The ROCK book

Szilvia Zörgő & Gjalt-Jorn Peters

13:52:02 on 2019-12-11 UTC (GMT+0000)

Contents

Introduction **5**

I Qualitative Research **7**

1 Qualitative research **9**

2 Qualitative Ethnography? **11**

3 Qualitative methods in psychology **13**

4 Cognitive interviews **15**

5 Reporting Qualitative Research **17**

5.1 Introduction 18

5.2 Methods 18

5.3 Results 19

5.4 Discussion 19

II The ROCK **21**

6 The ROCK vocabulary **23**

7 The ROCK standard **25**

7.1 Examples 28

8 The `rock` R package **31**

8.1 Downloading and installing R and RStudio . . . 31

8.2 Downloading and installing the `rock` package . . . 32

8.3 Functions in the `rock` package . . . 33

9 The iROCK interface **37**

9.1 Background . . . 37

9.2 Using iROCK . . . 37

10 A ROCK workflow **39**

10.1 A basic ROCK workflow . . . 39

10.2 An extensive ROCK workflow . . . 43

11 Using the ROCK for Epistemic Network Analysis **49**

11.1 Starting point . . . 50

11.2 Planning coding . . . 50

11.3 Planning Segmentation . . . 52

11.4 Designating Source and Cases . . . 55

11.5 Designating Attributes . . . 55

11.6 Coding and Segmentation . . . 58

11.7 Merging Coded Sources (if necessary) . . . 60

11.8 Creating Networks . . . 60

12 Using the ROCK for Decentralized Construct Taxonomies **61**

12.1 Introduction to Decentralized Construct Taxonomies . . . 61

12.2 Creating a DCT . . . 62

12.3 Coding with DCTs . . . 65

12.4 Analysing DCT-coded sources . . . 66

13 Using the ROCK for Cognitive Interviews **67**

14 References **69**

Introduction

This is the Reproducible Open Coding Kit book, a living[1] Open Access[2] book. The Reproducible Open Coding Kit, the ROCK, is a standard for qualitative research. It was developed to facilitate reproducible and open coding, enabling explicit documentation of the various analysis steps without compromising the flexibility that lends much of the qualitative research its strength.

At its core, the ROCK specifies conventions for including codes and attributes in plain text files. This standard enables coding and reading qualitative data without the requirement of any software other than a plain text file editor. Simultaneously, it allows the development of software to, for example, further process those files to support specific analyses, or to provide a graphical user interface to facilitate coding. This book will discuss two such implementations of the ROCK. The first is the `rock` R package, an R package originally developed to facilitate using the ROCK standard to support Epistemic Netwok Analyses, but since then extended to also use the ROCK for cognitive interviews and to work with decentralized construct taxonomies (DCTs). The second is iROCK, a rudimentary graphical user interface to code text files using the ROCK.

The ROCK helps qualitative researchers navigate the imperatives created by Open Science and the General Data Protection Regulation. On the one hand, because no proprietary software is required, researchers retain complete control over the data they process. This means that they do not need to enter into data processing agreements with third parties if they process identifiable data. On the other hand, if they work with anonymized data, because the data are stored in plain text files, they can easily be shared with other researchers, who can then easily re-run (or adapt and re-run) the analyses.

Note that this book is mostly written to cover reproducible coding of qualitative data using the ROCK, although it does cover some basic underpinnings of

[1] A living document is a document that can be updated over time. Conventional books have editions; living books can be updated in smaller steps.

[2] Open Access means that it is free. Specifically, the license attached to this book for now is the CC-BY-NC-SA license. This roughly means you're allowed to download, copy, and share the book; you're allowed to change it, as long as you apply the same license to the adapted version; but you're not allowed to sell it. The authors can of course always grant specific rights anyway.

qualitative methods in Part I. Chapter 1 will start with a brief introduction of a number of qualitative approaches which are then discussed more in detail in the other Chapters in Part I. Part II will the continue, based on this introduction, with Chapter 6, which will establish a vocabulary. At the hand of this vocabulary, Chapter 7 will introduce the ROCK standard. Chapter 8 will discuss the `rock` R package, and Chapter 9 will discuss the iROCK interface. Chapter 10 combines all this in an example of a ROCK-based workflow. Chapter 11 concerns applying the ROCK to Epistemic Network Analysis, Chapter 12 concerns applying the ROCK to work with decentralized construct taxonomies, and Chapter 13 concerns applying the ROCK to work with cognitive interviews. Because this is a living book, more chapters may be added.

