knitrdata: A Tool for Creating Standalone Rmarkdown Source Documents

by David M. Kaplan

Abstract Though Rmarkdown is a powerful tool for integrating text with code for analyses in a single source document exportable to a variety of output formats, until now there has been no simple way to integrate the data behind analyses into Rmarkdown source documents. The `knitrdata` package makes it possible for arbitrary text and binary data to be integrated directly into Rmarkdown source documents via implementation of a new `data` chunk type. The package includes command-line and graphical tools that facilitate creating and inserting `data` chunks into Rmarkdown documents, and the treatment of `data` chunks is highly configurable via chunk options. These tools allow one to easily create fully standalone Rmarkdown source documents integrating data, ancillary formatting files, analysis code and text in a single file. Used properly, the package can facilitate open distribution of source documents that demonstrate computational reproducibility of scientific results.

Introduction

The basic principles of open science are that the data, research methodologies and analysis tools (e.g., the specific computational tools) used for scientific research should be made publicly available so that others can confirm and validate scientific analyses. Open science is particularly important for studies and disciplines for which true experimental replication is often difficult or impossible due to spatial, temporal or individual specificity [e.g., we cannot replicate Earth; Powers and Hampton (2019)]. In these cases, computational reproducibility, i.e., the ability to reproduce analytic results given the original data and analysis code, can still be achieved and can provide significant credibility to results (Powers and Hampton 2019). Though scientists, governments and journals often place great emphasis on access to raw data (Cassey and Blackburn 2006; Lowndes et al. 2017), it is important to remember that computational reproducibility can only be assured if data, methods, computational tools and the relationships between these are *all* openly accessible. Even when data are made publicly available, there are often significant gaps between the Methods section of a publication and the raw data that complicate reproducibility without access to the detailed code used to generate results. It is, therefore, essential for computational reproducibility that the code used to generate results be distributed along with the data and the publication itself. Though there are a number of potential ways to distribute all these elements together, probably the most common current approach is to place the data in a publicly accessible data store (e.g., Dryad) and to associate the code with the publication via the supplementary material and/or by including it in the data store. Though this is a perfectly viable approach that can greatly enhance transparency of research, it physically separates data from analysis code and interpretation of results, potentially leading to confusion and/or loss of information regarding how these different element interrelate. At times, it would be more convenient, transparent and/or effective to join all the elements into a single document. The R package presented here, **knitrdata** (Kaplan 2020a), provides tools for doing just that - integrating data directly into Rmarkdown source documents so that data, code for analyses and text interpreting results are all available in a single file.

Rmarkdown (Allaire, Xie, McPherson, et al. 2022) has become an increasingly popular tool for generating rich scientific documents while maintaining a source document that makes explicit the relationship between text and specific analyses used to produce results (Xie 2014; Lowndes et al. 2017). In a nutshell, Rmarkdown source documents are text documents comprised of two major elements: structured text that make up the headings and paragraphs of the document, and blocks of code (typically, but not exclusively, R code) for doing analyses and generating figures and tables. Rmarkdown source documents can be processed into a variety of final output formats, including PDF documents formatted for scientific publication. During this processing, the blocks of code in the source document are executed and used to augment the final output document with figures, tables and analytic results. In addition to providing a single source document that includes both written text and code for carrying out analyses, Rmarkdown has other benefits for open science, such as requiring the user to provide fully functioning code that runs from start to end without errors and facilitating reuse and updating of documents when new data arrives.

Until now, however, it has been difficult to integrate the raw data itself that are the bases for analyses directly into Rmarkdown source documents. Typically, data are placed in external files that are accessed via R code contained in the Rmarkdown source document that is executed during the knitting. As previously mentioned, this has the disadvantage of physically separating data from analysis code and text contained in the Rmarkdown source document, potentially leading to confusion

and/or information loss. Furthermore, on a practical level, it often can be extremely convenient to merge all pertinent information into a single source document (e.g., to facilitate collaboration on an Rmarkdown source document). `knitrdata` provides a simple mechanism for integrating arbitrary text or binary data directly into Rmarkdown source documents, thereby allowing one to create standalone source documents that include all the elements necessary for conducting analyses. This integration is done with minimal additional formatting of the data (e.g., allowing one to insert comma-separated value (CSV) data without escaping quotation marks directly into Rmarkdown documents) and in a way that clearly visually separates data from R code, thereby facilitating comprehension of the different elements that contribute to analyses. `knitrdata` also facilitates encryption of data integrated into Rmarkdown source documents, thereby allowing one to merge data with analysis code and text even in cases where industrial or ethical privacy constraints restrict data access to a specific group of individuals.

