

Codebook package

Step-by-step & Exercises

Jürgen Schneider

2/16/23

Table of contents

Installation	2
Aufgabe 1 (Erstes Codebook)	3
Aufgabe 2 (Codebooks vergleichen)	4
Aufgabe 3 (Variablenlabels)	5
Selbst anlegen	5
Aus “Dictionary” importieren	6
Aufgabe 4 (Wertelabels)	7
Manuell ergänzen/überarbeiten	7
Auf ein Set an Variablen anwenden	7
Aufgabe 5 (negative Items definieren)	9
Aufgabe 6 (Skalen definieren)	10
Voraussetzungen bereits erfüllt?	11
Manuell definieren	11
Troubleshooting	12
Reliabilitäten nicht berechnet	12
Viele Warnings angezeigt	12
Mittelwerte sind sehr klein/groß	13
Reliabilitäten sind sehr schlecht	13
Grafiken sind zu klein	13

Weitere Optionen	14
Daten in das HTML einbetten	14
Metadaten über gesamten Datensatz	14
Datensatz mit Metadaten exportieren	15
Einführung ins Codebook	16
Layout Themes	16
Session Info	17

i Wann muss ich die Code-Schnipsel aus dieser Anleitung ergänzen/verändern?

Oftmals können Sie die Code-Chunks einfach kopieren und in Ihr Skript an die passende Stelle einfügen. Manchmal müssen Sie aber auch etwas verändern. Diese Stellen habe ich in den Kommentaren mit **# DO:** gekennzeichnet. Beispiel:

```
print(paste("David Hasslehoff",           # DO: Ergänzen Sie Ihre Spotify Nr. 1
           "ist auf Spotify meine Nr. 1"))
```

Manche Kommentare habe ich aber auch nur eingefügt, um zu erklären was in der Funktion passiert. Diese Kommentare sind entsprechend *nicht* mit **# DO:** gekennzeichnet.

Tutorial

Dieser Workshop basiert stark auf Ruben Arslans [Tutorial des Codebooks](#).

Webseite

rubenarslan.github.io/codebook/index.html

Paper

Arslan RC. How to Automatically Document Data With the codebook Package to Facilitate Data Reuse. *Advances in Methods and Practices in Psychological Science*. 2019;2(2):169-187. doi:[10.1177/2515245919838783](https://doi.org/10.1177/2515245919838783)

Installation

Bitte installieren Sie die **Development-Version** des Codebook Packages!

Hier sind schon einige Bugs gefixed, die in der CRAN-Version noch drin sind.

Oftmals vermeidet es Installationsprobleme, wenn Sie die **Installation in R** (und nicht RStudio) **vornehmen**. Je nach Ihren gewählten Einstellungen, werden beim Start von RStudio nämlich bereits Pakete geladen mit denen dann Konflikte entstehen könnten.

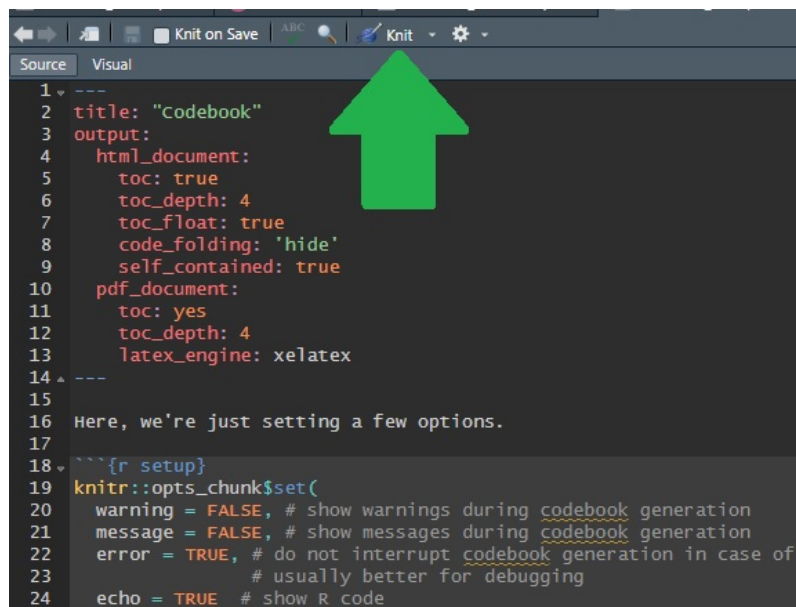
```
install.packages("remotes")
remotes::install_github("rubenarslan/codebook") # installiert direkt von github
```