Impatient readers may want to skip ahead to Chapters 10, 11, 12, and 13, if those match one of their use cases. At present, this book is still under development: the first complete edition has not yet been released. Since the `rock` R package used in this book is also under active development, at this point you may want to install the development version of the `rock` R package (see Chapter 8).

Tot cite this book, you can use:

Zörgő, S. & Peters, G.-J. Y. (2019) The ROCK book (1st Ed.). doi:10.5281/zenodo.3571020

Part I

Qualitative Research

Chapter 1

Qualitative research

Disciplines that directly or indirectly study the human psychology, such as anthropology, sociology, criminology, and psychologyy, face the problem that the studied objects are not directly observable (and often are not natural kinds[1]). The methods with which the indirect observations that are used instead are collected can be divided into two types: quantitative and qualitative methods.

Quantitative methods are comparable to most measurement instruments, and aim to map unobservable variables unto a quantitative scale. The means through which this occurs is called operationalization of those variables, and without a valid operationalization, quantitative methods cannot be applied. Qualitative methods do not require such quantification. This is simultaneously a strength and a weakness.

[...]

[1] Explain plus link to Eiko's commentary.

Chapter 2

Qualitative Ethnography?

Or quantitative?

It’s a bit weird that quantitative ethnography is in fact qualitative :-)

[...]

Chapter 3

Qualitative methods in psychology

Within psychology, there are broadly two ways in which qualitative methods are used, one fundamentally inductive, the other using a combination of inductive and deductive approaches. First, if (almost) no theory exists about a subject, qualitative data can be collected to contribute to the creation of a large evidence base in which patterns can be identified that can then give rise to theory. Second, if theory does exist, the constructs posited by those theories often have content that differs for different populations, contexts, and behaviors. Once a theoretical construct has been clearly defined, it is possible to derive guidelines for eliciting the construct's content in a given population, context, or otherwise relevant circumstance.

Chapter 4

Cognitive interviews

Cognitive interviews are a method to study the cognitive validity of operationalizations of psychological constructs. [...]

Chapter 5

Reporting Qualitative Research

Best practices for reporting differ between different types of research. For example, in most quantitative research, the employed statistical models usually require that sampling procedures are designed to sample randomly from the population, and accurate estimation with those statistical models usually requires considerable sample sizes. Because the sampling and measurement error can then be modelled, uncertainty can be estimated, which then enables reporting quantitative results in a transparent and integreous manner. In most qualitative research, on the other hand, sampling procedures are designed to optimize sample heterogeneity, data collection can be partly driven by the observations and is therefore less systematic, and the error in sampling and observation cannot be modeled.

Fundamental differences such as these have a number of implications for how research is reported. In this Chapter, we will list a number of best practices for reporting qualitative research, paying special attention to similarities and differences between qualitative and qualitative methods. Consistent with the Justify Everything principle, we will also provide justifications for these best practices. We will follow the conventional manuscript structure of introduction, methods, results, and discussion.

5.1 Introduction

5.2 Methods

5.2.1 Full Disclosure

This section includes the link to the repository containing the Full Disclosure package for this study. A Full Disclosure Package consists of a Replication Package and an Analysis Package.

The Replication Package commonly contains everything required for, or facilitating, replication of the study, such as the request for ethical approval and the confirmation letter granting ethical approval; the communication templates for communicating with participants; the recruitment protocols; the interview scheme or topic list; the protocols for support for participants and for interviewers, for studies where the interviews might touch upon sensitive topics; and the transcription procedures; the data management plans. The Replication Package should allow other researchers to replicate your data collection with minimal effort.

The Analysis Package commonly contains the (anonymized) raw data; the (anonymized) processed data; the documentation of the analysis steps and decisions taken; justifications of those decisions; and your results. The Analysis Package should allow other researchers to replicate your data analysis with minimal effort.

5.2.2 Materials

This section describes the used materials, such as the interview scheme or topic list; the communication templates for communicating with participants; the recruitment protocols; and the protocols for support for participants and for interviewers, for studies where the interviews might touch upon sensitive topics. In this section, also describe how data will be recorded.

5.2.3 Sampling strategy

In this section, describe the sample composition you aim to achieve, and how you operationalized these goals.

