Making Datasets Truly Interoperable and Reusable in R

A working paper to develop the dataset R package

Daniel Antal

Table of contents

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.

The development of the *dataset* R package started in 2022 with a very experimental first release of some code and articles about what this software package extension to the R statistical system should do. The initial package idea was released on CRAN (Antal 2023) and send for peer review to rOpenSci. In 2023, a more thorough analysis of the non-functional and functional requirements started. This document aims to review the literature on possible solutions and provide some guidance to start the development. This document is accompanies the development version of the *dataset* package (Antal 2022b), which are only sent to CRAN for release when they reach a new level of maturity, and add new functionality that is well-tested and documented.

You can refer to this document with DOI 10.5281/zenodo.10091666.

Problem statement

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 (Wickham 2014). 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 (Dublin Core 2020; 'DataCite to Dublin Core Mapping 4.4', n.d.). Our critique of the approach of describing datasets with DC or DataCite alone 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 (Boettiger et al. 2021) 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 (Tennison 2016).

A broader problem that we will describe in more detail is that a dataset, even a *tidy dataset* does not contain enough semantic information for re-use. A reuse by a third party, or even by the creator of the dataset after 1-2 years requires at least a precise description of the *meaning* of variables and *codelists*. Various metadata standards and models, such as *DCAT* or the *W3C Data cube* (strongly aligned with the SDMX Datacube model) and *Schema.org* require additional metadata on the codelists and variable concepts used in the dataset. (Albertoni et al. 2020; W3C 2014) Additionally, they recognise the fact that for greater interoperability, the same dataset may be serialised into different file formats (for example, `.csv`, `.xlsx`, and `.sav` files) with essentially the same meaning but conforming the differnet ways how these file formats represent data and metadata. While DataCite and Dublin Core may create a separate data record for each file verison, it is important to connect the same data file with different file formats in a data catalogue. Even in Dublin Core or DataCite, a diligent record keeping would record the *relation* of the CSV and Excel representation of the dataset with the very same meaning.

Therefore, we would like to provide an intermediate R user with the opportunity to correctly record metadata for greater interoperability and real third-party reuse. In a well-regulated working environment with large IT and documentation resources, our problems are well handled by extensive processes and procedures. GSIM is a statistical vocabulary that harmonises the uses of the SDMX statistical data standard (SDMX 2021) and the DDI (Alliance 2012), a standard for documenting, coding and maintaining microdata files. Most

R users do not work in such a well-regulated environment, and we want to give them a practical tool to improve their process documentation for much-improved interoperability and reusability with the introduction of the dataset R package (Antal 2023).

Requirements and usability

- Rebeca, intermediate R user, able to use the *eurostat* package and *tidyverse*, wants to become FAIR compliant: eurostat-user1 – source stories/eurostat-user1.qmd

Literature

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.

The DataCite to Dublin Core Mapping 4.4 is very helpful in generalising our package, because we do not want to simply implement DataCite (or Dublin Core.)

The most relevant work for designing the metadata aspects of our new class is the unofficial CiteDCAT-AP specification of the Joint Research Centre of the European Commission (Perego et al. 2021), which is seeking a solution to bridge the library-oriented approach of the European FAIR applications and the data cataloging approach of statisticians and open data providers by trying to create a mapping of DataCite and DCAT-AP, the DCAT Application profile for data portals in Europe, a specification based on W3C's Data Catalogue vocabulary (DCAT).

Methodology

In our methodology, we first follow existing mappings between various metadata standards. To describe the dataset and further map them to the R language and ecosystem. This will undoubtedly help publication, findability and accessibility but adds less to the interoperability and reuse. While finding a book in a natural language that we know in a library lends itself to a natural way to reuse the book (start reading it), an analyst finding a dataset in a library or a data catalogue will need metadata to begin using the find.

The librarian DataCite and Dublin Core talk about a dataset file (or printed format) that is stored in a library; SDMX and DCAT talk about an abstract dataset that can be accessed (together with its represented meaning) through various file serialisations that may take a different format in an API designed for machines and flat file intended for download by humans.

SDMX and DCAT particularly, and the not data-focused Schema.org partly help by providing metadata about the data. They allow opening up the hoods of the dataset file and seeing what is inside. Dublin Core and DataCite do this only in two aspects: by choosing to describe an overarching Subject of the dataset as a whole and by describing the geographical area the dataset contains information about.

DCAT also describes the temporal frame (the times of the observations in the dataset); SDMX aims to describe every variable and, if needed, every observation in the dataset, including the geographical and time coverage of the observations.

We will come back to this problem when we open the hoods of the dataset, but we observe that adding every variable (column) subject to the subject definition of the dataset itself would be very misleading. All statistical datasets contain a description of observation time and geography; however, adding time and geography to the Subject would not be helpful to finding a dataset about gender pay gap differences (which, of course, have a geographical and time dimension.) Some concept descriptions from under the hood should go to the Geographical coverage of the dataset or the Temporal coverage, and some to the Subject. There may be some technical concepts that do not need to be mapped to DataCite or Dublin Core in any way.

Let us start with the simplest mapping. We only want to describe datasets, so the mandatory DataCite property `resourceType` (which corresponds to Type in Dublin Core) can be expressed as constant.

Literal values

Several DataCite properties can be expressed with literal values.

From the mandatory fields: - `Title`, which is a literal string.

- `PublicationYear` is a literal integer or an integer representing a year.