Below, I briefly provide a conceptual overview of how `knitrdata` works, presenting some simple examples of its use and the tools available to facilitate integrating data into Rmarkdown source documents. I then discuss typical use cases and workflows for development of Rmarkdown source documents with `knitrdata`, as well as a number of potential caveats for its use. I conclude by reflecting on the value of `knitrdata` for achieving computational reproducibility and its place within the growing pantheon of tools that make Rmarkdown an increasingly essential tool for research.

knitrdata installation and usage

The `knitrdata` package is available on CRAN, though the latest version can be installed from github using the **remotes** (Csárdi et al. 2021) package:

```
remotes::install_github("dmkaplan2000/knitrdata",
                        build_vignettes=TRUE)
```

Once the package has been installed, all that is needed to use the functionality provided by the package in a Rmarkdown source document is to load the library at the start of the document, typically in the `setup` chunk:

```
library(knitrdata)
```

Conceptual overview of `knitrdata`

To understand how `knitrdata` works and the functionality it provides, one must first understand some of the terminology and functioning of Rmarkdown itself. As previously mentioned, Rmarkdown documents are a combination of text written in *markdown*, a simple, structured text format that can be translated into a large number of final output formats, and code for doing analyses that can augment the final output document with analytic results, tables and figures. The code is contained in specially delimited blocks, referred to as *chunks*. For example, adding the following to an Rmarkdown document:

````
```{r}
plot(-5:5,(-5:5)^2,type="l")
```
````

would add a plot of a parabola to the final output document. The process of translating a Rmarkdown document into a final output document is known as *knitting*, and this process is carried out using (often implicitly via RStudio) the **knitr** package (Xie 2015).

Though code chunks typically contain R code, `knitr` supports a large number of other *language engines*, allowing one to integrate analyses in a number of other computer languages, including C, Python and SQL. For example, one could use the SQL language engine to import the contents of a database table into the R environment by including the following chunk in a Rmarkdown source document:

````
```{sql connection="dbcon",output.var="d"}
SELECT * FROM "MyTable";
```
````

During knitting, this will create in the R environment a variable `d` containing the contents of the table `MyTable` accessible via the (previously created) active R database connection `dbcon`. Note that the name of the database connection and the name of the output variable are supplied in the chunk header via what are known as *chunk options*. Though this database table could be imported into the R environment without the SQL language engine using R code:

````
```{r}
d = dbGetQuery(dccon,"SELECT * FROM \"MyTable\";")
```
````

the use of the SQL language engine has both practical and conceptual advantages. On the practical side, it avoids the need to escape quotation marks and allows text editors to recognize and highlight the code as SQL, both of which becoming increasingly valuable as the length and complexity of SQL queries increase. On the conceptual side, using the SQL engine visually separates database queries from R code and text, thereby better communicating the structure and functioning of analyses in Rmarkdown documents.

The `knitrdata` package works in many ways analogously to the SQL language engine, adding a new `data` language engine to the list of language engines known to `knitr` that is specifically designed to import raw "data" into the R environment and/or export it to external files. Here the term "data" is used in a very wide sense, including not only standard data tables (e.g., CSV text files) or binary data (e.g., RDS files, NetCDF files, images), but also text and formatted text (e.g., XML files, BibTeX files). For example, placing the following chunk in a Rmarkdown source document will, during the knitting process, create in the R environment a data frame `d` containing the contents of the comma-separated values (CSV) data in the chunk (provided that the `knitrdata` package has been previously loaded as described above):

````
```{data output.var="d",loader.function=read.csv}
name,score
David M. Kaplan,1.2
The man formerly known as "Prince",3.4
Peter O'Toole,5.6
```
````

As with the SQL language engine, the name of the output variable for the chunk is supplied with a chunk option and in this example a `loader.function` option instructs `knitrdata` how to translate the contents of the chunk into a usable R object (in this example the R function `read.csv` is used to translate the CSV data into a data frame).

There are of course a number of other ways that such a simple data table could be imported into the R environment, including via an external data file or directly in R code, one approach to which might be:

````
```{r}
d = read.csv(textConnection(
"name,score
David M. Kaplan,1.2
The man formerly known as \"Prince\",3.4
Peter O'Toole,5.6
"))
```
````

However, using the `data` language engine has much the same practical and conceptual advantages as the SQL data language engine, avoiding the need for escaping certain characters and visually separating data from code, both of which become increasingly valuable as dataset size increases.

Incorporating binary data into Rmarkdown source documents is a bit more complicated as the data must first be *encoded* as ASCII text (see the Section below on Binary data chunks for details), but the basic principles are the same - encoded binary data is incorporated into a `data` chunk and chunk options are used to tell `knitrdata` how to decode the data and load it into the R environment during knitting (see Table 1 for a full list of `data` chunk options). There is also the possibility of saving `data` chunk contents out to external files using the `output.file` chunk option. This option is particularly useful for integrating into Rmarkdown source documents ancillary text files used in the final formatting of the output of the knitting process, such as BibTeX files with references, LaTeX style files for PDF output and CSS style files for HTML output. For example, the following chunk would export a BibTeX reference to a file named `refs.bib`, taking care not to overwrite an existing file with

the same name [though note that similar functionality can also be achieved with the `cat` language engine; Xie, Dervieux, and Riederer (2020)]:

````
```{data output.file = "refs.bib", eval=!file.exists("refs.bib")}

@book{allaireRmarkdownDynamicDocuments2020,
  title = {Rmarkdown: {{Dynamic}} Documents for r},
 author = {Allaire, JJ and Xie, Yihui and McPherson, Jonathan and Luraschi, Javier and Ushey, Kevin and Atkins, Ar
  year = {2020}
}
```
````

As code chunks are processed during knitting before generating the final output document, these files can be generated at any point during the knitting process using `data` chunks (in particular, it is often most practical to place this information at the end of a Rmarkdown document).