5.2.4 Analysis plan

Here, describe the analysis plan. This includes the type of coding (e.g. inductive, deductive, or a combination), how many coders will be involved and how the sources will be divided, etc etc etc

5.3 Results

5.3.1 Participants

In this section, describe the participants of the study. This ideally happens in a similar manner for quantitative and qualitative research. Where the Sampling Strategy section in the Methods section described the aims and strategies, this section describes the results of those efforts. Note that unless the study is a case study of one or more cases, the individuals are not the primary interest. Thereofore, this section should normally describe your *sample*, and not be a list of descriptions of each participant. To the degree that specific participant characteristics are important for contextualizing specific source fragments as described later on, list those characteristics at that point in the narrative - do not burden readers with mixing and matching throughout the manuscript. Note that more extensive descriptions can always be include in the repository accompaying the manuscript.

5.3.2 Background

In some cases it may be beneficial to start with a description of the context of the participants and data collection efforts. Such contextualisation may be important to properly interpret the results, even though the information itself may not pertain to the relevant research questions. Therefore, it can be useful to separate this description from the primary results.

5.3.3 Results

Then, describe the results.

5.4 Discussion

Part II

The ROCK

Chapter 6

The ROCK vocabulary

In the plethora of qualitative approaches, many different terms exist and often partially overlap. The ROCK standard uses the terms listed below.

attribute A property or characteristic of a *case*, for example demographic variables such as interviewee age, gender, education level. Attributes can also be characteristics of *case* types that are not persons, such as interview venue (an attribute can be e.g. whether it was crowded or not) or interviewer (an attribute can be e.g. the interviewer's age).

case A data provider or context. In interview studies, a case is usually a specific person. Assigning utterances to cases is a means to efficiently associate *attributes* to many utterances in one go. Cases can also be used to associate other information to many utterances, such as the interviewer, the place where an interview took place, or the time of day. Examples of cases are "participant 4" to identify a person, "14:00" to identify the time of an interview, "meeting room B" to identify the location of an interview.

code A brief *identifier* applied to a *fragment*. Such a code usually represents a concept. Codes can vary from simple descriptions, for example to denote that the coded fragment concerns a topic such as "leisure activities", to complex constructs, for example to denote that the coded fragment likely espresses psychological aspects that fall within the definition of a construct called "perceived autonomy".

coding tree The hierarchy of codes used to *code* one or more *sources* (also called *coding structure*).

fragment A part of a *source* (one or more consecutive characters, such as one or more words, sentences, or paragraphs).

identifier A unique character sequence that uniquely identifies something. For example, a Uniform Resource Locator (URL) is an identifier (commonly for a website); a Digital Object Identifier (DOI) is an identifier (commonly for a scientific article); and an International Standard Book Number (ISBN) is an identifier (commonly for a book). The ROCK implements

a way to generate and specify identifiers for *utterances* and a way to add other identifiers to a *source*, such as for *cases*.

section A delimited *fragment* of a source.

section break A sequence of characters that represents a *section* delimiter. In other words, section breaks split up *sources* into *sections*. The ROCK standard allows parallel use of multiple types of section breaks: for example, one type of section break can indicate paragraph breaks, whereas another type of section break can indicate where an interviewer asks a new question, and yet another type can indicate where there is a turn of talk between participants in a discussion.

source A plain text file that describes or captures a bit of reality. The most common sources in research with humans (e.g. anthropology and psychology) are interview transcripts, but sources can also be internet content, archive materials, meeting minutes, descriptions of photographs, or timestamped descriptions of video material.

utterance The shortest *codable fragment* of a *source*. In the ROCK, these are by default delimited by line breaks (“`\n`”), and utterances will usually correspond to sentences.

YAML YAML is a standard for encoding data in plain text files in a way that is easily readable by humans. The ROCK standard uses the YAML standard for specifications of *attributes* as well as deductive code structures. YAML is a recursive acronym that stands for “YAML Ain’t Markup Language”, and is technically a JSON (Javascript Serial Object Notation) superset, which means that all JSON is valid YAML.

Chapter 7

The ROCK standard

The ROCK standard has been developed as an open standard for qualitative data analysis. It follows the principles that also guided the development of the Markdown and YAML standards: prioritizing human-readability while retaining machine-readability. The aim of the ROCK is to provide a standard that enables researchers to exchange data and analyses in a format that is readable even without running any specific software. In other words, coded transcripts should be readable as is.

This open standard enables development of programs or scripts to perform specific functions that are not yet present in any of the existing applications that support the ROCK format. In addition, this enables all existing qualitative data analysis programs to import data files in this format and to export to this format.