The `utils` package (which is distributed with very R installation, it is maintained by the R-core team) includes the `bibentry()` function which is intended to be used with bibliographic entries, and therefore can be mapped to Dublin Core and more or less to DataCite, too. The `Title` field can be recorded as `bibentry(title = "" )` and `PublicationYear` as `bibentry(year = "" )`.

DataCite and DublinCore differ in the way we can add alternative titles, such as subtitles, translation to a record, but adding a `Title` is a necessity in both cases.

From the recommended and optimal fields:

- `Version`, which is a literal string.

`Version` is not present in Dublin Core, but it is present in SDMX. Version can be added as `bibentry(version = "" )` as a literal string.

In these cases, we only need to ensure that the atomic classes of the R objects representing them are correctly translated into XML.

Agents

An important group of properties related to agents, i.e., people and institutions, who are related to the dataset in different roles. As DataCite (and Dublin Core) work mainly with library catalogues, often dating back before the semantic web, their representation of people and institutions could be made richer; CiteDCAT-AP implicitly builds an Agent model to meet the various metadata standard's expressiveness levels.

Recording agency with more semantics is a very important step towards giving an account of data provenance, which includes reviewability, and eventually can foster replicability and trust in the data: we should know who, in what capacity did what with the data; for example, who was the person who ran a software agent to impute missing numbers. Modelling data provenance is not independent of the (business) process of the data creation, and it is inherently very complex. Our dataset package is not a data provenance tracking tool; but a more complex modeling of agency can open ways to these important aspects of reproducible research.

In R, we consider that `utils::person()` already has an R model that is used to contain agency information which is used embedded to bibentry objects, for example, describing a person as author with affiliations.

CRAN generally uses the comment field to record unique identifiers such as ORCiD URIs. Because this model is used in the entire R ecosystem, we will rely on it to record data for agents and provide a parser into RDF. This agent model should be able to serve the following properties:

- Creator (mandatory in DataCite)
- Publisher (mandatory in DataCite)
- Contributor (recommended) - in DataCite, they can take many roles; in a statistical process, these roles can be even further described.
- Related item's children, which are again the Creator, Publisher and Contributor.

Dates

The representation of Dates is not uniform in R; the base R solutions are rather cumbersome to use, and several modern R packages exist to work better with identifying time and time intervals. We keep in mind their interoperability and display them in an ISO standard format. We will only need simple coercion or formatting rules.

- `PublicationYear` is mandatory but should be truncated to the year.
- `Date` allows a list of dates relevant to the resource to be added to DataCite, which is recommended.
- DCAT and Schema.org also uses dates, but different ones.

With dates, we can work similarly to literals; we need to translate the dates into the standard RDF/XML notation. Even if somebody is not working with XML, this notation can easily be parsed and truncated to a date string.

Identifiers

We have one more important mandatory field, the `Identifier`, where DataCite is more particular than DublinCore; furthermore, DCAT, SDMX and Schema.org use a different model. DublinCore and DataCite aim to identify a physical or digital object in a catalogue that contains the dataset. DCAT, SDMX and Schema.org aim to describe the meaning of the abstract dataset semantically and conceptualise serialisations (files that may follow different formats, such as JSON, XML, CSV, and Excel) as other instances (distributions) of the dataset. So a data catalogue has a more complex identification than a library which collects one particular physical or digital file representation of the dataset.

For a user who wants to comply with DataCite (and EU open science FAIR adoptions), it is a good practice to create a DOI to the first release of the dataset and all subsequent published versions; for example, in Zenodo this is possible with versioned DOIs. Yet for data services (Data feeds in Schema.org), there are more practical solutions than this, for example, when the dataset changes daily because new observations are added and released daily.

The simplest solution is using a URI identifier for the dataset that we describe, reminding the user that for DataCite compliance, it must be a DataCite DOI, but our R solution allows using any URI. In the next step, we can extend this with the more structured information required by DCAT, SDMX, and Schema.org to find the dataset's (various) downloadable files on the net.

Identifiers, however, are used as properties of Agents and relatedItems. We should have two models for Identifiers: one that works well with DataCite and Dublin Core, and one that works on DCAT, SDMX and Schema.org.

Related Items

The DataCite relatedItems and the Dublin Core relation (dcterms:relation) concepts are comprehensive because they relate to many information objects, including books and journal articles. We only want to deal with datasets and potentially open ways for a more granular quality control of a statistical process. Consider a simple use case: our analysts download a GDP dataset and a population dataset from a data source and produce their own GDP/capita calculation. We want to relate both initial datasets (and the researchers working on the combination.)

The original datasets should be recorded in the isDerivedFrom property of the relatedItem in DataCite, and they are recorded in Dublin Core as dcterms:source. This information is not sufficient for review or replication in our data analytical workflow; we may want to record who (the researchers) used what software agents to achieve the combination that we describe. As a starting point, we should provide a subjective view of how the dataset package should work with these metadata standards. For reproducibility, we should provide helper functions to provide a higher level of computational reproducibility (for example, recording elements of the users running environment, or providing more data provenance.) Dublin Core's DCTERMS allows a flexible definition of such relations.

Subject

Going forward with the recommended fields, the most important one is the Subject. While DataCite allows the use of simple keywords, it also suggests the use of public thesauri for interoperability. The incorporation of Subjects with URIs is straightforward on the one hand but opens up the question of going further into the data structure definition of the dataset itself. For use with DCAT, SDMX or Schema.org, we seek a SKOS solution for Subjects, which semantically extends DataCite and Dublin Core without contradiction, as we keep the URIs of the Subjects but add equivalence, broader and narrower relationships to place the dataset on a richer concept map.

we must be pragmatic here: for findability and future reuse, the user can add the most detail here, however, after a point, the user must change to a different metadata language. Our aim is not to provide a parser to/from RDF or OWL. We aim to find a simple-to-use representation form that can be the starting point of a far richer (and still connected) semantic description of the dataset.