Aufgabe 1 (Erstes Codebook)

1. Erstellen Sie ein neues Projekt in R (z.B. “codebook_package”)
2. Erstellen Sie einen Unterordner “data”, in den Sie Ihre Daten kopieren
3. Geben Sie in der Console `codebook::new_codebook_rmd()` ein
4. Im neu geöffneten Dokument wird ein Beispieldatensatz geladen. Ersetzen Sie diesen durch Ihren Datensatz:
 - Löschen Sie den Code `codebook_data <- codebook::bfi`
 - Ersetzen Sie ihn mit Code, der Ihren Datensatz in das Objekt `codebook_data` lädt. Mein Lieblingspaket hierfür ist `{rio}`, da es mit fast allen Dateiformaten umgehen kann.

```
codebook_data <- rio::import("data/meineDaten.csv") # DO: Pfad zum und Namen Ihres
# Datensatzes einfügen
```

5. Exportieren (“kniten”) Sie das Codebook in ein HTML indem Sie auf den Button “knit” klicken.



```
1 ---
2 title: "Codebook"
3 output:
4   html_document:
5     toc: true
6     toc_depth: 4
7     toc_float: true
8     code_folding: 'hide'
9     self_contained: true
10  pdf_document:
11    toc: yes
12    toc_depth: 4
13    latex_engine: xelatex
14 ---
15
16 Here, we're just setting a few options.
17
18 ```{r setup}
19 knitr::opts_chunk$set(
20   warning = FALSE, # show warnings during codebook generation
21   message = FALSE, # show messages during codebook generation
22   error = TRUE, # do not interrupt codebook generation in case of
23                # usually better for debugging
24   echo = TRUE # show R code
```

Aufgabe 2 (Codebooks vergleichen)

Schauen Sie Ihr Codebook genau an und vergleichen Sie es mit dem [Codebook aus einem meiner früheren Projekte \(Link\)](#).

- Was sind die kleinen und großen Unterschiede?
- Was sind Aspekte, die Sie in Ihrem Codebook noch ergänzen wollen?
- Was sind Aspekte, die bei meinem Codebook noch ergänzt werden könnten?

💡 Aufgaben

1. Vergleichen Sie die Codebooks (Individualarbeit)
2. Beantworten Sie die Fragen (für sich) (Individualar-

beit)

3. Zeigen Sie sich gegenseitig ihre Codebooks und sprechen Sie über die Unterschiede und was Sie noch ergänzen wollen.

Fertig?

Lassen Sie es mich wissen: pollev.com/js123

Aufgabe 3 (Variablenlabels)

Wenn Daten, wie in SAV-Dateien, bereits Metadaten (Variablen-, Wertelabels) beinhalten, dann sollten diese automatisch durch das `codebook` package übernommen werden. Sollten Sie aber z. B. CSV-Datendateien ohne Metadaten importiert haben, müssen diese nachträglich definiert werden.

Aufgaben

Ergänzen oder überarbeiten Sie Variablenlabels Ihrer Wahl. Verwenden Sie hierzu eine der beiden nachfolgenden Optionen.

Selbst anlegen

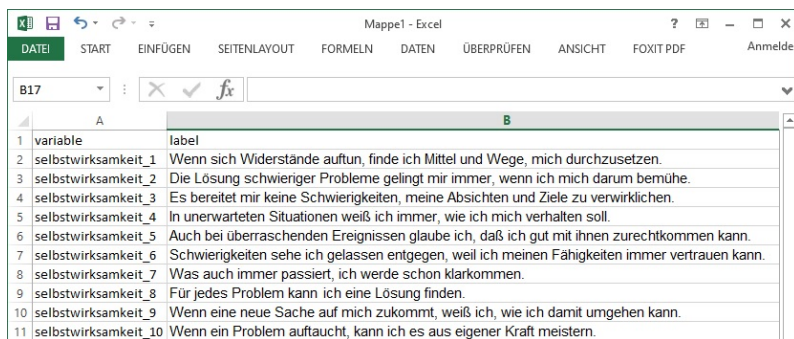
Legen Sie Variablenlabels selbst an, indem Sie die Funktion `var_label()` aus dem `{labelled}` package verwenden. `var_label()` nimmt eine Liste an, in der Sie mehrere Variablenlabels gleichzeitig definieren können. Auf der linken Seite des `=` geben Sie den Namen der Variablen an, wie sie im Datensatz erscheint und auf der rechten Seite des `=` das Label der Variablen.

```
library(labelled)
var_label(codebook_data) <- list(      # DO: Unten Ihre Variablennamen und -labels einfügen
  selbstwirksamkeit_7 = "Was auch immer passiert, ich werde schon klarkommen.",
  selbstwirksamkeit_8 = "Für jedes Problem kann ich eine Lösung finden."
)
```

