Making Datasets Truly Interoperable and More Reusable in R

Daniel Antal

Table of contents

Making Datasets Truly Interoperable and More Reusable in R

The dataset package extension to the R statistical environment aims to ensure that the most important R object that contain a dataset, i.e. a `data.frame` or an inherited `tibble`, `tsibble` or `data.table` contains important metadata for the reuse and validation of the dataset contents. We aim to offer a novel solution to support individuals or small groups of data scientists working in various business, academic or policy research functions who cannot count on the support of librarians, knowledge engineers, and extensive documentation processes.

Note

This document aims to orient the work starting from [eurostat] and then generalising it into the new packages [dataset] and [statcodelists].

The term `dataset` and `data set` or even `Data set` currently follows the context to where the text refers to. Some standards capitalise terms, others not, and dataset has two alternate spellings.

The R language offers generic and inherited data objects to store tabular data, which all use R's flexible metadata system ("attributes"). Each object has its default properties or attributes, for example, a `data.frame` has `row.names` and `col.names`, and the user can freely add further attributes. User-defined S3 or other objects inherit some attributes added at construction or coercion to the new type. Using metadata attached to the data is very useful because it minimizes the chances that the metadata will be lost. However, perhaps because of its flexibility and lack of a well-trodden path, few R users add metadata to the objects themselves. Metadata recorded in other objects, if at all, is often detached or lost for the reuse of the R object.

```r
myiris <- iris
attr(myiris, "row.names") <- NULL
attr(myiris, "title") <- "The Famous [iris] Dataset"
attributes(myiris)
```

```
$names
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"

$class
[1] "data.frame"

$title
[1] "The Famous [iris] Dataset"
```

Tip

The current version of [dataset] already implements this. We should now focus on how to retain such information without relying (yet) on dataset in [eurostat].

We want to strike a good balance between prescribing the mandatory use of necessary metadata and letting the user choose. Users will refrain from using a complicated initialisation process, especially in pre-existing pipelines or during experimental analysis; on the other hand, most data analysts are not metadata experts and often need to become more familiar with even the most basic metadata standards.

One way of helping them is the creation of flexible, new, derived classes from the most frequently used classes, such as `data.frame`, `data.table` or `tibble` that helps add FAIR and provenance metadata for supporting interoperability, reuse and review. This approach would allow us to increase usability with better printing or summary functions. At this point, we will create new S3 classes only to show the feasibility of this way; we would need many uses and user experience to design suitable S3 classes, which should be stable.

The other way is designing functions that add helpful metadata to tabular data objects. Because we can add almost any attributes to an R object, in the early stages, we will mainly focus on developing helpful and user-friendly metadata functions and export/import functions that retain the metadata. Although offering less user comfort, this allows more experimentation because we can add the same metadata to a `tsibble` (a `tibble`-like time series object) or a `data.frame` (which is rather different from a `tibble`) without modification.

The best pragmatic solution is to start experimenting with recording more metadata from the data we download from Eurostat. That data fully complies with the best SDMX and W3C metadata standards, but the eurostat package just does not work with the metadata parts much.

Why we need more semantics?

A dataset in its tidy form has enough semantic information to allow a knowledgeable to easily create pipelines to process or manipulate the data in a reproducible way. The tidy data form is a statistical representation of Cobb's 3rd normal form, which is ideal for incorporation into a relational database. Most heavy R users will come across this concept sooner or later: the modernisation of the R language and the tidyverse packages, which are used in all of our packages that we want to modernise, all use this concept.

The tidy data format contains the semantics necessary for the analyst to work with the dataset in a pipeline. This amount of semantic information is usually not enough for interoperability or reuse: years after, even the initial user or analyst may struggle to understand the concepts that the data is supposed to represent; the provenance of the data becomes unclear; and a third-party user will almost certainly require further metadata to use the dataset. Like relational databases, a third-party user has difficulty accessing the data without a user-friendly schema description.

The FAIR recommendations focus on describing the dataset as a whole when released and mandate the use of Dublin Core (DC) or DataCite descriptive metadata. Our critique of this approach is that librarians developed DC with a focus on texts; DataCite, while considering datasets, also wanted to be format-agnostic and offer a solution that can describe texts, still images, video and datasets alike. While using DC and DataCite can significantly increase findability and encourage proper attribution, they could be more helpful for interoperability and reuse in the case of datasets. A user can immediately reuse a book in her domain of expertise if it is written in a natural language she understands. This is not the case with a dataset.

Data is only informative with metadata. The kind of metadata that is required to read and functionally use a dataset is not the descriptive metadata that helps find the file containing the dataset but metadata that makes the numbers and their organisation intelligible for the data analyst. How are observations organised in the dataset? What do columns mean? Are the numbers expressed in euros or dollars? Kilograms or tons? Without such semantic information, the dataset cannot be reused. It is also barely interoperable because addition into a relational or graph database is hard or impossible.