Data Structure Definition

The Data Structure Definition aims to bring the dataset into relation with the following objects:

- A conceptual, semantic description of the constitutents of the dataset;
- Distributions or file serialisations of the dataset;
- Codebooks requires to understand instances of the observations recorded in each variable.

In DataCite and Dublin Core, the focus is on the distribution or file; and the conceptualised description appears in Subject, and Geolocation (DataCite) or `dcterms:spatial` (Dublin Core.)

Codebooks are very important for SDMX and for microdata management, for example, under the DDI standard.

Codebooks

The R ecosystem offers various ways to deal with codebooks. Base R offers the use of the atomic factor class to record labelled data; various more modern packages, such as labelled, offer better support for the handling of values and labels of categorical data.

Making labeling consistent among resources is a very important requirement in research. For example, survey harmonisation allows the combination of data recorded in different surveys, which requires the matching of codebooks. When we combine different datasets, even from the same source, for example, from Eurostat, we must ensure that we use a consistent labelling for dimensions to keep the semantic integrity of the analysis. If male and female data subjects are labelled in one data source as `M`; `F`, while in others `Man` and `Woman`, we must harmonise the coding.

Interoperability can be greatly enhanced with the use of standardised codebooks. SDMX, which is an ISO standard, offers global codebook standards for many cross-domain concepts that are likely to appear in almost any dataset: for geographical reference areas, for points in time and time intervals, for various data quality attributions, or for sex at birth or gender.

We should allow the user to crosswalk `Man` and `Woman` to the international standard `M`, `F` for a far greater interoperability and ease of reuse, and to be able to attach metadata about each variable (and eventually, if needed, each observation.)

The accompanying *statcodelists* data package contains many internationally standardised codelists in native R objects (Antal 2022a).

Conceptualisation of variables

The conceptualisation of variables means is required already to match the variable with a good codebook. We need to know that the reliable contains information about sex at birth or declared gender to be able to match it with the SDMX `CL_SEX` (or other standard) sex and gender codebooks.

The conceptualisation of variables constituting a dataset should not be mixed up with the subject of the dataset itself. All statistical datasets should contain a variable about the reference geographical area and timeframe. Yet, it would be as misleading to add to the subjects of the dataset "Time" or "Geography" as adding "Table of contents" or "Bibliography" to the subject keywords of a book. Just because a book has a part that can be described well with the concept of a reference list or a statistical dataset has a reference list of observation times, they are not reference books or datasets about time.

In our previous simple example, our users combined a dataset about GDP and another about population to a dataset about GDP/capita. Suppose they worked with national (not subnational) data; their resulting dataset can be best titled GDP Per Capita by Country. We may qualify the countries to a larger region or for a time interval; for example, if they somehow backcast inferred GDP data to the former Soviet Union, both time and geographical area are particular: Estimated GDP/Capita of Former Soviet Republics in the 1980s. It would be very natural to relate this dataset to concepts of the Soviet Union and the 1980s already in the library subject headings, regardless of whether we will add the reference area to DataCite's Geolocation or dcterms:spatial. The GSIM statistical vocabulary (that connects SDMX and DDI) would describe such an interesting dataset in a far more expressive way than the more general-use DataCite or Dublin Core. Our role here is to provide good support for both.

Our hypothesis is — and this requires broader user validation — that generally, the concepts that describe the `measure` variables of a statistical dataset should be recorded as the general Subject of the dataset itself. In our hypothetical example above, the reason why users would look for such a dataset would be the specific geographical and time coverage; therefore, the user should be able to add to Subjects the concepts that describe current countries and former Soviet republics or the time frame. To give a counter-example, a dataset that publishes observed currency exchange rates and adds a new observation about the daily opening and closing rate has new versions (distributions) and a new timeframe every day. In this case, the vital Subject is the exchange rate, and if it contains information about a few specific currencies, then these currencies, too. We would refrain from using the timespan in the Subject description, though we may want to use the daily frequency concept (SDMX:FREQ).

Eventually, it is a subjective data curatorial decision on the critical constituents of the datasets that describe well the dataset as a singular concept: this should find its way to the descriptive DataCite, Dublin Core, DCAT or Schema.org fields to help search for the dataset. Where our approach goes beyond this use case is where the statistical processing starts: we need to understand every variable in the dataset, and we have to understand the observations reasonably well, too; in the language of tidy data, we have to know what the columns and the rows mean.

We could add endless semantic information about each variable, or perhaps even about each observation, but it is unlikely that we would use R for this purpose. Our task is to enable a richer semantic description of the dataset by providing as much information as DCAT, SDMX, or QB asks for. We should be able to add a single subject keyword to each variable combined with a URI and a more descriptive label than the R variable name. Furthermore, in the case of using standard statistical concepts, we should be able to match this concept URI with the code list URI. If the variable records the sex of the statistical units (groups of women and men), then we must connect the variable (column) containing this information with the SEX concept of the statistical agency and the CL_SEX accompanying code list for full interoperability and later reuse.