In this chapter, the vocabulary explained in Chapter 6 is used to describe the ROCK standard. Qualitative data files that implement the ROCK standard can be recognized by their extension: `.rock`. These files normally follow the conventions set out in this chapter.

7.0.1 Codes

Codes are by default any string of characters (specifically, lower or uppercase letters, digits, periods, underscores, larger-than signs, and dashes) in between two pairs of square brackets (`[[` and `]]`). This is described by the regular expression `\[\[([a-zA-Z0-9._>-]+)\]\]`; note that the escaping backslashes must be escaped themselves by prepending a second backslash when specifying this regular expression in R. Codes are designated per utterance, or in other words, per line. As many codes can be specified per line as one wishes. For example, see these two lines (utterances):

```
So what went right [[reflection-positive]]
What went wrong [[reflection-negative]]
```

The first line is coded with `reflection-positive`, and the second line with code `reflection-negative`.

7.0.2 Structuring inductive codes

When engaging in inductive coding (i.e. when not working with a prespecified code structure, but instead developing the code structure as one goes along; see the section below re: deductive coding), it can be desirable to structure the codes hierarchically. For example, perhaps a researchers wants to specify a parent code such as `reflection` with two child codes such as `positive` and `negative`. This helps one to identify patterns in the data, and makes it possible to easily extract all utterances coded as any type of reflection. By default, the marker that can be used to structure inductive codes is the greater than sign (specified by the regular expression `>`). For example, see the same fragment but coded in two levels:

```
So what went right [[reflection>positive]]
What went wrong [[reflection>negative]]
```

When this source is parsed by `rock`, it will recognize these deductive codes and their structure, and it will generate the corresponding hierarchical coding structure, as illustrated in the more extensive example below.

7.0.3 Specifying identifiers

It is often desirable to attach specific attributes to utterances. For example, one may want to compare the patterns in codes between different categories of participants, such as those who do and do not own a car, or those that listen to progressive metal versus those that listen to psychedelic trance. Instead of coding all utterances with all relevant attributes, instead, it is possible to specify identifier to easily link utterances to characteristics of the data provision (such as data providers, for example participants, or the moment of data collection, for example daytime or nighttime, or winter or summer, or the location of data collection, such as in a busy place or in a silent office).

This can be done by specifying identifiers. These are again specified using regular expressions. By default, two types of identifiers are specified: case identifiers and stanza identifiers. They are again specified using two pairs of square brackets, but this time, the opening brackets are immediately follow by a string of identifying characters (the 'identifier identifier', so to speak), followed by an equals sign, and then by the unique identifier. This may seem a bit abstract; it will become clearer as we look at the first example.

7.0.3.1 Case identifiers

Case identifiers can be used to link utterances to data providers, such as participants. Their 'identifier identifier' is `cid`, and by default, their full regular expression is `\[\[cid=([a-zA-Z0-9._-]+)\]\]`. A source excerpt coded with only case identifiers may look like this:

```
CAIAPHAS: No, wait!   We need a more permanent solution to our problem. [[cid=1]]

ANNAS: What then to do about Jesus of Nazareth?   Miracle wonderman, hero of fools. [[cid=2]]

PRIEST THREE: No riots, no army, no fighting, no slogans. [[cid=3]]

CAIAPHAS: One thing I'll say for him -- Jesus is cool. [[cid=1]]

ANNAS: We dare not leave him to his own devices.   His half-witted fans will get out of control.
```

(Note that in this example, the names of the participants were retained; normally, the researcher would anonimyze the transcripts so as to allow publication of the coded transcripts.)

WHen `rock` parses this source, it will know that the first and fourth utterances belong to the same case, as do the second and fifth. The attributes specified for these cases will then be attached to these utterances (see the section about metadata below).

7.0.3.2 Stanza identifiers

A stanza is a unit of analysis in ENA analysis (see the glossary for the exact definition).

7.0.4 Specifying deductive coding structures

When a researcher works with a prespecified coding structure (i.e. engages in deductive coding), they only use codes that were determined a priori. Like in inductive coding, there are often multiple levels in such a coding structure, with the codes organised hierarchically. To efficiently be able to collapse codes to higher levels, `rock` needs to know the deductive coding structure. This can be specified using YAML fragments in the sources. YAML fragments are, by default, delimited by two lines that each contain only three dashes (`---`). Between those delimiters, YAML (a recursive acronym that stands for 'YAML ain't markup language') can be specified. Specifically, in YAML terminology, each fragment should be a sequence of mappings that is named `codes`.

