Experiences with R-based data editing systems

Mark P.J. van der Loo (m.vanderloo@cbs.nl)
Statistics Nederland

ABSTRACT

In this short paper, we review our experiences with building an R-based automated data editing street, pointing out lessons learned in both software applications and user expectations. The lessons include a number of features that will be useful for the further development of our R packages and led to the discovery of a simple but very useful property of reliability weights used in the process of error localization. Finally, we discuss future directions for our data editing software.

INTRODUCTION

The task of preparing data prior to data analyses is as an important part of statistics production as the actual analyses. Indeed, the consequences of (not) cleaning data for the quality of final output have been widely investigated, especially for the case where data is obtained from a survey based on probability sampling [see De Waal *et al.* (2011) and references therein]. In general, survey data as well as data obtained from other sources (administrative data, or more recently big data sources) should be expected to suffer from missing items, technical errors and logical inconsistencies. To produce statistics based on such data, one needs to make an informed decision on how to either treat such cases or how to properly ignore them for reliable statistical analyses to take place.

Since the resources involved in data editing can be substantial [De Waal et al (2011) report that the cost of a statistic may be determined for up to 40% by data editing], one typically would like to automate the process as much as possible. The approach taken by the official statistics community is to combine a automated, rule-driven approach with a so-called 'selective editing' approach where a selection of records with the (potentially) most influcential errors are treated by domain experts [See e.g. Pannekoek *et al.* (2013); De Waal (2013)]. In this paper we shall focus on the fully automated, rule-based part of the process.

In the rule-driven approach, domain experts configure a generic data editing system by specifying a set of restrictions the data must satisfy and/or a set of data transformation rules. The software then edits the data to satisfy those restrictions in a controlled way which very generally may be separated in the following steps:

1. Check data against predefined (in-record) restrictions.
2. Determine or estimate the field(s) causing rule violations.
3. Derive or estimate a better value for the erroneous fields.

The first step is referred to as *data validation*, the second step is called *error localization* and the third step consists of data amendment and imputation.

Examples of software following a rule-based strategy include Statistics Candada's CANCEIS (Guertin et al, 2014) for automated editing of Census data, and the SAS-based BANFF software of the same office, for editing business survey data (Kozak, 2005). The current author has co-authored several R packages in the area of rule-based data editing, notably:

- The editrules package for rule-based data validation and error localization based on the principle of least change (De Jonge and Van der Loo, 2014).
- The deducorrect package for deductive and deterministic error correction and imputation (Van der Loo, 2014).
- The rspa package for minimal numeric record adjustment under edit restrictions (Van der Loo, 2014).

The modular setup of these packages is aimed to provide a flexible set of tools that can be used for both command-line use for one-off jobs as well as to build production grade data editing systems.

In this paper, we discuss our experiences with the implementation of such a production street based on the packages mentioned above. As the project involved translating a custom system build by domain experts to a more general and more modular setup, it gave an excellent opportunity to compare user needs and expectations in terms of rule definition, methodology, and process flow with the rules, methodology and flow defined by the R packages, allowing us to determine future directions for their development.

The rest of this paper is structured as follows. In the next section we give a short overview of the functionality provided by our R packages. Next, we discuss the general architecture of the data editing street build based upon them, pointing out how components of the R-packages have been used and re-used several times over in the process. After that we point out a number of user needs that were not (yet) provided by the packages which necessitated building a few custum-build components for this system. In particular, it was necessary to provide a more general interface to rule definitions that amongst other things allows dynamically derived variables to enter data validation rules. Also, certain wishes on how error localization should be performed led us to make a simple, yet interesting and useful observation about so-called reliability weights for error localization. In the final section we summarize our conclusions and discuss a further outlook for the development of our packages.

