WP. 33
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.

AUTOMATIC DATA EDITING WITH OPEN SOURCE R

Submitted by Statistics Netherlands [1]

I. INTRODUCTION

1. In recent years, the free statistical software tool R [R Development Core Team, 2012] has received an enormous surge in popularity that is illustrated by the near perfect exponential growth of user contributed extension packages Fox [2009] since 2001. R has been embraced as a strategic tool for statistics production by Statistics Netherlands since 2010.

2. Although R and its extension packages offer a wide range of methods for statistical analyses and data manipulation, methods for rule-based automated data editing have thus far been missing. Statistics Netherlands has developed two R extension packages, called deducorrect [Van der Loo et al., 2012] and editrules [De Jonge and Van der Loo, 2012] which help to fill this gap in functionality. The packages are part of a broader project that aims to deploy SN's validated methodology [Statistics Netherlands, 2010-2012] in a user-friendly way. The editrules package allows users to define, visualize and manipulate data editing rules, apply those rules to data to discover edit violations and to localize errors based on the generalized principle of Fellegi and Holt. The deducorrect package implements algorithms to solve typos, rounding errors and sign errors in numerical data. The package also implements some *deductive imputation* methods, that derive unique valid imputations based on the edit rules and the submitted data when possible. Both packages have been released via the comprehensive R archive network.

3. In this paper we will demonstrate some of the packages' core functionalities with simple R coded examples. Although some familiarity with the R language is helpful, it is no prerequisite for understanding the rest of the paper. The commands given at the interactive R prompt are preceded by a >, comments are preceded by a #. All other text in the code blocks is output. References to the theory behind the packages will be skipped. In stead, in section VII we give an overview of references pertaining to the theory and algorithms underlying the packages. The interweaving of text, code, and results in this paper is made possible by the knitr package of Xie [2012].

[1] Prepared by Mark van der Loo and Edwin de Jonge (m.vanderloo@cbs.nl). The views expressed in this paper are those of the authors and do not necessarily reflect the policies of Statistics Netherlands.

II. EDIT RULE DEFINITION AND DATA CHECKING

4. As a running example, we will use a block of data of the Dutch Structural Business Survey pertaining to staff numbers. The variables, and the rules they have to obey are given in the table and equations below.

Label	variable
p_1	Staff on salary list
p_2	Staff lent to others
p_3	Subtotal
p_4	Temporary from staffing agency
p_5	Temporary, other
p_6	Other persons
p_7	Subtotal
p_8	Total number of persons

Edit rules:

$$p_1 = p_2 + p_3$$
$$p_7 = p_6 + p_5 + p_4$$
$$p_8 = p_3 + p_7$$
$$p_2 < p_1 + p_6$$
$$p_i \geq 0, \quad i = 1, 2, \ldots 7$$
$$p_8 \geq 1.$$

The example used here only contains numerical variables. However, our packages can handle the following classes of edit rules.

- Linear (in)equations: $\sum_i a_i x_i \leq b$.
- Categorical restrictions in if-else form. Example: if $gender =$ male then $pregnant =$ FALSE.
- Conditional restrictions on numerical or mixed data. Examples include if $x > 0$ then $y > 0$, and if $age < 16$ then $Marital\ status =$ unmarried.

5. There are several ways of reading edits into R with the editrules package. The most convenient way is probably to write them in a free-format text file. For example, to define the edits above one can create a file edits.txt with the following content.

```
## contents of edits.txt

# balance account
p2 + p3 == p1
p7 == p6 + p5 + p4
p8 == p3 + p7

# sanity check
p2 < p1 + p6

# range edits
p4 >= 0
p5 >= 0
p6 >= 0
p7 >= 0
p1 >= 0
p2 >= 0
p8 >= 1
```

The lines starting with a # are comments, and will be ignored by the parser. Reading edits into R can be done by loading editrules and using the editfile command.

```
> library(editrules)
> E <- editfile("edits.txt")
```

The edits are read and parsed by editfile and stored into an editset object called E. This function is capable of distinguishing between numerical, categorical and mixed data edits, and accepts an optional argument type, telling the parser to extract edits of a specific type.