In R, most metadata packages [for example, dataspice, we need a literature review here] add the metadata to a new dataset, creating an initial separation of data and metadata. Our approach, i.e., adding metadata, including rich semantics, into the attributes metadata fields of an object, keeps the data and its crucial metadata attached. This creates sufficient interoperability and reuse within the R ecosystem, provided the user serialises the data and metadata into a native `.rds` file.

The problem with this approach is that for interoperability and reuse reasons, we often serialise the contents of R objects into more interoperable formats, such as CSV or Excel files. At this point, we will have to work with at least two files: a data and a metadata file. For example, the W3C recommendation for releasing data in CSV format is to accompany the data with semantic and other metadata in an accompanying JSON file.

There is a standard way to keep all metadata and data in one file: to convert the data using the RDF format (Resource Description Format) into one of the serialisation formats of RDF, for example, the default XML; or any other serialisation formats used to store information that support the RDF format. This format is ideal for inclusion in a graph database. It can be easily included in a relational database management system because it contains not only the data but also schematic information on how to use it.

If the analyst works in a more extensive, well-regulated business process, then adding semantics is possible by stacking various IT solutions. However, we would like to find solutions that are workable for a typical R user who may work in a small team or alone.

While most analysts are unlikely to use the RDF serialisation, we believe it is the only future-proof, long-term safe storage for the dataset. It is a format that meets the five-star requirements of open science's FAIR, and aligns well with the way the EU suggest the release of open data. Therefore, we want to create functions that:

- Add necessary metadata to the attributes of tabular data container objects in R;
- Help the user to go beyond the bear minimum and easily add further metadata;
- Offer serialisation solutions that meet international standards, including the W3C standard on releasing data in CSV format;
- Offers serialisation to the W3C-SDMX standard on statistical datasets and RDF.

Methodology

The SDMX-W3C Data cube model describes the statistical dataset "as a multi-dimensional space, or hyper-cube, indexed by those dimensions. This space is commonly referred to as a cube for short".

Statistical datasets are compiled from micordatasets (which contain observations about the statistical population), which may be compiled from nano-level datasets (when we need to consult multiple datasets about a single statistical object.)

For example, our statistical dataset contains the average remuneration of female and male music creators by country and year. This dataset is created from a mico-level dataset(s) that contained the earnings of male and female music creators in each year in various countries. We may have had several datasets about the earnings of each artist on various royalty accounts. The statistical dataset sufficiently identifies Slovakia 2019 case, while the creation of this dataset needs averaging of male and female earnings in Slovakia for the year 2019; the data is connected to persons with an identifier.

Statistical datasets, microdata, and nano-level datasets

Practically speaking, the base R `data.frame` is a good information object for a statistical dataset because it does not have an explicit row identifier. It uses the `row.names` attributes, which are by default just integer numbers. The `tibble` package offers a function to make an identifier column (or primary key column) from any column in case we work with microdata or more structured data created from several statistical datasets.

Suppose we want to map a user's `data.frame`, or `tibble` or `data.table` object to a dataset following the Data cube model, we need to ensure that columns that contain a unique combination and can potentially identify an observation (row) are made dimensions.

In either case, we need to be able to identify **observations**. They constitute the measured values. In a statistical table, the observations would be the values in the table cells.

In the case of statistical data, to **locate an observation** within the hypercube, one has at least to know the value of each dimension at which the observation is located, so these values must be specified for each observation. Datasets can have additional organisational structures in the form of slices as described that we do not implement in the first phase. In the case of microdata, the most obvious way to locate an observation is by using a unique observation identifier. If no such identifier is present, we treat various variables as dimensions. For example, within a small workplace, a unique combination of given name and family name stored in one or two columns may be sufficient to identify an observation about Jane Doe. In larger organisations, where there may be multiple people named as Jane Doe, or there may have been previous employees with the same name, they are likely to use a unique personal identifier.

Once we locate the observation, we need specific **structural metadata** to be able to interpret it. What is the unit of measurement? Is the value measured or estimated? These metadata are provided as attributes and can be attached to individual observations or to higher levels. When they are attributed to individual observations, they should be part of the dataset; when the entire dataset contains only actual data or data measured in the same unit, then they can be attached to the dataset's metadata.

The Data cube model offers two solutions for structuring datasets with multiple measures; however, choosing between these approaches already requires a solid data model for the dataset itself.

Cataloguing

Dublin Core and DataCite are metadata standards used in open science and libraries to make the datasets findable and attributable. They are instrumental in being found in libraries and their repositories, but these differ from the typical places where we would look for quality datasets. Therefore, we want to add further descriptive metadata that complies with the W3C DataCatalogue definition that large data providers and statistical offices use. We also want to comply with Schema.org's definition of the Dataset and DataDownload, because this is the descriptive metadata standard that internet search engines use.