AN OVERVIEW OF OUR R-BASED DATA EDITING PACKAGES

In this section we give a quick overview of functionality of the R-packages editrules, deducorrect, and rspa. An introduction to their use with references to methodology can also be found in the lecture notes of a tutorial given at the *useR!*2013 conference (De Jonge and Van der Loo, 2013).

The editrules package is principally aimed at definition of in-record data restriction rules ('edits'), validating data against those restrictions, and error localization based on the principle of minimal change in the sense defined by Fellegi and Holt (1976). It also contains some

functionality for rule management, including de-duplication, detection of contradictions, and detection and removal of simple redundancies. Rules may be defined through the R command line or in a text file. Figure 1 below shows a text file readable by the package that illustrates some typical cases.

Examples of restrictions allowed by the editrules package

Figure 1

```
# Numerical, linear restrictions
turnover >= 0
profit + cost == turnover
profit < 0.6 * turnover

# (Conditional) restrictions on mixed data
gender %in% c('male', 'female')
if ( gender == 'male' ) pregnant == FALSE
 # Mixed conditional restrictions (numeric and categorical)
 if ( n_employees > 100 ) company_type == 'large'
```

The deducorrect package exports a number of methods that repair rule violations in a deterministic or deductive way. With deductive, we mean that a value for an erroneous field can be derived uniquely from the edit restrictions and the valid data in a record. For example, consider a record (*profit = 10*, *cost = 5*, *turnover = NA)* subject to the rule *profit + cost = turnover.* In this case the missing value for turnover is easily derived. Such and more general cases with systems of (in)equalities can be solved by methods implemented in the deducorrect package. Similarly, methods for categorical or mixed categorical under restrictions are available as well.

With deterministic methods, we methods for repairing commonly occurring errors (typing errors in numerical data, variable swaps) that can under certain circumstances be easily detected and repaired. Deterministic methods also include simple user-defined data transformations that may be executed on the data so they can be logged for reproducibility. Such rules include the derivation of new variables and transformation of variables when some user-defined condition is satisfied. Figure 2 shows some simple examples.

Examples of variable derivation and data transformation rules

Figure 2

```
# deriving a new variable
x <- y - z

# Conditional transformation (empty a variable)
if ( x > y ) x <- NA

# Conditional transformation (altering a variable)
if ( x > y & y > 0 ){
   z <- y + w
}
```

The deducorrect package translates such transformation rules to vectorized R-statements, executes them and logs their result. These transformation rules should be interpreted as a way to translate knowledge of domain experts directly to executable rules which are then executed in a reproducibly way by the software. It should be noted that although in the example we use only linear transformations, this is not a restriction of the package. Moreover, conditional statements may be nested and multiple transformations can be defined within a consequent. In the next section we shall see that these deterministic, user-defined execution rules play a role in several phases of the automated data cleaning process.

The rspa package implements methodology to apply a minimal adjusting to (subsets of) numerical variables such that the end result obeys a predefined set of linear (in)equations. Specifially, denote with $\boldsymbol{x}^{\mathbf{0}}$ a record of numerical values, subject to a set of linear (in)equality restrictions $\boldsymbol{Ax} \leq \boldsymbol{b}$. The rspa package uses the successive projection algorithm of Hildreth (1950) [but see Pannekoek and Zhang (2012) for a recent discussion in the context of official statistics] to find a new value $\boldsymbol{x}^{\mathbf{1}}$ such that the weighted sum

$$(\boldsymbol{x}^{\mathbf{0}} - \boldsymbol{x}^{\mathbf{1}})^T \boldsymbol{W} (\boldsymbol{x}^{\mathbf{0}} - \boldsymbol{x}^{\mathbf{1}}),$$

is minimized. Here, $\boldsymbol{W}$ is a diagonal matrix whose nonzero elements contain positive variable weights. Variables with a higher adjustment weight will be adjusted by a smaller amount then variables with a lower weight.

