. The probabilistic model is in contrast to imputation via rules like substituting a value from an ordered list or applying the smallest possible change (by some metric, such as Fellegi-Holt's). TEA provides a limited mechanism for such substitutions, and instead focuses on imputation via specifying, estimating, and drawing from statistical models.

12. TEA strives to simplify the addition of new imputation models. If an analyst decides to replace simple draws from a Normal distribution with draws based on an OLS regression on a set of other variables, the analyst need only change a line or two in the specification. Because edits and imputes are segregated, the edit segment of the specification needs no changes to accommodate this change in model.

13. TEA focuses on probabilistic models aimed at describing the data source itself, and are more appropriate for situations where data are entirely missing or withheld. It is appropriate to use the fact that the logarithm of income tends to be Normally distributed to impute the income of a person who chooses to not report her income. TEA was designed for environments where non-reporting is the primary data problem, and so its focus has been on probabilistic models rather than determining the cause of error and imputing accordingly.

14. Compared to having a definite answer reported by the respondent, imputing a value adds uncertainty, and we would ideally be able to report the uncertainty added by the imputation model. Multiple imputation provides a useful framework for estimating the added variance [Rubin, 2004]. Because of its focus on the sort of probabilistic models around which multiple imputation was designed, TEA facilitates adding multiple imputation to the survey production process.

15. After a brief review of previous work in Section B and an overview of TEA in Section II, this paper will largely follow the process followed by TEA: Section IV discusses the editing step, Section V covers inference control, and Section VI covers the imputation process.

B. Previous literature

16. Pierzchala [1990] gives an overview of the problem of editing (including a useful glossary). Written in 1990, during the midst of the transition from mainframe computers to desktop PCs, it described systems that were a mix of both paradigms. In the present day, the PC has won, and the typical mainframe is a cluster of PCs-on-a-blade. The first consequence is that we can focus on desktop-friendly software. Care must still be taken to process data efficiently, but loading a data set with 300 million observations onto a desktop PC is no longer an impressive feat.

17. The models covered in that paper are uniformly in the class of what I called mechanical edits above, including several implementations of the method of Fellegi and Holt [1976] and deterministic nearest-neighbor methods. Chen et al. [2002] discusses three newer systems, each of which focuses on determining the cause of the error and imputing accordingly: the editing system for the American Community Survey (ACS, from the US Census Bureau) consists of a long sequence of if-then rules that specify what substitutions are to be made given every type of failure; the DISCRETE system [Winkler, 1995] uses Fellegi-Holt's method to derive imputation rules from the edits; what is now CANCEIS [Bankier et al., 2002] uses a relatively sophisticated nearest-neighbor rule for imputation.

18. Probabilistic models are typically more computationally-intensive. Again, some care must be taken, but it is no longer computationally impossible to estimate and draw from tens of thousands of statistical models.

II. AN OVERVIEW OF TEA

19. TEA provides a data pipeline that provides a series of modifications to the input data.

(1) Read in the data from a plain text file to the database
(2) Check all edits against the database. Mark those elements that fail.
(3) Run the disclosure avoidance tests to flag fields in records that may cause disclosure risk.
(4) Apply each imputation model in sequence to fill in data that are missing (either in the original data set, due to a failed edit, or due to a disclosure issue).
  (a) Verify that imputed record is consistent.
  (b) If a record fails an edit, re-impute.

A. Environment and underlying systems

20. TEA can be run on any POSIX-compliant system, including Windows with Cygwin, Linux, and Mac. It is written using three languages: C, R, and SQL. Each provides facilities that complement the others.

21. **SQL** is designed around making fast queries from databases, such as finding all observations within a given age range and income band. Any time we need a subset of the data, we will use SQL to describe and pull it. SQL is a relatively simple language, so users unfamiliar with it can probably learn the necessary SQL in a few minutes—in fact, a reader who claims to know no SQL will probably already be able to read and modify the SQL-language conditions in the `checks` and `recodes` sections below.