Distribution

The DataCite data model does not distinguish between a dataset and its embodiment(s) ("distribution(s)" in the DCAT terminology). Dublin Core assumes that a library has a copy of the dataset; DataCite goes further by allowing different versions of them and asking for each version to be a DOI identifier. Zenodo implemented a solution for this with its versioned DOIs.

Yet a statistical distribution service would find this approach unworkable because DOIs are supposed to be permanent, long-term identifiers, and a statistical service will likely update a dataset daily. In DataCite (or Dublin Core), there is no attribute equivalent to dcat:accessURL or dcat:downloadURL; or Schema.org's DataDownload concept.

We do not want to model a data feed or an API in dataset, but we want to offer interoperability outside the R ecosystem, which will always require the serialisation of a dataset from an R object (our dataset object) to various, non-R specific file formats. Within the R ecosystem, our dataset S3 class can ensure that the dataset concept can be saved as a native .rds file and reused with all metadata. But naturally, R users often want interoperability with other systems or users who do not use R, and they export the dataset into different file serialisations: for example, into a CSV, JSON, Excel and SPSS "manifestation" of the same R object. Creating a DOI for every file manifestation of every new version would be inefficient; therefore, we need to use the more generic DCAT/SDMX/Schema.org identification for different serialisations. This way, we can significantly increase the interoperability and the reuse capacity because we make the reuse with varying software tools far more accessible than a library catalogue (which is not designed for this purpose.)

The distribution will not start before the dataset is serialised to be transferable from the computer memory of our R user to other computers or data storage; therefore, unlike most of the metadata that we had worked on before, distribution metadata cannot be specified during the analytical work. We must leave this to the moment of saving the file or exporting the dataset from the computer memory (as an R object) to files. Another point to make here that before serialisation the DataCite metadata property Size (in Dublin Core dcterms:extent) is not known; and it is not even a property of the dataset (concept) but the actual serialisation, it will differ from file format to file format, and possibly even depend on the computer used to save the file. And of course, we may create several type of serialisations, which would require each a different DataCite record becasue of their different `Format` or `dcterms:format` properties.

For full interoperability, the best is to export the entire dataset and its metadata in RDF because it keeps all the data and the metadata in a single, standardised file. Because there are excellent solutions for this conversion, our package will not include such functionality, only a vignette on how to work with other packages to achieve this goal.

Turning the dataset object into a resource is optimal for semantic richness but reduces the usability to those other researchers who know how to work with an RDF resource. As a second-best solution, we also offer export to W3C flat CSV files, which are by design accompanied by a machine-readable JSON file.

Implementation

We implement an s3 data class, dataset, which can be created from any R object that is inherited from base R's data.frame; a `tibble` from tidyverse, a data.table, or a tsibble for a modern time series dataset. We must very carefully add methods to the dataset class to avoid any inconsistencies among the widely used packages for the various dataset containers. Currently, we only implement `describe()`, which gives a human-readable oversight of the metadata attached to our dataset.

It is unlikely to conflict with any classes derived from `tibble`, `data.frame` or `tsibble`, because it only interacts with attributes of the objects, not the object contents itself.

Metadata Management

Literals and agents

```
iris_bibentry <- bibentry(
  bibtype = "Misc",
  title = "Iris Dataset",
  author = person(family ="Anderson",
                  given ="Edgar",
                  role = "aut"),
  doi = "https://doi.org/10.1111/j.1469-1809.1936.tb02137.x",
  description = "The famous iris dataset distributed with R.",
  publisher = "American Iris Society",
  year = 1935,  # datacite:PublicationYear and dc:date
  version = "1.0", # datacite:Version
  url = "https://en.wikipedia.org/wiki/Iris_flower_data_set",
  language = "eng",
  resourceType = "Dataset"
)
```

Even though the default print of a `bibentry` item does not show all the metadata, we can easily retrieve them.

```
iris_bibentry$publisher
```

```
[1] "American Iris Society"
```

```r
iris_bibentry$version
```

```
[1] "1.0"
```

```r
iris_bibentry$description
```

```
[1] "The famous iris dataset distributed with R."
```

We hard-wire the Dataset type into the attribute `type`.

Related Items

To handle Related Items, we use again the `utils::bibentry` and `utils:person` functions. The Iris dataset is distributed with each installation of R. Let's add this information to `myiris` using `utils::citation()`, which will create the bibentry of the R installed on our user's computer.

```r
rref <- citation()
rref
```

```
To cite R in publications use:

  R Core Team (2022). R: A language and environment for statistical
  computing. R Foundation for Statistical Computing, Vienna, Austria.
  URL https://www.R-project.org/.

A BibTeX entry for LaTeX users is

  @Manual{,
    title = {R: A Language and Environment for Statistical Computing},
    author = {{R Core Team}},
    organization = {R Foundation for Statistical Computing},
    address = {Vienna, Austria},
    year = {2022},
    url = {https://www.R-project.org/},
  }

We have invested a lot of time and effort in creating R, please cite it
when using it for data analysis. See also 'citation("pkgname")' for
citing R packages.
```

```r
relatedItem <- list(
  relatedItem1 = list(
    relatedItem = rref,
    relatedItemType = "Manual",
    relationType = "IsPartOf"
  )
)
```

Dates

Dates can be added as a list to the attributes. The dates should use for interoperability base R's "POSIXlt" and "POSIXct" classes, or a simple integer for years, or a simple character string that can be unambigously coerced into a date type.

