Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *doi: 10.18637/jss.v000.i00*

cleanR: Your Personal Maid for Cleaning Datasets in R

Claus Thorn Ekstrøm
Biostatistics
Department of Public Health
University of Copenhagen

Anne Helby Petersen
Biostatistics
Department of Public Health
University of Copenhagen

Abstract

Data cleaning and validation is the first step in any data analysis since the validity of the conclusions from the analysis hinged on the quality of the input data. Ideally, a human investigator should go through each variable in the dataset and look for potential errors — both in input values and coding.

We describe an R package which implements an extensive and customizeable suite of checks to be applied to the variables in a dataset in order to identify potential problems in the corresponding variables. The typical output is a stand-alone document that summarizes the variables and lists potential errors. The results are typically presented in a stand-alone document that could be perused by an investigator with an understanding of the variables in the data and the experimental design.

The **cleanR** package is designed to be easily extended with custom user-created checks that are relevant in particular situations.

Keywords: data cleaning, quality control, R.

1. Introduction

Statisticians and data analysts spend a large portion of their time on data cleaning and on data wrangling. Packages such as **data.table**, and **plyr** have made data wrangling a lot easier in R, but there are only a few options available for automated data cleaning.

Data cleaning is a time consuming endeavour and it inherently requires human interaction since every dataset is different and the variables in the dataset can only be understood in the proper context of the experiment. While each dataset is different and requires unique attention there are often a number of similar tasks that are undertaken as part of the data cleaning and quality control process. This is especially true when data are received repeatedly

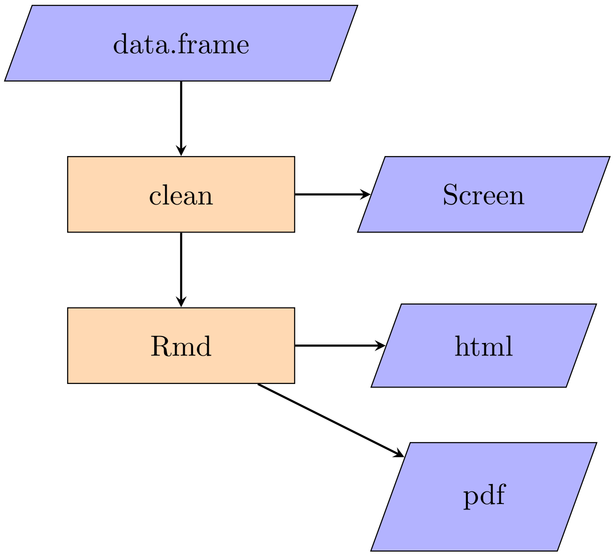


Figure 1: The process of cleaning data frames using the **cleanR** package. A series of check is applied to each variable in a `data.frame` and a summary of the result is either printed to the screen, or an R markdown file is produced which is subsequently rendered.

from the same source as data providers tend to use a stable setup (and hence introduce the same set of potential mistakes that need to be identified and corrected).

In many situations these errors are discovered in the process of the data analysis (e.g., a categorical variable with numeric labels is wrongly classified as a numeric variable), but in other cases a human with knowledge about the data context area is needed to identify possible mistakes in the data (e.g., if there are 4 categories for a variable that should only have 3). It is necessary to summarize information — both numerically and graphically — about each variable in order for an investigator to detect possible errors, and help raise flags of warning to draw the attention of investigator for the situations where there *might* be a error.

The **cleanR** package supports the automated checking of errors in a dataset and produces an output document with detailed information about each variable that can be The process is illustrated in Figure 3. A dataset is cleaned and an R markdown file is produced and possibly rendered into html or pdf.

The manusccript is organized as follows. Section 2 presents a worked example and tutorial on how to use the **cleanR** package to create a summary of potential errors. Section 3 illustrates the methodology of modeling proportional data using simplex generalized regression. The generalized estimating equations for longitudinal proportional outcomes are given in Section 4. Then we address model diagnostics in Section 5. Section 6 presents the details of the simplexreg package. Section 7 further conducts analyses based on the simplex distribution in R with real data sets. Finally, plans for extending the package are described in Section 8.