A typical use of the package in a data editing process is as follows. Once erroneous fields have been selected, they will be emptied. After this imputation takes place without regarding the linear (in)equality restrictions so imputed records will generally violate restrictions. The imputed values can then be minimally adjusted according to the criterion mentioned above so every restriction is guaranteed to hold.

AN R-BASED DATA EDITING PRODUCTION STREET

The aim of building the production system we're about to discuss was to replace an older data editing system that had been used for a number of statistics, mainly in the area of healthcare economics. To ensure continuity of production and statistical output, it was decided to keep the work flow and statistical methodology consistent with the old system as much as possible although some methodological improvements have been implemented as well. The most important aim was to build a system that was easier to maintain by setting it up in a more modular fashion. The system is currently developed to edit numerical data, that have to obey linear (in)equality restrictions and it can be used by multiple users simultaneously.

The data editing system developed consists of three main parts: a database, storing raw data, data at various stages of editing and a log of all edits performed on the data; an R-based automated data cleaning system for the more complex and data editing specific methodologies; and a two-part user interface, written in C#, allowing users to load data and start the data editing process and to manually alter individual records.

Figure 3 gives a high level overview of the data editing work flow and a number of the system's components. The solid arrows indicate the flow of data through the system while the dashed arrows show where user-defined configuration data like edit restrictions and execution rules enter the system. For reasons of clarity we left out data flowing from and to the database as we wish to focus on the methodological aspects in this picture.

The basic work flow is as follows. As a new dataset arrives, a user loads in the dataset and starts an initial data editing run. In that case, all records pass through two stages of data editing (Editing pt. I and II, to be discussed shortly) and are written to the database. At that point, all records are guaranteed to satisfy every restriction imposed on it, up to perhaps a small numerical tolerance. Next, a domain expert may choose to export the dataset and investigate it interactively (here, SPSS is used most often), checking for outliers or other features that might indicate errors that have been missed by the system. If additional errors are found, records may be retrieved by the manual editing application, altered and fed back into the system. This process may repeat a number of times for various records until the output is approved for statistical estimation.

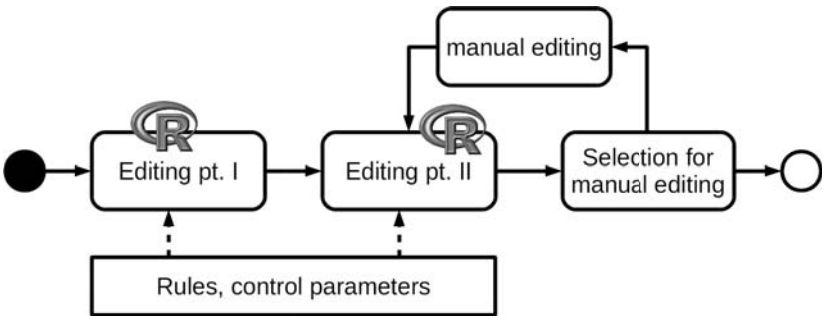
As shown in Figure 3, the R-based data editing street consists of two main steps, both controlled by a set of user-defined rule sets and other parameters. The first main step consist of a number of deterministic methods including: deriving a set of indicator variables; applying a set of user-defined mutations; correcting unit of measure errors; correcting typos in numerical data; correcting roundoff errors. The second step consists of error localization based on user-defined execution rules and additional error localization based on the principle of minimum change. Next a sequence of imputation steps imputes all missing values and finally the rspa package is used to adjust imputed values so all linear (in)equality rules are satisfied. An description of the substeps involved is also given in Pannekoek and Van der Loo (2014).

There are two interesting points about these steps. The first, technical point is that there are four different substeps that are controlled by user-defined execution rules (like those in Figure 2). These steps are the user-defined variable derivations and mutations, part of the correction methods for detection and resolving unit measure errors, and the error localization based on user-defined execution rules. In the old system, these rules were partly defined outside the system and partly hard-coded. In the new system, configuration of these rules is done fully outside of the system. Moreover, although for users these are four logically different configuration steps, under the hood, the exact same R-function is called four times, making sure that the actions are logged such that they are recognizable afterwards.

