

# How to automatically document data with the codebook package to facilitate data re-use

*forthcoming in Advances in Methods and Practices in Psychological Science*

## Author

Ruben C. Arslan

Center for Adaptive Rationality, Max Planck Institute for Human Development, Berlin

[ruben.arslan@gmail.com](mailto:ruben.arslan@gmail.com)

## Abstract

Data documentation in psychology lags behind not only many other disciplines, but also basic standards of usefulness. Psychological scientists often prefer to invest the time and effort necessary to document existing data well into other duties such as writing and collecting more data. Codebooks therefore tend to be unstandardised and stored in proprietary formats, and are rarely properly indexed in search engines. This means that rich datasets are sometimes used only once—by their creators—and left to disappear into oblivion; even if they can find it, researchers are unlikely to publish analyses based on existing datasets if they cannot be

confident they understand them well enough. My *codebook* package makes it easier to generate rich metadata in human- and machine-readable codebooks. By using metadata from existing sources and by automating some tedious tasks such as documenting psychological scales and reliabilities, summarising descriptives, and identifying missingness patterns, I aim to encourage researchers to use the package for their own or their team's benefit. The *codebook* R package and web app make it possible to generate rich codebooks in a few minutes and just three clicks. Over time, this could lead to psychological data becoming findable, accessible, interoperable, and reusable, and to reduced research waste, thereby benefiting the scientific community as a whole.

## Links

Open data: <https://osf.io/k39bg/> DOI:10.17605/OSF.IO/K39BG

Open code/materials: <https://github.com/rubenarслан/codebook> DOI:10.5281/zenodo.2574896

Preprint <https://psyarxiv.com/5qc6h> doi:10.31234/osf.io/5qc6h

Documentation and vignettes: <https://rubenarслан.github.io/codebook/>

Package on CRAN <https://cran.r-project.org/web/packages/codebook/index.html>

Web app <https://codebook.formr.org>

Codebook gallery [https://rubenarслан.github.io/codebook\\_gallery/](https://rubenarслан.github.io/codebook_gallery/)

Codebooks as indexed by Google

<https://toolbox.google.com/datasetsearch/search?query=site%3Arubenarслан.github.io&docid=pQmFCzVqf%2B43vYZYAAAAAA%3D%3D>

# Introduction

Psychologists rarely ensure that their data are findable, accessible, interoperable, and reusable (FAIR; (Borghi & Van Gulick, 2018; Hardwicke et al., 2018; Kidwell et al., 2016; Wilkinson et al., 2016). Data re-use is therefore difficult for those who lack intimate knowledge of the dataset—including co-authors and one’s future self. Furthermore, datasets are rarely documented in standard formats palatable to search engines and other algorithmic approaches to data, and often only a processed and subset form of the data without item-level information is released.

There are at least three likely reasons for insufficient documentation. First, the curse of knowledge: It is difficult to imagine which descriptions will be cryptic to readers who are not familiar with the data. Second, a lack of standards and training: Psychological scientists tend to receive little to no training in data documentation, which, when offered, is often done ad hoc. Because many datasets in psychology are small and specific to certain research questions, few departments hire specialised data librarians who can concentrate knowledge on the topic and support researchers in archiving useful data. However, even disciplines with a stronger history of data sharing, such as ecology, struggle to share re-usable data (Roche, Kruuk, Lanfear, & Binning, 2015). Third, the tragedy of the commons: Career incentives do not favor the significant time investment necessary for the data documentation that would best advance knowledge generation in the long term.

The *codebook* package gives psychological scientists something that is useful for them in the short term, potentially even saves them time, in order to create a resource that is useful for all in the long term. It automates commonly performed data summary steps such as

generating descriptive statistics and plots. It can compute state-of-the-art reliability indices automatically (Crutzen, 2007; McNeish, 2018; Peters, 2014). It makes variable and value labels easily accessible in R, while at the same time generating standardised metadata about a dataset that can be read by other researchers and by search engines (e.g., Google Dataset Search<sup>1</sup>) and other data processors.

## Roles of a Codebook

A good codebook<sup>2</sup> fulfills several roles. By giving a high-level summary of the data and providing meaningful labels, it can make it easier to discover errors, miscoded missing values, oddly shaped distributions, and other cues for problems, thereby allowing data creators to clean datasets more efficiently and reproducibly. A high-level summary, ideally combined with text explaining the structure and nature of the dataset, also helps to explain unfamiliar datasets for those who seek to reproduce analyses or re-use data. This should lead to fewer errors resulting from researchers analyzing a dataset without understanding measurement specifics, which values encode missingness, whether certain survey weights have to be used, and so on. Codebooks also offer standardization. There are few exceptions (Gorgolewski et al., 2016) to the general lack of standards in psychological data description. Projects using integrated data analysis—that is, performing the same analysis across multiple cohorts to boost power and generalisability (Leszko, Elleman, Bastarache, Graham, & Mroczek, 2016)—currently devote ample amounts of time getting multiple datasets into the same format. Human analysts benefit from standardisation, but they can make do with unstandardised data if need be. Search engines and other algorithmic approaches, however, have a more difficult task – and we rely on

---

<sup>1</sup> <https://toolbox.google.com/datasetsearch>

<sup>2</sup> Other widely used related terms are data dictionary, data documentation, and simply metadata (data about data).

search engines a lot in the search for information. Without a good codebook, how can a search engine tell whether it is dealing with a dataset in which intelligence was measured or with one that asks whether elderly people would be willing to use an intelligent household robot? The task becomes even more challenging when it comes to structural aspects of the data: How do search engines identify a study using peer reports, or dyadic data? How do we differentiate experiments in which mood was manipulated from those in which it was the outcome? And how can we filter our search by sample size to find only datasets which have measured at least 100 individuals at least 10 times? For example, the Open Science Framework (OSF) currently relies on user-supplied tags—a very limited approach—and is not indexed in Google Dataset Search. As a result, it is difficult to find a dataset on the OSF without either knowing exactly where to look or investing a lot of time.

## The codebook package

### Immediate Benefits

The codebook package (Arslan, 2018) makes it easy to share codebooks in five commonly used formats: PDFs, HTML websites, spreadsheets, R data frames, and proprietary dataset formats (e.g., SPSS or Stata files). These make it easy for a researcher to share information about a dataset with co-authors, reviewers, and readers. The codebooks generated by the package are more extensive and more portable than most other current approaches: Not only do they include metadata (i.e., data about data) such as variable names and variable and value labels, but they also contain high-level summaries of the data, such as the means and standard deviations, plots of the distribution, number of missing values and complete entries,

and missingness patterns in the data. If other metadata are also available, such as information about which items were shown to whom or the order in which questions were asked, this is also displayed.

By making dataset contents and structure more transparent, the codebook package makes it easier for researchers to find useful datasets for their research question using search and to decide whether to (re-)use a dataset for a particular research question without seeing the dataset itself. This is particularly useful in cases when the data cannot be openly shared and access needs to be requested, for instance due to privacy constraints; it is also more convenient than downloading and examining a plethora of datasets when data is open. It is especially useful for search between adjacent fields where researchers do not come across datasets that could be repurposed for their question in the literature.

Even after downloading a dataset, an analyst will go back and forth between metadata and data frequently in order to find relevant variables, refresh their memory, or just label axes accurately. By making metadata available while working in R, the codebook package puts this information at the analyst's fingertips.

## Long-Term Benefits

When analysts and would-be data sharers use the codebook package to document their datasets, they also create machine-readable metadata. Hidden in the generated HTML files are JSON-LD (Javascript Object Notation - Linked Data) blocks. This format is an extensible shared vocabulary for datasets which is supported by search engines such as Google. Large-scale providers of public data such as IPUMS<sup>3</sup> already generate this sort of metadata. Other, older solutions that cater to providers of large datasets also exist, such as the Data

---

<sup>3</sup> <https://www.ipums.org/>, Integrated Public Use Microdata Series

Documentation Initiative (Rasmussen & Blank, 2007). My focus was on making this functionality available as a bottom-up tool that is suitable for nonspecialists, uses a modern metadata format, is open source and freely available (both in the sense of cost and in not being tied to one platform), and can be improved by its users.