6. We now read in some data and show a few records using R's built-in head function (which displays the first six records).

```
> dat <- read.csv2("data.csv")
> head(dat)

   p1  p2   p3  p4  p5  p6  p7   p8
1 188   0  188   0   0   0   0  188
2   0   0    0   0   0   0   0    0
3  37   0   37   0   0  51  51   88
4  13   0   13   0   0   0   0   13
5  23   0   23   0   2   0   2   25
6  66   0   66   0   2   0   2   68
```

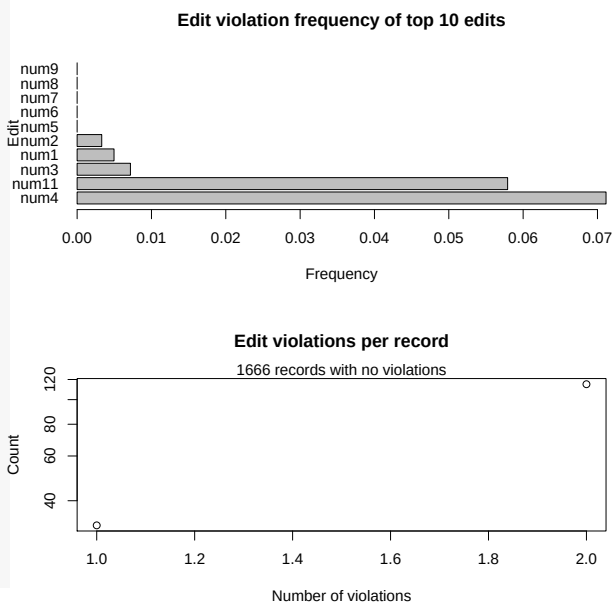
This particular file contains 1813 rows and has no missing values. We can detect how many rules are violated for each record as follows.

```
> ve <- violatedEdits(E, dat)
```

The function violatedEdits returns an object of class violatedEdits which is stored here in ve. The variable ve is a 1813×11 boolean array indicating which record violates what edit. R's native summary and plot have been extended in our package to represent some aggregates.

```
> summary(ve)

Edit violations, 1813 observations, 0
   completely missing (0%):

 editname freq  rel
     num4  129 7.1%
    num11  105 5.8%
     num3   13 0.7%
     num1    9 0.5%
     num2    6 0.3%

Edit violations per record:

 errors freq   rel
      0 1666 91.9%
      1   32  1.8%
      2  115  6.3%
```

```
> plot(ve)
```



The summary function (on the left) determines the number of times each edit is violated and how many edits each record violates. Results are sorted and represented in absolute and relative numbers. The same counts are represented graphically when calling plot, as shown on the right. The edits are labeled by editrules' parser with numbers, prefixed with three letters indicating their type. Numeric (in)equations are prefixed with num, restrictions on categorical data with cat and others with mix.

7. Subsets of edits can be retrieved by standard indexing. For example, the most violated rule in the example above is shown below.

```
> E[4, ]


Edit set:
num4 : p2 < p1 + p6
```

8. The initial quality of the example data is unusually high: 92% of the records violate no restrictions. When the full records are considered in stead of this subblock, typically about 50% of all records violate at least one restriction. We will return shortly to this in Section VI.

III. CORRECTING OBVIOUS ERRORS

9. The deducorrect package offers functionality to detect and repair some commonly occurring causes of errors in survey or administrative data. The errors are of such nature that once detected, they are easily solved. Errors that can currently be solved by the package include typing errors, sign errors and value swaps, and rounding errors.

10. An important design principle behind the deducorrect package is that the interface to various correction methods should be as uniform as possible. That means that both input and output for various correction functions are very similar: the minimal input is always a set of editrules and a data.frame (the way that R stores a rectangular dataset). The output is always an object of class deducorrect. A deducorrect object holds the corrected data as well as logging info such as a list of applied corrections, a status indicator and a timestamp. In the following we demonstrate how typing errors and rounding errors are solved with this package.