A second interesting point involves the the interaction between the manual and automated data editing. At the manual editing stage, the domain expert is not required to deliver a fully consistent record. Indeed, manual editing often takes place based on extra external information such as public financial reports that only contain part of the information necessary to fill a record. Sometimes a domain expert may spot a value that is clearly wrong but without having good information for an accurate better value. In such cases, he or she may empty the variable and let the system impute a value.

Figure 3

High level overview of the data editing process flow



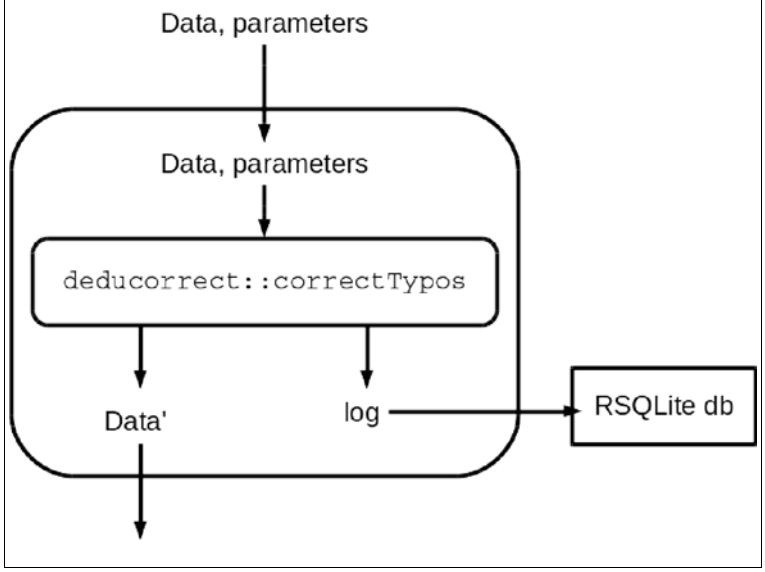
TECHNICAL DETAILS

The R-part of the system is implemented as an R package that depends on editrules, deducorrect, rspa, RSQLite and a few others indirectly. The package is used to allow for uniform calls to various data cleaning routines and to ensure that logging data is written to a temporary RSQLite database in a uniform matter for all methods. Most of the package consists of simple wrappers around pre-existing functionality. The main steps 'Editing pt. I' ad 'Editing pt. II' consists of a simple R scripts that reads parameters and input files, calls a sequence of of functions of the package and writes out various output files.

Figure 4 shows the typical design of a wrapper function. The function `correctTypos` of the deducorrect package takes a dataset and some parameters (including a set of edit restrictions) as input and outputs an object that contains (possibly) altered data and a log of all the alterations that were performed. The wrapper takes as input a dataset and the same parameters, but diverts the logging to an in-memory RSQLite database and outputs the data only directly. The main advantage of this approach is that it becomes very easy to add or remove methods or to interchange their order. In principle one could use the chaining operator of the magrittr package of Bach and Wickham (2014) that has recently become very popular through its use in packages like dplyr (Wikham and Francois, 2014) and ggvis (RStudio, 2014).

Wrapper function

Figure 4



The interface between the R and other parts of the system was kept very simple. The R-street is activated by a C# component simply by calling the Rscript program that comes with the base R installation. Rscript is passed the .R file that performs the data editing and a single command line argument pointing to the location of a (generated) XML file that contains information on where the R script can find rule sets, input data and reference data and where it has to write its output. The XML package of Temple Lang (2013) was used to parse the file.

RULE DEFINITION ISSUES

As stated above, the system described here is aimed to translate features of an older system to a more flexible and maintainable environment. This also meant translating the rules and parameter settings of the earlier system to the new one. Although much of the work could be automated in a fairly simple matter, we needed to add a few features on top of existing functionality of editrules and deducorrect to accommodate all user requirements. Moreover, we've found that the syntax commonly used to define conditional edit restrictions can be interpreted in various ways by users.

