Journal of Statistical Software

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

dataMaid: your assistant for data documentation in R

Anne Helby Petersen
Biostatistics
Department of Public Health
University of Copenhagen

Claus Thorn Ekstrøm
Biostatistics
Department of Public Health
University of Copenhagen

Abstract

Data cleaning and -validation are important steps in any data analysis, as the validity of the conclusions from the analysis hinges on the quality of the input data. Mistakes in the data can arise for any number of reasons, including erroneous codings, malfunctioning measurement equipment, and inconsistent data generation manuals. Ideally, a human investigator should go through each variable in the dataset and look for potential errors — both in input values and codings — but that process can be very time-consuming, expensive and error-prone in itself.

We describe an R package, **dataMaid**, which implements an extensive and customizeable suite of quality assessment tools that can be applied to a dataset in order to identify potential problems in its variables. The results are presented in an auto-generated, non-technical, stand-alone overview document intended to be perused by an investigator with an understanding of the variables in the data, but not necessarily knowledge of R. Thereby, **dataMaid** aids the dialogue between data analysts and field experts, while also providing easy documentation of reproducible data cleaning steps and data quality control. Moreover, the **dataMaid** solution changes the data cleaning process from the usual ad hoc approach to a systematic, well-documented endeavor. **dataMaid** also provides a suite of more typical R tools for interactive data quality assessment and -cleaning, where the data inspections are executed directly in the R console.

Keywords: data cleaning, quality control, R, data documentation.

1. Introduction

Though data cleaning might be regarded as a somewhat tedious activity, adequate data cleaning is crucial in any data analysis. With ever-growing dataset sizes and complexities,

statisticians and data analysts find themselves spending a large portion of their time on data cleaning and data wrangling. While a computer should generally not make unsupervised decisions on what should be done to potential errors in a dataset, it can still be an extremely useful tool in the data cleaning process. Some errors can be tracked down and flagged by a computer without further ado, while other types of errors need an analytic context in order to be identified. Even in this latter case, well-designed software can aid the process tremendously by providing the human investigator with the information needed for identifying issues.

But even when tools are available for identifying problems in a dataset, the activity of data cleaning still suffers from a challenge that has recieved increasing attention in the scientitic communities in the later years: Data cleaning is not very straight forward to document and therefore, reproducibility suffers. **dataMaid**'s most central purpose is to facilitate a data cleaning workflow where documentation is thoroughly integrated rather than a secondary add-on. This is accomplished by centering the data cleaning process around auto-generated data overview reports that summarize the contents of the current version of a dataset and flags potential problems that could be candidates for the next bullet on the cleaning agenda.

A number of R packages made for other pre-analysis tools are already available, including **janitor** (Firke 2016), **dplyr** (Wickham *et al.* 2017), **tidyr**(Wickham and Henry 2017), **data.table** (Dowle *et al.* 2016), **DataCombine** (Gandrud 2016) , **validate** (van der Loo and de Jonge 2016), **assertr** (Fischetti 2017), and **DataExplorer** (Cui 2016). These packages focus on different stages of the pre-analysis work, most of which are not really data cleaning. **janitor** provides tools for data import with a particular emphasis on the challenges of getting neat data frames from Microsoft Excel data files. **dplyr**, **tidyr**, **data.table** and **DataCombine** go a few steps further by providing a wide array of extremely powerful tools for data wrangling, including a number of particularly useful functions for merging and working with very large datasets. When it comes to actual data cleaning, however, the options are fewer. **validate** (and the similar packages **editrules** (de Jonge and van der Loo 2015) and **deducorrect** (van der Loo *et al.* 2015) from the same authors) does offer a few tools for identifying and auto-correcting errors in a dataset, but, as the name suggests, the focus is on internal validity rather than general data cleaning. In practice, this means that quite elegant tools for, e.g., linear restraints among the variables in a dataset can be applied, but looking for potentially miscoded missing values is not really feasible. The main difference between these two challenges is the direction in which the data is inspected: While linear constraints work observation-wise with no ambiguity, determining whether or not something is a miscoded missing value often requires knowledge about the full variable (e.g. range or data type), and thus it should be performed variable-wise. **validate** does not allow for user-defined extensions of the latter type, thereby limiting its data cleaning potential. Automatic data correction functions are also provided by **validate** which we consider to be quite a dangerous cocktail: all power is given to the the computer with no human supervision, and investigators are less likely to make an active, case-specific choice regarding the handling of the potential errors. Finally, no tools have been made to easily document exactly which checks and preliminary results were used in the data cleaning process. Something about **assertr** here.

All in all, the large role of data cleaning in any data analysts everyday endeavors is hardly matched in the amount of available R software solutions. In particular, few packages attempt to implement systematic, reproducible data cleaning. And while the available tools attempt to alleviate the ubiquitous ad hoc approach to data cleaning, they are primarily intended for the data savvy users and less so for the general researcher with a knowledge about a specific field

and the context of the available data. The **dataMaid** package (Petersen and Ekstrøm 2016) presented here tries to address this by providing a framework that both allows for extendable, systematic, reproducible data cleaning, and summarizing findings for researchers from other fields such that they can act as human experts when tracking down potential errors.

