meta.data() R package usage vignette

Brett Buttliere

2023-11-10

Variable level meta.data as the core problem and the core solution

Data become available online, but without readable, and ultimately standard meta.data, repositories will remain unsearchable and major reuse will be inhibited, because of the need to recode and harmonize the different sets. Especially with the development of over 2,000 data repositories in the last years (re3data.org), the need for a way to systematically search across these to find useful instances of variables becomes high. Having clarity at the variable level, especially by using standards so that the computer can identify them, is probably the best and most useful way of solving these problems. Using standard templates for the meta.data makes it easier for the researcher, since they don't have to create their own, and it allows for e.g., community standard analysis and plot scripts, which ease the work enormously, even while making the work more transparent and reusable.

The essential problem is that variables are not indicated systematically in data sets (Buttliere, 2021). This means that even if you and I are studying the 'same things', e.g., a standard scale, we cannot exchange data sets or analysis scripts without having to recode one or the other. Additionally, data.sets cannot be systematically identified in repositories, because the questions have different labels in the data.sets. If variables are indicated systematically, then data sets are clear, analysis scripts are clear, and one can e.g., search for all data.sets containing a particular variable name by searching for it and then apply community standard data scripts to see how e.g., results change (forthcoming).

A simple demonstration and proposed solution using the 'gender' variable.

The difficulty in understanding how variables are coded into the data set can be seen in even such a 'simple' variable as gender. Looking at any number of data sets one will find different ways of referencing gender e.g., var1, V25, sex, SEX, mf, Geschlecht, and many other abbreviations in different languages.

This is only the variable label, and does not consider yet how the variable is coded in terms of whether males are 1 or 2, M & F, M & W, whether there are other labels afforded, e.g., other, they/them, unidentified, or any of the many other potential coding schemes.

The problem is not that people don't agree on one single way to code it, but that there is not a clear way to indicate which coding scheme one is using (e.g., through a standard variable label or 'VOI'). Once one has several agreed upon schemes and templates to apply, translators can be built. The idea here is to build these templates and standards, making them available in an online library that can be pulled from, within R, and then applied to the variables in your data set. This then also opens e.g., standard analysis scripts and searchable repositories (Buttliere, 2021).

The meta.data R package solves these problems/ affords these possibilities

The meta.data() R package, which this vignette demonstrates, solves these problems by enabling researchers to **create.meta(), write.meta(), and apply.meta() data templates onto variables.**

These templates indicate what the variable is and how it is coded in the data set, and in turn allow e.g., standard analysis scripts, since recoding can be avoided. *I believe this holds fantastic potential for communities that use the same or similar variables like sets of demographic variables or standard scales in the social*

sciences. These templates can be stored online, pulled into a programming environment and applied onto variables. **It makes work easier, even while being more transparent and reusable.**

These templates can include not only scales, but also experimental variables like condition, trial number, participant id, measurement number, or anything else. Once again, there is no need to get everyone using one single standard, and different versions of variables (e.g., height, gender) are simply different specified templates which can have translators between them.

##meta.data templates make work easier and more reusable Utilizing meta.data templates *makes it easier for data creators* because they do not need to create a new meta.data and create it, while also allowing them to plug into these standard analysis and plot scripts, which saves analysis effort. Using the templates also *makes it easier for data reusers* to understand and use the data for e.g., meta-analysis, since the data can be consistently identified and do not have to be recoded. Currently, meta-analyses focus on effect size estimates, because it is so hard to recode the raw data (e.g., metaphor package). But in a world of standard variable labels and AI we can already imagine a machine that can search for all instances of e.g., the Rosenberg Self-Esteem scale which also contains e.g., location and get a premelded data set back. It could be quite useful, i think.

The **long term goal of the project is to maintain an online library** with such integration that e.g., Qualtrics can create a pre-meta.data'd question in their software given a variable or scale label (VOI). This begins with researchers establishing standard variable labels for at least newly published scales, as well researchers in general adopting better data set and analysis script hygiene norms (Buttliere, 2021). In this vignette, the basic use of the meta.data package is demonstrated.

Getting started with the meta.data() R package.

The meta.data() package begins this quest by enabling users to create.meta() data templates, write.meta() templates out to e.g., a meta.data file, and apply.meta() data templates that one could get online onto variables in one's data sets.