Aus "Dictionary" importieren

Sie haben eine Excelliste o.ä. in dem Sie die Variablen und deren Variablenlabels bereits notiert haben? Gut! Diese können Sie importieren und einfach auf Ihren Datensatz loslassen.

Voraussetzung ist, dass im sogenannten "Dictionary" die Namen der Variablen in der Spalte "variable" und die Variablenlabel in der Spalte "label" stehen. Ähnlich wie hier:



variable	label
selbstwirksamkeit_1	Wenn sich Widerstände auftun, finde ich Mittel und Wege, mich durchzusetzen.
selbstwirksamkeit_2	Die Lösung schwieriger Probleme gelingt mir immer, wenn ich mich darum bemühe.
selbstwirksamkeit_3	Es bereitet mir keine Schwierigkeiten, meine Absichten und Ziele zu verwirklichen.
selbstwirksamkeit_4	In unerwarteten Situationen weiß ich immer, wie ich mich verhalten soll.
selbstwirksamkeit_5	Auch bei überraschenden Ereignissen glaube ich, daß ich gut mit ihnen zurechtkommen kann.
selbstwirksamkeit_6	Schwierigkeiten sehe ich gelassen entgegen, weil ich meinen Fähigkeiten immer vertrauen kann.
selbstwirksamkeit_7	Was auch immer passiert, ich werde schon klarkommen.
selbstwirksamkeit_8	Für jedes Problem kann ich eine Lösung finden.
selbstwirksamkeit_9	Wenn eine neue Sache auf mich zukommt, weiß ich, wie ich damit umgehen kann.
selbstwirksamkeit_10	Wenn ein Problem auftaucht, kann ich es aus eigener Kraft meistern.

Importieren Sie dieses Dictionary und wenden Sie es auf den Datensatz an, anhand der Funktion `dict_to_list()`.

```
library(dplyr)      # Ermöglicht die Zeilenschreibweise mit %>%
dict <- rio::import("data/meinDictionary.xlsx") # DO: Speicherort des Dictionary einfügen

var_label(codebook_data) <- dict %>%
  dplyr::select(variable, label) %>% # wählt die Spalten "variable" und "label" aus
  dict_to_list()                    # wendet das Dictionary auf die Daten an
```

Fertig?

Lassen Sie es mich wissen: pollev.com/js123

Aufgabe 4 (Wertelabels)

Auch die Wertelabels sollten automatisch aus Daten übernommen werden, die bereits Metadaten beinhalten. Auch hier gilt: Diese können fehlerhaft oder (im Falle fehlender Metadaten) nicht vorhanden sein.

Sie können Wertelabels für jede Variable **manuell** ergänzen/überarbeiten. Zudem können Sie sich zunutze machen, dass manchmal mehrere Items (z. B. Likert-Skalen) dieselben Wertelabels aufweisen. In diesem Fall müssen Sie die Wertelabels nicht einzeln für jede der Variablen definieren, sondern können sie gleich **auf ein ganzes Set an Variablen anwenden**.