TWO INTERPRETATIONS OF CONDITIONAL EDIT RESTRICTIONS

Conditional restrictions are not uncommon in for example business statistics. For example, we have encountered restrictions such as

```
if ( n_employees > 0 )  salary_payed > 0          [1]
```

several times during development of the project. The common interpretation of such a rule is to think of it as a logical implication

$$n>0 \Rightarrow s>0, \qquad [2]$$

where n is the number of employees and s the amount of salary payed. Using the implication replacement rule we may also write $\neg(n>0)\vee(s>0)$. Given a record (n,s) one may check the truth value of the implication by substituting the values of n and s in Equation [2]. For example, the record $(n=1,s=0)$ yields `FALSE`, indicating that the rule is violated and $(n=1,s=2)$ yields `TRUE`. For records violating the rule, one needs to either alter n, s, or both variables for the record to satisfy the rule. Deciding which variables to alter is, in fact, error localization.

Writing such an implication rule using an if-else like syntax sometimes causes confusion amongst users. We have encountered an alternative interpretation that needed to be taken into account when translating rules from the old system to the new one. In most cases, if-statements were not to be interpreted as logical implications but rather as stratifying statements whose conditional statement indicates when the statement in the consequent must hold. That is, we found that in some cases the variables used in the condition (such as company type) were deemed so reliable that if the implication was violated, the variable in the consequent was always determined to be erroneous. Such an interpretation clearly mixes work flow management – *i.e.* what rules apply to what stratum, with semantic rule definition where ale variables are treated equal.
This issue was met by setting up the configuration such that users may define a general file with rules holding for each stratum as well as separate files defining rules specific to a stratum. In many cases, we could simplify the process by translating the user-defined rules to deterministic error localization rules. That is, in the interpretation of the old system, the rule of Equation [1] could be replaced simply with the following execution rule:

```
if ( n_employees > 0 & salary_payed <= 0 ) salary_payed <-  NA
```

so in records where the condition is met, the salary variable is emptied for imputation further on in the process.

RULE DEFINITION FEATURES

Although the syntax of editrules and deducorrect is in principle strong enough to define all edit restrictions and execution rules needed for the process. We added for the current system a few features significantly enhances the user experience when defining rules. These features are the definition of transient variables, defining variable groups, and allowing derived variables to occur in edit restrictions. Below we describe how those features are now implemented in the system. Neither of them have been published (yet) as part of our packages.

The first feature, defining transient variables occurs for example when several execution rules depend (partially) on the same calculation. It is very convenient to be able to store the result of such a calculation rather then performing it twice. This was solved by introducing the (in R unused) `:=` operator as an extention to the existing syntax allowed by deducorrect. For example, rather then defining the following:

```
if ( a + b > 0 & c < 0 ) <do X>
if ( a + b > 0 & c > 0 ) <do Y>
```

which forces R to compute `a + b > 0` twice, one may define

```
T :=  a + b > 0
if ( T & c < 0 ) <do X>
if ( T & c > 0 ) <do Y>
```

Here, the variable `T` is computed and stored only for the duration of the call to the rule execution function (hence the name 'transient variable').

The second feature involves defining variable groups, which is basically a simple form of templating. In some cases, a similar rule must hold for a series of variables or variable combinations. For example, suppose we have the following set of rules.

```
A > X
A > Y
B > X
B > Y
```

This may be replaced with the following syntax

```
G := {A;B}
H := {X;Y}
G > H
```

Internally, the system will expand the group definitions to the extensive set which will then be applied to the data. A similar syntax can be used for executable rules. For example, suppose that for a series of variables A, B, C, D, E we wish to check whether they are larger then zero, and if not so, set them to `NA`. Using the grouping feature we may simply write

```
G := {A;B;C;D;E}
if ( G <= 0 ) G <- NA
```

rather than repeating nearly the same statement five times.

