Journal of Statistical Software

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

dataMaid: your assistant for documenting supervised data quality screening in R

Anne Helby Petersen
Section of Biostatistics
Department of Public Health
University of Copenhagen

Claus Thorn Ekstrøm
Section of 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 customizable suite of quality assessment aids 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 quality screening. Moreover, the **dataMaid** solution changes the data screening 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 -screening, where the data inspections are executed directly in the R console.

Keywords: data screening, 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 a subject 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 received increasing attention in the scientific communities in the later years: Data cleaning is not very straight forward to document and therefore, reproducibility suffers. We present a new R package, **dataMaid** (Petersen and Ekstrøm 2016), whose most central purpose is to facilitate a supervised data quality screening workflow where documentation is thoroughly integrated rather than an add-on. This is accomplished by structuring the data screening around auto-generated data overview reports that summarize and flags potential problems in the dataset.

But no matter how clever software tools we make, data cleaning remains to be a time consuming endeavor, which 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. 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 screening, -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 summaries through **dataMaid** will help this necessary knowledge sharing.

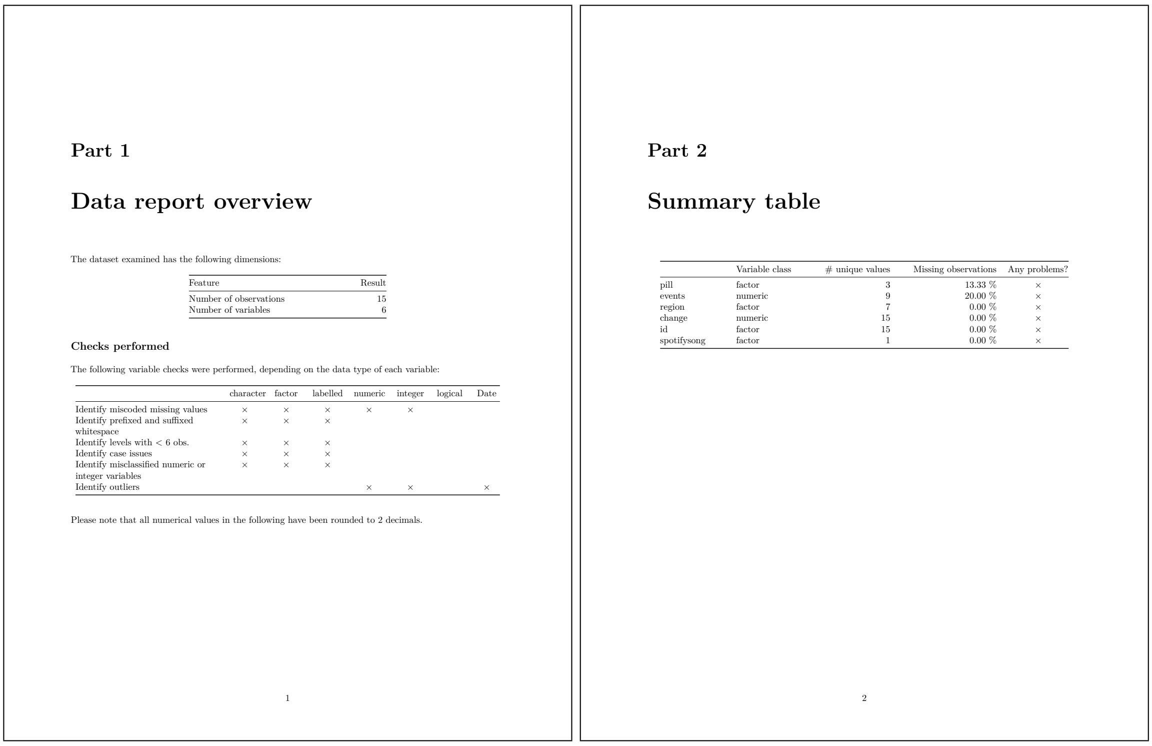
While **dataMaid**s primary raison d'être is auto-generating data quality assessment overview documents, we still wish to emphasize that it is *not* a tool for unsupervised data cleaning. This qualifies as the second paradigm of **dataMaid**: Data cleaning decisions should always be made by humans. Therefore, **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.

A number of R packages made for other pre-analysis steps are already available, including **janitor** (Firke 2016), **assertive** (Cotton 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), and **assertr** (Fischetti 2017). These packages focus on different stages of the pre-analysis work. **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) and **assertive** offer tools for identifying errors in a dataset by checking the state of the variable given a set of pre-specified rules, and their focus is on internal validity rather than general data screening. 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 currently 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. The **assertr** package provide very similar — and very nice — tools to those of **validate**, but without any ambitions of conducting auto-cleaning.

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 above. 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.

This manuscript is structured as follows: First, in Section 2, we introduce the main representative of the first paradigm, namely the `makeDataReport()` function, which generates data overview documents. In the **dataMaid** package, we have provided a number of default generic checks that cover the data cleaning challenges we find to be most common and these are also summarized in Section 2. Next, in Section 3, we present the interactive mode of **dataMaid**, as motivated by the second paradigm above. Next, we show step-by-step how the data report mode and the interactive mode of **dataMaid** can be combined to conduct a well-documented, systematic data cleaning in Section 4. Here, we assess and clean a dirty dataset with information about the US presidents. At last, in Section 5, we discuss a number of examples of specific data cleaning- and documentation challenges and how **dataMaid** can be used to solve them.

dataMaid was designed to be easily extended with user-supplied functions for summarizing, visualizing and checking data. In the package, we have provided a vignette in which



Part 1

Data report overview

The dataset examined has the following dimensions:

Feature	Result
Number of observations	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

Summary table

	Variable class	# unique values	Missing observations	Any problems?
pill	factor	3	13.33 %	×
events	numeric	9	20.00 %	×
region	factor	7	0.00 %	×
change	numeric	15	0.00 %	×
id	factor	15	0.00 %	×
spotifysong	factor	1	0.00 %	×