Going beyond search engines, projects like WikiData (“Introduction to Wikidata,” n.d.) plan to link structured data from sources across the web (Google, 2018; Noy & Brickley, 2017). If data were sufficiently complete and structured, Wikipedia could, for instance, automatically display information about the sex differences in various traits in their respective entries, or synthesize the meta-analytic heritability of traits studied using genetic methods. A Wikipedia for traits that arranges items and traits into ontologies (or nomological networks) by collecting bivariate correlations could simply emerge for free instead of being painstakingly assembled, as in the metaBUS and SAPA projects (Bosco, Steel, Oswald, Uggerslev, & Field, 2015; Condon, 2018). Structured data would also enrich existing data for researchers, by tying locations recorded in psychological data to geographic or administrative metadata, or by tying the time of study participation to current events and news. There are many ways in which existing data could be re-used for purposes not imagined by those who released the data. Findability and accessibility of datasets are crucial for this. Meta-analysis would also become much easier. Especially unpublished effect sizes could be discovered more easily by finding studies and datasets that measured all relevant constructs in the relevant way. This approach compares favourably to sending requests out through mailing lists and browsing conference abstracts for clues. Including more unpublished and unpredicted effects could reduce selection bias in estimated effect sizes (Bosco, Aguinis, Field, Pierce, & Dalton, 2016). Even published work can be hard to find using keyword searches and requires a lot of human filtering. Currently, ontologies and meta-analysis databases can be built by determined teams, but have no

economy of scale and little potential for automation because every step of the way requires the efforts of qualified researchers and research assistants.

## Alternatives

Several other ways to create codebooks exist. The closest relative to the codebook package is the `dataspice` R package (Boettiger et al., 2018), which also generates machine-readable metadata but does not provide an overview of distributions, reliabilities, and missing data for users; nor does it allow users to easily re-use existing metadata in SPSS and Stata files (i.e., labels for variables, values, and missing values). However, it excels at helping inexperienced R users interactively enter metadata, if it does not yet exist in a structured form. Conceivably, the codebook and `dataspice` packages could be integrated in the future by working with the same metadata substrate—the `dataspice` R package is currently only available on Github. The `dataMaid` R package (Helby Petersen & Thorn Ekstrøm, 2018) also offers a codebook function. It generates a similar, but PDF-focused, overview document, but neither computes reliabilities nor generates machine-readable metadata. The `summarytools` R package (Comtois, 2018) generates similar overviews as `dataMaid`, in HTML, with similar limitations. These packages focus on helping users find errors in their own data and do not prioritise sharing metadata. The Dataset Nutrition Label (Holland, Hosny, Newman, Joseph, & Chmielinski, 2018) project generates similar high-level dataset overviews for machine learning, but does not yet offer a public-facing product; furthermore, it does not generate machine-readable metadata. There are also several online interfaces in which one can enter metadata. The ZPID DataWiz platform<sup>4</sup> (Kerwer, Bölter, Dehnhard, Günther, & Weichselgartner, 2017) helps users generate dataset documentation that complies with funder statutes, but lacks

---

<sup>4</sup> <https://datawiz.leibniz-psychology.org>



a dedicated outlet for sharing machine-readable metadata that can be indexed by search engines and instead focuses on administrative information that is not particularly interesting for other researchers, such as funders and the data management plan. I have also tried describing the same dataset on various well-known commercial, nonprofit, and governmental platforms (including the OSF, Figshare, DataDryad, ReShare, ICPSR, Dataverse, PsychData, and Zenodo). None makes use of metadata that is already stored in a datafile. Some, like the OSF, do not use metadata related to the dataset contents at all. With others, like ReShare, I had to give up after the sign up process led to errors. Yet others, like the Dataverse, allow for metadata, but they have to be entered in a cumbersome online interface, even if they already exist in a structured format, and the researcher does not derive any personal benefit from entering the metadata. These platforms are thus better suited to storing data; the rich description of the data for other humans and search engines might be housed more comfortably elsewhere. The most mature platform for social science research appears to be openICPSR.<sup>5</sup> In sum, the codebook solutions I found, with the exception of dataspace, do not provide rich metadata for humans and machines, can make heavy demands on researchers' time, and give little in return.

	codebook	dataspace	summarytools	Dataset Nutrition Label	dataMaid	dataWiz	Other online providers
Machine-readable metadata	for dataset and variable labels	for dataset and variable labels	no	not yet	no	only citation related	usually only citation related
Distribution plots of the data	yes	no	yes	yes	yes	no	varies
Reliability	yes	no	no	no	no	no	no

---

<sup>5</sup> <https://www.openicpsr.org>

computation							
Missingness patterns	yes	no	no	yes	no	no	no
Web interface	yes	yes (Shiny)	no	no	planned	web-only	web-only
Interactively enter metadata	no	yes (locally)	no	no	no	yes (online)	yes (online)
Storage independent	yes	yes	yes	yes	yes	yes	usually tied to upload of data
Metadata available locally during analysis	yes	no	yes	no	no	no	no

**Table 1.** Feature comparison of various codebook solutions.

---

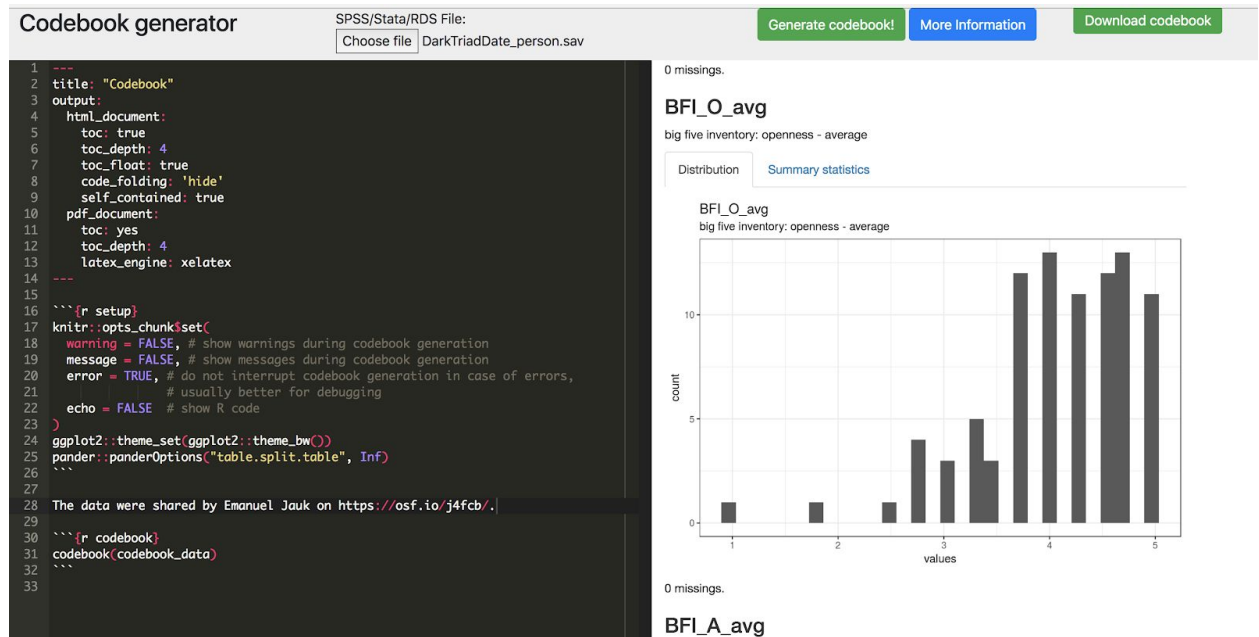
### **Box 1. Generate a codebook in three clicks**

The codebook web app (<https://codebook.formr.org>) is an easy entry point to generating codebooks using SPSS or Stata files on your computer that already have properly labelled variables and values—especially for researchers who do not use R. It allows you to turn these files into codebooks in an open (i.e., non-proprietary) format that you can easily share. The codebook web app is a simple website<sup>6</sup> (see Figure 1) that shows an editable R Markdown (Allaire et al., 2018) document on the left and what initially appears as an empty space on the right. You can ignore this document for the moment; you will learn how to edit it in the remainder of this tutorial. To generate a codebook, visit <https://codebook.formr.org> and choose a file from the bar at the top. This can be in a file format such as .rds (R data serialization), .sav (SPSS), .dta (Stata), .xlsx (Excel format), .csv (comma-separated values), or one of many any other

---

<sup>6</sup> This website was constructed using OpenCPU (Ooms, 2014), a way of using R on the web that is similar but not identical to Shiny (Chang, Cheng, Allaire, Xie, & McPherson, 2018).

standard file formats for data.<sup>7</sup> Then simply click the green button labelled "Generate codebook." Depending on the size of your dataset, you may have to wait a few minutes. When it is ready, the codebook will appear on the right. You can browse the codebook there or download it for later using the second green button.



**Figure 1.** A screenshot of the codebook web app after a codebook has been generated.

The web app sets reasonable defaults and it is possible to edit the text and the R code to improve the resulting codebook. However, the web app does not store edits, is less interactive than working in R, and requires the user to temporarily upload the dataset to a server. This is not permissible for certain restricted-use datasets. Moreover, very large datasets may result in an error message due to individual resource limits imposed by the server. To document large, private, or many datasets, or if you first need to add the metadata in R, I recommend installing the codebook package locally.