Finally, recall that the system has a phase where new variables may be derived by users. In our cases, there was a desire to define edit restrictions in terms of those derived values. For example in one step, a derivation could look like this:

```
balance <- total_profit - total_cost
```

and in another step one would have a condition such as

```
balance > 0
```

If the above rule is violated, an error localization algorithm may choose to adapt the derived balance variable without altering the underlying components. This was solved by substituting automatically, the right hand side of derivation rules whenever a derived variable occurs in an edit restriction.

All the above features are easily implemented in R when the syntax for restriction rules and execution rules are a subset of R syntax (as is the case for editrules and deducorrect). In particular, R's '`parse`' funtion returns the abstract syntax tree of an R statement which makes it possible to filter, alter or re-interpret statements in a customized fashion.

TURNING THE PRINCIPLE OF LEAST CHANGE

Error localization that takes the edit restrictions into account is a central step in any data editing process. The editrules package implements error localization routines that localizes errors based on the generalized principle of Fellegi and Holt which reads as follows. Given 3 record $\boldsymbol{r}=(r_1, r_2, \ldots, r_n)$ violating one or more edit restrictions imposed on it, find a subset of variables $S \subset \{1, 2, \ldots, n\}$ that (1) can be imputed such that all edit restrictions will be obeyed and (2) the sum

$$\sum_{j \in S} w_j \qquad [3]$$

for positive weights w_j is minimized. The weights w_j are sometimes referred to as 'reliability weights', since variables associated with larger assigned weights are less likely to be part of a solution to the error localization problem. We will call solutions that satisfy property (1) a *feasible solution*, and solutions that satisfy (1) and (2) a *minimal feasible solution*. If S is a solution then we denote with $|S|$ the size (number of variables) of the solution. A solution that has a minimal number of variables is called a *solution of minimal size*.

Observe that depending on the weights used in Equation [3], the generalized principle of Fellegi and Holt does not guarantee that the minimal number of variables is altered: the minimal feasible solution is not necessarily of minimal size. For example, suppose we have $n=3$ variables and the feasible solutions to the error localization problem are $\{1\}$ and $\{2,3\}$. If all weights are equal the first solution will be chosen, minimizing the number of variables to be edited. However, if we set $w_1=1$ and $w_1=w_2=1/3$, the second solution will be preferred.

In the automated data editing system described above, the reliability weights are computed automatically based on the difference between a predicted and observed value so that on one hand a larger difference reduces the reliability weight. On the other hand, we wish to ensure that no more variables are edited then strictly necessary, so as to stay as close to the observed data as possible.

Over time several algorithms for solving (specific cases of) de generalized error localization problem have been proposed [See e.g. the thesis of De Waal (2004)]. In editrules, an approach based on the branch-and-bound algorithm of De Waal and Quere (2003) and an approach based on Mixed Integer Programming (MIP) [De Waal (2003)] are implemented. The advantage of the branch and bound method is that it allows for fine-grained control of the optimization process. In particular, it is possible to produce all feasible solutions, then select the ones that alter the smallest number of variables, and choose the one of lowest weight from that subset. However, this is cumbersome and in fact as demonstrated by for example De Jonge and Van der Loo (2014) the MIP-based approach is much faster for typical error localization problems and also has better memory scaling properties. The advantage of a MIP-based approach is that one can offload the work to existing, well established MIP-solvers. The disadvantage is that one typically looses some control over the algorithm, most notably, it is usually not possible to return all feasible solutions.

As it turns out, it is possible to rescale any weight vector in such a way that guarantees (1) that the relative order of the weights does not change and (2) the minimal feasible solution is also a solution of minimal size. In particular we will prove the following observation.