The package is currently on Github (https://github.com/NabuKudurru/meta.data), which means you will have to use the install_github() function instead of the install.packages() function, as shown below. To use install_github(), you will need devtools(), which is another package for R packages in development. The following code should manage this - remove the '#'s to run this code if you do not have e.g., devtools or meta.data downloaded or installed yet. Sometimes you might need to update some of the packages that meta.data() uses (dependencies), if they are older versions.

```
#install.packages('devtools')
#library(devtools)

#install_github('NabuKudurru/meta.data')
library(meta.data)
```

From here you should be able to access the function help pages, and find it in your packages library.

Example dataset: Iris in base r

To demonstrate the problem and solution that the meta.data() R package presents, we will use the Iris data set, which is often used to teach R (https://rpubs.com/moeransm/intro-iris).

If you have ever used this data.set, it is likely you have asked e.g., what exactly are the variables, who collected the data, what does the scale mean, and is there some paper describing with these data? These are all good questions, and they arise because of the lack of meta.data associated with the variables in the data.set, so we provide them here.

Wikipedia has an excellent article about the iris data set, but it is not linked anywhere to the actual data. It describes how the Iris data set was introduced by Ronald Fisher to example Linear Discriminant Analysis (Fisher, 1936). It describes how the data set has four variables about three types of Irises. The metrics are

indicated in centimeters, the sepal is the green petal like part below the petal, and the petal is the colored part of the flower.

So, let's use the meta.data() R package to indicate more completely what these variables mean. Since there is no existing meta.data for the variable, we first need to create.meta().

create.meta() creates new meta.data template about a given variable.

Since there is no existing meta.data for the variables in the Iris data.set, let's set some using the create.meta() function, which allows us to indicate the following attributes:
long.label - a long label, for searching the library/ repository with, unique to it, a 'VOI'.
full.text – the full specific text of the question or indication of the metric.
value.labels – what the content of the variable means, scale points and labels, type.
missing – what is the missing indicator (often but not always NA or NaN).
associated.ids – dois, ORCIDs, UniIDs, GrantIDs, ScriptIDs, etc.
comments – a general catchall into which one can implement any comments, an ISO standard, or any other thing, link, or etc.

Essentially what this package does is make it easier and more systematic to assign 'attributes' to variables, as well as a way to apply e.g., templates.

Below you see first the original variable iris$Sepal.Length, line 2 creates a copy of the variable to be able to compare them later, then we use the create.meta() function. Each attribute has its own line in order to increase readability, but one can use all some or only one e.g., comments.

```
iris$Sepal.Length
#>   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
#>  [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0
#>  [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5
#>  [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
#>  [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5
#>  [91] 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
#> [109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2
#> [127] 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
#> [145] 6.7 6.7 6.3 6.5 6.2 5.9
iris.1 <- iris
iris.1$Sepal.Length <- create.meta(iris$Sepal.Length,
    long.label = 'iris.Sepal.Length',
    full.text = 'Sepal.Length is an indicator, in cm, of the length of the sepal, which are the green l
    value.labels = 'cm',
    missing = 'NA' ,
    associated.ids = 'http://dx.doi.org/10.1111/j.1469-1809.1936.tb02137.x',
    comments = 'this is an example of more detailed meta.data, one could implement any other meta.data
#Below is the edited variable with the meta.data attached. You can see there are no differences
iris.1$Sepal.Length
#>   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
#>  [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0
#>  [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5
#>  [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
#>  [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5
#>  [91] 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
#> [109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2
#> [127] 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
#> [145] 6.7 6.7 6.3 6.5 6.2 5.9
#> attr(,"variable.label")
#> [1] "Sepal.Length"
```

```
#> attr(,"long.label")
#> [1] "iris.Sepal.Length"
#> attr(,"full.text")
#> [1] "Sepal.Length is an indicator, in cm, of the length of the sepal, which are the green leaves sur
#> attr(,"value.labels")
#> [1] "cm"
#> attr(,"missing")
#> [1] "NA"
#> attr(,"associated.ids")
#> [1] "http://dx.doi.org/10.1111/j.1469-1809.1936.tb02137.x"
#> attr(,"comments")
#> [1] "this is an example of more detailed meta.data, one could implement any other meta.data or meta.
#You can compare iris.1 to the original iris$Sepal.Length by running it if you like.
mean(iris$Sepal.Length)#the means
#> [1] 5.843333
mean(iris.1$Sepal.Length)#are the same
#> [1] 5.843333
#As far as we know, it does not change any computation. If you find one it changes, please let us know!
```

Technically we do not need to create a new version of the variable, but we do need to assign the output of the function to something at the end, since it is only returned by the function. The function will automatically take the variable name and enter it as a variable label, which is important for combining sets of templates. We can enter any character vector we like into each of the sections, including other data sets which will be stored as meta.attributes of the variable e.g., particular other ISO standards in JSON variables.