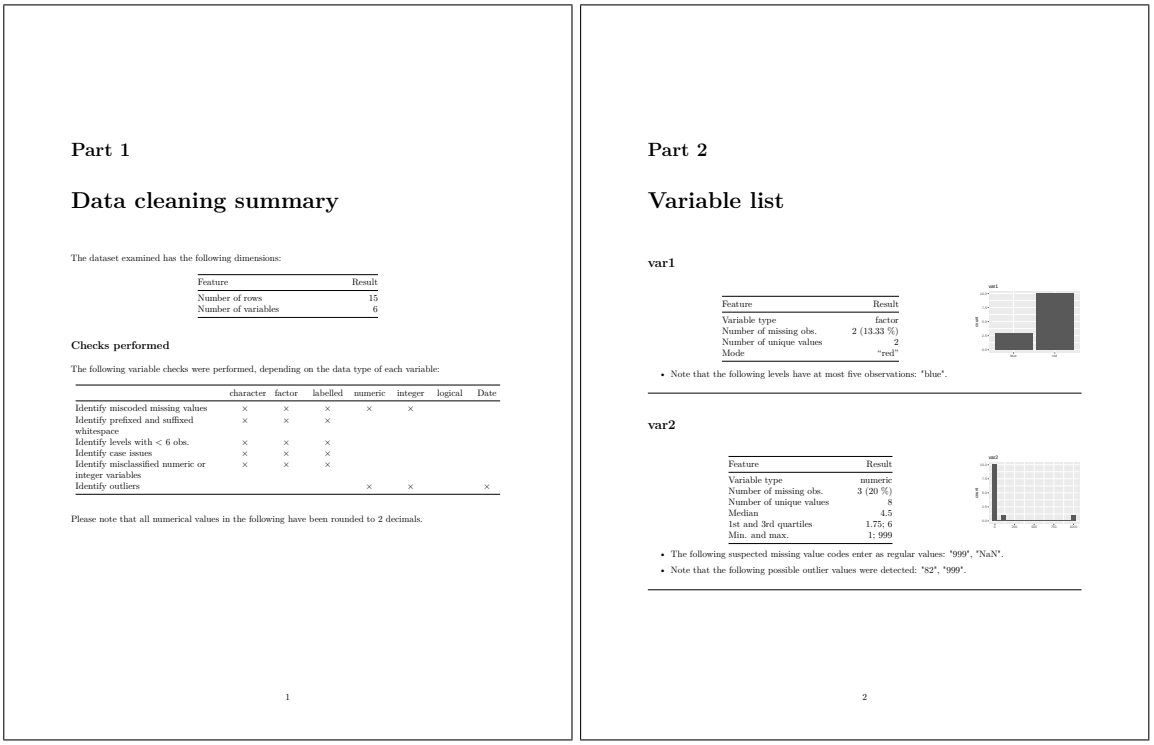
One last package that should be mentioned in this context is **DataExplorer** (Cui 2016). While this package does not address data cleaning issues *per se*, its general strategy is quite similar to that of **dataMaid** and to the paradigms presented below. This package provides a few simple, but practical, tools for exploratory data analysis, including automated documentation. Therefore, we find **DataExplorer** to be a good candidate for a next-step package after data cleaning is finished.

But no matter how clever software tools we make, data cleaning remains to be a time consuming endeavor, as 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 their origin. This often requires a collaborative effort between an expert in the field and a statistician or data scientist, which may be why the process of proper data cleaning is not always undertaken. In many situations, these errors are discovered in the process of the data analysis (e.g., a categorical variable with numeric labels for each category may be wrongly classified as a quantitative variable or a variable where all values have erroneously been coded to the same value), 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).

The **dataMaid** approach to data cleaning, -quality assessment and -documentation is governed by two fundamental paradigms. First of all, there is no need for data cleaning to be an ad hoc procedure. Often, we have a very clear idea of what flags are raisable in a given dataset before we look at it, as we were the ones to produce it in the first place. This means that data cleaning can easily be a well-documented, well-specified procedure. In order to aid this paradigm, **dataMaid** provides easy-to-use, automated tools for data quality assessment in R (R Core Team 2016) on which data cleaning decisions can be made. This quality assessment is presented in an auto-generated overview document, readable by data analysts and field experts alike, thereby also contributing to an inter-field dialogue about the data at hand. Oftentimes, e.g., distinguishing between faulty codings of a numeric value and unusual, but correct, values requires problem-specific expertise that might not be held by the data analyst. Hopefully, having easy access to data descriptions through **dataMaid** will help this necessary knowledge sharing.

While **dataMaid**s primary raison d'être is auto-generating data quality assessment overview documents, we still wish to emphasize that it is *not* a tool for unsupervised data cleaning. This qualifies as the second paradigm of **dataMaid**: Data cleaning decisions should always be made by humans. Therefore, **dataMaid** does not supply any tools for "fixing" errors in the data. However, we do provide interactive functions that can be used to identify potentially erroneous entries in a dataset and that can make it easier to solve data issues, one variable at a time.

This manuscript is structured as follows: First, in Section 2, we introduce the main representative of the first paradigm, namely the `clean()` function, which generates data overview documents. In the **dataMaid** package, we have provided a number of default checks that cover the data cleaning challenges, we find to be most common. Next, in Section 3, we present the



Part 1

Data cleaning summary

The dataset examined has the following dimensions:

Feature	Result
Number of rows	15
Number of variables	6

Checks performed

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

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

Please note that all numerical values in the following have been rounded to 2 decimals.

1

Part 2

Variable list

var1

Feature	Result
Variable type	factor
Number of missing obs.	2 (13.33 %)
Number of unique values	2
Mode	"red"

- Note that the following levels have at most five observations: "blue".

var2

Feature	Result
Variable type	numeric
Number of missing obs.	3 (20 %)
Number of unique values	8
Median	4.5
1st and 3rd quartiles	1.75; 6
Min. and max.	1; 999

- The following suspected missing value codes enter as regular values: '999', 'NaN'.
- Note that the following possible outlier values were detected: '82', '999'.

2

Figure 1: Example output from running `makeDataReport()` on the `toyData` dataset. First a summary of the full dataset is given and then type-dependent information on each variable is given in a table and a graph. Larger versions of the pages can be seen in Appendix A.

interactive mode of **dataMaid**, as motivated by the second paradigm above. But, as any data analyst knows, every dataset is different, and some datasets might include problems that cannot be detected by our data checking functions. In Section **??**, we turn to the question of how **dataMaid** extensions can be made, such that they are integrable with the `clean()` function and with the other tools available in **dataMaid**. At last, in Section 4, we discuss a number of examples of specific data cleaning challenges and how **dataMaid** can be used to solve them.