Manuell ergänzen/überarbeiten

Zur Ergänzung der Wertelabels nutzt codebook die Funktion `val_labels()` aus dem `{labelled}` package. Für die Definition müssen Sie die Wertelabels links des `=` als character und die Werte rechts des `=` bereitstellen. Auf welche Variable Sie diese Wertelabels anwenden wollen, tragen Sie innerhalb der Funktion `val_labels()` ein:

```
val_labels(codebook_data$sprache) <- # DO: Name der Variable ergänzen
  c("Deutsch ist Muttersprache" = 1, # DO: Wertelabels & Werte ergänzen
    "Deutsch ist nicht Muttersprache" = 2) # DO: Wertelabels & Werte ergänzen
```

Auf ein Set an Variablen anwenden

Wenn sich Wertelabels für eine Reihe an Items wiederholen, dann definieren Sie zunächst die Wertelabels und Werte (links und rechts des `=`). Mit diesem Code erstellen Sie eine Funktion, die wir anschließend auf ein Set an Variablen anwenden werden:

```

add_likert_labels <- function(x) {
  val_labels(x) <- c("Stimmt gar nicht" = 1,      # D0: Wertelabels
                    "Stimmt eher nicht" = 2,      #      und
                    "Teils/teils" = 3,              #      Werte
                    "Stimmt eher" = 4,             #      ergänzen
                    "Stimmt voll und ganz" = 5) #
  x
}

```

Jetzt müssen Sie noch einen Vektor mit den Namen der Variablen erstellen (innerhalb der Funktion `c()`), auf welche die neuen Wertelabels angewendet werden sollen.

```

codebook_data <- codebook_data %>%
  dplyr::mutate(across(c(selbstwirksamkeit_1,      # D0: Definition aller
                        selbstwirksamkeit_2,      #      Items mit denselben
                        selbstwirksamkeit_3,      #      Wertelabels
                        selbstwirksamkeit_4,      #
                        selbstwirksamkeit_5,      #
                        selbstwirksamkeit_6,      #
                        selbstwirksamkeit_7,      #
                        selbstwirksamkeit_8,      #
                        selbstwirksamkeit_9,      #
                        selbstwirksamkeit_10), #
                  add_likert_labels)) # oben definierte Wertelabels
                                     # werden angewendet.

```

Aufgabe

1. Definieren Sie die Wertelabels für eine Variable manuell
2. Definieren Sie die Wertelabels für ein Set an Variablen (falls anwendbar)
3. Kontrollieren Sie in einem vorher-nachher-Vergleich, ob die Umcodierung korrekt geklappt hat.

Fertig?

Lassen Sie es mich wissen: pollev.com/js123

Aufgabe 5 (negative Items definieren)

Sind Items negativ formuliert und man bildet eine Skala, sind die Reliabilitätswerte logischerweise nicht sehr hoch. Auch hier liefert das `codebook` package Funktionen mit: `addR()` und `reverse_labelled_values()`

Negativ formulierte Items kennzeichnen

Zunächst müssen Sie die negativ formulierten Items, die umcodiert werden sollen als solche kennzeichnen. Wir nehmen das vor, indem wir dem Variablennamen ein “R” anhängen. Das geht automatisch mit der Funktion `addR()`, sie müssen lediglich die negativ formulierten Items auswählen (hier z. B. `Item1`, `Item2`)

```
library(dplyr)    # Ermöglicht die Zeilenschreibweise mit %>%

codebook_data <- codebook_data %>%
  rename_with(add_R,
              c(Item1, Item2)) # DO: wählen Sie die negativ formulierten Items aus
```

Variablen mit “R” im Namen umcodieren

Der folgende Code wählt alle Items aus, die auf “R” enden und codiert die Zahlenwerte um, lässt aber die Wertelabels gleich. So ist immer ersichtlich, ob das Item umcodiert wurde oder nicht. Im Code müssen Sie nichts ändern.

```
codebook_data <- codebook_data %>%
  dplyr::mutate(across(matches("\\dR$"), # wählt alle Items aus, deren Name auf R enden
                 reverse_labelled_values) # codiert die Zahlenwerte um,
               )                          # belässt aber die Wertelabels
```

Aufgabe

1. Benennen Sie die negativ formulierten Items ihres Datensatzes anhand der Funktion `addR()` um.

2. Codieren Sie diese Items dann um.
3. Schauen Sie sich die Werte im bisherigen Codebook an, knüpfen Sie das neue Codebook und kontrollieren Sie, ob das Ergebnis stimmt (Vorher-Nachher-Vergleich): Sind die richtigen Items umcodiert worden?

Fertig?

Lassen Sie es mich wissen: pollev.com/js123

Aufgabe 6 (Skalen definieren)

Items einer Skala werden im Codebook automatisch als Skala berichtet. Sollte das package aber nicht erkennen, dass es sich um eine Skala handelt, dann werden sie als Einzelitems im Codebook berichtet.