22. The TEA system stores data using an SQL database. Currently, TEA is written to support SQLite as its database interface; however it would be possible to implement other interfaces such as Oracle or MySQL. Output at each step is also to the database, to be read as input into the next step. Therefore, the state of the data set is recorded at each step in the process, so suspect changes may be audited. Outside the database, to control

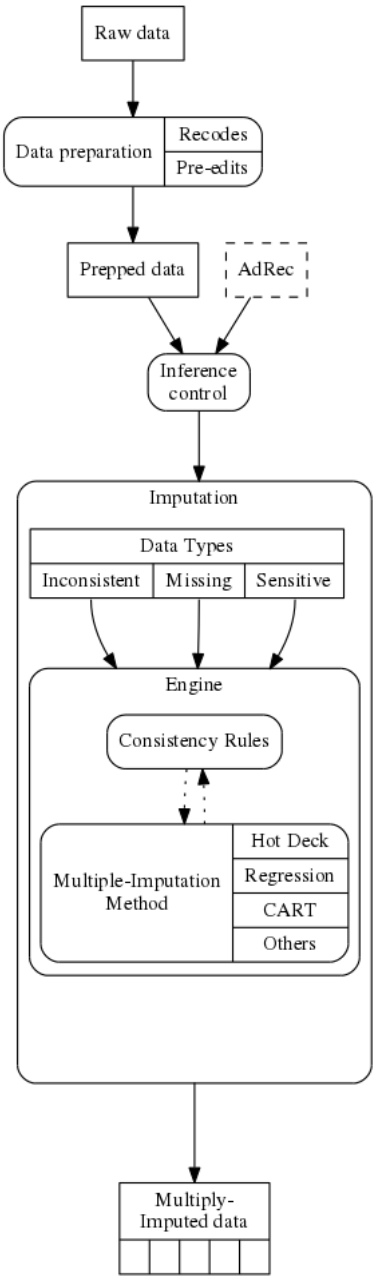

FIGURE 1. The flow of processing. First, read in the data, add artificial variables (recodes) and do pre-edits on the data. Administrative records in the database can be joined to the data using a standard SQL join. Then, flag data that need to be processed, because they are is missing, fail a consistency check, or put personally identifiable information at risk. Having found the data points that need to be fixed, apply an imputation process—there are many that one can choose from—to fill in multiple candidates for a new value. Once those imputations are stored, report a final value and its total variance calculated via the imputation set. Diagram by Rolando Rodríguez.

what happens and do the modeling, the TEA package consists of roughly 3,500 lines of R code and 5,000 lines of C.

23. **R** is a relatively user-friendly system that makes it easy to interact with data sets and write quick scripts to glue together segments of the survey-processing pipeline. Its graphical toolbox is unparalleled. R is therefore the interactive front-end for TEA. Users will want to get familiar with the basics of R. As with SQL, users well-versed in R can use their additional knowledge to do additional analysis beyond the tools provided by the system. But due to technical limitations, R is inappropriate for large-scale data analysis, so where possible we store data in SQL tables and do modeling in C.[3]

24. **C** is the fastest human-usable system available for manipulating matrices, making draws from distributions, and other basic model manipulations. Most of the numerical work will be in C. The user is not expected to know any C at all, because R procedures are provided that do the work of running the underlying C-based procedures.

25. Binding them all together, we can have a system that is friendly to interact with, effectively handles Census-sized data sets, and quickly does the processing for even relatively sophisticated methods.

B. Interactive R

26. TEA is intended for use by analysts with limited programming experience, so the specification of survey details, including the variables and their ranges, the list of edits, the models for imputation, and so on, should be written in a format that has minimal programming formalities (like an excess of semicolons or deeply-nested parentheses). Even in the most compact notation, the details that need to be specified for a typical survey processing pipeline often take up several pages. In many environments, different people write different portions of the specification. These practicalities, and the general programmer's principle that data and procedure should be separate, advised that the details of the process be segregated into separate specification files.

27. Given the specification file, the procedure to be executed from the R command prompt is rather simple. This script loads the TEA library into R's memory, reads the spec file, flags inconsistencies, flags records at risk of disclosure, and multiply imputes substitute or fill-in values:

```
library("tea")
read_spec("spec")
doChecks()
doFingerprint()
doMImpute()
```

The script is so simple because the specification file will give all of the details of the procedure, so at run-time, the analyst need only indicate what procedures to run.