2. Creating a data overview report

The `makeDataReport()` function is the primary workhorse of **dataMaid** and this is the only function needed if a standard battery of tests are used to generate data reports. The data report itself is an overview document, intended for reading by humans, in either pdf or html format. Appendix A provides an example of a data report, produced by calling `makeDataReport()` on the dataset `toyData` available in **dataMaid**. The first two pages of this data report are shown in Figure 1. `toyData` is a very small (15 observations of 6 variables), artificial dataset which was created to illustrate the main capabilities of **dataMaid**. The following commands load the dataset and produce the report:

```
R> library("dataMaid")
R> data("toyData")
```

```
R> toyData
```

```
   var1 var2  var3       var4 var5       var6
1   red    1     a -0.6264538    1 Irrelevant
2   red    1     a  0.1836433    2 Irrelevant
3   red    1     a -0.8356286    3 Irrelevant
4   red    2     a  1.5952808    4 Irrelevant
5   red    2     a  0.3295078    5 Irrelevant
6   red    6     b -0.8204684    6 Irrelevant
7   red    6     b  0.4874291    7 Irrelevant
8   red    6     b  0.7383247    8 Irrelevant
9   red  999     c  0.5757814    9 Irrelevant
10  red   NA     c -0.3053884   10 Irrelevant
11 blue    4     c  1.5117812   11 Irrelevant
12 blue   82     .  0.3898432   12 Irrelevant
13 blue   NA       -0.6212406   13 Irrelevant
14 <NA>  NaN other -2.2146999   14 Irrelevant
15 <NA>    5 OTHER  1.1249309   15 Irrelevant
```

```
R> makeDataReport(toyData)
```

By default, an R markdown file and a rendered pdf overview document is produced, saved to the working directory and opened for immediate inspection. Turning to Figure 1, we see that such a data report consists of three parts. First, an overview of what was done is presented under the title *Data report overview*. Secondly, an index listing each variable along with an indication of whether it was found to be problematic or not is provided. Thirdly, each variable in the dataset is presented in turn using (up to) three tools in the *Variable list*: A table summarizing key features of the variable, a figure visualizing its distribution and a list of flagged issues, if any. For instance, in the `numeric`-type variable `var2` from `toyData`, `makeDataReport()` has identified two values that are suspected to be miscoded missing values (`999` and `NaN`), while two values were also flagged as potential outliers that should be investigated more carefully.

Though the `makeDataReport()` function is very easy to use, it should not be mistaken to be inflexible: By using the optional arguments of the function, both the contents and the look of the data report can be molded according to the user's needs. The most commonly used arguments are summarized in Table 1 and they are grouped according to the part of the data assesment and report generation they influence. In order to understand this distinction, a glimpse of the inner structure of `makeDataReport()` is shown in Figure 2. Below, we present a few examples on how to use the arguments from Table 1 to influence the output of a `makeDataReport()` call.

2.1. Polishing off the arguments

We begin with an example that is intended as an illustration of how `makeDataReport()` might be used in the very first stages of data cleaning, when the we are uncertain about the complexities of the errors and how much time should be allocated to data cleaning. At this stage, what is really needed, is a very rough idea of the severity of errors in the dataset.

Argument	Description	Default value
Control input variables and summary		
`useVar`	What variables should be used?	`NULL` (corresponding to all variables)
`ordering`	Ordering of the variables in the data summary (as is or alphabetical)	`"asIs"`
`onlyProblematic`	Should only variables flagged as problematic be included in the *Variable list*?	`FALSE`
`listChecks`	Should an overview of what checks were performed be listed in the *Data report overview*?	`TRUE`
`preChecks`	What check functions should be called to determine whether a variable is suitable for summarization, visualization and checking?	`c("isKey", "isEmpty")`
Control summarize, visualize, and check steps		
`mode`	What steps should be performed for each variable (out of the three possibilities *summarize*, *visualize*, *check*)?	`c("summarize", "visualize", "check")` (corresponding to all three steps)
`smartNum`	Should numerical values with only a few unique levels be flagged and treated as a factor variable?	`TRUE`
`maxProbVals`	Maximum number of problematic values to print, if any are found in data checks	`Inf`
`maxDecimals`	Maximum number of decimals to print for numeric values in the variable list	`2`
`twoCol`	Should the summary table and visualizations be placed side-by-side (in two columns)?	`TRUE`
Control output and post-processing		
`output`	Type of output file to be produced (html, or pdf)	`"pdf"`
`render`	Should the output file be rendered from markdown?	`TRUE`
`openResult`	If a pdf/html file is rendered, should it automatically open afterwards, and if not, should the `rmarkdown` file be opened?	`TRUE`
`replace`	Overwrite an existing file with the same name?	`FALSE`

Table 1: A selection of commonly used arguments to `makeDataReport()` separated into the parts they control.

