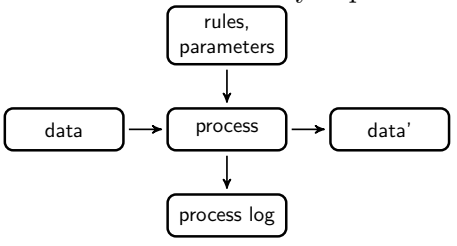
Systematic data cleaning using R

To allow for blinded review:
do NOT indicate author information or affiliation

Keywords: *data editing, data validation, automation, reproducibility, monitoring*

1 Introduction

Over the last decades there have been several attempts to set up frameworks for statistical data processing and statistical data cleaning [4, 5]. One of the key notions is that a data cleaning procedure can be decomposed into a sequence of fundamental steps, where each step is controlled by external information defined by experts.



In this model, some imperfect data set data is input for a processing step. The processing step is generally parameterized by two types of metadata. First, a set of validation rules (rules) describe the desired ultimate state of the dataset. Second, there are parameters that control the details of process. For example, if the processing step concerns an imputation procedure an imputation model specification may enter as a parameter. The `process` then yields an improved dataset `data'`, while keeping a log of its activities in a `process log` that can be used for monitoring.
In the following sections we demonstrate how a set of tools build in R[6] can be flexibly combined to follow precisely this model. A detailed overview of methodology and R packages can also be found in a recent publication of the authors [11].

2 Validation rules

The core of the tool set presented here is formed by R package `validate`[12]. This package offers an infrastructure for validation rule management, including CRUD (create, read, update, delete) functionality and the ability to confront data with a validation rule set. The package treats rule sets as first-class citizens that can be programmatically manipulated. In its implementation, it closely follows the definitions and methodology described in the ESS handbook on validation[2]. In what follows, we will use a few columns of the `retailers` data set that comes with the package as a running example.

```
> library(validate)
> data(retailers)
> # we select a few columns for brevity and add a unique key
> dat <- cbind( id=sprintf("RET%02d", seq_len(nrow(retailers)))
+     , retailers[3:6])
> head(dat, 3)
```

```
     id staff turnover other.rev total.rev
1 RET01    75       NA        NA      1130
2 RET02     9     1607        NA      1607
3 RET03    NA     6886       -33      6919
```

Validation rules can be defined in a text file, in a table (as in a data base), in structured `YAML` file[13] or on the command line, which is what we use in the current example.