Table 1: Full list of knitrdata chunk options.

Chunk option	Description
`decoding.ops`	A list with additional arguments for `data_decode`. Currently only useful for passing the `verify` argument to gpg::`gpg_decrypt` (Ooms 2022) for gpg encrypted chunks.
`echo`	A boolean indicating whether or not to include chunk contents in Rmarkdown output. Defaults to `FALSE`.
`encoding`	One of `'asis'`, `'base64'` or `'gpg'`. Defaults to `'asis'` for `format='text'` and `'base64'` for `format='binary'`.
`eval`	A boolean indicating whether or not to process the chunk. Defaults to `TRUE`.
`external.file`	A character string with the name of a file whose text contents will be used as if they were the contents of the data chunk.
`format`	One of `'text'` or `'binary'`. Defaults to `'text'`.
`line.sep`	Only used when `encoding='asis'`. In this cases, specifies the character string that will be used to join the lines of the data chunk before export to an external file, further processing or returning the data. Defaults to `knitrdata::platform.newline()`.
`loader.function`	A function that will be passed (as the first argument) the name of a file containing the (potentially decoded) contents of the data chunk.
`loader.ops`	A list of additional arguments to be passed to `loader.function`.
`max.echo`	An integer specifying the maximum number of lines of data to echo in the final output document. Defaults to 20. If the data exceeds this length, only the first 20 lines will be shown and a final line indicating the number of ommitted lines will be added.
`md5sum`	A character string giving the correct md5sum of the *decoded* chunk data. If supplied, the md5sum of the decoded data will be calculated and compared to the supplied value, returning an error if the two do not match.
`output.file`	A character string with the filename to which the chunk output will be written. At least one of `output.var` or `output.file` must always be supplied.
`output.var`	A character string with the variable name to which the chunk output will be assigned. At least one of `output.var` or `output.file` must always be supplied.

Using `data` chunks, just about any data or information that would typically be stored in external files can be integrated directly into Rmarkdown source documents. In particular, this permits creating standalone Rmarkdown source documents that can be knitted without need for external data files, thereby uniting text, code and data in a single source document.

Note that this is different from the `self_contained` YAML header option permitted by some Rmarkdown output formats, notably HTML output formats. This option attempts to create a single *output* file that contains everything needed to visualize the final output document (e.g., in the case of HTML documents, the output HTML file will contain any CSS styles, javascript libraries and/or images used by the document), but it says nothing about whether or not external files are needed to

knit the Rmarkdown source document (i.e., it is relevant to the output side of knitting, not the input side). In fact, a source document can be standalone in that all data and formatting files needed for knitting are incorporated within it using `data` chunks, but the final output (HTML) document may not be self contained because it relies on external files or libraries for visualization, and vice-versa (i.e., standalone source documents and standalone output documents are two separate and independent concerns).

Under the hood, the way `knitrdata` works is by adding (using the `knitr::knit_engines$set()` function) to the list of language engines that `knitr` maintains internally a `data` entry that points to a function inside the `knitrdata` package that processes `data` chunks (specifically the `eng_data` function, though users would typically not interact directly with this function). When knitting a Rmarkdown document, `knitr` will call this function each time a `data` chunk is encountered, passing it both the textual contents of the chunk and any chunk options. The function then uses this information to process the chunk, decoding it if necessary (via the `format` and `encoding` chunk options) and returning it as either a variable in the R environment (`output.var` chunk option) and/or an external file (`output.file` chunk option) after any additional processing has been carried out (via, e.g., the `loader.function` chunk option).

Text data chunks

Though a basic example of a `data` chunk containing CSV tabular data has been presented in the previous section, it is useful to develop that example a bit more to better understand the functioning of `knitrdata`. The simplest `data` chunks contain plain text that is read, but not processed by `knitrdata`. For example, omitting the `loader.function` chunk option from the previously presented `data` chunk with CSV data produces a different outcome:

````
```{data output.var="txt"}
site,density
a,1.2
b,3.4
c,5.6
```
````

During the knitting process, this will place the text contents of the chunk into a R variable named `txt`, but no further processing of the text will be carried out (i.e., the variable `txt` will contain the literal text contents of the chunk, excluding the header with the chunk options and the tail). One could later convert the text into a R `data.frame` using the `read.csv` command in a R chunk placed after the `data` chunk:

````
```{r}
d <- read.csv(text=txt)
```
````

The `loader.function` chunk option used in the initial `data` chunk example above causes `knitrdata` to combine into one process the two steps of (1) reading in the chunk contents and (2) converting them into a usable R data object. Whereas the first of these steps, reading the chunk contents, is carried out for all `data` chunks, the second only occurs if the `loader.function` chunk option is given. `loader.function` should be a function that takes in the name of a file containing the chunk contents and returns the processed contents. Though `read.csv` is likely to be a common choice, there are many other possibilities including `readRDS`, `read.csv2`, `scan`, `png::readPNG` and custom, user-defined functions. One can also supply a list of additional input arguments to the loader function using the `loader.ops` chunk option (e.g., one could change the expected separator in CSV data using `loader.ops=list(sep="|")`).