---

<sup>7</sup> All of the formats that can be read by the R package rio; read its documentation for more information (Chan & Leeper, 2018).

---

# Using the codebook package Locally in RStudio

## One-time Preparatory Work and Assumed Knowledge

RStudio is an integrated development environment for R. The codebook package can make use of some of RStudio's features (such as the viewer tab, R Markdown editing, and addins), but it works independently of it. I strongly recommend using RStudio with the codebook package in order to prevent problems resulting from misspecified paths and to simplify the transition to self-contained reproducible project management, and will assume you are using RStudio for the remainder of the Tutorial. You can generate a codebook without previous experience with R and RStudio with the help of this Tutorial. For an introduction to installing R, RStudio, and first steps in R, please consult the excellent Tutorial for the package *apaTables* (Stanley & Spence, 2018), which also recommends working with RStudio projects.

## Installing R Markdown

The *codebook* package makes use of the *rmarkdown* package (Allaire et al., 2018). R Markdown is a form of living document that lets you intermesh text, graphics, and code in a fully reproducible manner. It is a simple plain text document that can be "knit" into rich HTML, PDF, or Word documents. To install R Markdown, simply click File -> New File -> R Markdown in the RStudio menu. RStudio will prompt you to install the necessary packages at this point. The resulting document serves as a succinct introduction to R Markdown. As you can see,

explanations in plain text surround code blocks introduced by ````{r}` and ending with `````, each on a new line. Click the "Knit" button at the top of the document and select a file name. Within seconds, the document will be turned into an HTML document with formatting and graphics, shown in the bottom-right viewer panel.

## Installing codebook

Once R and R Markdown are installed, run the following command in the RStudio console to install the *codebook* package. It will automatically install the *codebook* package as well as several other R packages, some of which will be used in this Tutorial to prepare data.

```
install.packages("codebook")
```

Another way to install the packages is by clicking the "Install" button in the bottom right "Packages" tab in RStudio, then typing in "*codebook*" and hitting "Enter."

## Creating Your Codebook

Now you need an R Markdown file to serve as the basis for your codebook. Because codebooks look best with a few defaults already set, you will load the template shown in Box 1 by executing the following command in the RStudio console:

```
codebook::new_codebook_rmd()
```

The file that just opened is the template you will be working with. For now, it is just an empty template without data. Try clicking on "Knit" at the top of the document. In the RStudio viewer pane on the bottom right, a codebook for a mock dataset included with the package will appear.<sup>8</sup> The codebook and its table of contents may look a little squished depending on the

---

<sup>8</sup> The generated document is named "codebook.html". You can open this file in your project directory anytime to view the codebook or share it with others.

size of your screen. You can expand it to a full browser window by clicking the little window with an arrow in the viewer tab.

## Loading Data

It is time to load some data. In this Tutorial, I will walk you through the process by using the "bfi" dataset made available in the psych package (Goldberg, 1999; Revelle et al., 2016; Revelle, Wilt, & Rosenthal, 2010). The bfi dataset is already very well documented in the psych R package, but using the codebook package, we can add automatically computed reliabilities, graphs, and machine-readable metadata to the mix. The dataset is available within R, but this will not usually be the case; I have therefore uploaded it to the OSF, which also features many other publicly available datasets. A new package in R, rio (Chan & Leeper, 2018), makes loading data from websites in almost any format as easy as loading local data. You can import the dataset directly from the OSF<sup>9</sup> by replacing the line

```
codebook_data <- codebook::bfi  
  
with  
  
codebook_data <- rio::import("https://osf.io/s87kd/download",  
"csv")
```

on line 34. R Markdown documents have to be reproducible and self-contained, so it is not enough for a dataset to be loaded locally; you must load the dataset at the beginning of the document. You can also use the document interactively, although this will not work seamlessly

---

<sup>9</sup> Loading local data is just as easy; remember to put the dataset you want to use in the same directory as the codebook.rmd file in order to avoid having to think about paths. One way of doing this is to go to the OSF at <https://osf.io/s87kd/>, download the csv file, and put it in the same folder, then type `codebook_data <- rio::import("bfi.csv")` on line 33. The package will automatically select how to import the file based on the file extension, and almost all common standard file extensions for tabular data are supported, including SPSS and Stata.

for the *codebook* package<sup>10</sup>. To see how this works, execute the line you just added by pressing Command + Enter (for a Mac) or Ctrl + Enter (for other platforms).

RStudio has a convenient data viewer you can use to check whether your command worked. In the environment tab on the top right, you should see "codebook\_data". Click that row to open a spreadsheet view of the dataset in RStudio. As you can see, it is not particularly informative. Just long columns of numbers with variable names like A4. Are we talking aggressiveness, agreeableness, or the German industrial norm for paper size? The lack of useful metadata is palpable. Click "Knit" again to see what the *codebook* package can do with this. This time, it will take longer. Once the result is shown in the viewer tab, scroll through it. You can see a few warnings stating that the package saw items that might form part of a scale, but there was no aggregated scale. You will also see graphs of the distribution for each item and summary statistics.

## Adding and Changing Metadata

### Variable labels

The last codebook you generated could already be useful if the variables had meaningful names and self-explanatory values. Unfortunately, this is rarely the case. Generally, you will need more metadata: labels for variables and values, a dataset description, and so on. The *codebook* package can use metadata that are stored in R attributes. Attributes in R are most commonly used to store the type of a variable; for instance, datetime in R is just a number with two attributes (a time zone and a class). However, they can just as easily store other metadata; the Hmisc (Harrell, 2018), haven (Wickham & Miller, 2018), and rio (Chan & Leeper, 2018)

---

<sup>10</sup> The output generated by the *codebook* package does not fit inside the interactive results box that RStudio uses.

packages, for example, use attributes to store labels. The benefit of storing variable metadata in attributes is that even datasets that are the product of merging and processing raw data retain the necessary metadata. The haven and rio packages set these attributes when importing data from SPSS or Stata files. However, it is also easy to add metadata yourself:

```
attributes(codebook_data$C5)$label <- "Waste my time."
```

You have just assigned a new label to a variable. Because it is inconvenient to do this over and over again, the labelled package (Larmarange, 2018) adds a few convenience functions. Load the labelled package by writing the following in your codebook.Rmd

```
library(labelled)
```

Now label the C5 item.

```
var_label(codebook_data$C5) <- "Waste my time."
```

You can also label values in this manner (label in quotes before the equal sign, value after):

```
val_labels(codebook_data$C1) <- c("Very Inaccurate" = 1, "Very  
Accurate" = 6)
```

Write these labelling commands after loading the dataset and click "Knit" again. As you can see in the viewer pane, the graph for the C1 variable now has a label at the top and the lowest and highest values on the X axis are labelled.<sup>11</sup> If the prospect of adding labels for every single variable seems tedious, do not fear. Many researchers already have a codebook in the form of a spreadsheet that they want to import in order to avoid entering labels one-by-one. The bfi dataset in the psych package is a good example of this, because it comes with a tabular dictionary. On the line after loading the bfi data, type the following to import this data dictionary:

```
dict <- rio::import("https://osf.io/cs678/download", "csv")
```

---

<sup>11</sup> Further information about adding labels with the labelled package can be found in their vignette. [https://cran.r-project.org/web/packages/labelled/vignettes/intro\\_labelled.html](https://cran.r-project.org/web/packages/labelled/vignettes/intro_labelled.html)



To see what you just loaded, click the "dict" row in the environment tab in the top right panel. As you can see, the dictionary has information on the constructs on which this item loads and on the direction with which it should load on the construct. You can make these metadata usable through the *codebook* package. You will often need to work on the data frames to help you do this; to make this easier, use the *dplyr* package (Wickham, François, Henry, & Müller, 2018). Load it by typing the following

```
library(dplyr)
```

Your next goal is to use the variable labels that are already in the dictionary. Because you want to label many variables at once, you need a list of variable labels. Instead of assigning one label to one variable as you just did, you can assign many labels to the whole dataset from a named list. Here, each element of the list is one item that you want to label.

```
var_label(codebook_data) <- list(  
  C5 = "Waste my time.",  
  C1 = "Am exacting in my work."  
)
```

There are already a list of variables and labels in your data dictionary that you can use, so you do not have to perform the tedious task of writing out the list. You do have to reshape it slightly though, because it is currently in the form of a rectangular data frame, not a named list. To do so, use a convenience function from the *codebook* function called `dict_to_list`. This function expects to receive a data frame with two columns: the first should be the variable names, the second the variable labels. To select these columns, use the `select` function from the *dplyr* package. You will also need to use a special operator, called a pipe, which looks like this `%>%` and allows you to read and write R code from left to right, almost like an English sentence. First, you need to take the `dict` dataset, then select the variable and label columns,

then use the `dict_to_list` function. You also need to assign the result of this operation to become the variable labels of `codebook_data`. You can do all this in a single line using pipes.