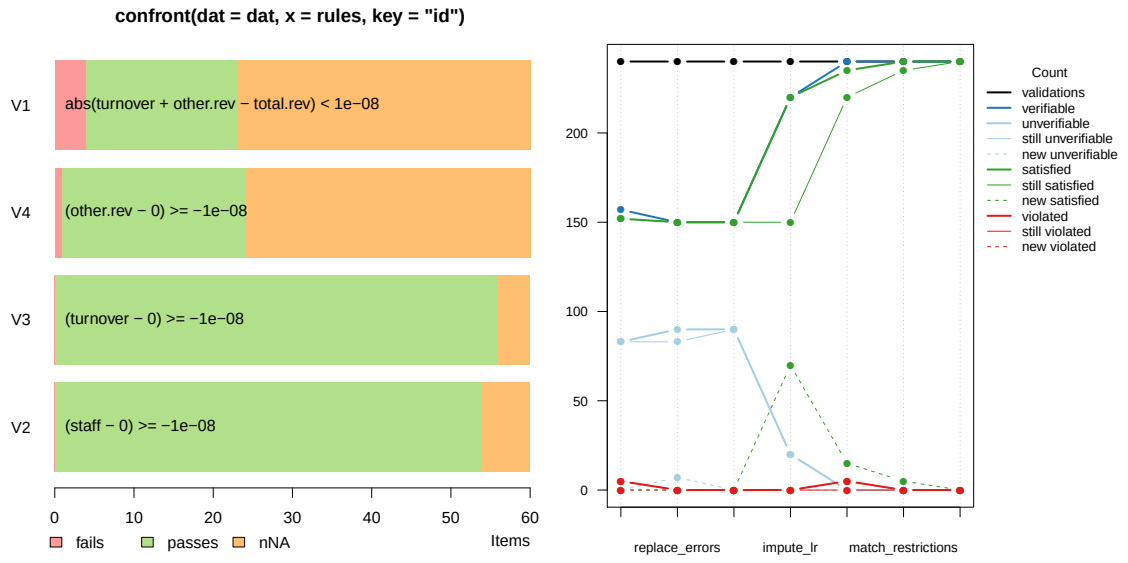


Figure 1: Left: quality of the initial data set, the number of items corresponds to the number of validated items (here: records) for each rule. Right: progress of the various components shown in Figure 2 across data processing steps.

```
> rules <- validator( turnover + other.rev == total.rev
+     , staff >= 0, turnover >= 0, other.rev >= 0
+ )
```

Data can be confronted with these rules using `confront`. The results can be inspected, for example by plotting a graphical summary (see Figure 1).

```
> results <- confront(dat, rules, key="id")
> plot(results)
```

The `results` object contains all results and can be inspected in various ways. Providing the name of the key column at confrontation time ensures that all validation results can be easily related to the original data, for example by turning the results into a standard R data frame

```
> head(as.data.frame(results),3)
```

```
     id name value                                        expression
1 RET01   V1    NA abs(turnover + other.rev - total.rev) < 1e-08
2 RET02   V1    NA abs(turnover + other.rev - total.rev) < 1e-08
3 RET03   V1 FALSE abs(turnover + other.rev - total.rev) < 1e-08
```

Here, we see that `validate` adapted the original rules to account for machine rounding. Also, note that in the first and second record the validation result is missing since in these cases the variables `turnover` and/or `other.rev` are missing. Both the tolerance for machine rounding and behaviour under missing values can be customized by the user.
Now that we identified quality demands on the output data and measured the quality of the initial data set we are ready to start cleaning it.

3 Cleaning data

As a first step we use the `errorlocate` package[1] to remove erroneous values (replace them with `NA`) based on Fellegi and Holt's principle[3]

```
> library(errorlocate)
> dat1 <- replace_errors(dat, rules, value=NA)
```

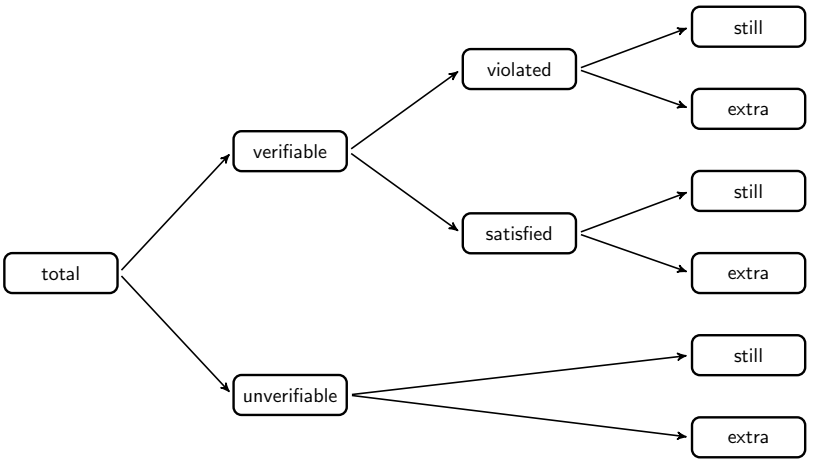


Figure 2: Decomposing the difference in validation results before and after data processing.

Next, we use a sequence of imputation models to estimate values for the missing data. Imputation happens without regard of the validation rules. Hence, we first locate the missing values, next we use a deductive imputation procedure that is based on substituting observed values in the rule set. If simplification of the rule set yields unique values for some of the variables these are then imputed. Next, we use the missForest algorithm[7, 8] to impute any remaining missings. After this we use the successive projection algorithm[14] as implemented in the `rspa` package[10] to adjust the imputed values minimally so all rules are satisfied.

```
> library(rspa); library(deductive); library(simputation)
> A <- is.na(dat1)
> dat2 <- deductive::impute_lr(dat1, rules)
> dat3 <- simputation::impute_mf(dat2, .-id ~ .-id)

> dat4 <- rspa::match_restrictions(dat3, rules, adjust=A)
> all( confront(dat4, rules, lin.eq.eps=0.01) )

[1] TRUE
```

Finally, we check whether all rules are satisfied within a tolerance of 0.01 (the default tolerance used in the `rspa` package). The `TRUE` result indicates shows that this is indeed the case.