You can see that iris.1$Sepal.Length has been assigned attributes 'attr' that correspond to the types of meta.data that are made available through the arguments of create.meta(). These are only example meta.data and they can in principle contain anything you would like, including any meta.data standard, so long as it can fit into a string and be parsed by a computer back into its native format, which is not that hard.

The final lines show that these meta.data attributes do not change the analysis of the variable, like the mean. We have not found any examples where it changes, but please let us know if you find one.

The code below creates meta.data for the other variables in the Iris data set.

```
iris.1$Sepal.Width <- create.meta(iris$Sepal.Width,
   full.text = 'Sepal.Width indicates the width of the sepal, the green leaves around the petals.',
   value.labels = 'cm',
   missing = 'NA' ,
   associated.ids = 'http://dx.doi.org/10.1111/j.1469-1809.1936.tb02137.x',
   comments = 'This is from  R.A. Fishers classic 1936 paper introducing ')

iris.1$Petal.Length <- create.meta(iris$Petal.Length,
   full.text = 'Petal.Length indicates the length of the petal, the delicate colored part of the flower
   value.labels = 'cm',
   missing = 'NA' ,
   associated.ids = 'http://dx.doi.org/10.1111/j.1469-1809.1936.tb02137.x',
   comments = 'This is from  R.A. Fishers classic 1936 paper introducing ')

iris.1$Petal.Width <- create.meta(iris$Petal.Width,
   full.text = 'Petal.Width indicates the width of the petal, the delicate colored part of the flower.'
   value.labels = 'cm',
   missing = 'NA' ,
   associated.ids = 'http://dx.doi.org/10.1111/j.1469-1809.1936.tb02137.x',
   comments = 'This is from  R.A. Fishers classic 1936 paper introducing ')
```

```
iris.1$Species  <- create.meta(iris$Species,
   full.text = 'Species indicates the type of Iris it is setosa, virginica, or versicolor.',
   value.labels = 'indicating the three plant genus (setosa, virginica, versicolor) ',
   missing = 'NA' ,
   associated.ids = 'http://dx.doi.org/10.1111/j.1469-1809.1936.tb02137.x',
   comments = 'this is an example of more detailed meta.data 5 ')
```

Now, all of the variables in our edited iris data set have meta.data about them. Still, that is a lot to have to write, especially if you have to do it for a larger data set. In the future, the idea is that you will be able to use the VOI to pull the template from the library and then apply.meta() as demonstrated below.

Even though the meta.data template library does not exist yet (we ask you to help us make it), we can still write out these meta.data and make them available at e.g., the data set level (i.e., to accompany a data.set), or at the scale level by selecting particular variables. For that, let's see the write.meta() function.

write.meta() writes out the meta.data about a single variable.

The write.meta() function writes out the meta.data for a single variable, and write.metas() demonstrated below, writes out a set of variables - for instance a particular scale or an entire data set. This is useful not only to communicate what is in the data.set, but also to e.g., make your templates available to other scholars to apply onto their data.sets.

The write.meta() function takes the meta.data for a single variable and writes it out to a list of its own, which can then be assigned to e.g., a data.frame of meta.data about the data.set.