Einzelitems im Codebook

Items als Skala im Codebook

Aufgabe

1. Definieren Sie eine Skala in Ihrem Datensatz anhand einer der zwei Möglichkeiten, siehe unten.
2. Überprüfen Sie die Plausibilität der Skala im Codebook.

Möglichkeiten (dass codebook Skalen als solche erkennt)

Voraussetzungen bereits erfüllt?

Wenn beide Voraussetzungen erfüllt sind:

- Die Items einer Skala haben dieselben Variablennamen und unterscheiden sich lediglich durch eine laufende Nummer am Ende (z. B. **neuro_1**, **neuro_2**, **neuro_3**)
- Es existiert eine Skalenvariable, die auch den Namen der Items trägt (ohne Zahl, z. B. **neuro**)

Übrigens

Das Survey-Tool formr berechnet automatisch die Skalenvariable, wenn die Items im Fragebogen so angelegt sind, wie oben beschrieben: Derselbe Variablennamen mit laufenden Nummern am Ende.

Manuell definieren

Sie können die Items einer Skala auch manuell definieren, codebook stellt hierfür die Funktion `aggregate_and_document_scale()` bereit. In diesem Fall benötigen Sie also nur die Variablen der Items, eine Skalenvariable muss noch nicht erstellt worden sein. Sie erstellen die Skalenvariable durch `aggregate_and_document_scale()` - in unserem Falle benennen wir die Variable `extraversion`.

Um das sogenannte Piping (`%>%`) und die Funktion `select()` zu benutzen, müssen Sie noch das package `{dplyr}` laden.

```
library(dplyr)      # Ermöglicht die Zeilenschreibweise mit %>%

codebook_data$extraversion <- codebook_data %>% # DO: Wählen Sie einen Namen für die neue Skala
  dplyr::select(E1:E5) %>%                     # DO: Wählen Sie die Items der Skala aus
  aggregate_and_document_scale()               # definiert Skala aus den Items E1:E5
```

FYI: In diesem Fall muss die neu erstellte Skalenvariable nicht zwangsläufig dem Namen der Items tragen.

Troubleshooting

Reliabilitäten nicht berechnet

Installieren Sie das package {ufs}.

```
install.packages("ufs")
```

Installieren Sie das package {GGally}.

```
install.packages("GGally")
```

Viele Warnings angezeigt

Die Warnings, Messages und Errors werden absichtlich angezeigt. Dies hilft beim debugging, also der Fehlersuche, wenn ein Teil des Codes nicht funktionieren sollte.

Natürlich kann man deren Anzeige auch deaktivieren. Die sogenannten Chunk-Options werden gleich im ersten R-Chunk definiert - dort können Sie die Benachrichtigungen also auch ausstellen: Setzen Sie hierfür die Argumente `warning`, `message` und `error` auf `FALSE`.

```
knitr::opts_chunk$set(  
  warning = FALSE, # don't show warnings during codebook generation  
  message = FALSE, # don't show messages during codebook generation  
  error = FALSE, # don't show errors  
  echo = TRUE # show R code  
)
```

Mittelwerte sind sehr klein/groß

Haben Sie Ihre fehlenden Werte im ursprünglichen Datensatz mit 99 oder 999 definiert?

Oder haben Sie fehlenden Werte mit negativen Werten definiert?

Wahrscheinlich wurden diese Werte in R nicht als fehlend interpretiert. Die Funktion `detect_missing()` hilft Ihnen dabei diese fehlend zu definieren.

```
codebook_data <- detect_missing(codebook_data,
  negative_values_are_missing = FALSE, # DO: Wenn negative Werte fehlend sind,
                                     # dann hier TRUE setzen
  ninety_nine_problems = TRUE,      # DO: Wenn 99 oder 999 fehlende Werte sind,
                                     # dann hier TRUE setzen
)
```

Reliabilitäten sind sehr schlecht

1. Haben Sie die negativ formulierten Items umgepolt? How to Items umpolen, siehe Aufgabe 5.
2. Haben Sie Ihre fehlenden Werte als solche definiert? Siehe ein Punkt weiter oben.