Add the following line after importing the dictionary.

```
var_label(codebook_data) <- dict %>% select(variable, label) %>%  
dict_to_list()
```

Click “codebook\_data” in the Environment tab again. You should now see the variable labels below the variable names. If you click “Knit” again, you will see that your codebook now contains the variable labels. They are both part of the plots and part of the codebook table at the end. They are also part of the metadata that can be found using, for example, Google Dataset Search, but this will not be visible to you.

## Value labels

So far, so good. But you may have noticed that education is shown as a number. Does this indicate years of education? The average is 3, so that seems unlikely. In fact, these numbers signify levels of education. In the `dict` data frame, you can see that there are value labels for the levels of this variable. However, these levels of education are abbreviated, and you can probably imagine that it would be difficult for an automated program to understand how these map to the values in your dataset. You can do better, using another function from the `labelled` package: not `var_label` this time, but `val_labels`. Unlike `var_label`, `val_labels` expects not just one label, but a named vector<sup>12</sup>, with a name for each value that you want to label. You do not need to label all values. Named vectors are created using the `c()` function. Add the following lines right after the last one.

```
val_labels(codebook_data$gender) <- c("male" = 1, "female" = 2)
```

---

<sup>12</sup> Vectors in R refer to variables that contain one or many elements of one type (e.g., numbers, or text). All variables in normal data frames are vectors.

```
val_labels(codebook_data$education) <- c("in high school" = 1,
    "finished high school" = 2,
    "some college" = 3,
    "college graduate" = 4,
    "graduate degree" = 5)
```

Click the “Knit” button. The bars in the graphs for education and gender should now be labelled.

Now, on to the many Likert items, which all have the same value labels. You could assign them in the same way you did for gender and education, entering the lines for each variable over and over, or you could let a function do the job for you instead. Creating a function is actually very simple. Just pick a name, ideally one to remember it by—I chose `add_likert_labels`—and assign the keyword function followed by two different kinds of brackets. Round brackets surround the variable `x`. The `x` here is a placeholder for the many variables you will use this function for in the next step. Curly braces show that you intend to write out what you plan to do with the variable `x`. Inside the curly braces, use the `val_labels` function from above and assign a named vector.

```
add_likert_labels <- function(x) {
  val_labels(x) <- c("Very Inaccurate" = 1,
    "Moderately Inaccurate" = 2,
    "Slightly Inaccurate" = 3,
    "Slightly Accurate" = 4,
    "Moderately Accurate" = 5,
    "Very Accurate" = 6)

  x
}
```

A function is just a tool and does nothing on its own; you have not used it yet. To use it only on the Likert items, you need a list of them. An easy way to achieve this is to subset the dict dataframe to only take those variables that are part of the Big Six. To do so, use the `filter` and `pull` functions from the `dplyr` package.

```
likert_items <- dict %>% filter(Big6 != "") %>% pull(variable)
```

To apply your new function to these items, use another function from the `dplyr` package called `mutate_at`. It expects a list of variables and a function to apply to each. You have both! You can now add value labels to all Likert items in the `codebook_data`.

```
codebook_data <- codebook_data %>% mutate_at(likert_items,  
add_likert_labels)
```

Click “Knit” again. All items should now have value labels. However, this display is quite repetitive. How about grouping the items by the factor that they are supposed to load on? And while you are at it, how can the metadata about keying (or reverse-coded items) in your dictionary become part of the dataset?

## Adding scales

The *codebook* package relies on a simple convention to be able to summarise psychological scales, such as the Big Five dimension extraversion, which are aggregates across several items. Your next step will be to assign a new variable, `extraversion`, to the result of selecting all extraversion items in the data and passing them to the `aggregate_and_document_scale` function. This function takes the mean of its inputs and assigns a label to the result, so that you can still tell which variables it is an aggregate of.

```
codebook_data$extraversion <- codebook_data %>% select(E1:E5) %>%  
aggregate_and_document_scale()
```

Try knitting now. In the resulting codebook, the items for extraversion have been grouped in one graph. In addition, several internal consistency coefficients have been calculated. However, they are oddly low. You need to reverse items which negatively load on the extraversion factor, such as "Don't talk a lot."

To do so, I suggest following a simple convention early on, when you come up with names for your items—namely the format `scale_numberR` (e.g., `bfi_extra_1R` for a reverse-coded extraversion item, `bfi_neuro_2` for a neuroticism item). That way, the analyst always knows how an item relates to a scale. This information is encoded in the data dictionary from the data you just imported. Rename the reverse-coded items so that you cannot forget about its direction. First, you need to grab all items with a negative keying from your dictionary. Add the following three lines above the `aggregate_and_document_scale()` line from above.

```
reversed_items <- dict %>% filter(Keying == -1) %>%  
pull(variable)
```

You can see in your Environment tab that names such as A1, C4, and C5 are now stored in the `reversed_items` vector. You can now refer to this vector using the `rename_at` function, which applies a function to all variables you list. Use the very simple function `add_R`, which does exactly what its name indicates.

```
codebook_data <- codebook_data %>%  
  rename_at(reversed_items, add_R)
```

Click “codebook\_data” in the Environment tab and you will see that some variables have been renamed: A1R, C4R, and C5R, and so on. This could lead to an ambiguity: Does the suffix R means "should be reversed before aggregation" or "has already been reversed"? With the help of metadata in the form of labelled values, there is no potential for confusion. You can reverse the underlying values, but keep the value labels right. So, if somebody responded "Very

accurate," that remains the case, but the underlying value will switch from 6 to 1 for a reversed item. The data you generally import will rarely include labels that remain correct regardless of whether underlying values are reversed, but the *codebook* package makes it easy to bring the data into this shape. A command using dplyr functions and the *reverse\_labelled\_values* function can easily remedy this.

```
codebook_data <- codebook_data %>%  
  mutate_at(vars(matches("\\dR$")), reverse_labelled_values)
```

All this statement does is find variable names which end with a number (*\d* is the regular expression codeword for a number; a dollar sign denotes the end of the string) and R and reverse them.<sup>13</sup> Because the extraversion items have been renamed, we have to amend our scale aggregation line slightly.

```
codebook_data$extraversion <- codebook_data %>% select(E1R:E5)  
%>% aggregate_and_document_scale()
```

Try knitting again. The reliability for the extraversion scale should be much higher and all items should load positively. Adding further scales is easy: Just repeat the above line, changing the names of the scale and the items. Adding scales that integrate smaller scales is also straightforward. The data dictionary mentions the Giant Three—try adding one, Plasticity, which subsumes Extraversion and Openness.

```
codebook_data$plasticity <- codebook_data %>% select(E1R:E5,  
01:05R) %>% aggregate_and_document_scale()
```

Note that writing *E1R:E5* only works if the items are all in order in your dataset. If you mixed items across constructs, you will need a different way to select them. One option is to list

---

<sup>13</sup> This function can only work automatically if the highest and lowest possible values are both encoded in the labels or levels attribute of the variable—otherwise *codebook* cannot infer the possible range of the values.

all items, writing `select(E1R, E2R, E3, E4, E5)`. This can get tedious when listing many items. Another solution is to write `select(starts_with("E"))`. Although this is quite elegant, it will not work in this case because you have more than one label that starts with E; this command would include education items along with the extraversion items you want. This is a good reason to give items descriptive stems such as `extraversion_` or `bfik_extra`. Longer stems not only make confusion less likely, they also make it possible for you to refer to groups of items by their stems, and ideally to their aggregates by only the stem. If you have already named your item too minimally, another solution is to use a regular expression, as I introduced above for matching reversed items. In this scenario, `select(matches("^E\\dR?$"))` would work.<sup>14</sup>

## Metadata About the Entire Dataset

Finally, you might want to sign your work and add a few descriptive words about the entire dataset. If you simply edit the R Markdown document to add a description, this information would not become part of the machine-readable metadata. Metadata (or attributes) of the dataset as a whole are a lot less persistent than metadata about variables. Hence, you should add your description right before calling the `codebook` function. Adding metadata about the dataset is very simple: Just wrap the `metadata` function around `codebook_data` and assign a value to a field. The fields "name" and "description" are required. If you do not edit them, they will be automatically generated based on the data frame name and its contents. To overwrite them, enter the following lines above the call `codebook(codebook_data)`:

---

<sup>14</sup> This means: Match only variables where the name starts with (^) the letter E, is followed by one digit (\\d), optionally the letter R (R?), and then ends (\$).

```
metadata(codebook_data)$name <- "25 Personality items  
representing 5 factors"  
  
metadata(codebook_data)$description <- "25 personality self  
report items taken from the International Personality Item Pool  
(ipip.ori.org)[...]"
```

It is good practice to give datasets a canonical identifier. This way, if a dataset is described in multiple locations, it can still be identified as the same dataset. For instance, when I did this I did not want to use the URL of the R package from which I took the package because URLs can change; instead, I generated a persistent document object identifier (DOI) on the OSF and specified it here.

```
metadata(codebook_data)$identifier <-  
"https://dx.doi.org/10.17605/OSF.IO/K39BG"
```

In order to let others know who they can contact about the dataset, how to cite it, and where to find more information, I set the attributes creator, citation, and URL below.

```
metadata(codebook_data)$creator <- "William Revelle"  
  
metadata(codebook_data)$citation <- "Revelle, W., Wilt, J., and  
Rosenthal, A. (2010) Individual Differences in Cognition: New Methods  
for examining the Personality-Cognition Link In Gruszka, A. and  
Matthews, G. and Szymura, B. (Eds.) Handbook of Individual Differences  
in Cognition: Attention, Memory and Executive Control, Springer."  
  
metadata(codebook_data)$url <-  
"https://CRAN.R-project.org/package=psych"
```



Lastly, it is useful to note when and where the data was collected, as well as when it was published. Ideally, you would make more specific information available here, but this is all I know about the BFI dataset.

```
metadata(codebook_data)$datePublished <- "2010-01-01"  
metadata(codebook_data)$temporalCoverage <- "Spring 2010"  
metadata(codebook_data)$spatialCoverage <- "Online"
```

These attributes are documented in more depth on <https://schema.org/Dataset>. You can also add attributes that are not documented there, but they will not become part of the machine-readable metadata. Click “Knit” again. In the viewer tab, you can see that the metadata section of the codebook has been populated with your additions.

## Exporting and Sharing the Data with Metadata

Having added all the variable-level metadata, you might want to re-use the marked-up data elsewhere or share it with collaborators or the public. You can most easily export it using the `rio` package (Chan & Leeper, 2018), which permits embedding the variable metadata in the dataset file for those formats that support it. The only way to keep all metadata in one file is by staying in R:

```
rio::export(codebook_data, "bfi.rds") # to R data structure file
```

The variable-level metadata can also be transferred to SPSS and Stata files. Please note that this export is based on reverse-engineering the SPSS and Stata file structure, so the resulting files should be tested before sharing.

```
rio::export(codebook_data, "bfi.sav") # to SPSS file  
rio::export(codebook_data, "bfi.dta") # to Stata file
```

## Releasing the Codebook Publicly

If you want to share your codebook with others, there is a `codebook.html` file in the project folder you created at the start. You can email it to collaborators or upload it to the OSF file storage. However, if you want Google Dataset Search to index your dataset, this is not sufficient. The OSF will not render your HTML files for security reasons and Google will not index the content of your emails (at least not publicly). You need to post your codebook online. If you are familiar with Github<sup>15</sup> or already have your own website, uploading the html file to your own website should be easy. The simplest way I found for publishing the HTML for the codebook is as follows. First, rename the `codebook.html` to `index.html`. Then create an account on netlify.com. Once you're signed in, drag and drop the folder containing the codebook to the Netlify web page (make sure the folder does not contain anything you do not want to share, such as raw data). Netlify will upload the files and create a random URL like `estranged-armadillo.netlify.com`. You can change this to something more meaningful, like `bfi-study.netlify.com`, in the settings. Next, visit the URL to check that you can see the codebook. The last step is to publicly share a link to the codebook so that search engines can discover it; for instance, you could tweet the link with the hashtag `#codebook`<sup>16</sup>—ideally with a link from the repository where you are sharing the raw data or the related study's supplementary material. For instance, I added a link to the bfi-study codebook on the OSF (<https://osf.io/k39bg/>), where I had also shared the data. Depending on the speed of the search engine crawler, the dataset should be findable on Google Dataset Search in anywhere from three to 21 days.

---

<sup>15</sup> If you want to learn about Github Pages, there are several guides available, for example, <http://www.the100.ci/2017/02/19/reproducible-websites-for-fun-and-profit/>.

<sup>16</sup> I would happily share the first 10 codebooks published this way and give feedback on them.

# Core Features of the Generated Codebook

You have just created a public good, but there are also personal benefits to generating codebooks this way. Codebooks are not just useful for other researchers, but also for the majority of us who struggle to keep all details about our datasets in mind at all times. Properly annotated data can help us complete rote tasks faster and help us make fewer errors. Codebooks will usually be easier and more accurate right after data collection, when the study is still fresh in mind. I hope the increased convenience of having a codebook to hand during analysis might motivate you to create it early on. The process of generating a dataset that can be automatically turned into a codebook should lead to more meaningful variable names and labels, and re-usable datasets. Currently, special variables that carry meaning reflect the conventions used in the survey software formr.org (Arslan, Walther, & Tata, 2018), so users of formr.org have it easy, but columns from other survey providers can be renamed accordingly if they carry the same meaning.<sup>17</sup> Pending agreement on name conventions for psychological variables, the *codebook* package will follow these conventions instead. To give you a useful codebook, the package draws on functionality supplied by many other R packages. I cite them below to give their authors credit, but you do not need to learn about all of them – part of the idea behind providing metadata at the outset is that automated data summaries can be more meaningful and require less additional user input. For example, your variable labels will be re-used in plots and model summaries generated using the *sjPlot* package (Lüdtke, 2018). However, nobody should be put off generating machine-readable codebooks because one of

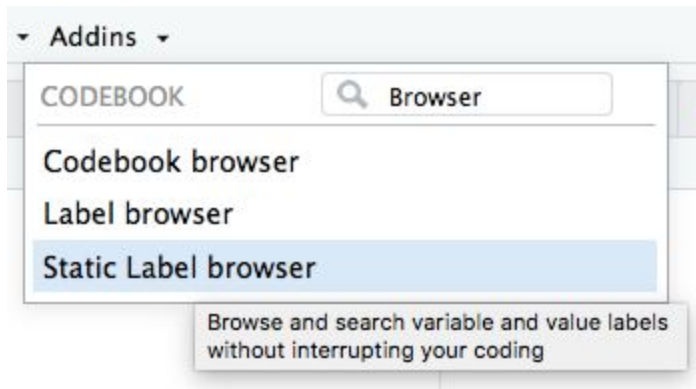
---

<sup>17</sup> An example for Qualtrics surveys can be found in the package documentation.

these automated summaries does not look right. Therefore, all components mentioned below can be turned off via arguments of the main codebook function.

## Checking the Codebook During Analysis

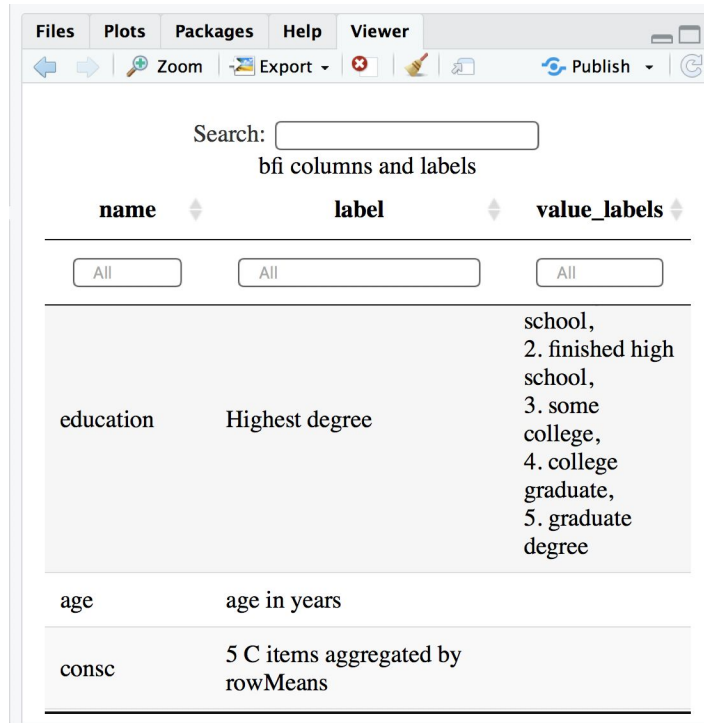
Often, during data analysis, we want to confirm that a variable is the one we intend to use or that values are in the correct order, or we need to find a variable and forgot what we called it. To solve these and similar problems, go to the Addins menu in the top bar in the RStudio window and choose "Static Label browser" (Figure 2). The static label browser shows the variable and value labels for a dataset in the bottom right viewer pane of RStudio. It selects the dataset that is alphabetically first in the environment, or the dataset with the name of any text that is selected in the editor. You can also pick a data frame by typing `label_browser_static(data_frame_name)` in the console.