This list of data frame can of course also be manipulated by its self or as part of a set. The input for the function is a variable name that contains some meta.data attributes. If the variable does not contain any attributes, it creates the attributes but assigns NA to them. The function basically collects the existing attributes and prints them as a list.

```
write.meta(iris.1$Sepal.Length)
#>   variable.label        long.label
#> 1   Sepal.Length iris.Sepal.Length
#>
#> 1 Sepal.Length is an indicator, in cm, of the length of the sepal, which are the green leaves surrou
#>   value.labels missing                                        associated.ids
#> 1            cm      NA http://dx.doi.org/10.1111/j.1469-1809.1936.tb02137.x
#>
#> 1 this is an example of more detailed meta.data, one could implement any other meta.data or meta.dat

meta.Sepal.Length <- write.meta(iris.1$Sepal.Length)
```

Remember that the first example will only print the meta.data to the console, and that one needs to assign it to a new object in the environment if one wants to do more than just look at it. Most of the time you will probably not be using single templates, but sets of templates that are part of e.g., a particular scale. Thus, we are introduced to the write.metas() function.

##write.metas() writes out the meta.data for a set of variables or entire data.set.

The write.metas() function simply goes through each variable in a defined list of variables, and applies the write.meta() function on to each variable, then combining each of these onto a meta.data data.frame. These variables do not necessarily need to even be in the same data.set, though this is not a recommended practice.

```
iris.1.metas <- write.metas(iris.1)
View(iris.1.metas)
#write.csv(iris.metas)
```

The above code creates a data.frame object that contains the meta.data for the iris data.set. This file can be about a scale or a data set and written out to the computer or anywhere, shared, read back in, and applied back onto the data set with apply.meta(). In the future we hope to have an online library that will be automatically indexed, but for now they can just be put into any repository.

##apply.meta() applies an existing template to a variable.

The data frame iris.1.metas of meta.data that we created with write.metas() will now be used as a set of templates that one can get imagine was gotten from anywhere e.g., from online. The idea is now to apply them with the apply.meta() function.

Remember that the original Iris data set is iris, our created set is iris.1, so now we will create iris.2, using the iris.1.metas which we can get from anywhere (e.g., our forthcoming integrated online library).

```
iris$Sepal.Length#there is no meta.data on the original variable.
#>   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
#>  [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0
#>  [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5
#>  [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
#>  [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5
#>  [91] 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
#> [109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2
#> [127] 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
#> [145] 6.7 6.7 6.3 6.5 6.2 5.9
meta.Sepal.Length #the single variable meta.data output from write.meta.
#>   variable.label        long.label
#> 1   Sepal.Length iris.Sepal.Length
#>
#> 1 Sepal.Length is an indicator, in cm, of the length of the sepal, which are the green leaves surrou
#>   value.labels missing                                        associated.ids
#> 1           cm      NA http://dx.doi.org/10.1111/j.1469-1809.1936.tb02137.x
#>
#> 1 this is an example of more detailed meta.data, one could implement any other meta.data or meta.dat
iris.1.metas[1,] #the first variable of the meta.data set from iris.
#>
#>
#>
#>
#>
#> "Sepal.Length is an indicator, in cm, of the length of the sepal, which are the green leaves surroun
#>
#>
#>
#>
#>
#>                                                                                          "http://dx.doi.
#>
#>                       "this is an example of more detailed meta.data, one could implement any other
#>
#>
#>
#>
```

Notice that we cannot call 'iris.1.metas$Sepal.Length' to view the meta.data for a variable because it is no longer a variable about a data point, it is now a data point about which there are variables. Thus the variables in iris.1.metas are things like 'variable.labels' or 'missing'

The meta.data template can be applied easily using the apply.meta() function. First, we will create another version of the iris dataset to edit, iris.2. From there, we can simply put the variable to be edited and then the template to apply as arguments.

```
iris.2 <- iris #create another copy of the variable.
iris.2$Sepal.Length <- apply.meta(iris.2$Sepal.Length, iris.1.metas[1,])#apply the template
iris.2$Sepal.Length#examine the new variable.
#>   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
#>  [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0
#>  [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5
#>  [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
#>  [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5
#>  [91] 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
#> [109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2
#> [127] 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
#> [145] 6.7 6.7 6.3 6.5 6.2 5.9
#> attr(,"variable.label")
#> [1] "original"
#> attr(,"long.label")
#> [1] "iris.Sepal.Length"
#> attr(,"full.text")
#> [1] "Sepal.Length is an indicator, in cm, of the length of the sepal, which are the green leaves sur
#> attr(,"value.labels")
#> [1] "cm"
#> attr(,"missing")
#> [1] "NA"
#> attr(,"associated.ids")
#> [1] "http://dx.doi.org/10.1111/j.1469-1809.1936.tb02137.x"
#> attr(,"comments")
#> attr(,"comments")[[1]]
#> template.2$comments[[1]]
```