Grafiken sind zu klein

Die Grafiken können in den Chunk-Options skaliert werden. Spielen Sie etwas an den Chunk-Options herum und schauen Sie, welche Einstellungen für Sie einen passenden Output liefern.

```
knitr::opts_chunk$set(
  fig.width = 9,      # DO: Setzen Sie unterschiedliche Werte für Breite
  fig.height = 7      # und Höhe der Abbildungen ein
)
```

)

Weitere Optionen

Daten in das HTML einbetten

Verwenden Sie das `xfun` package, um Ihre Daten direkt in das HTML einzubetten. Nutzer*innen können die Daten aus dem HTML heraus herunterladen (auch ohne Internetverbindung). Wenn Sie die HTML Datei weitergeben, sind die Daten also schon enthalten! Beispiel:

Click here to download the on the topic of never giving up.

Hier habe ich die `embed_file()` Funktion inline (also im Fließtext) verwendet:

```
"...download the `r xfun::embed_file("www/never.gif",  
text = "qualitative data")` on the topic..."
```

Metadaten über gesamten Datensatz

Genau wie Metadaten für einzelne Variablen definiert werden, können auf Ebene des gesamten Datensatzes Metadaten definiert werden. Diese geben Auskunft über Aspekte, wie

- Name des Datensatzes `name`
- Beschreibung des Datensatzes `description`
- Erhebungszeitraum `temporalCoverage`
- Lizenz (normalerweise URL zur Lizenz) `license`
- Persistenter Identifier (normalerweise DOI) `identifier`
- Name der*des Urheber*in `creator`
- URL (z. B. können Kontaktinformationen durch Link auf die ORCID hinterlegt werden) `url`
- Publikationsdatum `datePublished`

```
# DO: Hier können jeweils Informationen ergänzt werden
metadata(codebook_data)$name <- "Data of project SE-PRIME"
metadata(codebook_data)$description <- "Investigation of self-efficacy in ..."
metadata(codebook_data)$temporalCoverage <- "Apr 2022 - Feb 2023"
metadata(codebook_data)$license <- "https://creativecommons.org/licenses/by/4.0/"
metadata(codebook_data)$identifier <- "https://doi.org/10.1080/00461520.2021.1886103"
metadata(codebook_data)$creator <- "Erika Musterfrau"
metadata(codebook_data)$url <- "https://orcid.org/0000-0002-7867-2101"
metadata(codebook_data)$datePublished <- "2023-02-13"
```

Datensatz mit Metadaten exportieren

Bevor Sie das Codebook mit `codebook(codebook_data)` erzeugen, können Sie den Datensatz mit allen Metadaten exportieren. Wählen Sie hierzu ein Dateiformat aus, das es erlaubt Metadaten mitzuspeichern. Hierzu gehört das

- R data structure Format `.rds`
- SPSS data file Format `.sav`

Für das `.rds`-Format spricht, dass es auf einer Open Source Software aufbaut und somit besser accessible ist.

in `.rds` speichern

```
rio::export(codebook_data, "meinDatensatz.rds") # DO: Name der Datei ergänzen
```

in `.sav` speichern

```
rio::export(codebook_data, "meinDatensatz.sav") # DO: Name der Datei ergänzen
```

Einführung ins Codebook

Sie können den obersten Teil des Codebooks selbst gestalten. Es empfiehlt sich hier evtl. eine kurze Einführung in das Projekt oder den Datensatz bereitzustellen.

Bearbeiten Sie hierzu den Fließtext nach dem YAML-Teil. Welche Formatierungsoptionen Sie in R Markdown haben, können Sie im [R Markdown Cheat Sheet \(Link\)](#) nachschauen.

Layout Themes

Wollen Sie einen anderen Look für das Dokument ausprobieren? Eine Auswahl an Themes finden Sie in der [Theme Gallery \(Link\)](#).

Den Namen des Themes müssen Sie dann lediglich im YAML-Teil des R Markdown Dokuments definieren:

```
---
title: "Codebook"
output:
  html_document:
    toc: true
    toc_depth: 4
    toc_float: true
    code_folding: 'hide'
    self_contained: true
    theme: cerulean      # DO: Tragen Sie hier den Namen des Themes ein
---
```


Session Info

Zum Zwecke der Reproduzierbarkeit, hier noch mein Setup.
This was produced under the R-Version and with the packages:

```
sessionInfo()
```