Binary data chunks

Though text data chunks can integrate into Rmarkdown source documents many small- to medium-sized tabular data sets, binary data formats, such as RDS files, are more convenient for more complicated and/or larger data sets. Incorporating binary data into Rmarkdown documents requires additional steps relative to text data: encoding the binary data as text and telling the `data` chunk how to decode the encoded data. `knitrdata` provides tools for simplifying these two steps that currently support encoding and decoding of two widely-used encodings: `base64` and `gpg`. `base64` is a standard format for encoding binary data as ASCII text based on translation of 6 bits of information into one

of 64 alphanumeric and symbolic ASCII characters. Base64 encoded data looks like a somewhat intimidating jumble of characters, but the format is extremely widely used behind the scenes in many common web applications, such as email attachments and embedding images in HTML pages. In particular, `base64` is widely supported by a number of software packages and programming languages, including R, Python, Matlab and Julia, so `base64` encoded data is highly readable and likely to remain so for a very long time.

`gpg`, standing for GNU Privacy Guard, is a standard protocol for encrypting information so that only those with specific decryption keys can have access. This format can be used to ensure that only specific individuals can actually read and utilize the data contained in a Rmarkdown source document, as might be necessary when dealing with confidential (e.g., medical or trade-secret) data. Here, I focus primarily on `base64` encoding as this is the simplest and likely most common format for binary `data` chunks, and a full description of the configuration and use of GPG is beyond the scope of this document. The detailed use of `gpg` is, however, described in the package vignette.

Though `knitrdata` users rarely need to encode data by hand as the package provides graphical tools for this, it is instructive to have a basic understanding of the underlying functions for encoding and decoding data: `data_encode` and `data_decode`. `data_encode` takes the name of the file containing the data and the name of the encoding format, and it returns the encoded data that one would incorporate into a `data` chunk, either to the R command line or to a file. For example, if one saves the data frame `d` created in the previous section to a binary `RDS` file:

```
saveRDS(d,"data.RDS")
```

then one can encode this data as `base64` using:

```
b64 = knitrdata::data_encode("data.RDS","base64")
cat(b64)
```

```
#> H4sIAAAAAAAAA12OvQ+CMBDFKx+DGNTExPk2J1lc3HQwLsbIgInrBUokQmsKkbj5
#> PzsrXrEMekn72l/f693JY4zZzLEsZrt0ZO4x2s6XxCZ0sWiNvwbWJx1t8JYlsA9g
#> h9cchcEQnTkUKCCVquAqv8NFyFoAlhCqTMTc+PyQV1zBYRZJmXMCQ/336rloaz0w
#> Ok3bYjQVvfdM2BVY8NIMZBnoaNgZylgq/p+Kcyy7VAe9BCsMUqWzv/a+knXQNfJ1
#> owdtTdO8SN4fPb8RnS0BAAA=
```

This jumble of characters starting with `"H4sI"` is the base64 encoded contents of the binary file that one would place in a `data` chunk. For large files, it is often more practical to output the encoded data to a file by supplying the `output` argument:

```
knitrdata::data_encode("data.RDS","base64",
                       output="data.RDS.base64")
```

GPG encoding works similarly to base64 encoding, but one must change the format from `"base64"` to `"gpg"` and specify the encryption key (i.e., the receiver ID) to be used to encrypt the data.

Once one has the encoded data, one can use it in a `data` chunk by supplying the `format="binary"` chunk option and, optionally, an appropriate `loader.function` to convert the data into a R object:

````
```{data format="binary",output.var="d",loader.function=readRDS}
H4sIAAAAAAAAA12OvQvCMBDFz48OKn6A4Hybk11c3HQQFxE7VHA9aopimkgiFjf/
Z2etl5oOesO93I/3crdvA0ADmnXuAT8h2MWryYynIQ9MYfA1QIu1v6Tb6YCbENd0
kaQ8xvgoMCOFqTaZMPKOZ6VzhWQxMieVCO/rRuIqDG7HsdZSMOi5v+fPaVmLjtdR
WhaUV0HNhwNFmbD+oLqHTQcrg020Ef+pRJKtUhVsH+hKYWpc9tfeMjoPq0Vdt+jB
rSiKF8v7A6bdy9EtAQAA
```
````

During knitting, this chunk will be processed, decoding the encoded binary RDS file and loading it into the variable `d` using the `readRDS` function. `knitrdata` will by default assume that the encoding is base64 when `format="binary"`, but one can also specify the chunk option `encoding="base64"` for increased clarity. For GPG encoded data, one would use `encoding="gpg"`. As with text `data` chunks, one can alternatively output the decoded contents of the chunk to a file (`output.file` option) or return it to the R session as a `raw` binary vector (by not supplying a `loader.function`).