**Figure 2.** Opening an add-in.

In this case, only the `codebook_data` dataset and the `dict` dataset are loaded, so it will choose `codebook_data`. In the viewer tab, you can now see the variable and value labels (Figure 3). When the static label browser is open, you can keep working on and executing your R code. The codebook browser and the dynamic label browser have the advantage of allowing

you to select a dataset conveniently via a dropdown instead of via text selection, but because they are implemented as Shiny apps (Chang, Cheng, Allaire, Xie, & McPherson, 2018), code can only be executed after they have been stopped (using the red stop button).



name	label	value_labels
education	Highest degree	school, 2. finished high school, 3. some college, 4. college graduate, 5. graduate degree
age	age in years	
consc	5 C items aggregated by rowMeans	

**Figure 3.** The variable and value labels as seen in the RStudio viewer tab.

## Automatically Making Sense of Metadata When You Have Preprocessed the Data File Elsewhere

### Dealing with miscoded missing values

Sometimes not all the missing values in a dataset imported from SPSS or Stata will be set correctly. SPSS users in particular often code missing values as 99 or 999, but fail to

actually label these as missing value placeholders in SPSS. For this, the web app and the default codebook example run the function `detect_missing`. When its argument *ninety\_nine\_problems* is set to true, variables containing the value 99 or 999 (where that value is not in the plausible range of the other values for that variable) will be set as missing values. When the argument *only\_labelled* is set to true, this will only happen if the value 99 or 999 has a label. A similar option is available for negative values, a common convention used by Stata users. This function will not do anything for the BFI dataset because it has no labelled missings, but calling it by default should be harmless.

## Detecting scales that have been aggregated outside of R

You may prefer to aggregate your items outside of R, or you may have done so already for existing data without using the special `aggregate_and_document_scale` function in the *codebook* package. In this case, the function `detect_scales` is helpful. If your dataset contains a scale named, for instance, `bfi_extra`, and its items, `bfi_extra_1`, `bfi_extra_2R`, and `bfi_extra_3`, calling this function on the entire dataset will link items and scales. This link is a precondition for the Likert plots and reliability computations to work. The function is called by default when uploading data into the web app (Box 1) or when using the default codebook template. It will also warn you if it finds what looks like numbered items, but no aggregate, or when a new aggregate of the items is not perfectly correlated with the aggregate you already have (often indicating a missing item or ad hoc reverse-coded items).

## Survey Response Rates, Durations

If a data frame has the column *session* to identify participants, and the columns *created*, *modified*, *ended*, and *expired* in the form of datetimes, the *codebook* package can calculate a

few commonly desired summaries about participation in a survey. It can give the number of participants and the number of rows per participant. It can show the dates and times people enrolled, and, by subtracting *created* from *ended*, how long it took participants to fill out the survey. By checking whether *modified* is missing, it can differentiate people who filled out information in the survey from those who did not. By checking whether *expired* is missing, it can determine how many participants did not finish the survey in time. The resulting values will be summarised in the beginning of the codebook, if the necessary variables exist. See the manual for an example.<sup>18</sup>

## Reliability Estimates and Likert Plots

The *codebook* package automatically calculates an estimate of reliability for all defined scales. By default, this will be done using the internal consistency indices computed by the *scaleDiagnosis* function in the *userfriendlyscience* package (Crutzen, 2007; Crutzen & Peters, 2017; Peters, 2014) if there is just one row per participant. If there are either one or two rows per participant (*session* column), the package will calculate internal consistencies for each time point and a retest correlation between time points (data have to be sorted by time), again using the *userfriendlyscience* package. If there is a variable number of rows greater than two per participant, the *multilevel.reliability* function in the *psych* package (Revelle, 2018) reports the generalizability of changes, of the person average, and more (Shrout & Lane, 2012). For ordinal variables, the scale summary also includes a Likert plot (Figure 4) generated using the *likert* package (Bryer & Speerschneider, 2016). A boon of defining metadata upfront is that you do not have to get your data into the shape expected by these various functions, the *codebook* package can handle this for you, now that it understands how scales and items relate

---

<sup>18</sup> <https://rubenarslan.github.io/codebook/articles/codebook.html>

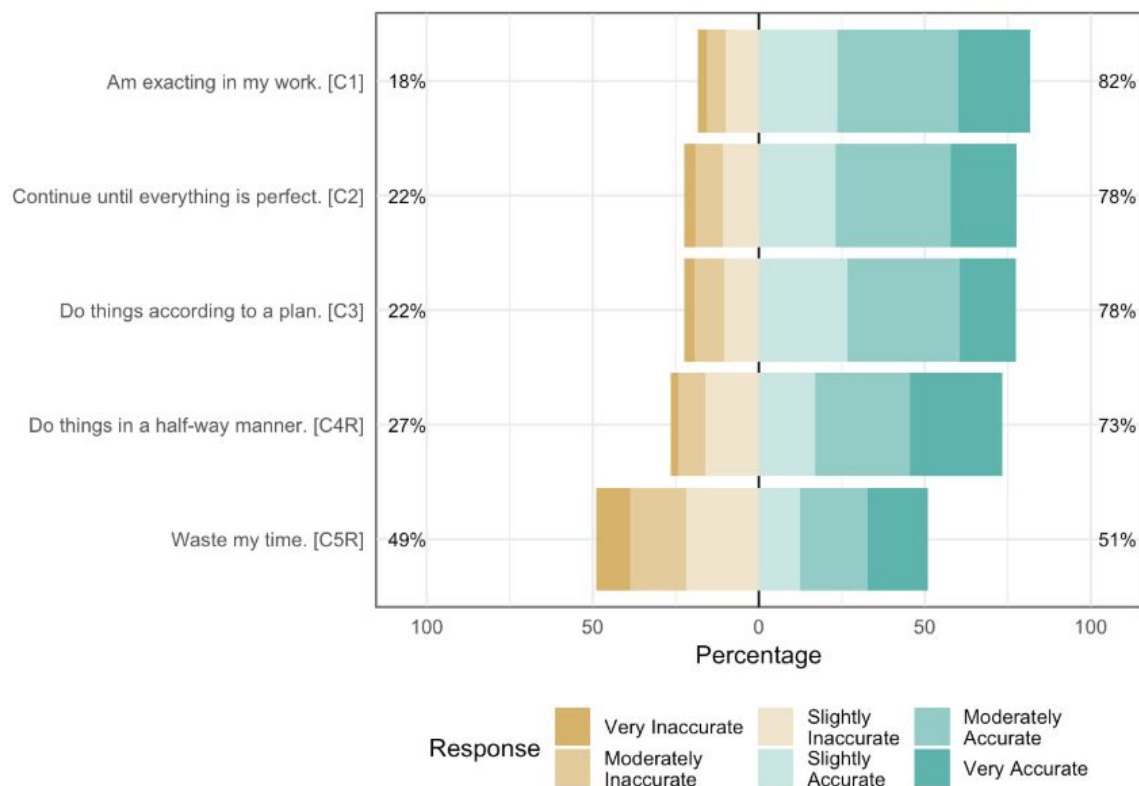
to each other. Given the right metadata, the package could also be extended to automatically compute measures of precision suitable for reaction times or other psychological data.

### Scale: conscientiousness

#### Overview

**Reliability:**  $\omega_{\text{ordinal}}$  [95% CI] = 0.77 [0.76;0.78].

**Missing:** 93.



**Figure 4.** A Likert plot for the conscientiousness scale in the bfi dataset in the psych package.

## Distribution Plots and Descriptives

The *codebook* package also shows a plot of the distribution for all individual items and all scales, except for large numbers of unique values (e.g., for free text responses it shows the



distribution of number of characters instead). These plots are labelled using the variable name, label, and value labels. They can be generated in isolation by calling the function `plot_labelled(codebook_data$E1R)` for a specific variable. Further, using the `skimr` package (McNamara, Arino de la Rubia, Zhu, Ellis, & Quinn, 2018), each item or scale is accompanied by a compact summary of the data, such as missing data, means, ranges, standard deviations for numeric data types, and top counts for categorical types, for dates, texts, and other data types. If there are labelled missing values (e.g., user did not do this part of the survey vs. user did not respond to this item), the summary includes the count of the types of missing values in a separate plot.