The first argument is the variable we want to apply the template onto, and the second argument is the template to apply. You can easily compare iris, iris.1, and iris.2 to see that everything happened as it should. i.e., iris has no meta.data iris.1 and iris.2 should have the same meta.data.

```
iris$Sepal.Length# original data set, no meta.data.
#>   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
#>  [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0
#>  [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5
#>  [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
#>  [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5
#>  [91] 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
#> [109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2
#> [127] 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
#> [145] 6.7 6.7 6.3 6.5 6.2 5.9
iris.1$Sepal.Length# meta.data written manually.
#>   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
#>  [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0
#>  [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5
#>  [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
#>  [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5
#>  [91] 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
#> [109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2
#> [127] 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
```

```
#> [145] 6.7 6.7 6.3 6.5 6.2 5.9
#> attr(,"variable.label")
#> [1] "Sepal.Length"
#> attr(,"long.label")
#> [1] "iris.Sepal.Length"
#> attr(,"full.text")
#> [1] "Sepal.Length is an indicator, in cm, of the length of the sepal, which are the green leaves sur
#> attr(,"value.labels")
#> [1] "cm"
#> attr(,"missing")
#> [1] "NA"
#> attr(,"associated.ids")
#> [1] "http://dx.doi.org/10.1111/j.1469-1809.1936.tb02137.x"
#> attr(,"comments")
#> [1] "this is an example of more detailed meta.data, one could implement any other meta.data or meta.
iris.2$Sepal.Length# meta.data written with a template.
#>   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
#>  [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0
#>  [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5
#>  [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
#>  [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5
#>  [91] 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
#> [109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2
#> [127] 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
#> [145] 6.7 6.7 6.3 6.5 6.2 5.9
#> attr(,"variable.label")
#> [1] "original"
#> attr(,"long.label")
#> [1] "iris.Sepal.Length"
#> attr(,"full.text")
#> [1] "Sepal.Length is an indicator, in cm, of the length of the sepal, which are the green leaves sur
#> attr(,"value.labels")
#> [1] "cm"
#> attr(,"missing")
#> [1] "NA"
#> attr(,"associated.ids")
#> [1] "http://dx.doi.org/10.1111/j.1469-1809.1936.tb02137.x"
#> attr(,"comments")
#> attr(,"comments")[[1]]
#> template.2$comments[[1]]
```

From here it should be clear how a researcher could make available a meta.data template for a particular variable, and then for other people to take that file and apply it onto their own data.sets to make them comparable and inter- analyzable.

This same function can be done for groups of variables using apply.metas(), below.

apply.metas() applies a series of templates to a series of variables.