28. These commands could be entered on R's command line as easily as in a script file, so an analyst who needs to verify the results of the consistency-checking step could copy and paste the first three lines of the script onto the R command prompt, where they will run and then return the analyst to the command prompt,

[3]The first key limitation is that R largely lacks a call-by-reference mechanism, meaning that a function that takes in a 1,000 $\times$ 1,000 matrix and changes one element may have to output a new copy of all million elements. The second is that R relies heavily on a data structure known as the S expression, which is largely modeled on LISP-like list elements. Due to issues of memory layout, processing on S expressions will be slower than on raw matrices. These limitations are not always binding, and can often be avoided via careful writing.

where he or she could print subsections of the output tables, check values, modify the specification files and re-run, continue to the imputation step, et cetera.

29. The remainder fo the paper will focus on the specification file. I will assume that it will eventually be read in and run by the R script above.

III. DATA INPUT AND PREP

30. The first step TEA executes is reading into the SQL database data from a text file provided by the user. In the simple R script above, the read-in occurs automatically during the `read_spec` step. There is nothing remarkable in how TEA reads the data set into the database, so this step makes a good introduction to the specification that a TEA user would write.

31. A specification file might begin with these lines:

```
database: survey.db
id: sid

input {
    input file: text_in.csv
    output table: dataset
}
```

The first line is a simple key/value pair specifying that the database in which all work will occur is named *survey.db*. This affects every aspect of the system (including the reading of the spec file itself!), so this key/value pair is the first in the file and is not in any subgroupings.
The second line gives the field in the survey that uniquely identifies each record. In this example, it is named `sid`. This is not necessary for input, but will be used for several steps in the pipeline, and is typically listed toward the top of the specification.
The `input file` and `output table` keys affect only the input step, so they are grouped into the `input` section. This section specifies that the text of `text_in.csv` shall be written to a new database table named `dataset`. Like all tables generated over the course of the process, it will be stored in the `survey.db` database file.

A. Recodes

32. The recode section of the specification assigns a category or class variable to selected input variables. For example, this segment will produce three new variables expressing age, income, and state groupings:

```
recodes {
    agecat {
        0 | age between 0 and 18
        1 | age between 19 and 64
        2 | age > 64
    }

    wagecat {
        0 | income == 0
        1 | income > 0 && income < 10000
        2 | income >= 10000 && income < 100000
        3 | income >= 100000
```

```
    }

    region {
        Northeast | state in ('ME', 'NH', 'VT', 'MA', \
                              'RI', 'CT', 'NY', 'PA', 'NJ')
        Midwest | state in ('WI', 'MI', 'IL', 'IN', 'OH'\
                              'MO', 'ND', 'SD', 'NE', 'KS', 'MN', 'IA')
        South | state in ('DE', 'MD', 'DC', 'VA', 'WV', 'NC', 'SC', 'GA',\
                              'FL', 'KY', 'TN', 'MS', 'AL', 'OK', 'TX', 'AK', 'LA')
        West | state in ('ID', 'MT', 'WY', 'NV', 'UT', 'CO', \
                              'AZ', 'NM', 'AK', 'WA', 'OR', 'CA', 'HI')
        Other |
    }
}
```

33. These recodes are generated using SQL views. The new variables may be used for edits, but they can not be used for imputation–if `agecat` in the example above were imputed to be one, what would `age` be? In the other direction, if `age` is changed by an imputation, the SQL view automatically reflects any change to `agecat`.

34. Now that the data has been read in and useful artificial variables have been specified, we can move on to the next step in the pipeline: consistency checking.

IV. IDENTIFYING ERRORS

35. There are typically a few dozen to a few hundred checks that every observation must pass, from sanity checks like *fail if age < 0* to real-world constraints like *fail if age < 16 and marital_status='married'*.