## Missingness Patterns

Although the number and types of missing values are always summarised for each item, this does not give a prospective analyst the answer to the question of how many data points have nonmissing data for a bivariate or multivariate analysis. The *codebook* package therefore displays a table of missingness patterns that shows the number of complete cases, cases with missing data in one variable, and frequently missing culprits, as well as whether there are common patterns of missingness (e.g., everything that depends on the same filter question). This information is mainly useful for re-usability. If other analysts need to request access through a cumbersome procedure such as NIH Dash, they might like reassurance that the combination of variables they are interested was actually measured often enough before immersing themselves in forms.

## Codebook Table

At the end of the codebook document is a rich tabular data dictionary (Figure 5). This searchable table is made possible through the DT package (Xie, 2018). It can be exported to Excel, CSV, and other formats from the browser. The variable names in the table are linked to the details in the HTML codebook. The table also includes variable and value labels, as well as the compact data summaries generated by skimr (McNamara et al., 2018). You can create this table directly without making the full codebook by calling the function `codebook_table`, but to share codebooks with others, I recommend also making the HTML document available, which contains the metadata in a search engine-friendly format.

Codebook table							
<div>Copy CSV Excel PDF Print</div>					Search: <input type="text"/>		
name	label	data_type	value_labels	scale_item_names	missing	complete	n
<input type="text"/>	All	A	All	All	<input type="text"/>	A	<input type="text"/>
A1R	Agreeableness: Am indifferent to the feelings of others.	numeric	6. Very Inaccurate, 5. Moderately Inaccurate, 4. Slightly Inaccurate, 3. Slightly Accurate, 2. Moderately Accurate, 1. Very Accurate		16	2784	2800

**Figure 5.** The first row of a codebook table.

## JSON-Linked Data (JSON-LD)

Unseen<sup>19</sup> by the researcher, the *codebook* package also writes JSON-LD into the HTML document. This type of data follows a flexible and extensible standard for metadata. Although it is presently limited to basic descriptive functionality, efforts are underway to extend it for use in biology and psychology. Currently, *codebook*'s implementation aims to make the fullest use of supplied information. If you publish a codebook generated using the package online, so that Google can index it, it will appear in the Google Dataset Search after some time (3–21 days). Unlike many other data platforms, the *codebook* package ensures that the dataset description contains all variable names and labels, thereby making it much easier to find relevant data. A heavily abbreviated JSON-LD example on the Big Five dataset you worked with above would look like this. It is basically a nested list of properties—some rather technical, but most with an obvious meaning. They are more fully documented at <https://schema.org/Dataset>, but most are irrelevant for normal users because the package creates them automatically.

```
{
  "@context": "http://schema.org/",
  "@type": "Dataset",
  "name": "25 Personality items representing 5 factors",
  "description": "25 personality self report items taken from the
International Personality Item Pool (ipip.ori.org)[...]",
  "identifier":
  "https://cran.r-project.org/web/packages/psych/index.html",
```

---

<sup>19</sup> By clicking on the small text "JSON-LD metadata" at the bottom of the generated codebook, you can see a copy of what search engines see too.

```

    "datePublished": "2010-01-01",
    "creator": {
      "@type": "Person",
      "givenName": "William",
      "familyName": "Revelle",
      "email": "revelle@northwestern.edu",
      "affiliation": {
        "@type": "Organization",
        "name": "Northwestern University"
      }
    },
    "citation": "Revelle, W., Wilt, J., and Rosenthal, A. (2010)
Individual Differences in Cognition: New Methods for examining the
Personality-Cognition Link In Gruszka, A. and Matthews, G. and
Szymura, B. (Eds.) Handbook of Individual Differences in Cognition:
Attention, Memory and Executive Control, Springer.",
    "url":
    "https://cran.r-project.org/web/packages/psych/index.html",
    "temporalCoverage": "Spring 2010",
    "spatialCoverage": "Online",
    "keywords": ["E1R", "E2R", "E3", "E4", "E5", "gender",
"education", "age", "extra"],
    "variableMeasured": [
      {

```

```

        "name": "E1R",
        "description": "Extraversion: Don't talk a lot.",
        "value": "6. Very Inaccurate,\n5. Moderately
Inaccurate,\n4. Slightly Inaccurate,\n3. Slightly Accurate,\n2.
Moderately Accurate,\n1. Very Accurate",
        "maxValue": 6,
        "minValue": 1,
        "@type": "propertyValue"
    },
    {
        "name": "gender",
        "description": "Self-reported gender",
        "value": "1. male,\n2. female",
        "maxValue": 2,
        "minValue": 1,
        "@type": "propertyValue"
    },
    {
        "name": "extraversion",
        "description": "5 extraversion items aggregated by
rowMeans",
        "@type": "propertyValue"
    }
]

```

}

In the future, psychologists could extend Schema.org to document certain psychological measurement scales, whether a variable is a self or informant report or a measured behaviour, types of datasets and research designs, and even single items such as demographic questions. Extending the schema is an open and community-driven process. Other research communities such as health and life sciences or biological sciences have started these schema extension processes on websites such as <https://health-lifesci.schema.org/> and <http://bioschemas.org/>. Discussions about extending schemas often take place on Github. The Society for the Improvement of Psychological Science has formed a workgroup for a psychological data specification.<sup>20</sup>

## Summary

Standardised, metadata-rich codebooks are useful to data creators, their teams, and the scientific community. The inconvenience and effort involved in creating such codebooks may have contributed to the current state of affairs in psychology, where codebooks, if any exist, are frequently unstandardised and lack information that is essential to understanding the data, and where datasets are not always available in open formats and are rarely machine-readable—and are therefore undiscoverable via web searches. In short, data is rarely easily findable, accessible, interoperable, and re-usable. The *codebook* package makes some common tasks easier: It speeds up the data cleaning and summary process, and makes data findable and accessible using tools like Google Dataset Search, independently of where the data is stored or whether it is even publicly available. Thanks to a public standard vocabulary, the metadata is

---

<sup>20</sup> Interested parties can find more information at <https://github.com/mekline/psych-DS>

interoperable. And by making sure codebooks are rich, descriptive, and interpretable by other researchers, data becomes more re-usable. The metadata are also portable – structured metadata can be imported and exported from and to many formats. A working codebook can be generated by inexperienced users within minutes. By following certain conventions or using specific survey providers, or by re-using metadata in closed-source formats such as SPSS files, researchers can save time, letting the *codebook* package take over graphing distributions, computation of descriptives, missingness patterns, and estimating reliabilities. It is my hope that the *codebook* package will encourage researchers to generate rich codebooks that benefit themselves and the scientific community as a whole.

## Author Contributions

RCA wrote software and manuscript.

## Conflicts of Interest

I have no conflicts of interest to report.

## Acknowledgements

I am grateful to Martin Brümmer for help setting up the proof-of-concept for JSON-LD and to early users like Christoph Schild, Caroline Zygar, Daniël Lakens, Matti Heino, and Mark Brandt, who reported bugs in earlier versions of the package. I also thank Bill Revelle who publicly released the BFI dataset used here and gave me permission to use it for this tutorial. I particularly thank Ioanna Iro Eleftheriadou, who tested the codebook package by generating a

gallery from several publicly available datasets on the Open Science Framework. I thank Deborah Ain for her scientific editing. All remaining errors are mine.

## References

- Allaire, J. J., Xie, Y., McPherson, J., Luraschi, J., Ushey, K., Atkins, A., ... Chang, W. (2018). *rmarkdown: dynamic documents for R*. Retrieved from <https://CRAN.R-project.org/package=rmarkdown>
- Arslan, R. C. (2018). *Cook codebooks from survey metadata encoded in attributes in R*. doi:10.5281/zenodo.1326520
- Arslan, R. C., Walther, M., & Tata, C. (2018). formr: A study framework allowing for automated feedback generation and complex longitudinal experience sampling studies using R. doi:10.31234/osf.io/pjasu
- Boettiger, C., Chamberlain, S., Fournier, A., Hondula, K., Krystalli, A., Mecum, B., ... Woo, K. (2018). *dataspice: Create lightweight schema.org descriptions of dataset*. ropenscilabs. Retrieved from <https://github.com/ropenscilabs/dataspice>
- Borghi, J. A., & Van Gulick, A. E. (2018). Data management and sharing in neuroimaging: Practices and perceptions of MRI researchers. *PloS One*, 13(7), e0200562. doi:10.1371/journal.pone.0200562
- Bosco, F. A., Aguinis, H., Field, J. G., Pierce, C. A., & Dalton, D. R. (2016). HARKing's Threat to Organizational Research: Evidence From Primary and Meta-Analytic Sources. *Personnel Psychology*, 69(3), 709–750. doi:10.1111/peps.12111
- Bosco, F. A., Steel, P., Oswald, F. L., Uggerslev, K., & Field, J. G. (2015). Cloud-based