Proposition. Given a positive weight vector $\boldsymbol{w}=(w_1, w_2, \ldots, w_n)$ and write each weight as $w_j = 1+\delta_j$ with $0 \leq \delta_j \leq \delta^{max}$. If $\delta^{max} < 1/n$ then the minimal feasible solution to an error localization problem is also a solution of minimal size.

Proof. Suppose that S and T are different feasible solutions with $|S|<|T|$. The associated weights of these solutions can be written as

$$w(S) = |S| + \sum_{j \in S} \delta_j,$$

and likewise for $w(T)$. We wish to ensure that $w(S) < w(T)$. Since variables occurring both in S and in T do not add to the difference in weight we assume without loss of generality that the two solutions have no variables in common ($S \cap T = \emptyset$). In the worst case we have $\delta_j = \delta^{max}$ for $j \in S$ and $\delta_j = 0$ for $j \in T$. The demand $w(S) < w(T)$ for this case is written as

$$(1+\delta^{max})|S| < |T|. \qquad [5]$$

Since we defined $|S|<|T|$ we certainly have $|T| \geq |S|+1$ and we replace [6] with the stricter demand

$$(1+\delta^{max})|S| < |S|+1,$$

which sets the following restriction on δ^{max} :

$$\delta^{max} < \frac{1}{|S|}.$$

The value of $|S|$ is generally unknown prior to optimization, but since we know that $|S|<n$ (since by assumption $|S|<|T| \leq n$) we may impose the more strict demand that $\delta^{max} < 1/n$. ∎

There are some special cases. If we set $\delta^{max}=0$, all weights are equal to one and we return to the unweighted error localization problem, which by definition already guarantees that minimal feasible solutions are solutions of smallest size. In that case, there is no preference between multiple solutions of minimum size. If we set $\delta^{max}=1/m$ with $1\leq m\leq n$, we may interpret m as the maximum value for $|S|$ for which it is guaranteed that the minimum feasible solution is also the solution of minimal weight.

The above result can be used to derive from any weight vector $\boldsymbol{w}=(w_1, w_2, \ldots, w_n)$ a new weight vector $\boldsymbol{w}'=(w'_1, w'_2, \ldots, w'_n)$ that guarantees that the minimal feasible solution is also of minimal size. In particular, choosing $\delta^{max}=1/n$ we can set

$$w'_j=1+\frac{w_j-w^{min}}{w^{max}-w^{min}}\times\frac{1}{n},$$

where w^{min} and w^{max} are respectively the smallest and largest coefficients of $\boldsymbol{w}$.

As an example we apply this transformation to the three-variable example from the beginning of this section. We had $S=\{1\}$ and $T=\{2,3\}$. Using the original weight vector $\boldsymbol{w}=(1,1/3,1/3)$ we get $w(S)=1$ $w(T)=2/3$. After rescaling we have $\boldsymbol{w}'=(4/3,1,1)$ giving $w'(S)=4/3$ and $w'(T)=2$.

SUMMARY, CONCLUSIONS AND OUTLOOK

We have successfully implemented an R-based data editing system that is currently taken into production. While developing this system we determined a number of features, requested b users that we feel should be incorporated more generally in our data editing libraries. These features especially include more flexible ways to define and transform edit restrictions and data transformation rules. Furthermore, we have shown that it is not necessary to generate all feasible (minimal) solutions of an error localization problem in order to ensure that feasible solutions of minimal weights are also solutions of minimal size.

Based on these experiences, and also on our recent work on data validation and process indicators [see e.g. Van der Loo and Pannekoek (2014)] it was decided to further generalize the rule management procedures of editrules, as well as allowing for more general data restrictions. For this reason, we are currently separating the rule management features from the error localization features offered by editrules. For this purpose we are working on a new and currently unpublished package named 'validate', that has a more suitable internal representation of rules and focuses on rule management and data validation, leaving error localization and other features to a package that will depend on it. Examples of features that are currently implemented in validate are: variable group definitions, commands to recursively include files into other (rule definition) files and generic data validation procedures. The package will also support the use of the increasingly popular 'piping' operator in R.