2

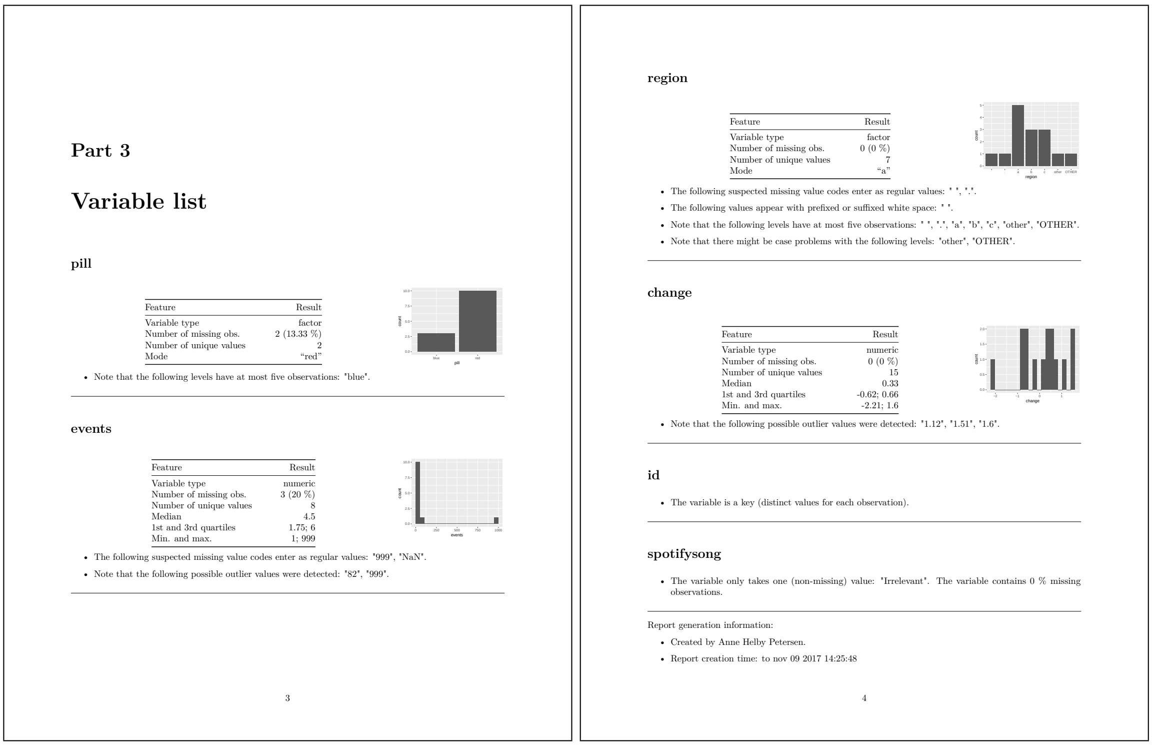
Figure 1: The two first pages of the report created by running `makeDataReport()` on the `toyData` dataset. First, a summary of the full dataset is given along with an overview of what checks were performed. Next, a summary of all the variables and whether or not they are problematic is provided. Larger versions of the pages can be seen in Appendix A.

we describe how **dataMaid** extensions can be made, such that they are integrate with the `makeDataReport()` function and with the other tools available in **dataMaid**.

2. Creating a data overview report

The `makeDataReport()` function is the primary workhorse of **dataMaid** and it is the only function needed to generate a data report using the standard battery of tests. The data report itself is an overview document, intended for reading by humans, in either pdf, html or word (.docx) 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 (excluding the front page) of this data report are shown in Figure 1 and the following two pages are shown in Figure 2. `toyData` is a very small (15 observations of 6 variables), artificial dataset which was created with a lot of potential errors to illustrate the main capabilities of **dataMaid**. Section 4 shows an example of a data screening process with a real dataset. The following commands load the dataset and produce the report:

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



Part 3

Variable list

pill

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".

events

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".

3

region

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".

change

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".

id

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

spotifysong

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

Report generation information:

- Created by Anne Helby Petersen.
- Report creation time: to nov 09 2017 14:25:48

4

Figure 2: The third and fourth pages of the report created by running `makeDataReport()` on the `toyData` dataset. Here, we see a description of each variable in the dataset, consisting of a summary table, a visualization and an indication of what problems were flagged for the variable (if any). At last, a few lines of metadata about the `makeDataReport()` are included for enhancing reproducibility. Larger versions of the pages can be seen in Appendix A.

```
# A tibble: 15 x 6
     pill events region     change     id spotifysong
   <fctr>  <dbl> <fctr>      <dbl> <fctr>      <fctr>
 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, word or html overview document is produced, saved to the working directory and opened for immediate inspection. Such a data report consists of three parts, two of which are presented in Figure 1. 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, as seen in Figure 2, 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 when relevant and a list of flagged issues, if any. For instance, as shown in Figure 2, for the `numeric`-type variable `events` 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.

The arguments to `makeDataReport()` can be used to modify the contents and the look of the data report 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 assessment and report generation they influence. In order to understand this distinction, a glimpse of the inner structure of `makeDataReport()` is shown in Figure 3. 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. Dusting 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 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. 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 2 printed potential problematic values per check 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 = 2, 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 outputted 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, replace=TRUE)
```

Argument	Description	Default value
Control input variables, looks and meta information		
`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`
`preChecks`	What check functions should be called to determine whether a variable is suitable for summarization, visualization and checking?	`c("isKey", "isSingular", "isSupported")`
`reportTitle`	What should the title displayed on the front page of the report be?	`NULL` (corresponds to the dataset name)
`twoCol`	Should the summary table and visualizations be placed side-by-side (in two columns)?	`TRUE`
Control summarize, visualize, and check steps		
`summaries`	What summaries should be performed for each variable type?	See Table 2
`visuals`	What type of visualization should be provided for each variable type?	See Table 2
`checks`	What checks should be applied to each variable type?	See Table 2
`mode`	What steps should be performed for each variable (out of the three possibilities *summarize*, *visualize*, *check*)?	`c("summarize", "visualize", "check")`
`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	`10`
`maxDecimals`	Maximum number of decimals to print for numeric values in the variable list	`2`
`treatXasY`	How should non-supported variable classes be handled?	`NULL` (no handling)
Control output and post-processing		
`output`	Type of output file to be produced (html, word (.docx) or pdf)	`NULL` (use pdf if LaTeX is found, otherwise Word (if on Windows), or html)
`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`
`vol`	Add a suffix to the file name of the outputted report	`""` (no suffix)

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

	Description	C	F	I	L	B	N	D
		Variable classes						
`summaryFunctions`								
`centralValue`	Compute median for numeric variables, mode for categorical variables	×	×	×	×	×	×	×
`countMissing`	Compute proportion of missing observations	×	×	×	×	×	×	×
`minMax`	Find minimum and maximum values			×			×	×
`quartiles`	Compute 1st and 3rd quartiles			×			×	×
`uniqueValues`	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 white space	×	×		×			
`isCPR`	Identify Danish CPR numbers	×	×	×	×	×	×	×
`isSingular`	Check if the variable contains only a single value	×	×	×	×	×	×	×
`isKey`	Check if the variable is a key	×	×	×	×	×	×	×
`isSupported`	Check if the variable is among the supported variable types	×	×	×	×	×	×	×

Table 2: Overview of all summary-, visual- and check functions 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. The default settings of `makeDataReport()` are marked in blue.