2. Checking a dataset for errors

In **cleanR** the clean function is the primary workhorse and it should be an ideal starting place

First we

```
> library(cleanR)
> data(diamonds)
> head(diamonds)

  carat       cut color clarity depth table price    x    y    z
1  0.23     Ideal     E     SI2  61.5    55   326 3.95 3.98 2.43
2  0.21   Premium     E     SI1  59.8    61   326 3.89 3.84 2.31
3  0.23      Good     E     VS1  56.9    65   327 4.05 4.07 2.31
4  0.29   Premium     I     VS2  62.4    58   334 4.20 4.23 2.63
5  0.31      Good     J     SI2  63.3    58   335 4.34 4.35 2.75
6  0.24 Very Good     J    VVS2  62.8    57   336 3.94 3.96 2.48

> data(testData)
> #head(testData)
```

To produce summary data-cleaning document

```
> clean(testData, replace=TRUE, openResult=FALSE)
```

By default, a ... is produced and the result is shown in Figure 2.
Arguments

3. The structure of cleanR

Bør vise DF -> prechecks -> for each variable do this ...

4. Extending cleanR by adding custom error checks

Lav en situation svarende til eksemplet

```
> characterFoo <- function(v) {
+     if (substr(substitute(v), 1, 1) == "_") {
+         out <- list(problem=TRUE, message="Note that the variable name begins with \\_")
+     } else out <- list(problem=FALSE, message="")
+     out
+ }
> class(characterFoo) <- "checkFunction"
> attr(characterFoo, "description") <- "I really hate underscores"
> #clean(testData, characterChecks=c(defaultCharacterChecks(), "characterFoo"))
>
```

Lav også et eksempel med rangecheck.

Data cleaning summary

The data frame examined has the following dimensions:

Feature	Result
Number of rows	15
Number of variables	14

Checks performed

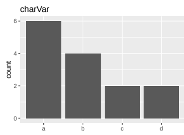
The following variable checks were performed, depending on the data type of each variable:

	character	factor	labelled	numeric	integer	logical
Identify miscoded missing values	×	×	×	×	×	
Identify prefixed and suffixed whitespace	×	×	×			
Identify levels with < 6 obs.	×	×				
Identify case issues	×	×				
Identify outliers				×	×	

Variable list

charVar

Feature	Result
Variable type	factor
No. missing obs.	1 (6.67 %)
No. unique values	4

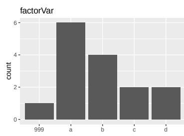


- Note that the following levels have at most five observations: "b", "c", "d"

---

factorVar

Feature	Result
Variable type	factor
No. missing obs.	0 (0 %)
No. unique values	5



- The following suspected missing value codes enter as regular values: "999"

1

Figure 2: Example output from running XXXX on the testData dataset. First a summary of the full dataset,

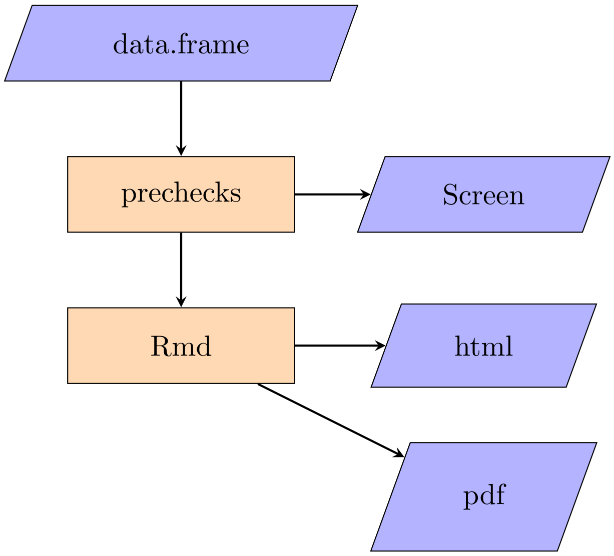


Figure 3: The process of cleaning data frames using the **cleanR** package. A series of check is applied to each variable in a `data.frame` and a summary of the result is either printed to the screen, or an R markdown file is produced which is subsequently rendered.