Eventually, for full interoperability, we will have to be able to serialise dates to XML Schema Part 2: Datatypes Second Edition.

```r
myirisdata <- list (
  PublicationYear = 1935,
  Created = 1935,
  Issued = as.Date("2022-01-01")
)
```

Subjects

The `subject` class is a simple, pre-defined class inherited from `list` that contains the elements requires for DataCite.

```r
source(here::here("R", "subjects3.R"))

irissubject <- subject(
  heading =  c("Data sets", "Irises (Plants)"),
  schemeURI = rep("https://id.loc.gov/authorities/subjects/", 2),
  valueURI = c("https://id.loc.gov/authorities/subjects/sh2018002256.html",
```

The dataset S3 Class

```r
data(iris)
source(here::here("R", "datasets3.R"))

myiris <- new_dataset (x = iris,
                       DataBibentry = irisref,
                       Subject = irissubject )
```

```r
attr(myiris, "Subject")
```

```
[[1]]
$heading
[1] "Data sets"

$subjectScheme
[1] ""

$schemeURI
[1] "https://id.loc.gov/authorities/subjects/"

$valueURI
[1] ""

attr(,"class")
[1] "subject" "list"

[[2]]
$heading
[1] "Irises (Plants)"

$subjectScheme
[1] ""

$schemeURI
[1] "https://id.loc.gov/authorities/subjects/"

$valueURI
[1] ""

attr(,"class")
```

```
[1] "subject" "list"

attr(,"class")
[1] "subject" "list"
```

```r
attr(myiris, "DataStructure")
```

```
$Sepal.Length
$Sepal.Length$name
[1] "Sepal.Length"

$Sepal.Length$label
$Sepal.Length$label[[1]]
[1] ""


$Sepal.Length$type
[1] ""

$Sepal.Length$range
[1] "xsd:decimal"

$Sepal.Length$comment
[1] ""

$Sepal.Length$concept
$Sepal.Length$concept$heading
[1] ""

$Sepal.Length$concept$schemeURI
[1] ""

$Sepal.Length$concept$valueURI
[1] ""


$Sepal.Length$defintion
$Sepal.Length$defintion$schemeURI
[1] ""

$Sepal.Length$defintion$valueURI
```

```
[1] ""



$Sepal.Width
$Sepal.Width$name
[1] "Sepal.Width"

$Sepal.Width$label
$Sepal.Width$label[[1]]
[1] ""


$Sepal.Width$type
[1] ""

$Sepal.Width$range
[1] "xsd:decimal"

$Sepal.Width$comment
[1] ""

$Sepal.Width$concept
$Sepal.Width$concept$heading
[1] ""

$Sepal.Width$concept$schemeURI
[1] ""

$Sepal.Width$concept$valueURI
[1] ""


$Sepal.Width$defintion
$Sepal.Width$defintion$schemeURI
[1] ""

$Sepal.Width$defintion$valueURI
[1] ""



$Petal.Length
```

```
$Petal.Length$name
[1] "Petal.Length"

$Petal.Length$label
$Petal.Length$label[[1]]
[1] ""


$Petal.Length$type
[1] ""

$Petal.Length$range
[1] "xsd:decimal"

$Petal.Length$comment
[1] ""

$Petal.Length$concept
$Petal.Length$concept$heading
[1] ""

$Petal.Length$concept$schemeURI
[1] ""

$Petal.Length$concept$valueURI
[1] ""


$Petal.Length$defintion
$Petal.Length$defintion$schemeURI
[1] ""

$Petal.Length$defintion$valueURI
[1] ""



$Petal.Width
$Petal.Width$name
[1] "Petal.Width"

$Petal.Width$label
$Petal.Width$label[[1]]
```

```
[1] ""


$Petal.Width$type
[1] ""

$Petal.Width$range
[1] "xsd:decimal"

$Petal.Width$comment
[1] ""

$Petal.Width$concept
$Petal.Width$concept$heading
[1] ""

$Petal.Width$concept$schemeURI
[1] ""

$Petal.Width$concept$valueURI
[1] ""


$Petal.Width$defintion
$Petal.Width$defintion$schemeURI
[1] ""

$Petal.Width$defintion$valueURI
[1] ""



$Species
$Species$name
[1] "Species"

$Species$label
$Species$label[[1]]
[1] ""


$Species$type
[1] ""
```

```
$Species$range
[1] "coded"

$Species$comment
[1] ""

$Species$concept
$Species$concept$heading
[1] ""

$Species$concept$schemeURI
[1] ""

$Species$concept$valueURI
[1] ""


$Species$defintion
$Species$defintion$schemeURI
[1] ""

$Species$defintion$valueURI
[1] ""
```

The describe General Method

The aim of the `desribe()` method is to provide a simple printing helper function for the rich metadata that we attach to the data.frame-inherited datasets.

Most R packages would create a new `print()` method for this purpose, but in our case, it would create unnecessary complications and risk losing the metadata focus. The `print()` functions should give an overview of the R object's data and some key metadata; we would like to print out the metadata only.

```r
source(here::here("R", "describe.R"))
describe(myiris)
```

```
The iris Dataset
Dataset with 150 observations (rows) and 5 variables (columns).
Description: The famous dataset that is distributed with R.
Creator: Edgar Anderson [aut]
```

The distribute General Method

The `distribute()` general method would provide a similar support to the `devtools::release()` function: a step-by-step check of the data and metadata before making it public. In this case, of course, the release would be the dataset embedded in the data.frame-inherited R object.

As earlier stated, we differentiate between the concept of a dataset, which is similar to but not equal to the R object in the users runtime environment, and the distribution of the dataset. While an abstract concept is R-language independent, an R object in memory is similar in the sense that it cannot be accessed by a third party.