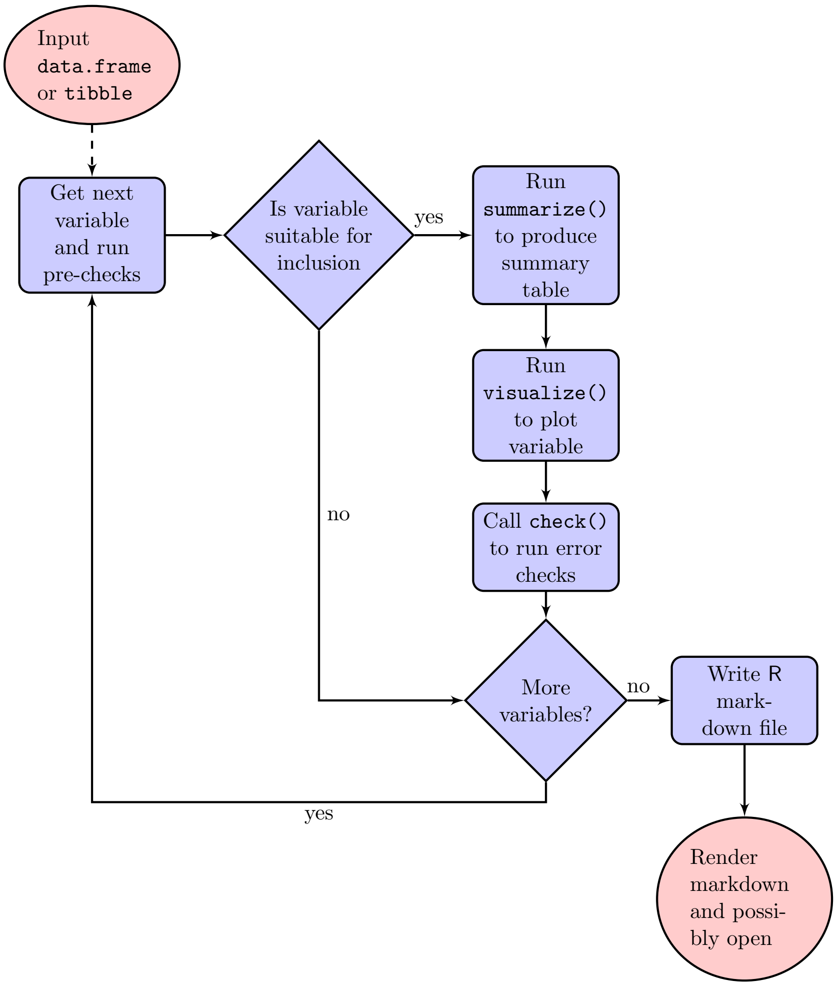


Figure 3: Schematic illustration of the stages undertaken when running `makeDataReport()`. Each variable is checked for eligibility before running `summarize()`, `visualize()`, and `check()`, and the resulting R markdown file may be rendered and opened.

2.2. Controlling contents through summaries, visualizations and checks

dataMaid works through three different steps — summarize, visualize, and check (SVC) — for each variable in the dataset (illustrated in Figure 3). Three different types of functions are used to perform these steps, namely `summaryFunctions`, `visualFunctions` and `checkFunctions`. By default, `makeDataReport()` 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, detection of outlier values might be interesting for numerical variables, but 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 `summaryFunctions` are:

```
R> allSummaryFunctions()

---------------------------------------------------------------------------
name           description                      classes
-------------- -------------------------------- -----------------------------
centralValue   Compute median for numeric       character, Date, factor,
               variables, mode for              integer, labelled, logical,
               categorical variables            numeric

countMissing   Compute proportion of missing    character, Date, factor,
               observations                     integer, labelled, logical,
                                                numeric

minMax         Find minimum and maximum         integer, numeric, Date
               values

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, 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 of `makeDataReport()`. Each of these arguments takes 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()` that contain the default settings of `makeDataReport()` and simple syntaxes for making small alterations of these default settings. We can inspect the default settings for summaries by calling:

```
R> setSummaries()

$character
[1] "variableType" "countMissing" "uniqueValues" "centralValue"

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

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

$numeric
[1] "variableType" "countMissing" "uniqueValues" "centralValue"
[5] "quartiles"    "minMax"

$integer
[1] "variableType" "countMissing" "uniqueValues" "centralValue"
[5] "quartiles"    "minMax"

$logical
[1] "variableType" "countMissing" "uniqueValues" "centralValue"

$Date
[1] "variableType" "countMissing" "uniqueValues" "centralValue"
[5] "minMax"       "quartiles"
```

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

```
R> defaultCharacterSummaries()
```

```
[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, replace=TRUE,
+    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, replace=TRUE,
+    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 white space for factor variables, then we would need provide this information for `setChecks()`:

```
R> makeDataReport(toyData, replace=TRUE,
+    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), replace=TRUE)
```

As with `summaryFunctions`, a complete list of available `checkFunctions` is obtained by calling `allCheckFunctions()`. Note however, that `checkFunctions` 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 `checkFunctions` that are applicable to all variable classes. The default pre-checks, the functions `isKey()`, `isSingular()` and `isSupported()`, check whether a variable has unique values for all observations, only a single value for all observations, and is not among the variable types supported by **dataMaid**, respectively. If one of these statements are true, the variable will not be subjected to the SVC steps. We can allow singular variables to move on to the SVC step by only checking for keys and non-supported variables in the pre-check step:

```
R> makeDataReport(toyData, preChecks = c("isKey", "isSupported"),
+    replace=TRUE)
```

Note that the data visualizations in the report 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"),
+    replace=TRUE)
```

In summary, and as indicated in Figure 3, 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.

The package contains a vignette that explains the details of how to modify and expand the possibilities by producing new summary, visual, and check functions.

One might also encounter datasets with variables that are not among the 7 classes mentioned in the above (`character`, `Date`, `factor`, `integer`, `labelled`, `logical` and `numeric`), for instance variables of type `complex` or user-defined classes. It is possible to tell `makeDataReport()` how to handle such variables by use of the argument `treatXasY`. This argument takes a list where the names correspond to "new" variable types (`X`), while the entries must be supported variable types (`Y`). For instance, we can instruct `dataMaid` to treat complex variables as numerics and generate a data report for a type `complex` variable like this:

```
R> complexData <- data.frame(complexVar = complex(100, real = 1:100,
+    imaginary = 3), numericVar = 1:100)
R> makeDataReport(complexData, treatXasY=list(complex = "numeric"),
+    replace=TRUE)
```

In this report, we will find that the two variables, `complexVar` and `numericVar` will be have identical presentations in the variable list, as treating a `complex` variable as a `numeric` means dropping the imaginary part of the complex numbers which was the only thing setting the two variables apart in the first place.

3. Using dataMaid interactively

While overview documents are great for presenting and documenting the data at various stages of the data cleaning process, it may be useful to be able to work more interactively when performing actual data cleaning. Aside from the `makeDataReport()` 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 `events`. 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$events)

$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 `checks` argument and the `setChecks()` helper function to specify this. 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$events, checks = setChecks(numeric = "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$events)
```

```
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> missEvents <- identifyMissing(toyData$events)
R> str(missEvents)
```

```
List of 3
 $ problem      : logi TRUE
 $ message      : chr "The following suspected missing value codes enter as regular values
 $ 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 entry `problemValues`. If we decide that the values `999` and `NaN` in `events` are in fact miscoded missing values, we can easily replace them with `NA`s:

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