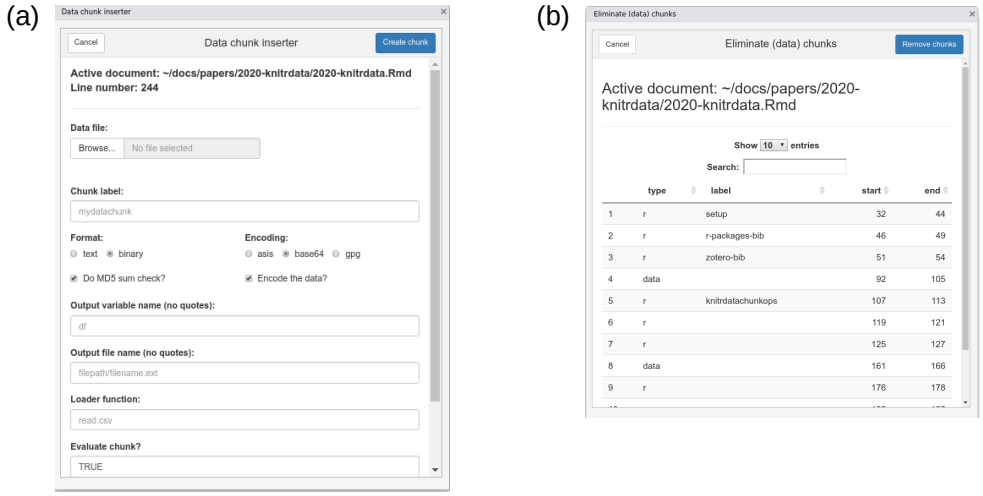

Figure 1: The (a) 'Insert filled data chunk' and (b) 'Remove chunks' RStudio add-ins included with knitrdata. The dialogues will open when selected from the 'Addins' menu of RStudio. They allow one to (a) insert a data chunk containing the contents of an existing external data file into an open Rmarkdown document, and (b) delete one or more chunks from an open Rmarkdown document.

RStudio add-ins for creating data chunks

As manually encoding data and creating `data` chunks can be complicated, particularly for large data files, `knitrdata` includes graphical RStudio add-ins that do all the hard work of incorporating data in Rmarkdown documents. The principal add-in is called `Insert filled data chunk` (Fig. 1a). Though its use is meant to be largely self-explanatory, an instructional video is available on YouTube (Kaplan 2020b). The basic idea is that one opens a Rmarkdown source document in the RStudio editor, places the cursor at the location one wants to insert a `data` chunk and then activates the add-in. The add-in prompts for the name of the data file to be incorporated, as well as values for various `data` chunk output and processing options. Based on the type of data file selected, the add-in will attempt to select or suggest various appropriate options. For example, if a RDS file is chosen, then format will be set to `binary`, encoding will be set to `base64` and the loader function will be set to `readRDS`. These defaults can be manually modified if not appropriate. The add-in also greatly facilitates and encourages the use of MD5 sum data integrity checks. After all options have been set, one clicks on `Create chunk` and an appropriately-formatted `data` chunk will be inserted in the active Rmarkdown source document at the cursor location.

`knitrdata` also provides a `Remove chunks` add-in that allows ones one to quickly delete unwanted (`data` and non-`data`) chunks (Fig. 1b), as well as a set of functions for command-line examination, creation and removal of chunks from Rmarkdown documents (e.g., `create_chunk`, `insert_chunk`, `list_rmd_chunks`, `remove_chunks`).

If one is not using RStudio to edit and knit Rmarkdown documents, then one can invoke the Skiny dialog to create `data` chunks directly from the command line using the `create_data_chunk_dialog` function contained in the `knitrdata` package. In this case, chunk contents will be (silently) returned on the command line for later insertion in a Rmarkdown document.

Use cases

There are a number of use cases for the functionality provided by `knitrdata`, primary among them providing a single source for public diffusion of all information related to a publication or report, and/or making collaboration on Rmarkdown source documents simpler by eliminating or reducing

the need for external files. A simple example of the prior is the Rmarkdown source document used to generate this publication, which includes text `data` chunks for the tabular data in Table 1, as well as the ancillary formatting files associated with the document (BibTeX and LaTeX style files), and encoded binary `data` chunks for the PNG images in Figs. 1 & 2.

A more complicated example is the Rmarkdown source document for Wain et al. (2021), publicly available on github. In this case, we wished to provide a permanent public record of the methods used in the paper and ensure that results could be verified, while at the same time respecting confidentiality agreements with respect to fine scale fishing activity data used in the paper. To achieve this we integrated the fine scale data in the Rmarkdown source document as an encrypted GPG `data` chunk. This approach may have value for a wide number of other studies using sensitive economic, social or medical data. To provide a complete record of the paper in a single document, we also integrated the Rmarkdown source document for the online supplementary materials into a `data` chunk within the Rmarkdown source document for the paper itself. As this supplementary materials document contains Rmarkdown chunks that would otherwise confuse the knitting process if integrated as raw text inside a `data` chunk, we base64 encoded this source document before including it in a `data` chunk. The document also contains `data` chunks for small data tables and for exporting to external files the ancillary formatting files required for knitting the document (BibTeX references, the LaTeX style file, the CSL citation style file, etc.). Finally, during the knitting process, the document also generates a lightweight version of itself that does not include the main data chunk, using the functionality of the `knitrdata` package to remove large data chunks. Overall, `knitrdata` provided a convenient way of generating a single document that contained all the necessary information for generating the final publication, thereby demonstrating computational reproducibility for the publication.