When we serialise this object into a file and make it available for the public, we must maintain the integrity of the link between the abstract dataset and the actual files, especially if we create paralel a CSV, SPSS, JSON serialisation; or, if we plan to release regularly updated version of the dataset.

The dataset object at the time of serialisation must include metadata that had not been known at the beginning of the production pipeline:

- The final identifier of the dataset (for example, before uploading to Zenodo, we can receive a reserved DOI for this purpose);
- The date of the release;
- The person responsible for the released dataset;
- The Publisher's name who is responsible for the dataset;
- The file name, the size of the file, and the URL where it will be made accessible;
- A Rights statement.

1. We record the information above into the `dataset` object. It can be saved as an `.rds` file, too.
2. If the user wants to serialise the `dataset` into an RDF resource, this information goes straight into the file; if we release a CSV file in concordance with the W3C recommendations, we place this metadata into the accompanying JSON file.
3. If we upload the CSV file or the SPSS file to Zenodo, we are expected to fill out this information on Zenodo by hand or the uploading utility we use for this purpose.

Nquad representation

The `convert_to_nquad()` function utilises the `normalise_df()` and `xs_class()` functions, which in turn are inspired by, but heavily changed from some internal functions of the `rdflib()` package, particularly the internal `poor_mans_nquad()`. These functions, after more heavy user testing, could be also included in `rdflib()` as they serve a rather different purpose.

The current `poor_mans_nquad()` optimises speed and converts any R `data.frame` fast into a syntactically correct Nquad, but in a way that is semantically unrealistically simple: each subject and each predicate has the same prefix, while object have no prefix at all. We are creating a more realistic, and less time-optimised version that is only using base R (no `tidyr` dependency) and correctly parses our `dataaset` objects columns, row names, and attributes into a `WRC Data cube` defined dataset.

```r
benelux <- data.frame (
  id       = as.character(1:3),
  refArea  = c("BE", "NE", "LU"),
  currency = rep("EUR", 3),
  obsValue = c(212, 214, 56)
)

# Context --------------------------
# sdmx-measure: http://purl.org/linked-data/sdmx/2009/measure#
# sdmx-dimension: http://purl.org/linked-data/sdmx/2009/dimension#

attr(benelux, "prefix") <- c(
  "id" = "eg",
  "refArea"  = "sdmx-dimension",
  "currency" = "sdmx-measure",
  "obsValue" = "sdmx-measure")
```

The `nquad_statement()` is a simple string pasting function where we add syntactic validation and helpers.

```r
source(here::here("R", "nquad_statement.R"))

nquad_statement(subject = "eg:myiris", "rdf:type", "<qb:Dataset>")
```

```
[1] "<eg:myiris> <rdf:type> <qb:Dataset> ."
```

```r
# We create a temporary file to save the statments into:
tempquads <- tempfile()

# We use write.table() instead of base R writeLines() to append
# the file with new statements:
utils::write.table(x = nquad_statement(
  subject="eg:myiris", predicate="rdf:type", object="<qb:Dataset>"),
  file = tempquads, col.names=FALSE, row.names=FALSE, quote=FALSE)
utils::write.table(x = nquad_statement(
  "eg:myiris", "dc:description", '"The famous iris dataset"@en'),
  file = tempquads, col.names=FALSE, row.names=FALSE, quote=FALSE,
  append=TRUE)

# Read the resulting temporary file:
readLines(tempquads)
```

```
[1] "<eg:myiris> <rdf:type> <qb:Dataset> ."
[2] "<eg:myiris> <dc:description> \"The famous iris dataset\"@en ."
```

Changing RDF serialisation formats

we would like to keep the dataset package lean with few dependencies, so any converstions should go into a vignette article that shows the interaction with other packages.

We can read these simple statements as RDF with the *rdflib* package (Boettinger 2023):

```
require(rdflib)
```

```
Loading required package: rdflib
```

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

```
rdf_parse(tempquads, format="nquads")
```

```
Total of 2 triples, stored in hashes
-------------------------------
<eg:myiris> <rdf:type> <qb:Dataset> .
<eg:myiris> <dc:description> "The famous iris dataset"@en .
```

And convert them (i.e., re-serialise them) to any RDF serialisation formats, for example, perhaps the most human-editing friendly Turtle:

```
iris_turtle <- rdf_serialize(
  rdf = rdf_parse(tempquads, format="nquads"),
  format = "turtle",
  namespace = c(
    rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
    dc = "http://purl.org/dc/elements/1.1/"
  )
)
# To maintain text formatting, use cat() instead of print().
cat(iris_turtle)
```

```
@base <localhost://> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .

<eg:myiris>
    <dc:description> "The famous iris dataset"@en ;
    <rdf:type> <qb:Dataset> .
```

Or the standard machine-readable output of RDF, i.e, XML:

```
iris_rdfxml <- rdf_serialize(
  rdf = rdf_parse(tempquads, format="nquad"),
  format = "rdfxml"
)

# To maintain text formatting, use cat() instead of print().
cat(iris_rdfxml)
```