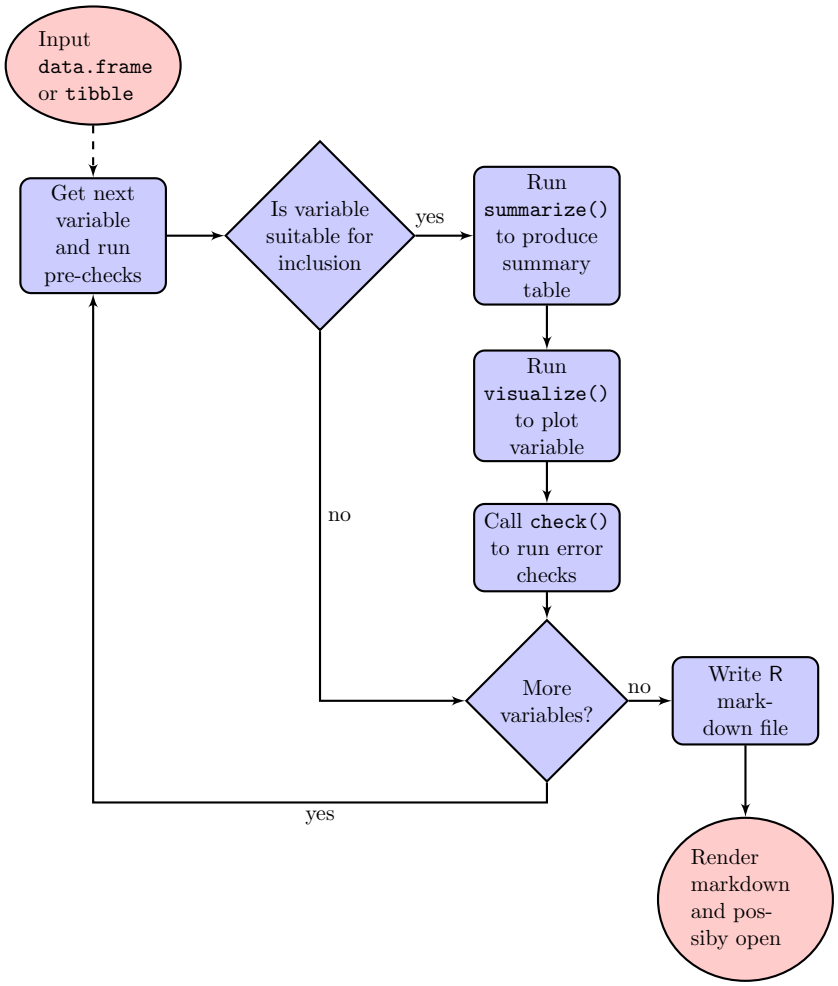


Figure 2: Schematic illustration of the stages undertaken when running `makeDataReport()`. Each variable is checked for eligibility before running `summarize()`, `visualize()`, and `check()`, and the resulting R markdown file may be rendered and opened.

In this scenario, we might wish to obtain a summary document in html format that only contains the variables with potential problems, and with a limit of, say, maximum 10 printed potential errors for each variable. Also, we can add the argument `replace=TRUE` in order to force `makeDataReport()` to overwrite any existing files produced by `makeDataReport()`. Using the `toyData` dataset as a guinea pig, we type:

```
R> makeDataReport(toyData, output = "html", onlyProblematic = TRUE, maxProbVals = 10,
+    replace = TRUE)
```

The final rendering of the generated markdown file is controlled by the `render` and `openResult` arguments, which both default to `TRUE`. `render` determines if the R markdown file produced should be rendered using the **rmarkdown** (Allaire *et al.* 2016) package and `openResult` decides whether the output html or pdf file should be opened. The following command produces an

R markdown file containing the information needed for generating a data report, but without rendering nor opening the markdown file:

```
R> makeDataReport(toyData, output="html", render=FALSE, openResult=FALSE)
```

We will now move on to discuss how not only the structure of the data assesment steps is manipulated, but also its very contents. This is done through the summarize/visualize/check (SVC) steps, as illustrated in Figure 2. **dataMaid** uses three different types of functions for performing these steps, namely `summaryFunction`s, `visualFunction`s and `checkFunction`s. By default, `clean()` runs selected summary, visualization and check functions on each variable in the dataset, and the exact choice of these functions depends on the classes of the variables. For instance, though detection of outlier values might be interesting for numerical variables, it holds little meaning for factor variables, and therefore, numerical and factor variables need different checks. Table 2 lists all available summarize/visualize/check functions, but we can also use the `allSummaryFunctions()`, `allVisualFunctions()`, and `allCheckFunctions()` functions in **dataMaid** to print overview lists in R. For example, the implemented `summaryFunction`s are:

```
R> allSummaryFunctions()
```

name	description	classes
centralValue	Compute median or mode	character, Date, factor, integer, labelled, logical, numeric
countMissing	Compute proportion of missing observations	character, Date, factor, integer, labelled, logical, numeric
minMax	Find minimum and maximum values	integer, numeric, Date
quartiles	Compute 1st and 3rd quartiles	Date, integer, numeric
uniqueValues	Count number of unique values	character, Date, factor, integer, labelled, logical, numeric
variableType	Data class of variable	character, Date, factor, integer, labelled, logical, numeric

Thus we can see, for example, that for `numeric`, `integer`, and `Date` variables, **dataMaid** provides functions for adding summary information about the minimum and maximum values,

	Description	Variable classes						
		C	F	I	L	B	N	D
`summaryFunctions`								
`centralValue`	Compute median or mode	×	×	×	×	×	×	×
`countMissing`	Compute proportion of missing observations	×	×	×	×	×	×	×
`minMax`	Find minimum and maximum values			×			×	×
`quartiles`	Compute 1st and 3rd quartiles			×			×	×
`uniqueValue`	Count number of unique values	×	×	×	×	×	×	×
`variableType`	Data class of variable	×	×	×	×	×	×	×
`visualFunctions`								
`basicVisual`	Histograms and barplots using base R graphics	×	×	×	×	×	×	×
`standardVisual`	Histograms and barplots using **ggplot2**	×	×	×	×	×	×	×
`checkFunctions`								
`identifyCaseIssues`	Identify case issues	×	×					
`identifyLoners`	Identify levels with < 6 obs.	×	×					
`identifyMissing`	Identify miscoded missing values	×	×	×	×	×	×	
`identifyNums`	Identify misclassified numeric or integer variables	×	×					
`identifyOutliers`	Identify outliers			×		×	×	
`identifyOutliersTBStyle`	Identify outliers (Turkish Boxplot style)			×		×	×	
`identifyWhitespace`	Identify prefixed and suffixed whitespace	×	×		×			
`isCPR`	Identify Danish CPR numbers	×	×	×	×	×	×	×
`isEmpty`	Check if the variable contains only a single value	×	×	×	×	×	×	×
`isKey`	Check if the variable is a key	×	×	×	×	×	×	×

Table 2: List of all summary functions (used in the summary table for each variable in the output), visual functions (used for visualization of each variable), and check functions (used for data checks for each variable) currently implemented in **dataMaid**. The variable classes C, F, I, L, B, N, and D, refer to `character`, `factor`, `integer`, `labelled`, `logical` (boolean), `numeric`, and `Date`, respectively.

while all seven variable classes dealt with in **dataMaid** have functions for central tendency summaries (i.e., mode or median).