The coding tree specified in the section on inductive coding, for example, can be efficiently specified as a deductive coding structure like this:

```
---
codes:
  -
    id: reflection
    children:
      -
        id: positive
      -
        id: negative
---
```

If all children of a code are so-called ‘leaves’ (i.e. in the coding tree, they have no children of their own[1]) they can be specified more efficiently:

```
---
codes:
  -
    id: reflection
    children: ["positive", "negative"]
---
```

When `rock` parses the sources, it will collect all such code specifications and combined them into one coding three using each code’s identifiers. It is possible to specify a parent in other code specification fragment by adding the field `parentId`. For example, in other sourrce, we could add this fragment:

```
---
codes:
  -
    id: neutral
    parentId: reflection
---
```

This would add `neutral` as a sibling to `positive` and `negative`.

7.0.5 Specifying metadata

7.1 Examples

7.1.1 Section breaks

```
So what went right
```

[1]This is less sad than it may look; voluntary childlessness is becoming more and more common, and not having children is one of the most effective choices one can make to save the environment.

```
What went wrong
---paragraph-break---
Was it a story
or was it a song
---paragraph-break---
Was it over night
Or did it take you long
---paragraph-break---
Was knowing your weakness
what made you strong
```

Source excerpt as example of section breaks (lyrics from Smiley Faces by Gnarls Barclay)

7.1.2 Identifiers

```
CAIAPHAS
No, wait!   We need a more permanent solution to our problem.

ANNAS
What then to do about Jesus of Nazareth?   Miracle wonderman, hero of fools.

PRIEST THREE
No riots, no army, no fighting, no slogans.

CAIAPHAS
One thing I'll say for him -- Jesus is cool.

ANNAS
We dare not leave him to his own devices.   His half-witted fans will get out of control.

PRIESTS
But how can we stop him?   His glamour increases   By leaps every moment; he's top of the poll.

CAIAPHAS
I see bad things arising.   The crowd crown him king; which the Romans would ban.
I see blood and destruction,   Our elimination because of one man.   Blood and destruction becaus

ALL (inside)
Because, because, because of one man.

CAIAPHAS
Our elimination because of one man.

ALL (inside)
```

```
Because, because, because of one, 'cause of one, 'cause of one man.

PRIEST THREE
What then to do about this Jesus-mania?

ANNAS
How do we deal with a carpenter king?

PRIESTS
Where do we start with a man who is bigger   Than John was when John did his baptism th

CAIAPHAS
Fools, you have no perception!   The stakes we are gambling are frighteningly high!
We must crush him completely,   So like John before him, this Jesus must die.   For the
```

This Jesus Must Die by Andrew Lloyd Webber

7.1.3 Codes

Chapter 8

The `rock` R package

The `rock` R package implements the ROCK standard for qualitative data analysis. It is an extension to R, a program that was originally a statistical programming language. R is not only open source, but also has a flexible infastructure allowing easy extension with user-contributed packages. Therefore, R is quickly becoming a multipurpose scientific toolkit, and one of its tools is the `rock` package.

When using R, most people use RStudio, a so-called integrated development environment. It has many features that make using R much more userfriendly and efficient. In this book, where we refer to using R, we actually mean using R through RStudio. Both R and RStudio are Free/Libre Open Source Software (FLOSS) solutions. This means that they are free to download and install in perpetuity.

8.1 Downloading and installing R and RStudio

Because RStudio makes using R considerably more userfriendly (and pretty), in this book, we will always use R through RStudio. Therefore, throughout this book, when we refer to R, we actually mean using R through RStudio.

R can be downloaded from https://cloud.r-project.org/:[1] click the "Download R for …" link that matches your operating system, and follow the instructions to download the right version. You don't have to start R - it just needs to be installed on your system. RStudio will normally find it on its own.

RStudio can be downloaded from https://www.rstudio.com/products/rstudio/download/. Once it is installed, you can start it, in which case you should see something similar to what is shown in Figure 8.1.[2]

[1] Yes, that page looks a bit outdated.