5. Customizing the cleanR output

Though the discussion in the above paints a picture of cleanR as a user-friendly package which requires practically no knowledge of R, one should not be mistaken to think that it is not customizable. In fact, the main function of cleanR, Rclean, is mainly a tool for formatting the results from various checking-, summary- and visualizaion functions as described... blabla. Thus, the actual work underlying a cleanR output file can be anything or nothing - depending on the arguments given to Rclean. This section consists of three parts. We commence this section with an overview of how contents are controlled in RcleanR in terms of what cleaning steps are performed. Secondly, we turn to the possibilities of customizing what the final cleanR file looks like. Lastly, we go through an example of how to put all these customization options together in practice.

5.1. Controlling contents

By the *contents* of the cleanR output, we refer to every step that actually involves a function being called on variables from the dataset. There are three stages in which this occurs (with ref. to figure/flowchart?):

1. In the precheck functions
2. In the summarize/visualize/check (SVC) step
3. In the multivariate visuzalizations Or whatever, in the stuff that we have not yet implemented. I'm pretending like this stage doesn't exist below.

Each of these stages are controllable using appropriate function parameters in Rclean. In the above, we presented the default RcleanR settings and how to tweak them into providing a slightly different data cleaning outputs. However, if for instance the dataset at hand requires completely different visualizations, more control is needed. RcleanR uses three different types of functions for performing all stages in the above, namely RsummaryFunctions, RvisualFunctions and RcheckFunctions. They each have distinct input-output structures and most instances of the functions are build as RS3 generics with methods for different data classes. By understanding how to construct proper RsummaryFunctions, RvisualFunctions and RcheckFunctions, the entire contents of the RcleanR output is at your hands. Therefore, this section is dedicated to introducing each in turn.

Something like a figure that gives an overview of all summaryFunctions, ... available in standard cleanR, including default settings and some kind of description of where they are called (in precheck or SVC step). Also something about what datatypes they can each be called on. Also, more text here about the relationship between the two (three) stages and the three function types.

Writing a summaryFunction

Though RcleanR provides a special class for RsummaryFunctions, there really is nothing special about these functions: They are nothing but regular functions with a certain input/output-structure. Specifically, they all follow the template below:

```
mySummaryFunction <- function(v) {
        res <- [result of whatever summary we are doing]
        list(feature = "Feature name", result = res)
}
```

and we recommend furthermore adding them to the overview of all summary functions by converting them to proper RsummaryFunction objects:

```
mySummaryFunction <- summaryFunction(mySummaryFunction,
        description = "Some text describing what your summaryFunction does")
```

which adds the new function to the output of RallSummaryFunctions(). Note that Rv is a vector and that Rres should be either a character string or something that will be printed as one. In other words, e.g. integers are allowed, but matrices are not. Though a lot of different things can go into the RsummaryFunction template, we recommend only using them for summarizing the features of a variable, and leaving tests and checks for the RcheckFunctions (presented below).

Writing a visualFunction

RvisualFunctions are the functions that produce the figures of a RcleanR output document. Writing a visual function is slightly more complicated than writing a summary function. This follows from the fact that visualFunctions need to be able to output standalone code for plots in order for Rclean to build standalone rmarkdown files. We recommend using the following structure:

```
myVisualFunction <- function(v, vnam, doEval) {
        thisCall <- call("[the name of the function used to produce the plot]",
                v, [additional arguments to the plotting function])
        if (doEval) {
                return(eval(thisCall))
        } else return(deparse(thisCall)
}

myVisualFunction <- visualFunction(myVisualFunction,
        description = "Some text describing your visualFunction")
```

In this function, Rv is the variable to be visualized, Rvnam is its name (which should generally be passed to Rtitle or Rmain arguments in plotting functions) and RdoEval controls whether the output is a plot (if RTRUE) or a character string of standalone code for producing a plot (if RFALSE). The latter RdoEval setting is not strictly necessary for its use in Rclean, but it makes it easier to assess what visualization options are available. In either case, it should be noted that all the parameters listed above, Rv, Rvnam and RdoEval, are mandatory, so they should be left as is (as are?), even if you do not want to use them. As with RsummaryFunctions, an overview of all available RvisualFunctions in the environment can be obtained by calling

```
allVisualFunctions()
```

and by calling

```
allVisual(v, vnam, output = "html")
```

an overview of all plotting options applied on Rv is produced and opened as a html document for easy comparison. Should we mention the side effect of producing .rmd and .html files on disc?