We can control what summaries and checks are applied for each variable type through the `summaries`, `visuals` and `checks` arguments. Each of these arguments should be a list with one entry for each variable type and a number of function names for each such entry. The easiest way to specify the arguments is by use of the built-in helper functions `setSummaries()`, `setVisuals()` and `setChecks()`.

For instance, we can inspect the default settings for summaries by calling

```
R> setSummaries()

$character
[1] "identifyMissing"  "identifyWhitespace"  "identifyLoners"
[4]  "identifyCaseIssues"  "identifyNums"

$factor
[1] "identifyMissing"  "identifyWhitespace"  "identifyLoners"
[4] "identifyCaseIssues"  "identifyNums"

$labelled
[1] "identifyMissing"  "identifyWhitespace"  "identifyLoners"
[4] "identifyCaseIssues"  "identifyNums"

$numeric
[1] "identifyMissing"  "identifyOutliers"

$integer
[1] "identifyMissing"  "identifyOutliers"

$logical
NULL

$Date
[1] "identifyOutliers"
```

This helper function really just calls several other helper functions, namely the `defaultXXXSummaries()` functions, where `XXX` refers to a variale class. For instance, we can see the default character checks by calling `defaultCharacterSummaries()`:

```
R> defaultCaracterSummaries()

[1] "variableType" "countMissing" "uniqueValues" "centralValue"
```

We can change the choice of summaries (and similarly the checks and visual functions) by setting the corresponding arguments when calling `makeDataReport()`. For example, to get only the variable type and the central tendency listed in the summary table for numeric and integer variables, we write

```
R> makeDataReport(toyData,
         summaries = setSummaries(numeric = c("variableType","centralValue),
                integer = c("variableType", "centralValue"))
```

In the case where we specify the same set of summary functions for each variable type, we can use a simpler argument for `setSummaries` which overrides the summary functions for all variable types:

```
R> makeDataReport(toyData,
        summaries = setSummaries(all = c("variableType", "centralValue"))
```

Similarly, the checks applied are set with the `checks` argument and the `setChecks` function. The default checks being applied to a factor are

```
R> defaultFactorChecks()

[1] "identifyMissing"    "identifyWhitespace" "identifyLoners"
[4] "identifyCaseIssues" "identifyNums"
```

Now, if we only wanted to apply the function to identify whitespace for factor variables, then we would need provide this information for `setChecks()`:

```
R> makeDataReport(toyData, checks = setChecks(factor = "identifyWhitespace"))
```

or we could remove checks for factors altogether by setting the corresponding argument to `NULL`, in which case factor variables will not be checked for any potential errors:

```
R> makeDataReport(toyData, checks = setChecks(factor = NULL))
```

As with `summaryFunction`s, a complete list of available `checkFunction`s is obtained by calling `allCheckFunctions()`. Note however, that `checkFunction`s have a usage beyond the `checks` arguments, namely in the pre-check stage. In this stage, it is determined whether or not each variable is suitable for the summarize/visualize/check (SVC) steps. The functions used in the pre-check stage should be `checkFunction`s that are applicable to all variable classes. The default pre-checks, the functions `isKey()` and `isSingular()`, check whether a variable has unique values for all observations or only a single value for all observations, respectively. If one of these statements are true, the variable will not be subjected to the SVC steps. We can allow empty variables to move on to the SVC step by only checking for keys in the pre-check step:

```
R> makeDataReport(toyData, preChecks = "isKey")
```

Note that the data visualizations in the cleaning summary are also controllable, though only a single function can be provided for each variable type. If, for instance, we wish to change the visualizations from the default **ggplot2** (Wickham 2009) style histograms and barplots to base R histograms and barplots, we type

```
R> makeDataReport(toyData, visuals = setVisuals(all = "basicVisual"))
```

As indicated in Figure 2, there are two stages where `makeDataReport()` applies functions to each of the variables:

1. In the pre-check stage.
2. As part of the summarize/visualize/check (SVC) steps.

Each of these stages are controllable using appropriate function arguments in `makeDataReport()`, and above we have shown examples of how to tweak them to modify the data cleaning outputs. However, if the dataset at hand requires new, additional checks, then more control is needed and we have provided a detailed vignette that explains how to modify and expand the possibilities by producing new summary, visual, and check functions in [REFERENCE].

3. Using dataMaid interactively

While overview documents are great for presenting and documenting the data cleaning checks, it may be useful to be able to work more interactively through the data cleaning process. Aside from the `clean()` function presented above, **dataMaid** also provides more standard R interactive tools, such as functions that print results to the console or return the information as an object for later use. This section describes how to use the functions `check()`, `summarize()` and `visualize()` to work interactively with **dataMaid**.

3.1. Data cleaning by hand: An example

Assume that we wish to look further into a certain variable from `toyData`, namely `var2`. The data cleaning summary found some issues in this variable, and we would like to recall what these issues were. This can be done using the `check()` command

```
R> check(toyData$var2)

$identifyMissing
The following suspected missing value codes enter as regular values: 999, NaN.
$identifyOutliers
Note that the following possible outlier values were detected: 82, 999.
```

Note that the arguments specifying which checks to perform, as described in the previous section, are in fact passed to `check()`, and thus they can also be used here. For instance, if we only want to check for potentially miscoded missing values, we can use the relevant `XXXChecks` argument (e.g., `numericChecks`, `factorChecks`, etc., as described in Section 2) to provide a vector of the check functions that should be applied. Recall that Table 2 or an `allCheckFunctions()` call provide overviews of the available check functions. Moving forward, we limit the numeric checks to only identify miscoded missing values:

```
R> check(toyData$var2, numericChecks = "identifyMissing")
```

```
$identifyMissing
The following suspected missing value codes enter as regular values: 999, NaN.
```

An equivalent way to call only a single, specific `checkFunction`, such as `identifyMissing`, is by using it directly on the variable, e.g.,

```
R> identifyMissing(toyData$var2)

The following suspected missing value codes enter as regular values: 999, NaN.
```