References

Bache, S.M., and Wickham, H. (2014) *magrittr - a forward-pipe operator for R*. R package version 1.0.1 http://CRAN.R-project.org/package=magrittr

Fellegi, I. P. and Holt, D. (1976), *A Systematic Approach to Automatic Edit and Imputation*, Journal of fthe American Statistical Association, **71** 17-35.

Guertin, L., Bureau, M. and Morel, J. (2014) *Editing the 2011 Census data with CANCEIS and options considered for 2016* Proceedings of the UNECE Conference of European Statisticians Work Session on Statistical Data Editing Paris, France (pdf).

Hildreth, C. (1957). *A quadratic programming procedure*. Naval research logistics quarterly **4**, 79–85.

De Jonge, E. and Van der Loo, M. (2013a) *editrules: R package for parsing, applying, and manipulating data cleaning rules*. R package version 2.7.2. http://cran.r-project.org/web/packages/editrules/

De Jonge, E., and Van der Loo, M. (2013) *An introduction to data cleaning with R*. Lecture notes for a course given at the *useR!*2013 conference in Albacete, Spain (pdf).

De Jonge, E and Van der Loo, M. (2014) *Error localisation as a mixed integer problem in the editirules package*, Discussion paper 201417, Statistics Netherlands.

Kozak, R. (2005) *The BANFF system for automated editing and imputation.* Proceedings of the SSC Annual Meeting, June 2005 Proceedings of the Survey Methods Section (pdf).

Van der Loo, M., and De Jonge, E. (2014) deducorrect: Deductive correction, deductive imputation, and deterministic correction. R package version 1.3-5 http://cran.r-project.org/web/packages/deducorrect/

Van der Loo, M. (2014) *rspa: Adapt numerical records to fit (in)equality restrictions with the Successive Projection Algorithm.* R package version 0.1-5 http://cran.r-project.org/web/packages/rspa/

Van der Loo, M., and Pannekoek J., *Towards generic analyses of data validation functions*, in *United Nations Economic Comission for Europe Work Session on Statistical Data Editing*, Paris, 2014.

Pannekoek, J., Scholtus, S. and Van der Loo, M. (2014). *Automated and manual data editing: a view on process design and methodology.* Journal of Official Statistics **29** 511-537.

Pannekoek, J. and Van der Loo, M. (2014). *Implementation and evaluation of automatic editing for business surveys*, Proceedings of the UNECE Conference of European Statisticians Work Session on Statistical Data Editing, Paris France (pdf).

Pannekoek, J. and Zhang, L.-C. (2012), *Optimal adjustments for inconsistency in imputed data*. Discussion paper 201219, Statistics Netherlands The Hague/Heerlen (pdf).

RStudio Inc. (2014) *ggvis: Interactive grammar of graphics.* R package version 0.4, http://CRAN.R-project.org/package=ggvis

Temple Lang, T. (2013) *XML: Tools for parsing and generating XML within R and S-Plus*. R package version 3.98-1.1 http://CRAN.R-project.org/package=XML

De Waal, T., and Quere, R. *A fast and simple algorithm for automatic editing of mixed data.* Journal of Official Statistics **19** 383-402.

De Waal, T. (2003) *Processing of erroneous and unsafe data.* PhD Thesis, Erasmus University of Rotterdam (pdf)

De Waal, T. (2013) *Selective editing: a quest for efficiency and data quality*. Journal of Official Statistics **29** 473-488.

De Waal, T., Pannekoek J., and Scholtus, S. (2011). *Handbook of Statistical Data Editing.* John Wiley & Sons.

Wickham, H. and Francois, R. (2014) *dplyr: A Grammar of Data Manipulation*. R package version 0.3.0.2 http://CRAN.R-project.org/package=dplyr