11. correctTypos tries to repair edit violations by replacing original values with values that differ no more than a certain string distance from the original. The string distance is measured as the number of transpositions, deletions, insertions or replacements of characters necessary to get from the original to the new value. By default, the maximum string distance is 1, corresponding to a single operation. Detecting and solving typing errors can be done with a single command.

```
> library(deducorrect)
> dat2 <- correctTypos(E, dat)
```

A list of applied corrections is stored in the $corrections attribute.

```
> head(dat2$corrections, n = 3)

  row variable old new
1 211       p1 452  45
2 292       p7  16   1
3 903       p8  34  35
```

Inspection of the first row shows that for the 211th record the original value 452 was replaced by 45. This corresponds to a string distance distance of 1 (a single deletion). Optionally, larger string distances can be allowed or one can assign separate weights to the operations constituting the string distance.

12. With violatedEdits and summary the increase in data quality can be inspected.

```
> summary(violatedEdits(E, dat2$corrected))

Edit violations, 1813 observations, 0 completely missing (0%):

 editname freq  rel
     num4  129 7.1%
    num11  105 5.8%
     num1    8 0.4%
     num3    5 0.3%
     num2    4 0.2%

Edit violations per record:

 errors freq   rel
      0 1673 92.3%
      1   29  1.6%
      2  111  6.1%
```

The treated data is stored under the dat2$corrected attribute. Observe that the number of records violating one or two edits have decreased while the number of fully clean records has increased from 1666 to 1673.

13. The correctRounding function randomly selects a variable occurring in edits which are violated by a small amount. If some or all edit violations can be fixed by adapting the variable by a small amount (typically, one or two units of measurement), the solution is accepted. Correcting for rounding errors and summarizing the resulting output can be executed as follows.

```
> dat3 <- correctRounding(E, dat2$corrected)
> summary(violatedEdits(E, dat3$corrected))

Edit violations, 1813 observations, 0 completely missing (0%):

 editname freq  rel
     num4  129 7.1%
    num11  105 5.8%
     num1    4 0.2%
     num2    4 0.2%
     num3    4 0.2%

Edit violations per record:

 errors freq   rel
      0 1677 92.5%
      1   26  1.4%
      2  110  6.1%
```

The number records violating one or two edits has decreased again and the number of fully clean records has increased from 1673 to 1677. By default, the rounding corrector looks for solutions to edit violations which are within 2 units of the original value. This can be tuned by passing an optional parameter (called delta) to correctRounding.

IV. ERROR LOCALIZATION AND VISUALIZATION

14. When all errors with a deducible cause have been repaired, Fellegi and Holt's paradigm offers a reasonable assumption to localize erroneous fields in a record. The function localizeErrors localizes the least (possibly weighted) number of fields in each record, such that they can be imputed with values without violating any explicitly defined or implied edits.

15. By default, the localizeErrors function first localizes variables violating range edits. Next, it divides the error localization problem in independent subproblems as much as possible and then performs a branch-and-bound search to localize errors in records under multivariate constraints. The localizeErrros function sets sensible defaults for many error localization problems. The search routines can be fine tuned either by passing optional arguments to errorLocalizer or by using one of the lower-level functions that have been exported to user space. These optional parameters include reliability weights (optionally per record), a maximum search time, maximum search depth and maximum solution weight, amongst others.

16. The basic interface to localizeErrors is similar to those of deducorrect's functions: at minimum it accepts a set of edits and a data.frame.

```
> el <- localizeErrors(E, dat3$corrected)
```

The return value of localizeErrors is an object of class errorLocation. The most important constituent of an errorLocation object is a boolean array indicating for each record which fields should be adapted. Below, we show the result for the first three rows.

```
> head(el$adapt, n = 3)

       adapt
record    p1    p2    p3    p4    p5    p6    p7    p8
     1 FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
     2 FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE
     3 FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE

> sum(el$adapt)

[1] 369
```

So in record 2, the variable p_6, p_7 and p_8 should be adapted. For this dataset, a total of 369 (2.5%) fields need to be adapted. Default R functionality quickly generates an overview of which variables are deemed erroneous by the error localization algorithm most often.

```
> sort(colSums(el$adapt))

 p2  p4  p5  p3  p1  p7  p6  p8
  2   6  17  53  55  56  75 105
```

So p_8 (total number of persons) must be adapted most often in this dataset.