The uses of `data` chunks tend to fall into one of four general, not mutually-exclusive use cases:

1) Integration of ancillary formatting files into the Rmarkdown source document, thereby reducing the number of external files needed to knit a document
2) Inclusion of small- to medium-sized tabular data used in analyses and/or for tables
3) Inclusion of larger data sets using encoded binary data
4) Inclusion of confidential data using GPG-encrypted `data` chunks

Though the first of these use cases, integration of ancillary formatting files, can also be achieved with the `cat` language engine that is included with the `knitr` package (Xie, Dervieux, and Riederer 2020), `knitrdata` provides functionality that make this task easier and more secure. First, `knitrdata` allows for integrity checks on chunk contents that can control for unintentional modification of chunk contents (see the section on data integrity below). Second, RStudio add-ins provided by the `knitrdata` package facilitate the integration of data into Rmarkdown source documents and the use of integrity checks. Finally, encoding of text documents permits integrating files that contain Rmarkdown chunks or other formatting that would otherwise be problematic within a `cat` chunk.

The second of these use cases, tabular data, can also in principle be achieved using other tools in R, such as a `textConnection` as shown above or via functionality in the **tibble** package (Müller and Wickham 2022). Nevertheless, the use of `data` chunks is generally more ergonomic and flexible for anything but the smallest data tables as it allows the user to format data exactly as it would be in an external CSV file, without additional markup or the need to escape quotation marks. As an example, the information contained in Table 1 was implemented in the source document for this paper as a `data` chunk as it contains lots of quotations and formatting that would have been tedious to include using other approaches.

The third and fourth use cases for `data` chunks, involving encoded binary data, are unique to `knitrdata` and allow for integration of complex data sets that would otherwise be very difficult to include in a Rmarkdown source document.

Workflow

When and in what ways to use the functionality provided by `knitrdata` during the development of a Rmarkdown source document requires some thought and depends to some degree on the project goals. If the goal is to create a final Rmarkdown source document that demonstrates computational reproducibility of a set of results, then it may not be necessary or practical to use `data` chunks during the development stages of the project as the use of `data` chunks necessarily weighs down a Rmarkdown source document with information (e.g., binary data) that may not be immediately useful to authors during development. In this case, it may be best to work initially as one has always done, relying on external files for data and formatting. External data and formatting files can be incorporated in `data` chunks at the end of development when it is time to generate a final archival/public version of a Rmarkdown source document.

On the other hand, if the objective of using `knitrdata` also includes reducing the complexity of collaborating on a Rmarkdown source document by reducing the number of external files necessary for knitting a document, then certain types of `data` chunks can be incorporated in a Rmarkdown source document during the initial phases of development with little impact on authors. Small- to medium-sized tabular data sets can be incorporated and this can have the benefit of making the tabular data visually available during the development process. Similarly, most ancillary formatting files can be placed at the very end of the Rmarkdown source document as these are only used after all chunks have been processed and, therefore, will not encumber the development process. Larger data sets, and in particular binary data sets, are a bit more problematic as they necessarily appear in the Rmarkdown source document before the data is used for analyses and will introduce significant amounts of text that are not human readable into the Rmarkdown source document. For this reason it may be best to leave incorporation of these data until the final stages of development, though see the sections below on file size and readability for workarounds to these issues.

This latter workflow involving incorporation of `data` chunks in two distinct stages of development is what was used when creating the source document for Wain et al. (2021). Small data tables and formatting files were incorporated directly into the document from the start, but the larger data set that was the basis for statistical analyses and the Rmarkdown source document for the supplementary materials were only incorporated at the end of development to provide an archival source document for the paper capable of demonstrating computational reproducibility.

Caveats and concerns

There are a number caveats and concerns with respect to the use of `knitrdata`, all of which have some validity, but for which a number of simple approaches exist to limit their impact. Below, I discuss four of them: file size, document readability, data integrity and security.

Won't this create huge Rmarkdown files?

Incorporation of large data sets into `data` chunks will significantly increase the file size of Rmarkdown source documents, potentially making them more difficult to work with. Though it is unlikely to be practical to place extremely large data sets in Rmarkdown source documents, there are many contexts where data sets are sufficiently small so as to be included directly in a Rmarkdown source document. For example, the 8 years of fine scale fishing data used in Wain et al. (2021) added about 2 MB to the size of the Rmarkdown document when incorporated as a (compressed) RDS file, a size that is manageable and well within the limits of typical email attachments. RStudio currently will not open Rmarkdown documents larger than 5 MB in size, effectively limiting the amount of data that can be placed in a document unless one is willing to forgo graphical editing tools (larger documents can be rendered from the command line using the `rmarkdown::render` function, but not the more convenient and common "Knit" button of RStudio). Despite being in the era of big data, many scientific studies use primary data sets that are smaller than this size limit. As for Wain et al. (2021), many experimental or field studies may rely on data sets that are relatively small, and building a standalone Rmarkdown source document for these studies is an effective approach to documenting all quantitative information needed to reproduce results.

Won't Rmarkdown source documents become unreadable?