36. These consistency rules are specified once, and reused in several contexts. The first pass is to check the input data against all rules, marking those records and fields which fail. Every time a change is made (such as by the imputation system) we need to re-check that the new value does not fail checks. An OLS imputation of age could easily generate negative age values, so the consistency checks are essential for producing valid imputations. Given the list of consistency checks, we can also use them for other purposes, such as setting structural zeros for the raking procedure.

A. Declarations

37. The user must specify the set of valid values that a variable can take, for two reasons. First, TEA's edit subsystem, based heavily on the DISCRETE system by Bill Winkler [Winkler, 1995], has an internal data structure whose size depends on the number of valid values. Second, the list of valid values provides the first set of edits: if `age` may take on values from zero to 110, then we already know that a value of 1980 is invalid.

38. Here is a sample set of fields, including two integer-valued fields, a categorical field, and a real-valued field.

```
fields {
    age int 0−116
    sex int 0, 1
    state cat AL, AK, AS, AZ, AR, CA, CO, CT, DE, DC, FM, FL, \
          GA, GU, HI, ID, IL, IN, IA, KS, KY, LA, ME, MH, MD, MA, \
```

```
            MI, MN, MS, MO, MT, NE, NV, NH, NJ, NM, NY, NC, ND, MP, \
            OH, OK, OR, PW, PA, PR, RI, SC, SD, TN, TX, UT, VT, VI, \
            VA, WA, WV, WI, WY, AE, AA, AE, AE, AP
    wage real
}

checks { #see below.
    include checks.spec
}
```

B. Edits

39. Edits are unordered, and conceptually, all relevant edits are checked at once (a real-world computer will process them sequentually, but the actual sequence is irrelevant). Imputations may be ordered; see below.

40. Edits are generated (and in the case of real-valued variables, tested) using SQL, so it makes sense to specify them using SQL syntax. In the example above, the section listing consistency checks was one line: `include checks.spec`, which allows the person writing and maintaining the consistency checks to maintain the `checks.spec` file without interfering with the others. To give an example, here is a short edit specification, listing the conditions that would indicate a failure (where `&&` can be read as 'and'):

```
age < 5 && wage > 0
wage < 0
```

41. The simplicity of the edit specification is a consequence of how superbly suited SQL is to finding the subset of a group of records that meets a given set of criteria. TEA can use the specification as written by the user by simply pasting the edit, without modification, into an SQL query. For example, it is a simple procedure to assemble the above information into a query which returns a list of the records that fail the edits:

```
select sid
from dataset
where
    (age < 5 && wage > 0)
    or
    (wage < 0)
```

42. This is how TEA handles edits involving real-valued fields. The procedure for integer and categorical values includes several time-saving tricks which will not be discussed in this overview (details are available upon request).

C. Editing outputs

43. The `check_consistency` function at the core of TEA (it is not directly user-callable) can be run in three ways.

- The simplest is a simple pass/fail check on a record.

- For more detail, the system can give the list of every field involved in any failed edit.[4]
- The most extensive return value is a full list of all of the alternatives that could pass the edits.

44. Different users of the `check_consistency` function may use this information in different ways. The R command `doChecks()` uses the list of fields involved in any failed edit for a record. It takes the direct approach of making a backup copy of the original data table, then blanking all of the fields involved in any failed edit, leaving the imputation system to reimpute all of them.
The imputation step will use the pass/fail facility to determine whether to keep the drawn values.
We have experimented with using the list of alternatives as the universe of options from which to draw. One could assign a uniform distribution to each alternative, or weight values according to a probability model.
Other methods of using the error identification information are planned for future work. Some involve more sophisticated edits with less of a focus on probabilistic modeling, including using the Fellegi-Holt algorithm and swapping values across records (e.g., when a father and son's ages have been switched).

V. INFERENCE CONTROL

45. Inference control as currently implemented in TEA is based on the number of elements sharing a given set of characteristics, such as female, under age 18, income over $100,000. If there are only two or three individuals in the data set with those characteristics, there is risk that an attacker can identify one of them, and thus use the survey to learn additional information about her. Here is a sample specification for this type of inference control:

```
fingerprint{
    combinations: 2
    frequency: 5
    key {
        sex
        agecat
        wagecat
    }
}
```

46. By setting `combinations` to two, the specification will find all 2-dimensional subsets of all of the listed keys: (sex $\times$ agecat), (sex $\times$ wagecat), (agecat $\times$ wagecat). If there are fewer than `frequency` records in a given category, the relevant variables are flagged for potential imputation.