17. It is instructive to visualize the dependency graph of edit sets. In editrules the default plot function of R has been overloaded to produce such a graph. Moreover, edits that are violated by a record as well as the solution to its error localization problem can be visualized by coloring the corresponding nodes.

```
> #
> #
> set.seed(1)
> plot(E)
```

```
> set.seed(1)
> v <- violatedEdits(E, dat3$corrected)[2, ]
> a <- el$adapt[2, ]
> plot(E, violated = v, adapt = a)
```

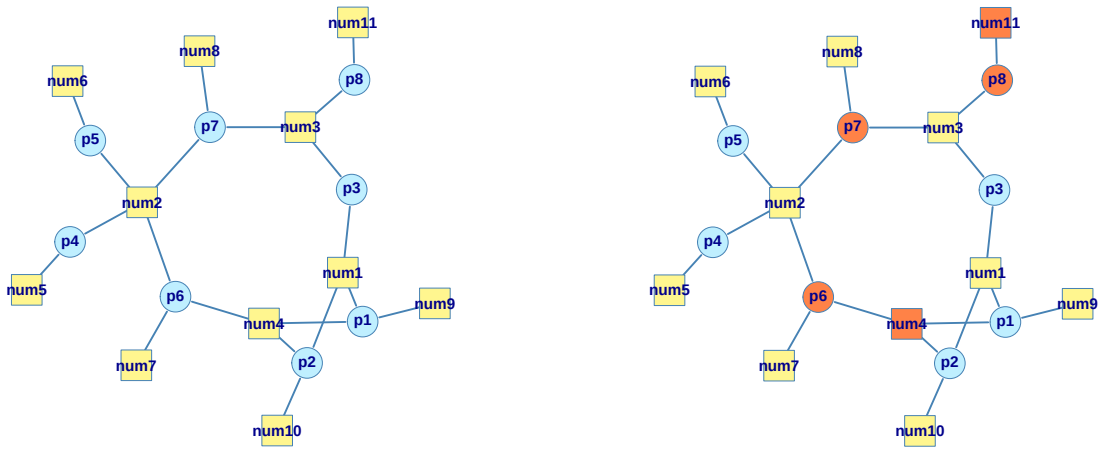


FIGURE 1. Plotting the dependency graph of a set of edit rules. The randomization seed is set prior to plotting to ensure that the layout algorithm yields comparable results in the left and right panel. In the left panel the dependency graph of E is plotted, with squares representing edit rules and circles representing variables. An edge indicates that a variable occurs in a certain edit. In the right panel, violated edits and the solution to the corresponding error localization problem are colored red. To plot dependency graphs, editrules utilizes the igraph0 package [Csardi and Nepusz, 2006].

18. In the left panel of Figure 1 the plot method is called on an editset without further options. The result is a dependency graph with squares representing rules and circles representing variables. Edges indicate variable occurrence in the rules. In the code of the right panel, we first determine which edits are violated by the second record in the dataset. Next, we collect the corresponding solution to the error localization problem. By passing both values to the plot function, the violated edits (num4 and num11) are colored red. The solution to the error localization problem (p_6, p_7 and p_8) is colored as well. The visualisation clearly illustrates that in certain cases variables that do not occur in violated edits need to be adapted as well.