[2] It is easy to change RStudio's appearance; simply open the options dialog by opening the

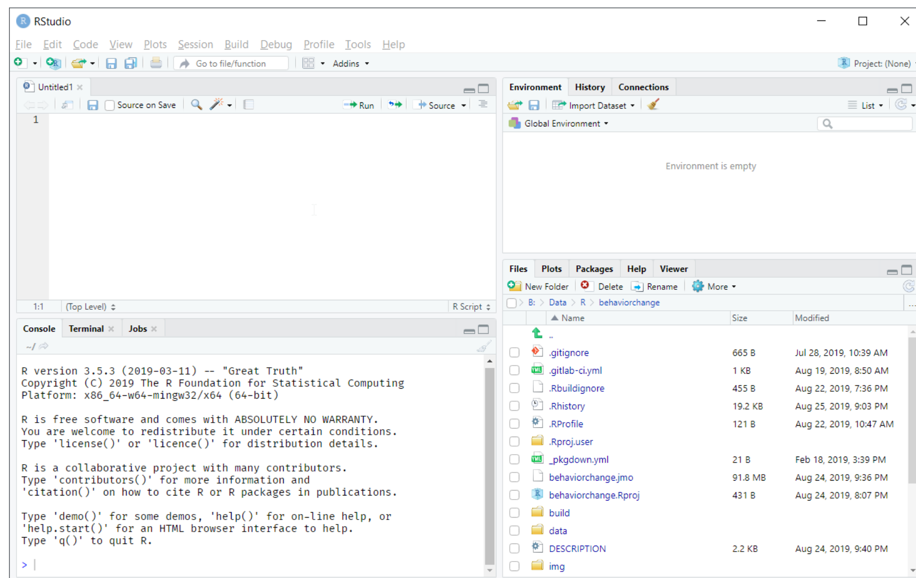

Figure 8.1: The RStudio integrated development interface (IDE).

R itself lives in the bottom-left pane, the console. Here, you can interact directly with R. You can open R scripts in the top-left pane: these are text files with the commands you want R to execute. The top-right pane contains the Environment tab, which shows all loaded datasets and variables; the History tab, which shows the commands you used; and the Connections and Build tabs, which you will not need. The bottom-right pane contains a Files tab, showing files on your computer; a Plots tab, which shows plots you created; a Packages tab, which shows the packages you have installed; a Help tab, which shows help ages about specific functions; and a Viewer tab, which can show HTML content that was generated in R.

8.2 Downloading and installing the `rock` package

The `rock` package can be installed by going to the console (bottom-left tab) and typing:

```
install.packages("rock");
```

Tools menu and then selecting the Global Options; in section Appearance, the theme can be selected.

This will connect to the Comprehensive R Archive Network (CRAN) and download and install the `rock` package. If you feel adventureous, you can instead install the one of the two development versions. One is the most current production version, and the other is the development ('dev') version. The most current production version will generally be as stable as versions on CRAN, and will contain more features. This version will contain all features discussed in this book. The dev version contains work on new features. This also means, however, that it may contain bugs.

To conveniently install the most recent production and dev versions, another package exists called `remotes`. You can install this using this command:

```
install.packages("remotes");
```

Then, to install the most up-to-date production version, use:

```
remotes::install_gitlab("r-packages/rock");
```

And to install the current dev version, use:

```
remotes::install_gitlab("r-packages/rock@dev");
```

More information about the `rock` package can be found at its so-called pkgdown website, which is located at http://r-packages.gitlab.io/rock.

8.3 Functions in the rock package

8.3.1 clean_source and clean_sources

Sometimes, sources are a bit messy.[3] In such cases, it can be efficient to preprocess them and perform some search and replace actions. This can be done for one or multiple source files using `clean_source` (for one file) and `clean_sources` (for multiple files; it basically just calls `clean_transcript` for multiple files).

For example, a researcher will often want every sentence, as transcribed, to be on its own line (as lines correspond to utterances). In fact, this is the basic function of the `clean_source` function: by default, if used without other arguments, they try to (more or less smartly) split a transcript such that each transcribed

[3] Well, they are messy more often than not, unfortunately.

sentence (as marked by a period (`.`), a question mark (`?`), an exclamation mark (`!`), or an ellipsis (`…`)) ends up on its own line. Before doing this, `clean_source` replaces all occurrences of exactly consecutive periods (`..`) with one period, all occurrences of four or more consecutive periods with three periods, and all occurrences of three or more newlines (`\n`) with two newlines.