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xml:base="localhost://">
  <rdf:Description rdf:about="eg:myiris">
    <ns0:type xmlns:ns0="rdf:" rdf:resource="qb:Dataset"/>
  </rdf:Description>
  <rdf:Description rdf:about="eg:myiris">
    <ns0:description xmlns:ns0="dc:" xml:lang="en">The famous iris dataset</ns0:description>
  </rdf:Description>
</rdf:RDF>
```

Conceptualisation

For interoperability, the best is to reuse already existing URIs for the description of variables (or even observations). In those cases we use of SKOS (Miles and Bechhofer 2009), as recommended by [QB] [DCAT-AP] and many other applications. The concept should be mapped to a `skos:Concept` and when we want to connect codelists, too, the overall set of admissible values using `skos:ConceptScheme` or `skos:Collection`.

DataCite allows the use of free text, literal keywords, or URIs, when it requires the separate recording of the schemeURI and the valueURI. Perhaps the most used scheme is the Library of Congress Subject Headings, which is identified by the `schemeURI=https://id.loc.gov/authorities/subjects` the concept of the dataset with the `value URI=https://id.loc.gov/authorities/subjects/sh2018002256.h` Recording the schemeURI and the valueURI is practical for our later use cases, too, but not strictly necessary.

The `namespace` element is not even defined in DataCite, however, it can be very practical to use prefix abbreviations within the `DataSetDefinition` (see later) and used together with `rdf_serialise`.

```
myirissubject <- list(
  Datasets = list (
    heading = "Data sets",
    namespace = "LCCH",
    schemeURI = "https://id.loc.gov/authorities/subjects/",
    valueURI = "https://id.loc.gov/authorities/subjects/sh2018002256.htm"
  ),
  Irises = list (
    heading = "Irises (plants)",
    context = "LCCH",
    schemeURI = "https://id.loc.gov/authorities/subjects/",
    valueURI = "https://id.loc.gov/authorities/subjects/sh85068079.html"
  )
)

attr(myiris, "Subject") <- myirissubject
```

The retrieval of the `iris` subject is straightforward, yet not very ideal if we want to add a matching taxonomic reference to the actual iris species that Anderson was describing.

```
attr(myiris, "Subject")$Iris
```

```
$heading
[1] "Irises (plants)"

$context
[1] "LCCH"

$schemeURI
[1] "https://id.loc.gov/authorities/subjects/"
```

```
$valueURI
[1] "https://id.loc.gov/authorities/subjects/sh85068079.html"
```

This leads us to the problem of providing a semantically richer description of the dataset that DataCite or Dublin Core, or perhaps even a DCAT would ask from us. Our solution is to allow (and encourage) the recording of more accurate semantic information, but as a separate attribute to the object.

Technically, we can add any kind of R objects to the attributes, but the portability of our object would be limited if we were going beyond base R classes for storing the metadata. For example, if we load the jsonld package, we can add without a problem a jsonld object to the attributes of our iris dataset instance and save it to an `.rds` file. However, we cannot ensure that a new user who loads this `.rds` file knows what JSON-LD is or how to read this information.

Reconciliation with other metadata editing tools and languages

Our initial recommendation is the use of RDF syntax that is easy to hand-edit and can be well represented in base R's character vectors. RDF 1.1 Turtle, RDF 1.1 N-Triples, RDF 1.1 N-Quads can all be represented well with character strings, and with an appropriate parser, they can be read into a graph database.

The *rdflib* package (Boettinger 2023) already uses N-Quads internally with the `poor_mans_nquad()` internal function. For interoperability among the two packages (and potential co-development) we will use in the *dataset* package the N-Quads notation.

Because the most likely use case of our concept description (and every other metadata descriptions) is that they will be translated to RDF and written to one of these (or other) serialisation format, basically the user can start an interative process of exporting from our R package the already recorded metadata, and add with hand or a more appropriate metadata editor further information to the dataset, which then can be re-attached to the dataset's representation as an R object.

We can, for example, record a German-language description (taken from Portal:Statistik/Datensätze of the German Wikipedia) to the following N-Triple statement in a character string and attach it to the object.

```
attr(myiris, "N-Triples") <- '<eg-myiris> <http://purl.org/dc/elements/1.1/description> "D
Iris-flower-Datensatz besteht aus jeweils 50
Beobachtungen dreier Arten von Schwertlilien (Iris)
(Iris Setosa, Iris Virginica und Iris Versicolor),
an denen jeweils vier Attribute der Blüten erhoben
wurden: Die Länge und die Breite des Sepalum
```

```
(Kelchblatt) und des Petalum (Kronblatt)."@de'