The result of a `checkFunction` is an object of class `checkResult`. By using the structure function, `str()`, we can look further into its components:

```
R> missVar2 <- identifyMissing(toyData$var2)
R> str(missVar2)

List of 3
 $ problem      : logi TRUE
 $ message      : chr "The following suspected missing value codes
    enter as regular values: \\\"999\\\", \\\"NaN\\\"."
 $ problemValues: num [1:2] 999 NaN
 - attr(*, "class")= chr "checkResult"
```

The most important thing to note here is that while the printed message is made for easy reading, the actual values of the variable causing the issue are still obtainable in the element `problemValues`. If we decide that the values `999` and `NaN` in `var2` are in fact miscoded missing values, we can easily replace them with `NA`s:

```
R> toyData$var2[toyData$var2 %in% missVar2$problemValues] <- NA
R> identifyMissing(toyData$var2)

No problems found.
```

Similarly, the `visualize()` and `summarize()` functions can be used to run the corresponding visualizations and summaries for each variable. See Figure 3 for the visualization output.

```
R> visualize(toyData$var2)
R> summarize(toyData$var2)

     Feature                   Result
[1,] "Variable type"           "numeric"
[2,] "Number of missing obs."  "4 (26.67 %)"
[3,] "Number of unique values" "6"
[4,] "Median"                  "4"
[5,] "1st and 3rd quartiles"   "1.5; 6"
[6,] "Min. and max."           "1; 82"
```

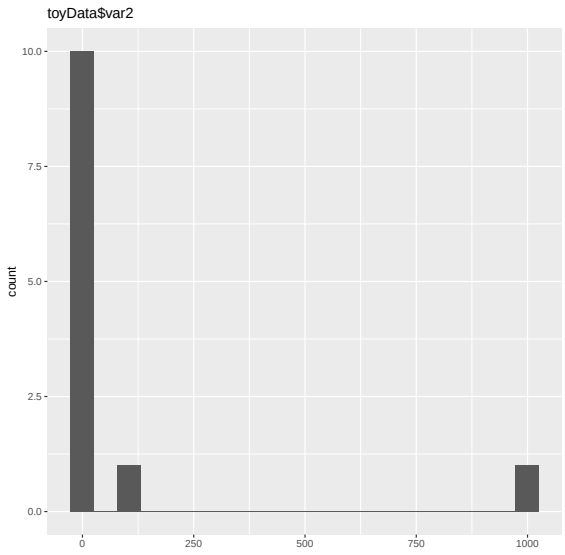


Figure 3: Output from running `visualize()` on the variable `var2` from the `toyData` dataset.

As we saw with the `check()` function, the summary can be modified by providing the relevant `XXXSummaries` arguments. Setting the `numericSummaries` argument, we can control the summary output by providing a vector of function names to run for a particular summary. To only get the median, minimum and the maximum we set `numericSummaries = c("centralValue", "minMax")`:

```
R> summarize(toyData$var2, numericSummaries = c("centralValue", "minMax"))
```

```
     Feature         Result
[1,] "Median"        "4"
[2,] "Min. and max." "1; 82"
```

Note that the `summarize()`, `check()` and `visualize()` functions are also available interactively for full datasets by calling e.g., `summarize(toyData)`. However, this produces an extensive amount of output in the console, and therefore, we generally do not recommend it, unless working with very small datasets or subsets of datasets.

4. Rubbing down data cleaning challenges

Finally, we present a few examples of how to make **dataMaid** solve specific issues related to data cleaning. First, we discuss the challenges related to cleaning large datasets, particularly in terms of memory use and computation speed. Next, we show how **dataMaid** can be used for problem-flagging. Lastly, we discuss how the **dataMaid** output document can be included in other R markdown documents as a way to produce clear and concise documentation of a dataset.

4.1. Cleaning large datasets

If the dataset becomes very large, the standard use of `clean()` outlined above might not be ideal. If there is a large number of variables, creation of the R markdown document might be quite slow, while a large number of observations will generally affect the rendering time of the document. This issue is especially relevant to Windows users, where memory usage is not very efficient. The problem is that older, 32-bit versions of Windows generally cannot access all physical RAM, while newer 64-bit versions of Windows cannot access virtual memory. Therefore, memory allocation issues becomes even more pressing under Windows.

Below we give a few practical examples of ways to deal with large data, while wishing to still produce (potentially very long) data cleaning overview documents. Note that the interactive tools of **dataMaid** can be used as usual or sequentially in small subsets of the large dataset, if no overview documents are needed.

Handling the figures

Though figures give a nice overview of each variable, they are also quite heavy objects in terms of memory allocation. Therefore, it might be beneficial to not include figures in the **dataMaid** outputs for very large datasets. This is controlled via the `mode` argument:

```
R> clean(toyData, mode = c("summarize", "check"))
```

If figures are indeed needed, a different approach is to choose the less memory heavy standard R figure style instead of the **ggplot2** figures that are the default option in `clean()`. This can be done by setting the `allVisuals = "basicVisual"` argument:

```
R> clean(toyData, allVisuals = "basicVisual")
```

Of course, even less heavy plots might be achieved by writing new `visualFunction`s, using the guidelines from Section **??**. For instance, a future extension of **dataMaid** might be the inclusion of ASCII plots, as e.g., represented in the R package **txtplot** (Bornkamp 2012).

Economic memory use

Another solution, which is especially relevant to Windows users due to the memory handling on this operating system, is simply splitting the two steps performed by `clean()`, namely producing the R markdown file and rendering it afterwards. If the `rmarkdown` file is very long, as it will typically be for very large datasets, keeping this file open in memory wastes precious memory capacities. Therefore, we advise users to instead split the two steps as shown in the following.

```
R> clean(toyData, render = FALSE, openResult = FALSE)
R> render("dataMaid_toyData.Rmd", quiet = FALSE)
```

This also deals with the fact that **dataMaid** can produce R markdown files that supersedes the upper size limit of code editors, e.g., RStudio, which currently has an editor file size limit of 5MB.