Oftentimes there is not just a single variable that one wants to apply to, but a series of variables in line. The apply.metas() function applies a series of templates to a series of variables in the data set. the templates do not need to be in order, so long as you specify the lists correctly. In this way, one can apply e.g., the Rosenberg Self Esteem Scales to the 10 questions in the data.frame which represent that scale without having to retype it 10 times.

```
iris.4 <- apply.metas(iris, iris.1.metas)
write.metas(iris.1)
#>      variable.label long.label
#> [1,] "Sepal.Length" "iris.Sepal.Length"
#> [2,] "Sepal.Width"  "NA"
#> [3,] "Petal.Length" "NA"
#> [4,] "Petal.Width"  "NA"
#> [5,] "Species"      "NA"
#>      full.text
#> [1,] "Sepal.Length is an indicator, in cm, of the length of the sepal, which are the green leaves su
#> [2,] "Sepal.Width indicates the width of the sepal, the green leaves around the petals."
#> [3,] "Petal.Length indicates the length of the petal, the delicate colored part of the flower."
#> [4,] "Petal.Width indicates the width of the petal, the delicate colored part of the flower."
#> [5,] "Species indicates the type of Iris it is setosa, virginica, or versicolor."
#>      value.labels
#> [1,] "cm"
#> [2,] "cm"
#> [3,] "cm"
#> [4,] "cm"
#> [5,] "indicating the three plant genus (setosa, virginica, versicolor) "
#>      missing associated.ids
#> [1,] "NA"    "http://dx.doi.org/10.1111/j.1469-1809.1936.tb02137.x"
#> [2,] "NA"    "http://dx.doi.org/10.1111/j.1469-1809.1936.tb02137.x"
#> [3,] "NA"    "http://dx.doi.org/10.1111/j.1469-1809.1936.tb02137.x"
#> [4,] "NA"    "http://dx.doi.org/10.1111/j.1469-1809.1936.tb02137.x"
#> [5,] "NA"    "http://dx.doi.org/10.1111/j.1469-1809.1936.tb02137.x"
#>      comments
#> [1,] "this is an example of more detailed meta.data, one could implement any other meta.data or meta
#> [2,] "This is from  R.A. Fishers classic 1936 paper introducing "
#> [3,] "This is from  R.A. Fishers classic 1936 paper introducing "
#> [4,] "This is from  R.A. Fishers classic 1936 paper introducing "
#> [5,] "this is an example of more detailed meta.data 5 "
#>      levels                                          class
#> [1,] "NULL"                                          "NULL"
#> [2,] "NULL"                                          "NULL"
#> [3,] "NULL"                                          "NULL"
#> [4,] "NULL"                                          "NULL"
#> [5,] "c(\"setosa\", \"versicolor\", \"virginica\")" "factor"
write.metas(iris.4)
#>      variable.label long.label
#> [1,] "Sepal.Length" "iris.Sepal.Length"
#> [2,] "Sepal.Width"  "NA"
#> [3,] "Petal.Length" "NA"
#> [4,] "Petal.Width"  "NA"
#> [5,] "Species"      "NA"
#>      full.text
#> [1,] "Sepal.Length is an indicator, in cm, of the length of the sepal, which are the green leaves su
#> [2,] "Sepal.Width indicates the width of the sepal, the green leaves around the petals."
#> [3,] "Petal.Length indicates the length of the petal, the delicate colored part of the flower."
#> [4,] "Petal.Width indicates the width of the petal, the delicate colored part of the flower."
#> [5,] "Species indicates the type of Iris it is setosa, virginica, or versicolor."
#>      value.labels
#> [1,] "cm"
```

```
#> [2,] "cm"
#> [3,] "cm"
#> [4,] "cm"
#> [5,] "indicating the three plant genus (setosa, virginica, versicolor) "
#>      missing associated.ids
#> [1,] "NA"    "http://dx.doi.org/10.1111/j.1469-1809.1936.tb02137.x"
#> [2,] "NA"    "http://dx.doi.org/10.1111/j.1469-1809.1936.tb02137.x"
#> [3,] "NA"    "http://dx.doi.org/10.1111/j.1469-1809.1936.tb02137.x"
#> [4,] "NA"    "http://dx.doi.org/10.1111/j.1469-1809.1936.tb02137.x"
#> [5,] "NA"    "http://dx.doi.org/10.1111/j.1469-1809.1936.tb02137.x"
#>      comments
#> [1,] "this is an example of more detailed meta.data, one could implement any other meta.data or meta
#> [2,] "This is from  R.A. Fishers classic 1936 paper introducing "
#> [3,] "This is from  R.A. Fishers classic 1936 paper introducing "
#> [4,] "This is from  R.A. Fishers classic 1936 paper introducing "
#> [5,] "this is an example of more detailed meta.data 5 "
#>      levels                                          class
#> [1,] "NULL"                                          "NULL"
#> [2,] "NULL"                                          "NULL"
#> [3,] "NULL"                                          "NULL"
#> [4,] "NULL"                                          "NULL"
#> [5,] "c(\"setosa\", \"versicolor\", \"virginica\")" "factor"
```

This function also makes it easy to apply the meta.data from one survey to another, in a series of surveys, which are largely the same except for some small changes.

Importantly, it is easy to apply the templates to the wrong variables, by e.g., starting or ending in the wrong place, so, be careful. At some point in the future there will be checks, and probably even an AI that can identify variables and apply templates automatically, but these do not exist yet (grants submitted). Once again, if you want to help us with this, reach out or just do it, we can try to help.

While the templates are very useful, they will also need to be edited sometimes. For this, we introduce change.meta().

change.meta() changes and adds to the meta for a variable.

Even if the library contains nearly all variables, there will be times when one needs to at least edit or add to an existing template, so for that is change.meta.