4 Monitoring output

The previous steps show that it is possible to produce an output data set that matches our restrictions defined in the `rules` object. However it does not give much insight into the influence of each step. This omission is filled with the logging framework provided by the `lumberjack` package[9]. This package implements a special function composition (pipe) operator that allows for logging differences between data transformation steps. In the example below, we use the `lbj_rules` logger provided by the `validate` package. This logger compares two consecutive versions of a dataset by decomposing the number of validations according to the scheme shown in Figure 2

```
> library(lumberjack)
> voptions(rules, lin.eq.eps=0.01)
> logger <- validate::lbj_rules(rules)
> output <- dat %L>%
+   start_log(logger) %L>%
+   replace_errors(rules, value=NA) %L>%
+   tag_missing() %L>%
+   impute_lr(rules) %L>%
```

```
+   impute_mf(. - id ~ . - id) %L>%
+   match_restrictions(rules) %L>%
+   dump_log(file="log.csv")
```

The lumberjack operator `%L>%` calls the function on its right-hand-side using the data on its left-hand-side as first input argument. If a logger is activated (using `start_logger`, then this logger is used to compare the difference between data before and after the function call. Logging information is stored in the logger and can be written to file using `dump_log`. In this case, the logger also has a plot method allowing one to quickly generate an impression of the effect of various data processing steps. The result is shown in the right panel of Figure 1.

```
> logger$plot()
```

5 Conclusion

We demonstrated a few features of a suite of R packages for data cleaning. The packages implement a modular approach to data cleaning that allows for a clear separation of concerns between programmer/data analyst and domain experts. Restrictions on output data (validation rules) are defined completely separate from the main process flow. We also demonstrated a powerful logging framework that is independent of the chosen data processing steps.

References

[1] de Jonge, E. and M. van der Loo (2016). *errorlocate: Locate Errors with Validation Rules*. R package version 0.1.2.

[2] Di Zio, M., N. Fursova, T. Gelsema, S. Giessing, U. Guarnera, J. Ptrauskiene, L. Q. von Kalben, M. Scanu, K. ten Bosch, M. van der Loo, and K. Walsdorfe (2015). Methodology for data validation. Technical Report Deliverable No. 2, ESSNet on validation.

[3] Fellegi, I. P. and D. Holt (1976). A systematic approach to automatic edit and imputation. *Journal of the American Statistical association 71*(353), 17–35.

[4] Oinonen, S., P. Ollila, M. Pyy-Martikainen, E. Gros, M. D. Zio, U. Guarnera, O. Luzi, L.-C. Zhang, and J. Pannekoek (2015). *Generic Statistical Data Editing Models*. High-Level Group for the modernisation of official statistics.

[5] Pannekoek, J., S. Scholtus, and M. van der Loo (2013). Automated and manual data editing: a view on process design and methodology. *Journal of Official Statistics 29*, 511–537.

[6] R Core Team (2018). *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing.

[7] Stekhoven, D. J. and P. Bühlmann (2012). Missforest–non-parametric missing value imputation for mixed-type data. *Bioinformatics 28*(1), 112–118.

[8] van der Loo, M. (2017). *simputation: Simple Imputation*. R package version 0.2.2.

[9] van der Loo, M. (2018a). *lumberjack: Track Changes in Data*. R package version 0.3.0.

[10] van der Loo, M. (2018b). *rspa: Adapt Numerical Records to Fit (in)Equality Restrictions*. R package version 0.2.2.

[11] Van der Loo, M. and E. de Jonge (2018). *Statistical data cleaning with applications in R*. John Wiley & Sons.

[12] van der Loo, M. and E. de Jonge (2018). *validate: Data Validation Infrastructure*. R package version 0.2.6.

[13] yaml.org (2015). YAML Aint Markup Language. http://yaml.org/ (accessed 2015-08-13).

[14] Zhang, L.-C. and J. Pannekoek (2015). Optimal adjustments for inconsistency in imputed data. *Survey methodology 41*(1), 127–144.