Pragmatic approach

Our primary goal is to provide a dataset solution that only partially implements the data cube model because such an implementation requires the user to think in dimensions, attributes, and measurements. By relaxing the data cube model, we can still add much functionality to the upgraded dataset. However, we should always keep the door open to embracing the cube model to gain full machine-actionability and interoperability. In other words, we should relax the cube model only in ways that it remain very easy to return to full compliance with the W3C Data cube definition.

We relax the datacube model by creating a simplified data structure definition where the user does not need to select which variables constitute *attributes*, *dimensions*, or *measurements*. We can make the dataset more machine-actionable by making such categorisation explicit, for which we may provide helper functions. For example, the combination of dimensions must be unique within a dataset; therefore we can rule out variables as dimensions if they breach this unicity condition. Any variable with a unit of measure attached to it is a measurement.

A pragmatic approach to development is the application of what we want to do generally with a `data.frame` or similar object to implement is, as a use case, for Eurostat's datasets. These datasets come from the Eurostat data warehouse, and therefore they are already formatted properly.

Features

Component definition - variables

Allow the addition of metadata for each column (component in the Data cube model), either to fully or partially comply with the Data cube models's dataset definition.

Minimum requirements

- Adding a `rdfs:label` property to each variable; for example, `"life expectancy" @en`;
- Explicitly adding, or guessing, when possible, the `rdfs:range` property of the column, for example, with `xsd:decimal` in a way that the dataset can be serialised to basic formats like CSV or Excel or HTML.

If you look at the following example, even without the `eg:lifeExpectancy` is a `qb:MeasureProperty` statement, every other statement can be recorded.

- The name of the column is `lifeExpectancy` [eg stands for example; i.e. this column name is not defined in a strict name space.]
- The column has a `life expectancy` label in English and `várható élettartam` [The user can add a label for any ISO language thus increasing the likelihood to be found in search terms in other languages]
- `rdfs:range xsd:decimal` states when read from a flat file such as an XML or a CSV file, this file should be read into a spreadsheet or statistical software as a numeric variable with decimals and not as text.
- The `sdmx-measure:obsValue` statement and `qb:MeasureProperty` should remain to be decided later; i.e., they need a more formal semantic knowledge about the dataset.

```
# Example with life expectancy data
eg:lifeExpectancy  a rdf:Property, qb:MeasureProperty;
    rdfs:label "life expectancy"@en;
    rdfs:label "várható élettartam"@hu;
    rdfs:subPropertyOf sdmx-measure:obsValue;
    rdfs:range xsd:decimal .
```

Recommended requirements

- Adding the valid range for `rdfs:range` for data validation explicitly.
- Conceptualising the variables, i.e., adding a well-defined concept that the variable represents.

- We can create helper functions to guess or help set widely used concepts, such as SEX, TIME, etc.
- Defining which variables should be seen as dimensions, measures, and attributes.

In the following example, we go beyond our minimum features. The `qb:concept sdmx-concept:refPeriod` statements makes it clear that our `refPeriod` variable (column in a tidy dataset) contains the reference period (in time) for the observations.

```
eg:refPeriod a rdf:Property, qb:DimensionProperty;
   rdfs:label "reference period"@en;
   rdfs:subPropertyOf sdmx-dimension:refPeriod;
   rdfs:range interval:Interval;
   qb:concept sdmx-concept:refPeriod
```

And going one step even further, for a `sex` variable we can add standardised codelists, by recording the following statements: `eg:sex qb:codeList sdmx-code:Sex`, i.e., the `sex` variable has a codelist which is the cross-domain standard codelist to code sex and gender.

For validation purposes, for which we may want to add helper functions in a later, more mature stage, we can also use the following statement: `eg:sex rdfs:range sdmx-code:Sex` which would allow a function to retrieve the valid coding of `Sex` variables, and check if the R `factor` variable indeed only contains levels that conform to this codelist definition.

```
eg:sex a qb:DimensionProperty, qb:CodedProperty;
    qb:codeList sdmx-code:Sex ;
    rdfs:range sdmx-code:Sex .
```

The aim of our statcodelists package is to ensure that these standard codelists are immediately available in native R variables for the user, making validation even simpler.

```
require(statcodelists)
```

```
Loading required package: statcodelists
```

```
Warning: package 'statcodelists' was built under R version 4.2.3
```

```
CL_SEX <- statcodelists::CL_SEX

# The valid code range:
CL_SEX$id
```

```
[1] "F"  "M"  "_N" "_O" "_T" "_U" "_Z"
```

```
# The valid labels (levels for R factors):
CL_SEX$name
```

```
[1] "Female"          "Male"            "Non response"    "Other"
[5] "Total"           "Unknown"         "Not applicable"
```

First steps

Our task should be to try to find this information in the eurostat data warehouse, and retain it, then create simple functions to modify or interact with them.