```
No problems found.
```

Similarly, the `visualize()` and `summarize()` functions can be used to run the corresponding visualizations and summaries for each variable. See Figure 4 for the visualization output.

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

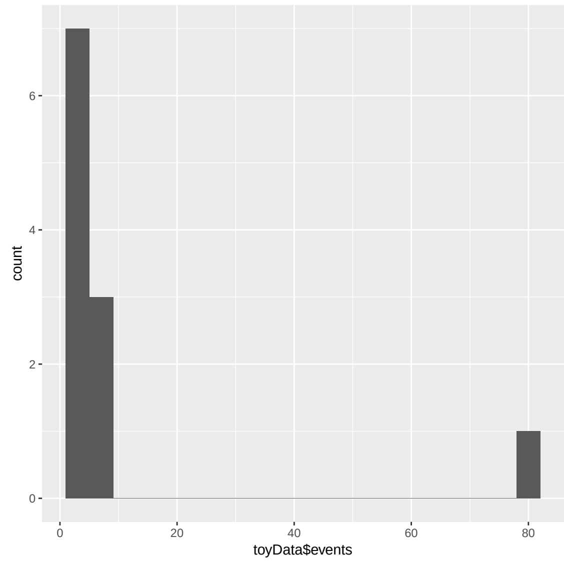


Figure 4: Output from running `visualize()` on variable `events` from the `toyData` dataset.

```
$variableType
Variable type: numeric
$countMissing
Number of missing obs.: 4 (26.67 %)
$uniqueValues
Number of unique values: 6
$centralValue
Median: 4
$quartiles
1st and 3rd quartiles: 1.5; 6
$minMax
Min. and max.: 1; 82
```

As we saw with the `check()` function, the summary can be modified by using the `summaries` argument and the `setSummaries()` helper function. If we want to remove the default summaries `variableType` and `countMissing` for numeric variables, we can use the function `defaultNumericSummaries()` and its argument `remove` that excludes a vector of summaries from the usual default summaries:

```
R> summarize(toyData$events,
+    summaries = setSummaries(
+    numeric = defaultNumericSummaries(remove = c("variableType",
+    "countMissing"))))

$uniqueValues
Number of unique values: 6
$centralValue
```

```
Median: 4
$quartiles
1st and 3rd quartiles: 1.5; 6
$minMax
Min. and max.: 1; 82
```

The syntax in this code chunk can be read as follows: "Summarize `events` in `toyData`, and for `numeric` variables, set the summaries to be the default summary functions, except `variableType` and `countMissing`."

Similar `defaultXXXSummaries()` functions are available for the other supported variable classes. For checks, the same syntax can also be used, but the helper functions are now named `defaultXXXChecks` with `XXX` as a placeholder for a supported variable class.

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. A worked example: Dirty presidents

We will now put the bits and pieces from above together and show how `makeDataReport()` can be used on a less artificial dataset to create a useful overview report and how the interactive tools can subsequently be used to assist the actual data cleaning process. More specifically, we will create a report describing the `presidentData` dataset, which is available in **dataMaid** and use the information from this report to clean up the data. `presidentData` is a slightly mutilated dataset with information about the 45 first US presidents, but with a few common data issues and a blind passenger. The dataset contains one observation per president and has the following variables:

`lastName` The last name of the president.

`firstName` The first name of the president.

`orderOfPresidency` The number in the order of presidents.

`birthday` The birthday of the president.

`stateOfBirth` The state in which the president was born.

`assassinationAttempt` Was there an assassination attempt on the president?

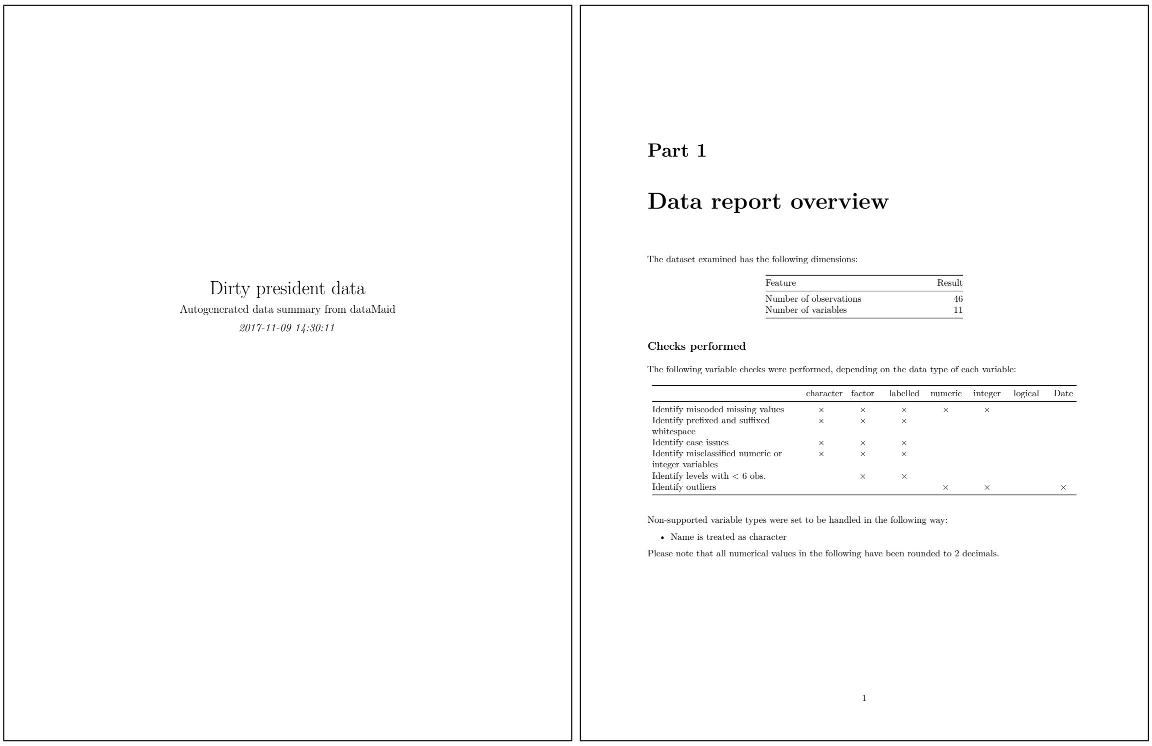
`sex` The sex of the president.

`ethnicity` The ethnicity of the president.

`presidencyYears` The duration of the presidency.

`ageAtInauguration` The age of the president at inauguration.

`favoriteNumber` The favorite number of the president (fictional).



Dirty president data

Autogenerated data summary from dataMaid

2017-11-09 14:30:11

Part 1

Data report overview

The dataset examined has the following dimensions:

Feature	Result
Number of observations	46
Number of variables	11

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 case issues	×	×	×				
Identify misclassified numeric or integer variables	×	×	×				
Identify levels with < 6 obs.		×	×				
Identify outliers				×	×		×

Non-supported variable types were set to be handled in the following way:

- Name is treated as character

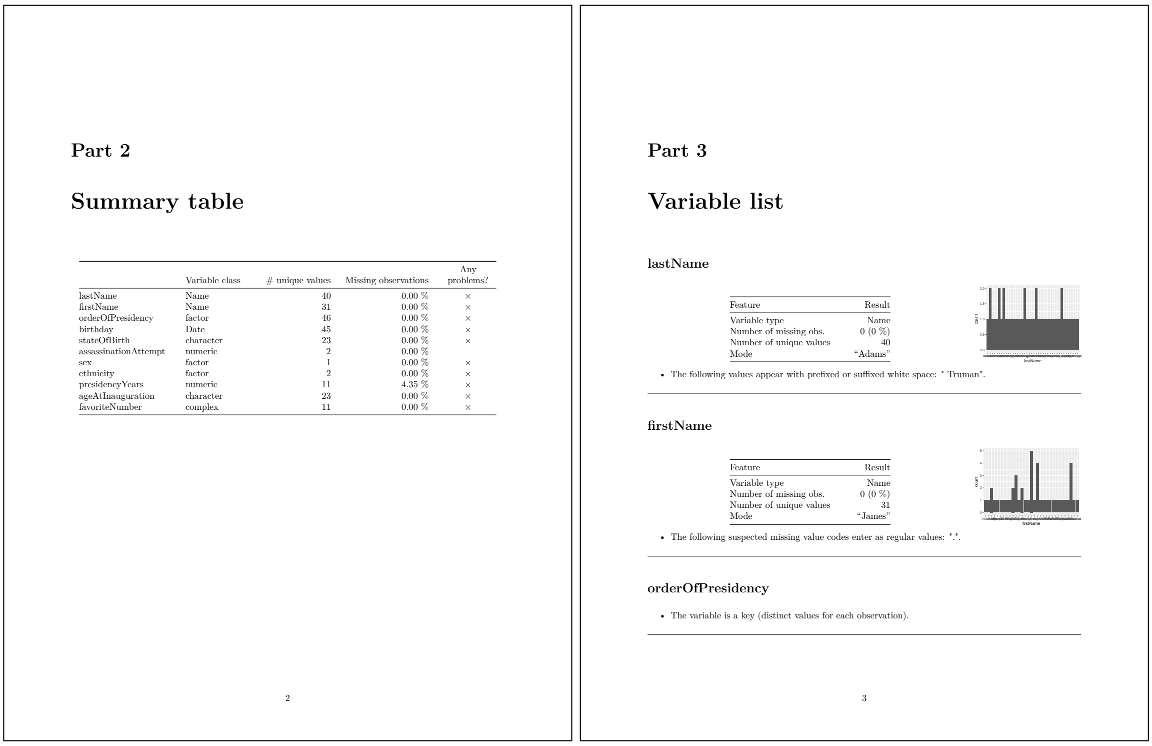
Please note that all numerical values in the following have been rounded to 2 decimals.

1

Figure 5: The front page and the first page of the data overview report for the `presidentData` dataset. Note that the report title has been customized (front page), `identifyLoners` has been removed from the checks performed on character variables ("Identify levels with <6 obs." is not checked for character variables in the table on page 1) and that variables of class `Name` have been set to be treated like `character` variables (page 1). Larger versions of the pages can be seen in Appendix B.