- meta-analysis to bridge science and practice: welcome to metaBUS. *Personnel Assessment and Decisions*, 1(1), 2. doi:10.25035/pad.2015.002
- Bryer, J., & Speerschnieder, K. (2016). *likert: analysis and visualization of Likert items*. Retrieved from <https://CRAN.R-project.org/package=likert>
- Chan, C.-H., & Leeper, T. J. (2018). *rio: a Swiss-army knife for data I/O*. Retrieved from <https://CRAN.R-project.org/package=rio>
- Chang, W., Cheng, J., Allaire, J. J., Xie, Y., & McPherson, J. (2018). *shiny: web application framework for R*. Retrieved from <https://CRAN.R-project.org/package=shiny>
- Comtois, D. (2018). summarytools: Tools to Quickly and Neatly Summarize Data (Version 0.8.8). Retrieved from <https://cran.r-project.org/web/packages/summarytools/>
- Condon, D. M. (2018). The SAPA Personality Inventory: An empirically-derived, hierarchically-organized self-report personality assessment model. *PsyArXiv*. doi:10.31234/osf.io/sc4p9
- Crutzen, R. (2007). Time is a jailer: what do alpha and its alternatives tell us about reliability? *European Health Psychologist*.
- Crutzen, R., & Peters, G.-J. Y. (2017). Scale quality: alpha is an inadequate estimate and factor-analytic evidence is needed first of all. *Health Psychology Review*, 11(3), 242–247. doi:10.1080/17437199.2015.1124240
- Goldberg, L. R. (1999). A broad-bandwidth, public domain, personality inventory measuring the lower-level facets of several five-factor models. In I. Merviele, I. Deary, F. De Fruyt, & F. Ostendorf (Eds.), *Personality psychology in Europe* (Vol. 7, pp. 7–28). Tilburg, The Netherlands: Tilburg University Press. Retrieved from <http://admin.umd.edu.pk/Media/Site/STD/FileManager/OsamaArticle/26august2015/A%20broad-bandwidth%20inventory.pdf>

- Google. (2018, August 2). Dataset Search. Retrieved August 2, 2018, from <https://developers.google.com/search/docs/data-types/dataset>
- Gorgolewski, K. J., Auer, T., Calhoun, V. D., Craddock, R. C., Das, S., Duff, E. P., ... Poldrack, R. A. (2016). The brain imaging data structure, a format for organizing and describing outputs of neuroimaging experiments. *Scientific Data*, 3, 160044. doi:10.1038/sdata.2016.44
- Hardwicke, T. E., Mathur, M. B., MacDonald, K., Nilsson, G., Banks, G. C., Kidwell, M. C., ... Frank, M. C. (2018). Data availability, reusability, and analytic reproducibility: evaluating the impact of a mandatory open data policy at the journal Cognition. *Royal Society Open Science*, 5(8), 180448. doi:10.1098/rsos.180448
- Harrell, F. E., Jr. (2018). *Hmisc: Harrell Miscellaneous*. Retrieved from <https://CRAN.R-project.org/package=Hmisc>
- Helby Petersen, A., & Thorn Ekstrøm, C. (2018). dataMaid: A Suite of Checks for Identification of Potential Errors in a Data Frame as Part of the Data Screening Process (Version 1.2.0). Comprehensive R Archive Network (CRAN). Retrieved from <https://cran.r-project.org/web/packages/dataMaid/index.html>
- Holland, S., Hosny, A., Newman, S., Joseph, J., & Chmielinski, K. (2018). *The Dataset Nutrition Label: A Framework to Drive Higher Data Quality Standards*. *ArXiv*. Retrieved from <https://arxiv.org/abs/1805.03677>
- Introduction to Wikidata. (n.d.). Retrieved September 4, 2018, from <https://www.wikidata.org/wiki/Wikidata:Introduction>
- Kerwer, M., Bölter, R., Dehnhard, I., Günther, A., & Weichselgartner, E. (2017). Projekt DataWiz. Retrieved from [https://e-science-tage.de/sites/default/files/2017-04/est\\_talk\\_kerwer\\_17-03-2017.pdf](https://e-science-tage.de/sites/default/files/2017-04/est_talk_kerwer_17-03-2017.pdf)

- Kidwell, M. C., Lazarević, L. B., Baranski, E., Hardwicke, T. E., Piechowski, S., Falkenberg, L.-S., ... Nosek, B. A. (2016). Badges to Acknowledge Open Practices: A Simple, Low-Cost, Effective Method for Increasing Transparency. *PLoS Biology*, 14(5), e1002456. doi:10.1371/journal.pbio.1002456
- Larmarange, J. (2018). *labelled: manipulating labelled data*. Retrieved from <https://CRAN.R-project.org/package=labelled>
- Leszko, M., Elleman, L. G., Bastarache, E. D., Graham, E. K., & Mroczek, D. K. (2016). Future Directions in the Study of Personality in Adulthood and Older Age. *Gerontology*, 62(2), 210–215. doi:10.1159/000434720
- Lüdecke, D. (2018). sjPlot: Data Visualization for Statistics in Social Science. doi:10.5281/zenodo.1308157
- McNamara, A., Arino de la Rubia, E., Zhu, H., Ellis, S., & Quinn, M. (2018). *skimr: compact and flexible summaries of data*. Retrieved from <https://CRAN.R-project.org/package=skimr>
- McNeish, D. (2018). Thanks coefficient alpha, we'll take it from here. *Psychological Methods*, 23(3), 412–433. doi:10.1037/met0000144
- Noy, N., & Brickley, D. (2017, January 24). Facilitating the discovery of public datasets. Retrieved August 2, 2018, from <http://ai.googleblog.com/2017/01/facilitating-discovery-of-public.html>
- Ooms, J. (2014). *The OpenCPU system: towards a universal interface for scientific computing through separation of concerns*. *arXiv [stat.CO]*. Retrieved from <http://arxiv.org/abs/1406.4806>
- Peters, G.-J. Y. (2014). The alpha and the omega of scale reliability and validity: why and how to abandon Cronbach's alpha and the route towards more comprehensive assessment of scale quality. *European Health Psychologist*, 16(2), 56–69. Retrieved from

- <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.974.9750&rep=rep1&type=pdf>
- Rasmussen, K. B., & Blank, G. (2007). The data documentation initiative: a preservation standard for research. *Archival Science*, 7(1), 55–71. doi:10.1007/s10502-006-9036-0
- Revelle, W. (2018). *psych: procedures for psychological, psychometric, and personality research*. Retrieved from <https://CRAN.R-project.org/package=psych>
- Revelle, W., Condon, D. M., Wilt, J., French, J. A., Brown, A., & Elleman, L. G. (2016). Web and phone based data collection using planned missing designs. *Handbook of Online Research Methods*. Thousand Oaks, CA: Sage Publications. Retrieved from <http://mobile.personality-project.org/revelle/publications/websapa.final.pdf>
- Revelle, W., Wilt, J., & Rosenthal, A. (2010). Individual Differences in Cognition: New Methods for Examining the Personality-Cognition Link. In A. Gruszka, G. Matthews, & B. Szymura (Eds.), *Handbook of Individual Differences in Cognition: Attention, Memory, and Executive Control* (pp. 27–49). New York, NY: Springer New York. doi:10.1007/978-1-4419-1210-7\_2
- Roche, D. G., Kruuk, L. E. B., Lanfear, R., & Binning, S. A. (2015). Public Data Archiving in Ecology and Evolution: How Well Are We Doing? *PLoS Biology*, 13(11), e1002295. doi:10.1371/journal.pbio.1002295
- Shrout, P., & Lane, S. P. (2012). Psychometrics. In T. S. Conner & M. R. Mehl (Eds.), *Handbook of research methods for studying daily life* (pp. 302–320). New York: Guilford Press. Retrieved from <https://nyu.pure.elsevier.com/en/publications/psychometrics>
- Stanley, D. J., & Spence, J. R. (2018). Reproducible Tables in Psychology Using the apaTables Package. *Advances in Methods and Practices in Psychological Science*, 1(3), 415–431. doi:10.1177/2515245918773743
- Wickham, H., François, R., Henry, L., & Müller, K. (2018). *dplyr: a grammar of data manipulation*. Retrieved from <https://CRAN.R-project.org/package=dplyr>

Wickham, H., & Miller, E. (2018). *haven: import and export “SPSS”, “Stata” and “SAS” files*.

Retrieved from <https://CRAN.R-project.org/package=haven>

Wilkinson, M. D., Dumontier, M., Aalbersberg, I. J. J., Appleton, G., Axton, M., Baak, A., ...

Mons, B. (2016). The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data*, 3, 160018. doi:10.1038/sdata.2016.18

Xie, Y. (2018). *DT: a wrapper of the JavaScript library “DataTables.”* Retrieved from

<https://CRAN.R-project.org/package=DT>