Writing a checkFunction

The last, but also most important, RcleanR function type is the RcheckFunction. These are the functions that flag issues in the data and control the flow of the overall data cleaning process in the precheck stage. A RcheckFunction can be written using the following template:

```
myCheckFunction <- function(v) {
        [do your check]
        problem <- [is there a problem? TRUE/FALSE]
        problemValues <- [vector of values in v that are problematic]
        problemStatus <- list(problem = problem, problemValues = problemValues)

        problemMessage <- "[The message that should be printed prior to listing
                        problem values in the cleanR output]"

        outMessage <- messageGenerator(list(problem = problem,
                problemValues = problemValues, message = problemMessage))
```

```
        list(problem = problem, message = message) #problem is TRUE/FALSE,
                # message is a text string
}
myCheckFunction <- checkFunction(myCheckFunction,
        description = "[A description of your checkFunction]")
```

Only the input parameter (Rv) and the output format strictly has to follow this structure. However, we recommend using RmessageGenerator for consistent styling of all RcheckFunction messages. This function simply pastes together the RproblemMessage and the RproblemValues, with the latter being quoted and sorted alphabetically. Note that printing quotes in rmarkdown requires an extensive amount of character escaping, so opting for RmessageGenerator really is the easiest solution.

While the descriptions of RsummaryFunctions and RvisualFunctions are only for internal use in the RallSummaryFunctions() and RallVisualFunctions() outputs, respectively, RcheckFunction descriptions are actually visible in the RcleanR output document. These are the brief descriptions presented in Part 1 eh? Is this clear? Whatever we call this section in the above in the output document. If a RcheckFunction does not have a description (for instance, if it is just a regular Rfunction using the RcheckFunction input/output-structure), the function name will be printed instead of the description.

5.2. Controlling formatting

Something about how to control what is printed where. Should be much briefer than the stuff in the above. This section could maybe include the following points:

1. Controlling "part 1", i.e. the stuff that is printed before the loop is started.
2. The effect of prechecks.
3. A table containing other parameters that control formatting/stuff like this, e.g. twoCol, checkDetails (if we ever implement it), listChecks...

5.3. Controlling flow?

I feel like we maybe need yet another section about customization. We have not yet described the following (rather important(?)) features:

- The mode argument - controlling which SVC steps are performed
- smartNum - chooosing whether or not numeric/integer variables with only a few levels are treated as factors
- standAlone - Describing how it is possible to produce .Rmd files that can be included in other .Rmd files using the child option and standAlone = F (removing the YAML preamble (and maybe also the cleanR commerical? Are we that nice?)).

5.4. A worked example

Maybe use the examples from the documentation to construct new summaryFunction etc. and add them to the default options.

6. To-dos in the code

... introduced by discrepancy between what I say we have done and what we have actually done:

- summaryFunction stuff:
  - Make constructor function
  - Make allSummaryFunctions()
  - Change class of all summary (and description) functions into summaryFunction (possibly letting description functions be a subclass?)
  - Maybe: Make clean check if summary functions are really summaryFunctions?
- visualFunction stuff:
  - Make constructor function
  - Make allVisualFunctions()
  - Change class of all visual functions into visualFunction
  - Maybe: Make clean check if visual functions are really visual functions?
- checkFunction stuff:
  - Make allCheckFunctions()
  - Work on messageGenerator() to make it better suited for being an exported function + export it
  - Make allVisual()

7. Using the online web-app

asd asd

Affiliation:

Claus Thorn Ekstrøm
Biostatistics, Department of Public Health
University of Copenhagen
Denmark
E-mail: ekstrom@sund.ku.dk
URL: http://eeecon.uibk.ac.at/~zeileis/

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
MMMMMM YYYY, Volume VV, Issue II *Submitted:* yyyy-mm-dd
doi:10.18637/jss.v000.i00 *Accepted:* yyyy-mm-dd