```
R> data(presidentData)
R> head(presidentData)
```

```
    lastName firstName orderOfPresidency   birthday  stateOfBirth
1 Washington    George                 1 1732-02-22      Virginia
2      Adams      John                 2 1735-10-30 Massachusetts
3  Jefferson    Thomas                 3 1743-04-13      Virginia
4    Madison     James                 4 1751-03-16      Virginia
5     Monroe     James                 5 1758-04-28      Virginia
6      Adams      John                 6 1767-07-11 Massachusetts
  assassinationAttempt  sex ethnicity presidencyYears
1                    0 Male Caucasian               7
2                    0 Male Caucasian               3
3                    0 Male Caucasian               8
4                    0 Male Caucasian               8
5                    0 Male Caucasian               8
6                    0 Male Caucasian               4
```


Part 2

Summary table

	Variable class	# unique values	Missing observations	Any problems?
lastName	Name	40	0.00 %	×
firstName	Name	31	0.00 %	×
orderOfPresidency	factor	46	0.00 %	×
birthday	Date	45	0.00 %	×
stateOfBirth	character	23	0.00 %	×
assassinationAttempt	numeric	2	0.00 %	
sex	factor	1	0.00 %	×
ethnicity	factor	2	0.00 %	×
presidencyYears	numeric	11	4.35 %	×
ageAtInauguration	character	23	0.00 %	×
favoriteNumber	complex	11	0.00 %	×

2

Part 3

Variable list

lastName

Feature	Result
Variable type	Name
Number of missing obs.	0 (0 %)
Number of unique values	40
Mode	“Adams”

- The following values appear with prefixed or suffixed white space: " Truman".

firstName

Feature	Result
Variable type	Name
Number of missing obs.	0 (0 %)
Number of unique values	31
Mode	“James”

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

orderOfPresidency

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

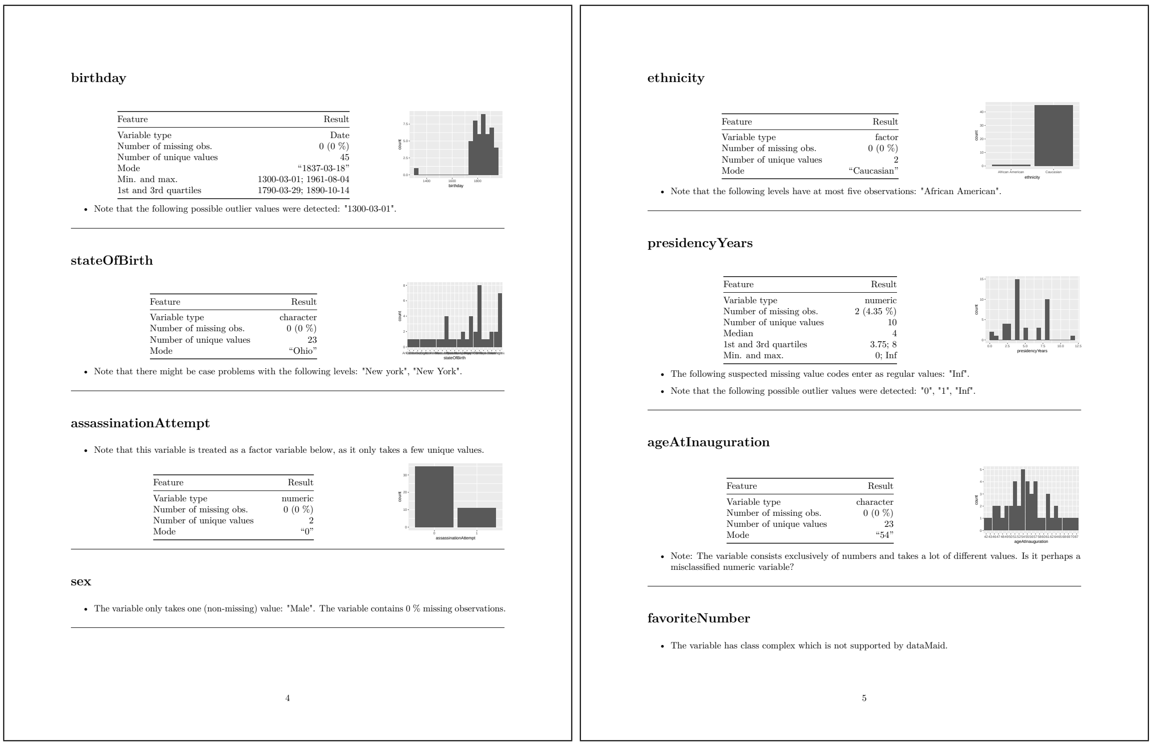
3

Figure 6: The second and third pages of the `presidentData` data report. We see that there are two `Name` variables in the overview on page 2 and see that these variables are indeed treated as `character` variables on page 3, as specified in the `makeDataReport` call by use of the `treatXasY` argument. Larger versions of the pages can be seen in Appendix B.