Large amounts of encoded binary data are undoubtedly not pretty to look at, but readability is not necessarily the primary benefit of using Rmarkdown. Rather, completeness and the articulation of text and analyses are the strengths of Rmarkdown, benefits that `data` chunks enhance as opposed to diminish. Many Rmarkdown documents are already a complex mix of text and code that is difficult to read and manage without the tools RStudio and other editors provide to navigate the document, such as the ability to jump between sections and chunks. `data` chunks are no different in this sense, and use of informative chunk labels can greatly facilitate document navigation. Furthermore, RStudio includes the possibility of hiding chunk contents with a single click (Fig. 2), which can be quite practical when dealing with large `data` chunks. Once hidden, `data` chunk contents can be ignored, allowing one to edit the document unhindered.

Won't a misplaced keystroke mess up my data?

It is possible to unintentionally corrupt a `data` chunk due to a misplaced keystroke, particularly if the data is encoded and not readily human readable. However, the use of navigation tools and hiding

Figure 2: Images demonstrating before (top) and after (bottom) a data chunk has been hidden from view in the RStudio editor. The top image shows two base64-encoded data chunks, one of which is hidden, whereas the other is visible. In the bottom image, both chunks have been hidden. The control for hiding chunk contents (top) and an indicator of a hidden chunk (bottom) are highlighted with red boxes.

of `data` chunk contents as described above (Fig. 2) can drastically reduce interaction with chunk contents, thereby limiting the possibility for error. Furthermore, there are a number of methods to validate chunk contents, the simplest of which is to do a MD5 sum check using functionality included in `knitrdata`. A MD5 sum is a very large number (typically encoded in hexadecimal) derived from a file's contents that has a vanishingly small probability of being equal to that of another file if the files are not identical. `data` chunks can include a `md5sum` chunk option that specifies a MD5 sum that will be checked against that of the decoded chunk contents, generating an error if the two do not match. In this way, data corruption can be swiftly identified and corrected. The `Insert filled data chunk` RStudio add-in will by default calculate and include a MD5 sum check when inserting binary `data` chunks (and such a check can be optionally requested for text `data` chunks) so that users can easily benefit from these checks without having to worry about the details.

Are there security concerns when using `knitrdata`?

Any time one runs code from a third party, there are security risks. Typically, code can write files to disk, potentially modifying essential files or installing malicious software. Rmarkdown source documents using the functionality of `knitrdata` are no different in this sense, though the practical risks may be more important as `knitrdata` may encourage users to knit entire documents to gain access to raw data and arbitrary data may be encoded in formats that are not human readable. Reducing these risks involves responsibilities for both the authors and the users of Rmarkdown source documents. For authors, the primary responsibilities are to assure that source documents cannot be modified by third parties between the author and the user, and to use best practices when carrying out file input and output during the knitting process. Integrity of source documents can be protected by using reputable websites with established security protocols for publishing Rmarkdown source documents, including, but not limited to, github, Zenodo and the Dryad. Authors can also publish MD5 sums for Rmarkdown source documents so users can verify the integrity of those documents, though the security of those MD5 sums is only as strong as the websites on which MD5 sums and Rmarkdown source documents are published. Best practices for file input and output include using temporary files and/or relative paths entirely within the base directory containing the Rmarkdown source document when writing files to disk, using file names that are unique (e.g., avoiding generic names like `data.csv`)

and performing checks for the existence of files with the same name before writing information to disk. The `Insert filled data chunk` RStudio add-in provided by `knitrdata` encourages the use of file existence checks in the `eval` chunk option controlling whether or not to process `data` chunks that write data to disk using the `output.file` chunk option, thereby avoiding overwriting existing files.

For users of Rmarkdown source documents, there are a number of simple steps one can take to avoid the most severe security risks. Knitting Rmarkdown source documents from an unprivileged user account and placing Rmarkdown source documents in new, empty directories can reduce the risks of the most malicious attacks. Users should also familiarize themselves with the workings of Rmarkdown source documents before knitting them and check for potentially problematic actions, such as use of absolute file paths and/or communication with external internet resources. This includes examination of the chunk options associated with `data` chunks (in particular, the `output.file` and `loader.function`). If one is primarily interested in just the raw data contained in `data` chunks within a Rmarkdown source document, then RStudio permits manual execution of individual chunks. This includes execution of `data` chunks, which can be processed individually using the `Run current chunk` button of RStudio once the `knitrdata` library has been loaded.

Conclusion

`knitrdata` provides a simple, but effective, tool for integrating arbitrary data into Rmarkdown source documents. If used appropriately, this can help assure computational reproducibility of many scientific documents by allowing one to integrate all relevant external files and data directly into a single Rmarkdown source document. Anyone who has attempted to validate the results in a publication by requesting the associated data has potentially encountered, if they managed to get the data, a set of one or more data tables with limited metadata and only the publication itself as documentation of the methods. Validating publication results under these conditions is often difficult and time consuming. By encouraging the integration of data, code for carrying out analyses, and text interpreting results in standalone Rmarkdown source documents, Rmarkdown with `knitrdata` can make it much easier to understand, reproduce and validate the details of scientific analyses. This combination can be particularly powerful when combined with other enhancements to Rmarkdown that make it possible to produce a wide variety of sophisticated scientific documents entirely within the confines of Rmarkdown, such as **bookdown** (Xie 2022), **rticles** (Allaire, Xie, Dervieux, et al. 2022) and **starticles** (Kaplan 2022) for producing books and scientific publications with Rmarkdown, **citr** (Aust 2019) for bibliographic citations, and **kableExtra** (Zhu 2021) for producing sophisticated data tables.