4.2. Using dataMaid for problem flagging

If the data is large, but memory issues and computation time are less of an issue than the time it takes a human to look through the data cleaning document, a viable solution might be not to include all information about all variables. This can be achieved by using the `onlyProblematic` argument in `clean()`. By specifying `onlyProblematic = TRUE`, only variables that raise a flag in the checking steps will be summarized and visualized. But perhaps we are not even interested in obtaining general information about these variables, but only in getting a quick overview of the problems they might have. This is obtained by controlling the `mode` argument:

```
R> clean(toyData, onlyProblematic = TRUE, mode = c("check"))
```

Now only the checking results are printed, and only for variables where problems were identified. An even more minimal output can be generated by also leaving out the checking results — then `clean()` essentially just produces a document with a list of the variable names that should be investigated further:

```
R> clean(toyData, onlyProblematic = TRUE, mode = NULL)
```

Of course, this can also be done without generating an overview document, by direct, interactive use of the `check()` function. When called on a `data.frame`, this function produces a list (of variables) of lists (of checks) of lists (or rather, `checkResult`s). Thus, the overall problem status of each variable can easily be unravelled using the list manipulation function `sapply()`:

```
R> toyChecks <- check(toyData)
R> foo <- function(x) {
+    any(sapply(x, function(y) y[["problem"]]))
+  }
R> sapply(toyChecks, foo)

 var1  var2  var3  var4  var5  var6
 TRUE  TRUE  TRUE  TRUE  TRUE FALSE
```

and we find that only a single variable in `toyData`, `var6` (for which all observations have the value `"Irrelevant"`), is problem-free.

4.3. Including dataMaid documents in other files

Sometimes, a **dataMaid** document might be a useful addition to a more general overview document, including additional information such as pairwise association plots, time series plots, or exploratory analysis results. **dataMaid** can produce a document to be included in other R markdown files by setting the `standAlone` argument in `clean()` to remove the preamble from the output R markdown file. Note that it is still necessary to indicate which R markdown output type is created; the pdf and html R markdown styles are unfortunately not identical.

If it is important that the embedded **dataMaid** document can be rendered to either of these two file types, we recommend setting `twoCols = FALSE` and `output = "html"` in `clean()`, thereby essentially removing almost all output type specific formatting code from the generated R markdown file.

On the other hand, if a pdf document is to be produced, a few extra lines need to be added to the preamble of the master R markdown document — otherwise, the two-column layout code will produce an error. The following is an example of how such a master document preamble YAML might look like and how the `dataMaid_toyData.Rmd` file can then be included:

```
---
output: pdf_document
documentclass: report
header-includes:
  - \renewcommand{\chaptername}{Part}
  - \newcommand{\fullline}{\noindent\makebox[\linewidth]{\rule{\textwidth}{0.4pt}}}
  - \newcommand{\bminione}{\begin{minipage}{0.75 \textwidth}}
  - \newcommand{\bminitwo}{\begin{minipage}{0.25 \textwidth}}
  - \newcommand{\emini}{\end{minipage}}
---

```{r, child = 'dataMaid_toyData.Rmd'}
```
```

In the this example, the `dataMaid_toyData.Rmd` file could have been created as follows:

```
R> clean(toyData, standAlone = FALSE)
```

and the more minimal, html-style R markdown file described above can be produced using

```
R> clean(toyData, standAlone = FALSE, output = "html", twoCols = FALSE)
```

Note that with the latter option, no special YAML preamble is needed in the R markdown document.

5. Concluding remarks

In this paper we have introduced the R package **dataMaid** for performing reproducible error detection and data cleaning summaries. The package provides a general and extendable framework for identifying potential errors and for creating human-readable summary documents that will help investigators to identify possible errors in the data.

While the the current release is stable, the authors have an interest in further developing the functionality of **dataMaid** by providing more summary, visual, and check function as part of the default package. We are also currently considering adding options to handle repeated measurement, where the visualizations — in particular — might be improved by visualizing measurements over time. In addition, an online **shiny** (Chang *et al.* 2016) application where users that are not R-savvy can upload their data and get a data cleaning document is currently planned.

References

Allaire J, Cheng J, Xie Y, McPherson J, Chang W, Allen J, Wickham H, Atkins A, Hyndman R (2016). ***rmarkdown**: Dynamic Documents for R*. R package version 1.3, URL http://rmarkdown.rstudio.com.

Bornkamp B (2012). ***txtplot**: Text based plots*. R package version 1.0-3, URL https://CRAN.R-project.org/package=txtplot.

Chang W, Cheng J, Allaire JJ, Xie Y, McPherson J (2016). ***shiny**: Web Application Framework for R*. R package version 0.13.2, URL https://CRAN.R-project.org/package=shiny.

Cui B (2016). ***DataExplorer**: Data Explorer*. R package version 0.3.0, URL https://CRAN.R-project.org/package=DataExplorer.

de Jonge E, van der Loo M (2015). ***editrules**: Parsing, Applying, and Manipulating Data Cleaning Rules*. R package version 2.9.0, URL https://CRAN.R-project.org/package=editrules.

Dowle M, Srinivasan A, Short T, Lianoglou S (2016). ***data.table**: Extension of Data.frame*. R package version 1.9.8, URL https://CRAN.R-project.org/package=data.table.

Firke S (2016). ***janitor**: Simple Tools for Examining and Cleaning Dirty Data*. R package version 0.2.1, URL https://CRAN.R-project.org/package=janitor.

Fischetti T (2017). ***assertr**: Assertive Programming for R Analysis Pipelines*. R package version 2.0.2.2, URL https://CRAN.R-project.org/package=assertr.

Gandrud C (2016). ***DataCombine**: Tools for Easily Combining and Cleaning Data Sets*. R package version 0.2.21, URL https://CRAN.R-project.org/package=DataCombine.

Petersen AH, Ekstrøm CT (2016). ***dataMaid**: A Suite of Checks for Identification of Potential Errors in a Data Frame as Part of the Data Cleaning Process*. R package version 0.9.2.