attr(myiris, "N-Triples")
```

```
[1] "<eg-myiris> <http://purl.org/dc/elements/1.1/description> \"Der \nIris-flower-Datensatz
```

It is unlikely that somebody wants to edit such statements in R, however, once the statements are clearly recorded, it may be a good idea to attach them back to the R object or its `.rds` file representation.

Needless to say that this `N-Triples` section can contain the translated `Subject` list into N-Triples and its extension with further taxonomic information, for example, in a plant species taxonomy on Wikispecies.

Connection to Foundational Ontologies

We aim not only to meet the DataCite or Dublin Core standards (or to force a choice between them.) Adding a creator to a dataset file can be unambiguously understood within the context of a library or repository catalogue. We can further increase the interoperability of our dataset by turning it into an RDF resource by using a more foundational ontology to describe the very same term. Defining the creator as a person in the context of FOAF, a machine-readable ontology (Brickley and Miller 2014), we can find on linked data or the web a lot more information about their related scientific activities or their relations to other people working on similar projects or other datasets and articles interpreting the data.

- We will describe our agents with the help of FOAF when turning our dataset into a resource, one of the oldest and most widely used foundational ontologies of the World Wide Web.
- For conceptualisation, we will use SKOS, one of the most used ontologies for this purpose and a quasi-universal standard.
- For describing the internal semantics of the dataset, i.e., providing information on how to use the data, we will rely on Data cube, a closely aligned W3C standard with SDMX, the statistical data and metadata global standard.

Of course, our definitions will use RDF and RDFS, the languages that make our RDF resource machine-readable, which are themselves described as ontologies.

Conclusion

This last step makes our resources particularly resilient and future-proof: they hard-wire into the same file that contains the data, besides the standardised metadata and codebook information, even the metadata language that allows reading this information, as if we placed our dataset into a library within a booklet carefully describing the data table, and providing even a single-language dictionary for future generations to keep the booklet readable.

References

Albertoni, Riccardo, David Browning, Simon Cox, Alejandra Gonzalez Beltran, Andrea Perego, and Peter Winstanley, eds. 2020. 'Data Catalog Vocabulary (DCAT) - Version 2'. W3C. https://www.w3.org/TR/2020/REC-vocab-dcat-2-20200204/.

———, eds. 2023. 'Data Catalog Vocabulary (DCAT) - Version 3. [Working Draft]'. W3C. https://www.w3.org/TR/vocab-dcat-3/.

Alliance, DDI. 2012. 'DDI-Codebook 2.5 | Data Documentation Initiative'. DDI Alliance. https://ddialliance.org/Specification/DDI-Codebook/2.5/.

Antal, Daniel. 2022a. 'Statcodelists: Use Standardized Statistical Codelists'. Vienna: The Comprehensive R Archive Network. https://CRAN.R-project.org/package=statcodelists.

———. 2022b. 'dataset: Create Data Frames that are Easier to Exchange and Reuse [development versions]'. Zenodo. https://doi.org/10.5281/zenodo.7440192.

———. 2023. 'dataset: Create Data Frames That Are Easier to Exchange and Reuse'. Vienna: The Comprehensive R Archive Network. https://CRAN.R-project.org/package=dataset.

Beckett, David, Tim Berners-Lee, Eric Prud'hommeaux, and Gavin Carothers. 2014. 'RDF 1.1 Turtle. Terse RDF Triple Language'. W3C. https://www.w3.org/TR/turtle/.

Biron, Paul V., Kaiser Permanente, and Ashok Malhotra, eds. 2004. 'XML Schema Part 2: Datatypes Second Edition'. W3C. https://www.w3.org/TR/xmlschema-2/#date.

Boettiger, Carl, Scott Chamberlain, Auriel Fournier, Kelly Hondula, Anna Krystalli, Bryce Mecum, Maëlle Salmon, Kate Kate Webbink, Kara Woo, and Irene Steves. 2021. 'dataspice: Create Lightweight Schema.org Descriptions of Data'. Vienna: The Comprehensive R Archive Network.https://CRAN.R-project.org/package=dataspice.

Boettinger, Carl. 2023. 'rdflib: Tools to Manipulate and Query Semantic Data'. Vienna: The Comprehensive R Archive Network. https://CRAN.R-project.org/package=rdflib.

Brickley, Dan, and Libby Miller. 2014. 'FOAF Vocabulary Specification 0.99, Paddington Edition'. https://iptc.org/thirdparty/foaf/.

Carothers, Gavin. 2014. 'RDF 1.1 n-Quads'. W3C. https://www.w3.org/TR/n-quads/.

Carothers, Gavin, and Andy Seaborne. 2014. 'RDF 1.1 n-Triples'. W3C. https://www.w3.org/TR/n-triples/.

'DataCite to Dublin Core Mapping 4.4'. n.d. DataCite e.V. https://doi.org/10.14454/qn00-qx85.

Dublin Core. 2020. ‘DCMI Metadata Terms’. http://dublincore.org/specifications/dublin-core/dcmi-terms/2020-01-20/.

Group, DataCite Metadata Working. 2021. ‘DataCite Metadata Schema 4.4’. DataCite e.V. https://schema.datacite.org/meta/kernel-4.4/.

Lebo, Timothy, Satya Sahoo, and Deborah McGuinness, eds. 2013. ‘PROV-o: The PROV Ontology’. W3C. https://www.w3.org/TR/prov-o/.

Miles, Alistair, and Sean Bechhofer, eds. 2009. ‘SKOS Simple Knowledge Organization System Reference’. W3C. https://www.w3.org/TR/skos-reference/.

Perego, Andrea, Timothy Austin, Anders Friis-Christensen, Lorenzino Vaccari, and Chrisa Tsinaraki. 2021. ‘DataCite to DCAT-AP Mapping’. Joint Research Centre of the European Commission. https://ec-jrc.github.io/datacite-to-dcat-ap/.

SDMX. 2021. ‘SDMX 3.0 Technical Specifications. Section 2 Information Model: UML Conceptual Design. Version 3.0’. Statistical Data; Metadata eXchange. https://sdmx.org/wp-content/uploads/SDMX_3-0-0_SECTION_2_FINAL-1_0.pdf.

Tennison, Jeni. 2016. ‘CSV on the Web: A Primer’. W3C. https://www.w3.org/TR/tabular-data-primer/.

W3C. 2014. ‘RDF Data Cube Vocabulary’. Edited by Richard Cyganiak and Dave Reynolds. W3C. https://rd-alliance.github.io/metadata-directory/standards/rdf-data-cube-vocabulary.html.

Wickham, Hadley. 2014. ‘Tidy Data’. *Journal of Statistical Software* 59 (September): 1–23. https://doi.org/10.18637/jss.v059.i10.