19. Error localization problems need not have a unique solution. The localizeErrors function keeps track of number of equivalent solutions (*degeneracy*) and draws a random solution when there is more than one option. The number of solutions for each record is recorded in the status attribute of an errorLocation object, but we can get a quick overview using summary.

```
> summary(el)

Summary of  errorLocation  object, generated by mark at Wed Jul  4 19:21:35 2012
by calling checkDatamodel.editmatrix(G, dat, weight) localizeErrors(E,
    dat3$corrected)

Results:
```

```
       err.per.var err.per.rec weight degeneracy   user system elapsed
min              2           0      0          1  0.008  0.000   0.010
median          54           0      0          1  0.012  0.000   0.012
max            105           3      4          3  0.080  0.004   0.081
total          369         369    474       1927 26.133  0.040  26.069
0 records exceeded maximum search time (0%)
0 records did not yield a solution (0%)
```

The summary shows the minimum, maximum and median of several counts, including the number of times a variable was pointed out as erroneous, the number of erroneous fields per record, and the number of equivalent solutions per record. Observe that the minimum number of equivalent solutions equals one. This is to be expected since our dataset has a number of fully valid records, for which the solution to the error localization problem is unique and trivial (doesn't change anything). For logging purposes, the summary also shows a sequence of calls to high-level functions used in error localization. The function checkDatamodel was not called explicitly by us, but is utilized by localizeErrors to quickly localize variables violating range edits.

V. DEDUCTIVE IMPUTATION

20. After error localization, the erroneous fields have to be replaced with valid values. In some cases, unique imputations may be derived based on the valid values in the record and the edit rules. In those cases no statistical modeling is needed, and assumption-free imputation is possible. The deduImpute function of the deducorrect package performs such imputations for numerical and categorical data.

```
> dat4 <- deduImpute(E, dat = dat3$corrected, adapt = el$adapt, checkFeasibility = FALSE)
```

Here, the optional argument adapt is passed to tell deduImpute which fields were deemed erroneous by localizeErrors. The process of deductive imputation can be accelerated by leaving out certain feasibility checks that become unnecessary when error localization algorithm has been run before.

21. After deductive imputation, every field that must be adapted according to the error localizer is either imputed or emptied (NA). Since some fields in the dataset are now empty, it is not possible to check each edit for every record. However, none of the edits that can be checked are violated anymore:

```
> sum(violatedEdits(E, dat4$corrected), na.rm = TRUE)

[1] 0
```

Some further investigation shows that the deductive imputation step yielded 10 imputations, leaving 359 fields (2.5%) to be imputed in 127 records. The number of fully consistent records has increased again from from 1677 (92.5%) to 1686 (93%). In a following step, restricted imputation based on statistical assumptions can be applied to generate a fully consistent dataset.

VI. DISCUSSION AND CONCLUSION

22. We pointed out several capabilities of our R extension packages editrules and deducorrect. Capabilities not shown or mentioned here include the processing of numerical or mixed data and the possibility of including conditional edits. Also, the editrules package has been set up as a toolbox for edit

rule management and manipulation. Some important unmentioned manipulations include feasibility checking and variable elimination, implicit edit derivation for categorical edits, value substitution, and block decomposition. To facilitate edit management, we have implemented methods to convert edits from and to text which allows one to store edits after manipulations using R's standard I/O capabilities.

23. All methods discussed in this paper can be applied to data using only the assumptions that are fixed in the edit rules. The only statistical assumption is made when localizing errors, where it is assumed that errors are made randomly across (possibly weighted) fields in the record. The example presented here has a high data quality to begin with and we gain just over one percent point of fully clean records. In a similar study, using 1347 records of 86 variables, we found an increase from 50% fully clean records at the start to 62% at the end. This shows that deductive methods have great potential to increase data quality without making assumptions apart from the knowledge locked in the edit rules. Out future work will focus on imputation and value adaptation under edit restrictions.