But this function can also be used to perform additonal (or other) replacements. For example, imagine that a transcriber used a dash at the beginning of a line, followed by a space, to indicate when a person starts talking, like this:

```
- Something said by one speaker
- Something said by another speaker
```

To easily group all utterances by the same person together, it would be convenient if this was expressed in the source file in a way that fits with ROCK's conventions. That sequence of characters (actually a newline character (`\n`) followed by a dash (`-`) followed by a whitespace character (`\s`)) can be converted into section break '`---turn-of-talk---`' with this command:

```
rock::clean_source(input = "
- Something said by one speaker
- Something said by another speaker
",
                   replacementsPre=list(c("\\n-\\s", "\n---turn-of-talk---\n")));
```

This will change those that bit of transcript into:

```
---turn-of-talk---
Something said by one speaker
---turn-of-talk---
Something said by another speaker
```

(You can copy-paste the command above into R and test this, assuming you have the `rock` package installed. Note that by default, R doesn't print newline characters as newline characters. To show newline characters as newlines, wrap the command in the `cat` command.)

To also maintain the default replacements, more can be added by specifying them in argument `extraReplacements` instead of `replacementsPre` (or `replacementsPost`). For `clean_source`, as the first argument (`input`), either a character vector (like in the example above) or a path to a file can be specified, in which case the files contents will be read. If the second argument (`outputFile`) is specified, the result is saved to that file; if not, it is returned (and printed by R).

8.3.1.1 A word of caution

If you use this function to clean one or more transcripts, make sure that whenever you edit the `outputFile`, you save it under another name! Otherwise, rerunning the script to clean the transcripts will overwrite your edits. By default, the `rock` option "`preventOverwriting`" is set to `TRUE`, so by default, if a file already exists on disk, it is never overwritten. You can change this behavior for one function by specifying `preventOverwriting=FALSE` as a function argument. You can also change this for all functions by changing the option, with the following command:

```
rock::opts$set(preventOverwriting=FALSE);
```

Chapter 9

The iROCK interface

9.1 Background

iROCK is an interface for coding sources that consists of an application built in HTML, CSS and Javascript. This means that it does not store any information on a server, which has as a benefit that its use is necessarily GDPR compliant. This means that even if you have not anonymized your sources, you can still comfortably use iROCK, resting assured that you cannot violate the GDPR by doing so.

iROCK is available at https://sci-ops.gitlab.io/irock/ - this is a version hosted by GitLab. You can also download all files to your local PC, and run it from there. To do that, simply visit the repository at https://gitlab.com/sci-ops/irock and click the download button at the right-hand side (next to the "History" and "Find file" buttons). You can download the archive, unpack it on your PC, store the files somewhere, and then double-click the "index.html" file in the "iROCK" directory.

Note that iROCK is Free/Libre Open Source Software (FLOSS), which means that you can inspect the source code, and add functionality, if desired (see the repository at https://gitlab.com/sci-ops/irock).

9.2 Using iROCK

Chapter 10

A ROCK workflow

This chapter describes, or perhaps more accurately, prescribes, a recommended workflow to use when working with the Reproducible Open Coding Kit (ROCK). Consistent with the aims of the ROCK, this workflow is designed to optimize transparency and reproducibility.

This ROCK workflow leans heavily on the `rock` R package and the iROCK interface, but of course, any of the actions described can be implemented in other ways as well.

10.1 A basic ROCK workflow

In qualitative studies where the collected data are already clean and only one coder is used, the workflow is very simple. This workflow is explained first: both because it is all some readers will need, and because it will give an impression of the core elements to those readers who will need the more advanced functionality.

10.1.1 A bit of project management

All projects require some minimal management. When working with computers, this management concerns, among other things, how to organise the related files. When working with qualitative data, there will usually be two types of files: files with participants' personal data, and files without personal data. The latter can safely (and relatively indiscriminately) be synchronized with other computers, while the former requires more care. It is important to have clear procedures for anonymizing data and for making sure the right files are backed up in the right way. For the anonymized transcripts, analysis scripts, and other scientific

materials, we recommend using a version control system such as Git. However, file and project management go beyond the scope of this book.

One trick that does fall within the scope of this book is the functionality of R Projects. An R Project is a collection of related files that sit in a directory. In RStudio, you can create an R Project though the New Project menu option in the File menu. You can either create a project in an existing directory; create a fresh project in a new directory; or connect to a version control system such as Git to clone an existing project to your computer.