```
  ageAtInauguration favoriteNumber
1                57    3+0.000000i
2                61    4+0.000000i
3                57    0+1.414214i
4                57   10+0.000000i
5                58    3+0.000000i
6                57    9+0.000000i
```

We discuss the results of a data overview report generated for this dataset below, but first there are a few special features of the dataset and wishes for the data report that require us to customize it using some of the arguments for `makeDataReport`. We have the following points of interest:

1. A couple of variables are used to store names, namely `lastName` and `firstName`. In order to use special bibliographical analysis tools on these, and only these, variables, it might be convenient to assign them a special class. Therefore, these variables have been set to have class `Name` by use of the base R function `class()`. When we wish to make a data report for the dataset, we have to tell `makeDataReport()` how to handle such `Name`-type variables, as they are not among the supported variable types mentioned in the above. This can be done using the `treatXasY` argument.



birthday

Feature	Result
Variable type	Date
Number of missing obs.	0 (0 %)
Number of unique values	45
Mode	“1837-03-18”
Min. and max.	1300-03-01; 1961-08-04
1st and 3rd quartiles	1790-03-29; 1890-10-14

- Note that the following possible outlier values were detected: "1300-03-01".

stateOfBirth

Feature	Result
Variable type	character
Number of missing obs.	0 (0 %)
Number of unique values	23
Mode	“Ohio”

- Note that there might be case problems with the following levels: "New york", "New York".

assassinationAttempt

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

Feature	Result
Variable type	numeric
Number of missing obs.	0 (0 %)
Number of unique values	2
Mode	“0”

sex

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

4

ethnicity

Feature	Result
Variable type	factor
Number of missing obs.	0 (0 %)
Number of unique values	2
Mode	“Caucasian”

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

presidencyYears

Feature	Result
Variable type	numeric
Number of missing obs.	2 (4.35 %)
Number of unique values	10
Median	4
1st and 3rd quartiles	3.75; 8
Min. and max.	0; Inf

- The following suspected missing value codes enter as regular values: "Inf".
- Note that the following possible outlier values were detected: "0", "1", "Inf".

ageAtInauguration

Feature	Result
Variable type	character
Number of missing obs.	0 (0 %)
Number of unique values	23
Mode	“54”

- Note: The variable consists exclusively of numbers and takes a lot of different values. Is it perhaps a misclassified numeric variable?

favoriteNumber

- The variable has class complex which is not supported by dataMaid.

5

Figure 7: Part of the data overview report generated for the `presidentData` dataset. Here, we see pages 4 and 5 from the *Variable list*. Larger versions of the pages can be seen in Appendix B.

2. We use the `character` class for a few categorical variables that have a lot of different levels, but where it is not a data mistake that these levels each generally have very few observations, e.g. the variable `stateOfBirth`. Therefore, we would like to disable the `identifyLoners` check (which flags variables with <6 observations in any of the levels) for this variable type. This can be done using the `checks` argument.

3. We would like the report to be called ”Dirty president data” in order to reflect that it is, indeed, a report concerning dirty data about presidents.

Incorporating these three customizations, we can load the data and generate a report by calling:

```
R> makeDataReport(presidentData, replace = TRUE,
+    treatXasY = list("Name" = "character"),
+    checks = setChecks(character =
+    defaultCharacterChecks(remove = "identifyLoners")),
+    reportTitle = "Dirty president data")
```

The first four pages of the resulting report can be inspected in Figures 5 and 6. Note that all the customized settings can be identified from the first two pages, without having to read through the report: The new title is written on the front page, the check settings are displayed in the *Checks performed* table and the strategy for handling `Name` variables is documented below this table. The full data report, except for the front page, is available in Appendix B.

The first problem that can be spotted from these first four pages is the surprising number of observations: Anno 2017, there have only been 45 US presidents. Therefore, having 46 observations reveal that the dataset contains a blind passenger. For instance, if the dataset was constructed as a subset of a more general "World leaders" dataset, this type of problem could occur due to wrongful nationality classification. We return to the extra president issue below.

On page 3, we see the contents for the three first variables. Here, we identify a prefixed white space in the last name entry for President Truman and we find that a dot was entered as a first name; this is a typical choice for coding missing values in e.g. Stata, and therefore, it is flagged as a potential miscoded missing value. The variable `orderOfPresidency` is not summarized, visualized or checked because it is categorical and contains unique values for each observation.

Figure 7 presents the remaining two pages with variable presentations. On pages 4 and 5, we find a few remarks:

- In the `birthday` variable, there is an entry with the date March 1, 1300 which is a bit of an outlier.
- Among the states in which the presidents were born, New York was mistakenly spelled with a lower case "Y" in at least one entry.
- The variable concerning assassination attempts is coded as a numeric, but the default `smartNum = TRUE` setting of `makeDataReport()` implies that such a numeric variable with only a few (less that 5) unique values is treated as a factor variable in the data report, thereby providing more relevant summaries, visualizations and checks. This is remarked in the variable presentation for `assassinationAttempt` and it can also be seen by the visualization being a barplot rather than a histogram.
- The variable concerning the sex of the president was skipped, as there is nothing to present when all US presidents have been male so far.
- The report flags the variable `ethnicity` to have suspiciously few observations in one category, "African American".
- A few presidents were found to have odd values in the variable describing the duration of their presidencies: Some had very short (outlier) presidencies of less than two years and one was registered to have an *infinite* presidency.
- The variable concerning age at inauguration was coded as a character variable, but consists exclusively of numbers and takes a lot of different values and therefore, it was flagged as a potentially misclassified numeric variable.
- It seems as if one or more presidents have complex numbers as their favorite numbers. As `complex` is not a supported variable type in `dataMaid` and no strategy for handling this class was provided in the `treatXasY` argument, the variable is simply flagged as non-supported.

A lot of these mistakes are easily fixable, and we will do so below. However, some of them require more delicate knowledge of the subject matter. For instance, `ethnicity` is very reasonably marked as a potentially problematic variable as it includes only a single observation of "African American". However, a human reading this report will know that this does *not* reflect a mistake in the data, but rather a peculiarity in the real world, and as such, it should not be cleaned out.

A few of the identified problems have easy fixes that need no further discussion. We remove the prefixed white space from Truman's name, fix the misspelling of New York, convert the binary variable `assasinationAttempt` to a factor and change the class of the `ageAtInauguration` variable to numeric:

```
R> presidentData$lastName[presidentData$lastName == " Truman"] <- "Truman"
R> presidentData$stateOfBirth[presidentData$stateOfBirth == "New york"] <-
+    "New York"
R> presidentData$assassinationAttempt <-
+    factor(presidentData$assassinationAttempt)
R> presidentData$ageAtInauguration <-
+    as.numeric(presidentData$ageAtInauguration)
```

Please note that if `ageAtInauguration` had been a `factor` rather than a `character` variable, an additional inner call should be added in order to ensure no conversion issues:

```
R> presidentData$ageAtInauguration <-
+    as.numeric(as.character(presidentData$ageAtInauguration))
```

Moving forward, we might be interested in inspecting the contents of the "."-coded entry of `firstName` closer, as we *do* know the first names of all the US presidents. We look at the last name for this president and fill in the first name correctly:

```
R> presidentData$lastName[presidentData$firstName == "."]