The change.meta() function allows one to overwrite the existing meta.data by indicating true for the 'replace' argument. The default is to attach the old meta.data after the new, to sustain provenance. Below we edit the meta.data on iris.3 using the change.meta() function. First we simply replace it, then we add to it. By default replace is false, so that it only adds new meta.data to the existing. Aside from that it is basically the same as create.meta().

```
iris.4$Sepal.Length #See the most recent sepal length, as applied from the template.
#>   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
#>  [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0
#>  [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5
#>  [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
#>  [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5
#>  [91] 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
#> [109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2
#> [127] 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
#> [145] 6.7 6.7 6.3 6.5 6.2 5.9
#> attr(,"variable.label")
```

```
#> [1] "Sepal.Length"
#> attr(,"long.label")
#> [1] "iris.Sepal.Length"
#> attr(,"full.text")
#> [1] "Sepal.Length is an indicator, in cm, of the length of the sepal, which are the green leaves sur
#> attr(,"value.labels")
#> [1] "cm"
#> attr(,"missing")
#> [1] "NA"
#> attr(,"associated.ids")
#> [1] "http://dx.doi.org/10.1111/j.1469-1809.1936.tb02137.x"
#> attr(,"comments")
#> [1] "this is an example of more detailed meta.data, one could implement any other meta.data or meta.
#> attr(,"levels")
#> [1] "NULL"
#> attr(,"class")
#> [1] "NULL"
change.meta(iris.4$Sepal.Length, comments ='Here we added to the meta.data', replace = F)
#>   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
#>  [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0
#>  [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5
#>  [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
#>  [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5
#>  [91] 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
#> [109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2
#> [127] 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
#> [145] 6.7 6.7 6.3 6.5 6.2 5.9
#> attr(,"variable.label")
#> [1] "Sepal.Length"
#> attr(,"long.label")
#> [1] "iris.Sepal.Length"
#> attr(,"full.text")
#> [1] "Sepal.Length is an indicator, in cm, of the length of the sepal, which are the green leaves sur
#> attr(,"value.labels")
#> [1] "cm"
#> attr(,"missing")
#> [1] "NA"
#> attr(,"associated.ids")
#> [1] "http://dx.doi.org/10.1111/j.1469-1809.1936.tb02137.x"
#> attr(,"comments")
#> [1] "Here we added to the meta.data (new:old) this is an example of more detailed meta.data, one cou
#> attr(,"levels")
#> [1] "NULL"
#> attr(,"class")
#> [1] "NULL"
change.meta(iris.4$Sepal.Length, comments ='Here we replaced the meta.data', replace = T)
#>   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
#>  [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0
#>  [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5
#>  [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
#>  [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5
#>  [91] 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
#> [109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2
```

```
#> [127] 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
#> [145] 6.7 6.7 6.3 6.5 6.2 5.9
#> attr(,"variable.label")
#> [1] "Sepal.Length"
#> attr(,"long.label")
#> [1] "iris.Sepal.Length"
#> attr(,"full.text")
#> [1] "Sepal.Length is an indicator, in cm, of the length of the sepal, which are the green leaves sur
#> attr(,"value.labels")
#> [1] "cm"
#> attr(,"missing")
#> [1] "NA"
#> attr(,"associated.ids")
#> [1] "http://dx.doi.org/10.1111/j.1469-1809.1936.tb02137.x"
#> attr(,"comments")
#> [1] "Here we replaced the meta.data"
#> attr(,"levels")
#> [1] "NULL"
#> attr(,"class")
#> [1] "NULL"
```

If the above changes are unclear, check the 'comments' attributes. In the first one, we added to the existing meta.data, and in the second we replaced it. Remember that these are simply printed, rather than assigned to come variable. In this way you can change an existing meta.data template to better suit your needs.

##Conclusion: something like this is probably the future.

Hopefully the above examples make clear the potential value of standardizing meta.data at the variable level (Buttliere, 2021), rather than focusing on data.sets or etc, it seems clear that something like this is the future, and the meta.data() tries to meet this need.

If there are other ideas or ways that you think we should extend this current package, we would be happy to hear about it. As well as ideas concerning how to port these functions into e.g., Python.

Please let us know of any errors on the github page, or you can email us, twitter or etc.

Thank you!