VI. IMPUTATION

47. At this point in the pipeline, we have found several records in need of new values, including data that was never reported or recorded, data that the edits report is clearly incorrect, and data that is flagged as a disclosure risk. For simplicity of exposition, I will refer to all of these data points as marked as missing, regardless of the reason.

[4] This is by no means the minimal set that would resolve the edits. For economic data, where there are often accounting identities that may point to one field being a typo, the minimal change is very desirable. For population data, edits tend to not have such a deterministic solution.

Given that the edit and inference control checks have already flagged problem data elements, and the specification file lists a sequence of imputation models, the procedure is as follows. The steps will be explained in greater detail below.

(1) The fields that fail edits or are otherwise missing are noted.
(2) The category in which the record falls is selected using the algorithm described above, giving us a subuniverse of complete data.
(3) For each imputation (the specification above set `draw count` to five):
    (a) For each model (such as the age and income models in the specification):
        (i) The model for the given field is estimated based on the subuniverse of the data.
        (ii) Given a fully estimated model, a draw is made.
        (iii) For integer and categorical data declared in the `fields` section above, if the drawn value is outside the declared range, redraw a new value.
    (b) If the now-filled record fails an edit, redraw new values and recheck the edits. In our TEA tests using ACS data, almost all records are filled after about four redraws.
    (c) Record a successful imputation in a separate table.

48. Because the imputations are recorded in a separate table, we are left with the original data and a list of imputations for all missing spaces in the data.
TEA's `checkout_impute` function will fill the missing elements in the data table with a single imputation. Users may choose to report all multiple imputations by calling `checkout_impute` five times, or report only the first. TEA also provides a function to calculate the total variance (within-imputation variance plus across-imputation variance) for a given statistic.
Alternatively, omitting the `draw count` from the specification indicates that TEA should run only one imputation.

A. Subuniverses

49. The parameters of the model will be estimated using non-missing data from the survey, but we typically presume that the best model comes from data that is 'close' to the subject whose data is missing, such as those respondents living in the same tract, with the same sex, or in the same income bracket. For the case of the Randomized Hot Deck model, in which missing values are filled in by simply drawing from the non-missing data in the same category as the record with missing data, the real design question is about what categories to divide the population into.

50. Here is a sample specification that will be dissected over the course of this section.

```
impute {
    min group size: 20
    draw count: 5

    categories {
        num_jobs_2008 = 0
        num_jobs_2008 = 1
        num_jobs_2008 => 2
                age < 18
        18 <= age < 35
        35 <= age < 65
        65 <= age
        state
    }
```

```
    age : hot deck
    income: lognormal
}
```

The `categories` subsection presents a set of subdivisions for the population, including three categories for the number of jobs held in 2008, four age groups, and state of residence (which, being listed in the spec as a bare variable with no restrictions, is understood to mean that each value is a distinct category). Given a record {age=43, num_jobs_2008 =1, state=OH, income=NA}, income needs to be imputed, TEA would first find those records that have a non-missing income value, living in Ohio, age between 35 and 65, holding one job in 2008. This specification requires a minimum group size of 20. If there are too few non-missing values, then TEA will drop the bottom category, in this case state, and thus use all records age between 35 and 65, holding one job in 2008. If this is still too small, then the age category is also dropped, continuing until there are enough non-missing values to meet the group size specification. At the extreme, the system may need to use the entire data set with no subcategories to satisfy the minimum group size restriction.

51. Having found the non-missing data for the category, TEA can go on to using that data to estimate a model for the missing data and draw from it to fill in missing values.

B. Model estimation

52. In this section, those items marked as missing are filled in using models chosen by the user. At its core, the process breaks down into two steps: estimation and drawing.