Dataset cataloguing

Dublin Core and DataCite are metadata standards used in open science and libraries to make the datasets findable and attributable.

The W3C Data Catalog Vocabulary (DCAT) - Version 3 and Schema.org's definition of the Dataset and DataDownload help to find datasets in data stores (as opposed to Dublin Core and DataCite, which are not designed to search for datasets.) The W3C and Schema.org definitions are highly interoperable (see).

There is an overlap between these four definitions, and we provide support for all four of them.

Simple fields

Simple metadata fields do not require validation; they can take the form of simple text or, integers or decimals. These are the fields that can take a literal value.

Adding them in a parameter list via a simple function to the dataset is straightforward.

```
myiris <- iris
attr(myiris, "row.names") <- NULL # To make results easier to read
add_title <- function(dataset, title) {
  attr(dataset, "title") <- title
  dataset
}
myiris <- add_title (myiris, "The Famous [iris] Dataset")
```

```r
attributes(myiris)
```

```
$names
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"

$class
[1] "data.frame"

$title
[1] "The Famous [iris] Dataset"
```

> First steps
>
> We already have this programed in the 0.2.0 version of [dataset] but we should now experiment with retaining and working with this data as they are in [eurostat] and on the eurostat data warehouse.

Persons, Unique identifiers

For increasing reusability and knowledge combination, it is a good practice to include globally unique identifiers among the metadata. Such unique identifiers include the creator's identifier (for example, ORCiD) or the use of the DOI.

> First steps
>
> In this regard, our packages already utilise how R manages bibliographic citations, and we should remain compatible with this approach.

Conceptualised fields

The SDMX standard includes a set of content-oriented guidelines (COG) which define a set of common statistical concepts and associated code lists that are intended to be reusable across data sets (for example, describing the frequency of the observations, reference area [geography] of the observations, whether the data is missing, estimated, or actual). Variables and metadata describing such information should recorded with COG because it is standardised and machine-actionable.

These COGs are conceptualised, meaning they utilise abstract concept objects described in machine-actionable and highly standardised ontologies, taxonomies or thesauri. DC and DataCite, as well as schema.org or W3C's Data Catalogue, support the use of conceptualised descriptions, for example, when recording the `Subject` of the dataset or recording the meaning of variables in the dataset.

In such cases, we will rely on the quasi-standard SKOS to describe conceptualised metadata. With SKOS, advanced users can create or curate their thesaurus that is machine-readable, but such a scenario is unlikely with a typical R user. Instead, we should point to the most generally used thesauri.

For example, the Library of Congress Subject Headings (LCSH) comprise a thesaurus (in the information science sense, a controlled vocabulary) of subject headings maintained by the United States Library of Congress for use in bibliographic records. The LCSH is the most used thesaurus in the world; many other thesauri, for example, ones created by national libraries, offer equivalence relations or translations to LCSH. Helping the user to use LCSH and 2-3 well-curated similar thesauri already adds a lot to the discoverability of the dataset.

```r
attr (myiris, "Subject") <- list (
  Heading = "Datasets",
  URI = "http://id.loc.gov/authorities/subjects/sh2018002256")

attributes (myiris)
```

```
$names
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"

$class
[1] "data.frame"

$title
[1] "The Famous [iris] Dataset"

$Subject
$Subject$Heading
[1] "Datasets"

$Subject$URI
[1] "http://id.loc.gov/authorities/subjects/sh2018002256"
```

The PURL/URI of the “Datasets” Heading in LCSH resolves to the following page:

https://id.loc.gov/authorities/subjects/sh2018002256.html

This page uses content negotiation, and it is accessible for both humans and computers. If we look at it via a web browser as humans, we can see that they contain the definition in various machine-readable formats, for example, in RDF/XML and SKOS JSON.

We have to offer exporting functions to W3C CSV+JSON (we can also if we want, create Excel+JSON or SPSS+JSON versions) and to RDF serialisation formats to retain this reach metadata. We can, of course, convert the Subject heading "Datasets" into a link that points to the PURL of the subject heading, but this is only useful for humans. The JSON or RDF formats place the URI/PURL into a well-defined place that makes computers immediately understand to look all the information up. On the other hand, the lack of such exporting shows why most datasets that are supposed to conform with FAIR remain so hard to find.

First steps

The question is how to find a comfortable way of entering such information: as `list`, nested list, `data.frame`? It is clear what information should be added to `attributes` but we should make it user-friendly to an R user who is not familiar with knowledge engineering, metadata standards, and perhaps find it hard to work with highly structured objects.