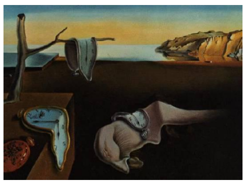

Journal of Statistical Software

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

Title

Firstname Lastname
Affiliation

Abstract

—!!!—an abstract is required—!!!—

Keywords: —!!!—at least one keyword is required—!!!—.

1. Introduction

2. Checking a dataset for errors

3. 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, `clean`, 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 `clean`. This section consists of three parts. We commence this section with an overview of how contents are controlled in `cleanR` 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.

3.1. 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 `clean`. In the above, we presented the default `cleanR` 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. `cleanR` uses three different types of functions for performing all stages in the above, namely `summaryFunctions`, `visualFunctions` and `checkFunctions`. They each have distinct input-output structures and most instances of the functions are build as `S3` generics with methods for different data classes. By understanding how to construct proper `summaryFunctions`, `visualFunctions` and `checkFunctions`, the entire contents of the `cleanR` 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 `cleanR` provides a special class for `summaryFunctions`, 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 `summaryFunction` objects:

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

which adds the new function to the output of `allSummaryFunctions()`. Note that `v` is a vector and that `res` 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 `summaryFunction` template, we recommend only using them for summarizing the features of a variable, and leaving tests and checks for the `checkFunctions` (presented below).

Writing a visualFunction

`visualFunctions` are the functions that produce the figures of a `cleanR` 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 `clean` 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, `v` is the variable to be visualized, `vnam` is its name (which should generally be passed to `title` or `main` arguments in plotting functions) and `doEval` controls whether the output is a plot (if `TRUE`) or a character string of standalone code for producing a plot (if `FALSE`). The latter `doEval` setting is not strictly necessary for its use in `clean`, 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, `v`, `vnam` and `doEval`, are mandatory, so they should be left as is (as are?), even if you do not want to use them. As with `summaryFunctions`, an overview of all available `visualFunctions` 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 `v` 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, `cleanR` function type is the `checkFunction`. 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 `checkFunction` 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 (`v`) and the output format strictly has to follow this structure. However, we recommend using `messageGenerator` for consistent styling of all `checkFunction` messages. This function simply pastes together the `problemMessage` and the `problemValues`, with the latter being quoted and sorted alphabetically. Note that printing quotes in rmarkdown requires an extensive amount of character escaping, so opting for `messageGenerator` really is the easiest solution.

While the descriptions of `summaryFunctions` and `visualFunctions` are only for internal use in the `allSummaryFunctions()` and `allVisualFunctions()` outputs, respectively, `checkFunction` descriptions are actually visible in the `cleanR` 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 `checkFunction` does not have a description (for instance, if it is just a regular `function` using the `checkFunction` input/output-structure), the function name will be printed instead of the description.

3.2. 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...

3.3. 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?)).

3.4. A worked example

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

4. 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()

Affiliation:

Firstname Lastname
Affiliation
Address, Country
E-mail: name@address
URL: http://link/to/webpage/

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