R Core Team (2016). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

van der Loo M, de Jonge E (2016). ***validate**: Data Validation Infrastructure*. R package version 0.1.5, URL https://CRAN.R-project.org/package=validate.

van der Loo M, de Jonge E, Scholtus S (2015). ***deducorrect**: Deductive Correction, Deductive Imputation, and Deterministic Correction*. R package version 1.3.7, URL https://CRAN.R-project.org/package=deducorrect.

Wickham H (2009). ***ggplot2**: Elegant Graphics for Data Analysis*. Springer-Verlag New York. ISBN 978-0-387-98140-6. URL http://ggplot2.org.

Wickham H, Francois R, Henry L, Müller K (2017). ***dplyr**: A Grammar of Data Manipulation*. R package version 0.7.4, URL https://CRAN.R-project.org/package=dplyr.

Wickham H, Henry L (2017). ***tidyr**: Easily Tidy Data with 'spread()' and 'gather()' Functions*. R package version 0.7.1, URL https://CRAN.R-project.org/package=tidyr.

A. Appendix A: cleaning the toyData data frame

Part 1

Data cleaning summary

The dataset examined has the following dimensions:

Feature	Result
Number of rows	15
Number of variables	6

Checks performed

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

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

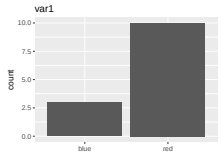
Please note that all numerical values in the following have been rounded to 2 decimals.

Part 2

Variable list

var1

Feature	Result
Variable type	factor
Number of missing obs.	2 (13.33 %)
Number of unique values	2
Mode	“red”

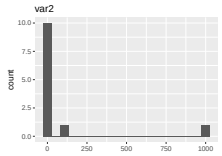


- Note that the following levels have at most five observations: "blue".

---

var2

Feature	Result
Variable type	numeric
Number of missing obs.	3 (20 %)
Number of unique values	8
Median	4.5
1st and 3rd quartiles	1.75; 6
Min. and max.	1; 999

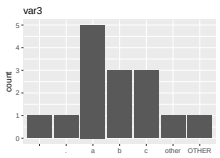


- The following suspected missing value codes enter as regular values: "999", "NaN".
- Note that the following possible outlier values were detected: "82", "999".

---

var3

Feature	Result
Variable type	factor
Number of missing obs.	0 (0 %)
Number of unique values	7
Mode	“a”

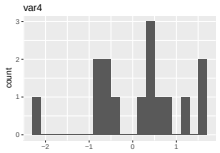


- The following suspected missing value codes enter as regular values: " ", ".".
- The following values appear with prefixed or suffixed white space: " ".
- Note that the following levels have at most five observations: " ", ".", "a", "b", "c", "other", "OTHER".
- Note that there might be case problems with the following levels: "other", "OTHER".

---

var4

Feature	Result
Variable type	numeric
Number of missing obs.	0 (0 %)
Number of unique values	15
Median	0.33
1st and 3rd quartiles	-0.62; 0.66
Min. and max.	-2.21; 1.6



- Note that the following possible outlier values were detected: "1.12", "1.51", "1.6".

---

var5

- The variable is a key (distinct values for each observation).

---

var6

- The variable only takes one (non-missing) value: "Irrelevant". The variable contains 0 % missing observations.

This report was created by dataMaid v0.9.2.

B. Appendix B: User-extended cleaning of exampleData

Part 1

Data cleaning summary

The dataset examined has the following dimensions:

Feature	Result
Number of rows	300
Number of variables	6

Checks performed

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

	character	factor	labelled	numeric	integer	logical	Date
Identify miscoded missing values	×	×	×	×	×		
Identify prefixed and suffixed whitespace	×	×	×				
Identify levels with < 6 obs.	×	×	×				
Identify case issues	×	×	×				
Identify misclassified numeric or integer variables	×	×	×				
Identify colons surrounded by alphanumeric characters	×	×	×				
Identify outliers				×	×		×

Please note that all numerical values in the following have been rounded to 2 decimals.

Part 2

Variable list

addresses

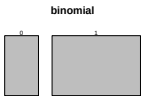
- The variable is a key (distinct values for each observation).

---

binomial

- Note that this variable is treated as a factor variable below, as it only takes a few unique values.

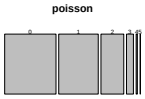
Feature	Result
Variable type	integer
Number of missing obs.	0 (0 %)
Number of unique values	2
Mode	"1"
No. zeros	83



---

poisson

Feature	Result
Variable type	integer
Number of missing obs.	0 (0 %)
Number of unique values	6
Median	1
1st and 3rd quartiles	0; 2
Min. and max.	0; 5
No. zeros	126



---

gauss

Feature	Result
Variable type	numeric
Number of missing obs.	0 (0 %)
Number of unique values	300
Median	-0.05
1st and 3rd quartiles	-0.66; 0.76
Min. and max.	-3.25; 3.64
No. zeros	0

- Note that the following possible outlier values were detected: "-3.25", "-3.21", "-2.8", "-2.11", "-2.04".

---

zigauss

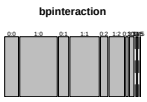
Feature	Result
Variable type	numeric
Number of missing obs.	0 (0 %)
Number of unique values	251
Median	0
1st and 3rd quartiles	-0.53; 0.47
Min. and max.	-2.97; 2.23
No. zeros	50

- Note that the following possible outlier values were detected: "-2.97", "-2.63", "1.67", "1.73", "1.74", "1.85", "1.87", "1.93", "2.02", "2.05", "2.11", "2.12", "2.23".

---

bpinteraction

Feature	Result
Variable type	factor
Number of missing obs.	0 (0 %)
Number of unique values	10
Mode	“1:0”
No. zeros	0



- Note that the following levels have at most five observations: "0:4", "1:4".
- Note: The following values include colons: "0:0", "1:0", "0:1", "1:1", "0:2", "1:2", "0:3", "1:3", "1:4", "1:5".

This report was created by dataMaid v0.9.2.

Affiliation:

Claus Thorn Ekstrøm
Biostatistics, Department of Public Health
University of Copenhagen
Denmark
E-mail: ekstrom@sund.ku.dk
URL: http://staff.pubhealth.ku.dk/~ekstrom/

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