Online supporting information

The Rmarkdown source documents for this publication and Wain et al. (2021) are available online at https://github.com/dmkaplan2000/knitrdata_examples. Additional examples and the package vignette are available in the `knitrdata` package itself.

Acknowledgements

I would like to thank my colleagues at the MARBEC laboratory in Sète, France for numerous conversations that encouraged me to develop the `knitrdata` package. I would also like to thank Yihui Xie for advice and encouragement regarding the development of the package. The handling editor and an anonymous reviewer provided valuable feedback that significantly improved the manuscript.

References

Allaire, JJ, Yihui Xie, Christophe Dervieux, R Foundation, Hadley Wickham, Journal of Statistical Software, Ramnath Vaidyanathan, et al. 2022. *Rticles: Article Formats for r Markdown*. https://github.com/rstudio/rticles.

Allaire, JJ, Yihui Xie, Jonathan McPherson, Javier Luraschi, Kevin Ushey, Aron Atkins, Hadley Wickham, Joe Cheng, Winston Chang, and Richard Iannone. 2022. *Rmarkdown: Dynamic Documents for r*. https://CRAN.R-project.org/package=rmarkdown.

Aust, Frederik. 2019. *Citr: RStudio Add-in to Insert Markdown Citations*. https://github.com/crsh/citr.

Cassey, Phillip, and Tim M. Blackburn. 2006. "Reproducibility and Repeatability in Ecology." *BioScience* 56 (12): 958–59. https://doi.org/10.1641/0006-3568(2006)56%5B958:RARIE%5D2.0.CO;2.

Csárdi, Gábor, Jim Hester, Hadley Wickham, Winston Chang, Martin Morgan, and Dan Tenenbaum. 2021. *Remotes: R Package Installation from Remote Repositories, Including GitHub*. https://CRAN.R-project.org/package=remotes.

Kaplan, David M. 2020a. *Knitrdata: Data Language Engine for Knitr / Rmarkdown*. https://github.com/dmkaplan2000/knitrdata.

———. 2020b. "Using Knitrdata to Create Standalone Rmarkdown Documents in Rstudio." https://www.youtube.com/watch?v

———. 2022. *Starticles: A Generic, Publisher-Independent Template for Writing Scientific Documents in Rmarkdown*. https://github.com/dmkaplan2000/starticles.

Lowndes, Julia S. Stewart, Benjamin D. Best, Courtney Scarborough, Jamie C. Afflerbach, Melanie R. Frazier, Casey C. O'Hara, Ning Jiang, and Benjamin S. Halpern. 2017. "Our Path to Better Science in Less Time Using Open Data Science Tools." *Nature Ecology & Evolution* 1 (6): 1–7. https://doi.org/10.1038/s41559-017-0160.

Müller, Kirill, and Hadley Wickham. 2022. *Tibble: Simple Data Frames*. https://CRAN.R-project.org/package=tibble.

Ooms, Jeroen. 2022. *Gpg: GNU Privacy Guard for r*. https://github.com/jeroen/gpg.

Powers, Stephen M., and Stephanie E. Hampton. 2019. "Open Science, Reproducibility, and Transparency in Ecology." *Ecological Applications* 29 (1): e01822. https://doi.org/10.1002/eap.1822.

Wain, Gwenaëlle, Loreleï Guéry, David Michael Kaplan, and Daniel Gaertner. 2021. "Quantifying the Increase in Fishing Efficiency Due to the Use of Drifting FADs Equipped with Echosounders in Tropical Tuna Purse Seine Fisheries." *ICES Journal of Marine Science* 78 (1): 235–45. https://doi.org/10.1093/icesjms/fsaa216.

Xie, Yihui. 2014. "Knitr: A Comprehensive Tool for Reproducible Research in R." In *Implementing Reproducible Computational Research*, edited by Victoria Stodden, Friedrich Leisch, and Roger D. Peng. Chapman; Hall/CRC. http://www.crcpress.com/product/isbn/9781466561595.

———. 2015. *Dynamic Documents with R and Knitr*. 2nd ed. Boca Raton, Florida: Chapman; Hall/CRC. https://yihui.org/knitr/.

———. 2022. *Bookdown: Authoring Books and Technical Documents with r Markdown*. https://CRAN.R-project.org/package=bookdown.

Xie, Yihui, Christophe Dervieux, and Emily Riederer. 2020. *R Markdown Cookbook*. 1st edition. The R Series. Boca Raton, Florida: CRC Press.

Zhu, Hao. 2021. *kableExtra: Construct Complex Table with Kable and Pipe Syntax*. https://CRAN.R-project.org/package=kableExtra.

David M. Kaplan
MARBEC
Univ Montepllier, CNRS, Ifremer, IRD
Sète, France
Institute de Recherche pour le Developpement (IRD)
UMR MARBEC
av. Jean Monnet
CS 30171
34203 Sète cedex, France
https://www.davidmkaplan.fr
ORCiD: 0000-0001-6087-359X
david.kaplan@ird.fr