53. The estimation procedure takes in a complete data set and returns the estimated parameters. For example, a Normal model returns μ=the mean of the input data and σ=the standard deviation of the input data, and an OLS model returns the coefficients of the linear regression (typically notated β).

54. Apophenia [Klemens, 2008] is a library of C functions and structures built with exactly this principle of providing a standardized model interface. As such, it already provides a number of models that are of exactly the form needed here, including Univariate and Multivariate Normal distributions, Probability Mass Functions (PMFs, aka histograms, aka Discrete distributions), Loess smoothing, OLS regression, Probit and Logit, et cetera. Each has an `estimate` function that takes in data and produces parameter estimates, and a random `draw` method that takes in a set of parameter estimates and produces synthetic data. TEA can use models in this standardized form directly.
Thus, the problem of standardizing models to the point that they can be interchanged by the user as desired is already solved, leaving TEA with only the logistics of parsing the user's text specification of the model and putting the data into the correct form for input to Apophenia's models.

C. The range of models

55. The intent of TEA is to not dictate to users the means by which imputation is executed. For example, users may choose to draw imputed values directly from the existing data–Hot Deck. The underlying assumption is that data is Missing Completely At Random (MCAR), and guarantees that any post-imputation distribution an analyst might be interested in will be approximately identical to the analogous pre-imputation distribution.

56. Data is Missing At Random (MAR) when the probability of a field being missing is correlated to one of the non-missing fields, but not the value of the missing data given the non-missing data. It is common to apply regression techniques, such as OLS or Logit estimation, to impute MAR data.

57. Rolando Rodríguez, also at the U.S. Census Bureau, has written a bridge for Apophenia and R, Rapophenia, that wraps an Apophenia model around estimate and draw routines written in R. At that point, they can be used by TEA like any other. Using Rapophenia, he has implemented one popular technique in the missing-data literature, Sequential Regression Multiple Imputation, which iteratively re-estimates a sequence of regressions to fill in multiple missing fields [Raghunathan et al., 2001].

VII. CONCLUSION

58. TEA provides a single platform for editing, inference control, and imputation, but it takes pains to keep these procedures separate. The specification is significantly simplified by the separation of components–with no prior experience, the specifications segments throughout this paper should have been basically readable. Also, because there is no need to customize imputation models to accommodate editing, much more sophisticated modeling techniques can be used. In most cases, usage is a relatively simple plug-and-impute routine. TEA is based on a system that already provides such a standardized model interface, so from the start it has a long list of available models and a mechanism for adding more.

59. The framework advises a few obvious paths for further work, as more methods can be added to each component. Many models in the statistical literature have not yet been applied to real-world surveys, but once they have a basic implementation via Apophenia or Rapophenia, they can be used in TEA. There is currently only one inference control routine, and concepts like differential privacy are not yet reflected.

References

M Bankier, P Mason, and P Poirier. Imputation of demographic variables in the 2001 Canadian Census of Population. In *American Statistical Association, Proceedings of the Section on Survey Research Methods*, 2002.

Bor-Chung Chen, Yves Thibaudeau, and William E Winkler. A comparison study of ACS if-then-else, NIM, and DISCRETE edit and imputation systems. In *ASAJoint Statistical Meetings: Section on Survey Research Methods*, 2002.

IP Fellegi and D Holt. A systematic approach to automatic edit and imputation. *Journal of the American Statistical Association*, 71:17–35, 1976.

Ben Klemens. *Modeling with Data: Tools and Techniques for Statistical Computing*. Princeton University Press, 2008.

Mark Pierzchala. A review of the state of the art in automated data editing and imputation. *Journal of Official Statistics*, 6(4):355–377, 1990.

Trivellore E Raghunathan, James M Lepkowski, John Van Hoewyk, and Peter Solenberger. A multivariate technique for multiply imputing missing values using a sequence of regression models. *Survey Methodology*, 27(1):85–95, June 2001.

Donald B. Rubin. *Multiple Imputation for Nonresponse in Surveys (Wiley Classics Library)*. Wiley-Interscience, 2004.

William E Winkler. Editing discrete data. In *American Statistical Association, Proceedings of the Section on Survey Research Methods*, pages 108–113, 1995.