VII. REFERENCES TO THEORY AND ALGORITHMS

24. Much of the theory and algorithms behind the implementations of deducorrect and editrules can be found in De Waal et al. [2011]. The algorithms for deductive correction were originally developed by Scholtus [2008, 2009, 2011] and were slightly generalized for implementation in R by Van der Loo et al. [2011]. Methods for deductive imputation of numerical data were developed by Pannekoek [2006] and implemented by Van der Loo and De Jonge [2011b]. Deductive imputation of categorical data was developed by Scholtus, but first published in De Waal et al. [2011] (Chapter 9). The error localization algorithm is based on methods of De Waal and Quere [2003]. The implementation for numerical data is described in De Jonge and Van der Loo [2011], the implementation for categorical and mixed data are described in Van der Loo and De Jonge [2011a] and Van der Loo and De Jonge [2012]. Much attention has been payed to implement efficient variable elimination techniques. A publication on algorithms for variable elimination in categorical data was recently submitted [Van der Loo, 2012].

References

G. Csardi and T. Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006. URL http://igraph.sf.net.

J. Fox. Aspects of the social organization and trajectory of the R project. *The R Journal*, 1:5–13, 2009.

E. de Jonge and M. van der Loo. Manipulation of linear edits and error localization with the editrules package. Technical Report 201120, Statistics Netherlands, The Hague, 2011.

E. de Jonge and M. van der Loo. *editrules: R package for parsing and manipulating of edit rules and error localization.*, 2012. URL http://cran.r-project.org/package=editrules. R package version 2.5-0.

M. van der Loo and E. de Jonge. Manipulation of categorical data edits and error localization with the editrules package. Technical Report 201129, Statistics Netherlands, 2011a.

M. van der Loo and E. de Jonge. Deductive imputation with the deducorrect package. Technical Report 201126, Statistics Netherlands, 2011b. In press.

M. van der Loo, E. de Jonge, and S. Scholtus. Correction of rounding, typing and sign errors with the deducorrect package. Technical Report 201119, Statistics Netherlands, 2011. R package version 1.0-0.

M. van der Loo, E. de Jonge, and S. Scholtus. *deducorrect: Deductive correction and imputation of restricted data*, 2012. URL http://CRAN.R-project.org/package=deducorrect. R package version 1.2-0.

M.P.J. van der Loo. Variable elimination and edit generation with a flavour of semigroup algebra. *submitted*, 2012.

J. Pannekoek. Regression imputation with linear equality constraints on the variables. In *UNECE Work session on statistical data editing in Bonn*, 2006. http://www.unece.org/stats/documents/2006.09.sde.html.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2012. URL http://www.R-project.org/. ISBN 3-900051-07-0.

S. Scholtus. Algorithms for correcting some obvious inconsistencies and rounding errors in business survey data. Technical Report 08015, Statistics Netherlands, Den Haag, 2008.

S. Scholtus. Automatic correction of simple typing errors in numerical data with balance edits. Technical Report 09046, Statistics Netherlands, Den Haag, 2009. The papers are available in the inst/doc directory of the R package or via the website of Statistics Netherlands.

S. Scholtus. Algorithms for correcting sign errors and rounding errors in business survey data. *Journal of Official Statistics*, 27:467–490, 2011.

Statistics Netherlands, 2010-2012. The methodological handbooks of SN contain a description of validated statistical methods that are in use at Statistics Netherlands for collecting, processing, analysing and publishing statistical information. English versions of the handbook can be downloaded from http://www.cbs.nl/en-GB/menu/methoden/gevalideerde-methoden/default.htm.

M. van der Loo and E. de Jonge. Manipulation of conditional restrictions and error localization with the editrules package. Technical Report 201215, Statistics Netherlands, 2012.

T. de Waal and R. Quere. A fast and simple algorithm for automatic editing of mixed data. *Journal of Official Statistics*, 19:383–402, 2003.

T. de Waal, J. Pannekoek, and S. Scholtus. *Handbook of statistical data editing and imputation*. Wiley handbooks in survey methodology. John Wiley & Sons, Hoboken, New Jersey, 2011. ISBN 978-470-54280-4.

Y. Xie. *knitr: A general-purpose package for dynamic report generation in R*, 2012. URL http://CRAN.R-project.org/package=knitr. R package version 0.6.3.