[1] "Trump"

R> presidentData$firstName[presidentData$firstName == "."] <- "Donald"
```

Next up is the unlikely US president birthday of March 1, 1300. In order to understand if this is a generally problematic observation, or if it is just a mistyped observation, we inspect the full data entry for this person. We can do this using the usual R selection syntax as above, or we can use the value of a `identifyOutliers` call to select this observation:

```
R> birthdayOutlierVal <-
+    identifyOutliers(presidentData$birthday)$problemValues
```

Now, we have the outlier birthday stored and can use it to select and print the appropriate observation in the dataset:

```
R> presidentData[presidentData$birthday == birthdayOutlierVal, ]

      lastName firstName orderOfPresidency   birthday stateOfBirth
46 Arathornson   Aragorn                 0 1300-03-01       Gondor
   assassinationAttempt  sex ethnicity presidencyYears
46                    1 Male Caucasian              NA
   ageAtInauguration favoriteNumber
46                87           8+0i
```

We see that this is not a proper US president and thus, it is likely to be the explanation of the faulty number of observations in the dataset. Therefore, we drop this observation from the dataset, e.g. by overwriting the dataset with a selection of all the other observations:

```
R> presidentData <-
+    presidentData[presidentData$birthday != birthdayOutlierVal, ]
```

Now, all that is left to fix is the `presidencyYears` variable. For this variable, we are concerned about three different things: First, it has some quite small outlier values. We need to identify whether these are really true. Secondly, one president is registered to have an infinite presidency, this should also be fixed. Third, we see from the summary table that there are two missing observations for this variable. One might have been fixed by removing Aragorn from the dataset, so we start by inspecting if this is indeed the case by calling `summarize()` interactively:

```
R> summarize(presidentData$presidencyYears)

$variableType
Variable type: numeric
$countMissing
Number of missing obs.: 1 (2.22 %)
$uniqueValues
Number of unique values: 10
$centralValue
Median: 4
$quartiles
1st and 3rd quartiles: 3.75; 8
$minMax
Min. and max.: 0; Inf
```

We see that there is one less missing value and that the small and large values pertain. Therefore, we look at all the observations that cause worry, namely the outliers and the missing value, and we select to see the variables `firstName`, `lastName` and `presidencyYears`:

```
R> presidentData[is.na(presidentData$presidencyYears) |
+    presidentData$presidencyYears %in%
+    identifyOutliers(presidentData$presidencyYears)$problemValues,
+    c("firstName", "lastName", "presidencyYears")]
```

```
   firstName lastName presidencyYears
9    William Harrison               0
12   Zachary   Taylor               1
20     James Garfield               0
44    Barack    Obama             Inf
45    Donald    Trump              NA
```

We see that Obama is listed as being president forever, which history has proven to be wrong. Trump, on the other hand, has a missing value for his presidency duration, which is in fact reasonable as we cannot know how long it will be yet (anno 2017). Presidents William Harrison, Zachary Taylor and James Garfield were identified to have very brief presidencies, but these are not mistakes, as any US history textbook can tell us. Thus, the only mistake left to fix is Obama's infinite presidency:

```
R> presidentData$presidencyYears[presidentData$lastName == "Obama"] <- 8
```

This does not mean we are necessarily done with the data cleaning process: There might be problems that **dataMaid** was not able to identify. But we have fixed some key issues in the data and thereby given ourselves a chance of a smoother sailing in the next steps of the data analysis.

Finally, we create a new data report, adding the suffix "cleaned" to the title, as well as the file name, so that we have documentation of the current state of the dataset. We also decide that it might be sufficient to be able to inspect only the real part of the presidential favorite numbers and therefore, we choose to treat the `complex` variable `favoriteNumber` as a `numeric` variable:

```
R> makeDataReport(presidentData, vol = "_cleaned",
+    treatXasY = list(Name = "character", complex = "numeric"),
+    checks = setChecks(character =
+    defaultCharacterChecks(remove = "identifyLoners")),
+    reportTitle = "Dirty president data - cleaned",
+    replace=TRUE)
```

This will create a new data report stored in the file `dataMaid_presidentData_cleaned.pdf`.

5. Rubbing down data cleaning challenges

Finally, we present a few examples of how to make **dataMaid** solve specific issues related to data documentation and -cleaning. First, we discuss how the data report generation functions of **dataMaid** can be used in a data science workflow where one is not necessarily interested in inspecting the results right away and most commands are run automatically. 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.

5.1. Incorporating dataMaid in automated workflow

The default settings of `makeDataReport()` have been set to facilitate easy and quick data report generation, but unfortunately, this also means that it is not ideal for a more programming-oriented workflow, where the function might not be called by a human. For instance, one might be interested in automatically running `makeDataReport()` on all datasets received from a certain client, perhaps via email or through a web upload, and returning a data report for them to inspect and comment before ever looking at the data. In this scenario, there are a few issues with the standard data report:

1. After rendering, the report is automatically opened. This is not very useful, if the processes are supposed to run in the background.
2. The report generation writes messages to the console while producing and rendering the report.
3. Unless specifically told otherwise, every report created for the same dataset (or different datasets with the same storage name in R) will have the same file name.

Please note that the data report does contain information about who, when and how concerning its generation, so even though the default choices for file names do not make it easy to tell different reports for the same dataset apart, it should be rather easy when inspecting the report manually.

The three problems can easily be solved by use of the arguments of `makeDataReport()`. Whether or not the outputted file is opened can be controlled through the argument `open`. How much information is printed in the console can be adjusted by using the argument `quiet`. And conveniently introducing small alterations of the file names can be obtained by use of the `vol` argument. For instance, we can make a data report for `toyData` that is not opened automatically, produces no output to the console and includes the date and time of its creation in the file name:

```
R> makeDataReport(toyData, open = FALSE, quiet = "silent",
+    vol = paste("_", format(Sys.time(), "%m-%d%-%y_%H.%M"), sep = ""))
```

Now, if e.g. the report is created at 3 pm on October 31, 2017, its file will be named `dataMaid_toyData_10-31-2017_15.00.pdf`, making it easy to find.

5.2. Using dataMaid for problem flagging

If the dataset is large and the time available for reading through the data report is scarce, it can be convenient to only make a report concerning the variables that were flagged to be problematic. This can be achieved by using the `onlyProblematic` argument for `makeDataReport()`. 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 using the `mode` argument:

```
R> makeDataReport(toyData, onlyProblematic = TRUE, mode = "check", replace=TRUE)
```

Now only the checking results are printed, and only for variables where problems were identified. An even more minimal output can be obtained directly in the console by using the `check()` function interactively. When called on a `data.frame`, this function produces a list (of variables) of lists (of checks) of lists (or rather, `checkResults`). Thus, the overall problem status of each variable can easily be unraveled 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)
```

```
       pill      events      region      change          id
       TRUE        TRUE        TRUE        TRUE        TRUE
spotifysong
      FALSE
```

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

5.3. Including dataMaid reports in other files

Sometimes, a **dataMaid** report 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=FALSE` argument in `makeDataReport()` 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. Note that the `word` output option is based on the `html` markdown style.

If it is important that the embedded **dataMaid** document can be rendered to any of these three file types, we recommend setting the `twoCols = FALSE` and `output = "html"` arguments in `makeDataReport()`. This essentially removes 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> makeDataReport(toyData, standAlone = FALSE)
```

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

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

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

Alternatively, one can create the usual output report, not render it and then manually edit the produced R markdown file as wished. This can be done with the following command:

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

After editing, the file can be rendered by calling the `render` function:

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

6. Concluding remarks

In this paper we have introduced the R package **dataMaid** for performing reproducible error detection, data overview reports and data screening. 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.

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 is currently being developed such that non-R-savvy users can upload their data online and get a data cleaning document.

Acknowledgements

This work was supported by The Lundbeck Foundation, Trygfonden and The Region of Southern Denmark.

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.

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.

Cotton R (2016). ***assertive**: Readable Check Functions to Ensure Code Integrity.* R package version 0.3-5, URL https://CRAN.R-project.org/package=assertive.

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. Data report for the toyData dataset

Part 1

Data report overview

The dataset examined has the following dimensions:

Feature	Result
Number of observations	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

Summary table

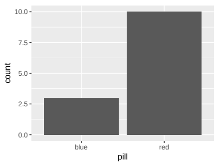
	Variable class	# unique values	Missing observations	Any problems?
pill	factor	3	13.33 %	×
events	numeric	9	20.00 %	×
region	factor	7	0.00 %	×
change	numeric	15	0.00 %	×
id	factor	15	0.00 %	×
spotifysong	factor	1	0.00 %	×

Part 3

Variable list

pill

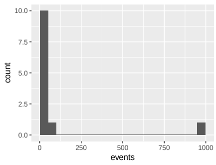
Feature	Result
Variable type	factor
Number of missing obs.	2 (13.33 %)
Number of unique values	2
Mode	"red"



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

events

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".

B. Data report for the presidentData dataset

Part 1

Data report overview

The dataset examined has the following dimensions:

Feature	Result
Number of observations	46
Number of variables	11

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 case issues	×	×	×				
Identify misclassified numeric or integer variables	×	×	×				
Identify levels with < 6 obs.		×	×				
Identify outliers				×	×		×

Non-supported variable types were set to be handled in the following way:

- Name is treated as character

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

Part 2

Summary table

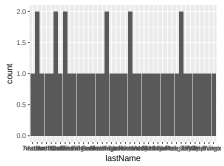
	Variable class	# unique values	Missing observations	Any problems?
lastName	Name	40	0.00 %	×
firstName	Name	31	0.00 %	×
orderOfPresidency	factor	46	0.00 %	×
birthday	Date	45	0.00 %	×
stateOfBirth	character	23	0.00 %	×
assassinationAttempt	numeric	2	0.00 %	
sex	factor	1	0.00 %	×
ethnicity	factor	2	0.00 %	×
presidencyYears	numeric	11	4.35 %	×
ageAtInauguration	character	23	0.00 %	×
favoriteNumber	complex	11	0.00 %	×

Part 3

Variable list

lastName

Feature	Result
Variable type	Name
Number of missing obs.	0 (0 %)
Number of unique values	40
Mode	“Adams”

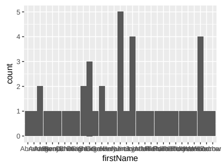


- The following values appear with prefixed or suffixed white space: " Truman".

---

firstName

Feature	Result
Variable type	Name
Number of missing obs.	0 (0 %)
Number of unique values	31
Mode	“James”



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

---

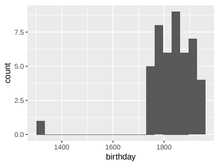
orderOfPresidency

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

---

birthday

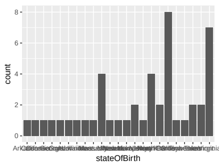
Feature	Result
Variable type	Date
Number of missing obs.	0 (0 %)
Number of unique values	45
Mode	“1837-03-18”
Min. and max.	1300-03-01; 1961-08-04
1st and 3rd quartiles	1790-03-29; 1890-10-14



- Note that the following possible outlier values were detected: "1300-03-01".

stateOfBirth

Feature	Result
Variable type	character
Number of missing obs.	0 (0 %)
Number of unique values	23
Mode	“Ohio”

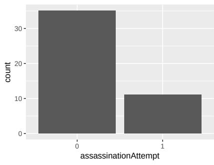


- Note that there might be case problems with the following levels: "New york", "New York".

assassinationAttempt

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

Feature	Result
Variable type	numeric
Number of missing obs.	0 (0 %)
Number of unique values	2
Mode	“0”



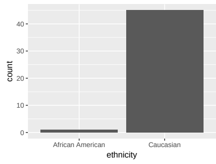
sex

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

4

ethnicity

Feature	Result
Variable type	factor
Number of missing obs.	0 (0 %)
Number of unique values	2
Mode	"Caucasian"

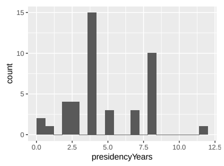


- Note that the following levels have at most five observations: "African American".

---

presidencyYears

Feature	Result
Variable type	numeric
Number of missing obs.	2 (4.35 %)
Number of unique values	10
Median	4
1st and 3rd quartiles	3.75; 8
Min. and max.	0; Inf

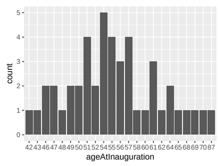


- The following suspected missing value codes enter as regular values: "Inf".
- Note that the following possible outlier values were detected: "0", "1", "Inf".

---

ageAtInauguration

Feature	Result
Variable type	character
Number of missing obs.	0 (0 %)
Number of unique values	23
Mode	"54"



- Note: The variable consists exclusively of numbers and takes a lot of different values. Is it perhaps a misclassified numeric variable?

---

favoriteNumber

- The variable has class complex which is not supported by dataMaid.

5

Affiliation:

Claus Thorn Ekstrøm
Section of 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