Once you created a project, RStudio will remember which files you had opened in your script file panel (top-left). It also conveniently shows the files and directories in your project in the Files tab of the bottom-right panel, and if you use Git, allows you to synchronize your changes using the Git tab in the top-right panel. Most importantly for our present purposes, using an R Project allows easy access to your files and directories regardless of where on the PC they are located. Therefore, start by creating an R Project. One you created it, to continue working on this project, simply open the associated file (with the "`.Rproj`" extension).

10.1.2 Source collection and preparation

We will assume that the data are organised into one or more sources, which are plain-text files where each smallest codable element is placed on a separate line (i.e. each utterance is separated by newline characters). Note that the smallest codable element is not the smallest element that could be coded in theory, but instead represents the smallest element that the researchers are interested in coding. If the data are not yet organised like this, you will first need to clean and organise them; please refer to those sections in the extensive workflow.

In each source, add case identifiers. Case identifiers indicate which utterances belong to which case. Cases are usually data providers, such as participants or organisation. Case identifiers can be, for example, numbers, letters, codes, or pseudonyms. Case identifiers are added to the ROCK like this:

```
[[cid=1]]
```

The "`1`" is the identifier itself; this could also be, for example, "`F23b`", "`Alice`", or "`F`", depending on the system used to identify the sources. This case identifier is used as an efficient way to attach attributes to the relevant utterances. By default, case identifiers are so-called "persistent identifiers", which means that once they have been specified on a line, all subsequent utterances in that source will be considered to have been coded with that case identifier, until a new case identifier is encountered.

10.1.3 Attribute specification

Cases function as a method for attaching attributes to utterances. For example, if sources are transcripts from interviews, cases can be the participants that were interviewed. This enables attaching participants' attributes to utterances, such as their age, gender, or area of residence (of course, constructs measured by measurement instruments such as questionnaires can also be used, such as scores on extraversion or self-efficacy).

These attributes can be defined in so-called YAML fragments that are delimited with three dashes ("`---`"), such as this fragment:

```
---
ROCK_attributes:
  -
    caseId: 1
    sex: female
    age: 50s
---
```

Such fragments can be placed in sources (usually at the beginning or the end, although the `rock` doesn't care), but it may make more sense to combine them all in one separate file. To combine the attribute specifications for multiple cases, simple repeat the same information, including the dash:

```
---
ROCK_attributes:
  -
    caseId: 1
    sex: female
    age: 50s
  -
    caseId: 2
    sex: male
    age: 30s
---
```

Note that when working with YAML, indentation is very important. The word "`ROCK_attributes`" must always start at the beginning of the line; the dash that indicates that attributes for a new case start must always be indented *exactly* two spaces; and the `caseId` and attribute names and their values must always be indented exactly fout spaces. If this is violated, the `yaml` package will throw a "Parser error".

10.1.4 Coding

To code, one can use any text editor able to edit plain text files, such as Notepad, TextEdit, Notepad++, Vim, or BBEdit. However, in this workflow, we will work with iROCK, an interface optimized for working with the ROCK. iROCK is a simple, userfriendly and GDPR-compliant interface for coding sources. The iROCK interface is discussed in detail in Chapter 9. To load it, visit https://sci-ops.gitlab.io/irock/ in your browser (or follow an alternative method as explained in Chapter 9).

In iROCK, import a source by clicking the red rectangle marked "Sources". Then, if you want to engage in inductive coding, you can simply start coding. At the right-hand side, you can create codes, and once created, a code can be dragged and dropped from the list onto the utterance you want to apply it to. Click an applied code (in the source) to remove it again. To indicate that a code falls under another code, use the ROCK hierarchy marker: "`>`", e.g. "`parentCode>childCode`".

If you want to use deductive coding, import your codes by clicking the red rectangle marked "Codes". You can import a plain text file: every line of the file will be imported as one code. If you already coded one or more sources, you can use the `rock` R package to efficiently create this list from the used codes. First, use the `rock::parse_sources()` function to import the sources. This is explained more in detail in the next section, but looks roughly like this:

```
parsedSources <-
  rock::parse_sources(input = here::here("data", "coded"));
```

Then, use the `rock::export_codes_to_txt()` function to export the codes. For example, to export all codes, including their so-called "paths" (their explicitly specified position in the hierarchy using the ROCK hierarchy marker, "`>`", use:

```
rock::export_codes_to_txt(input = parsedSources,
                          output = here::here("codes", "exported-codes.txt"));
```

Alternatively, you can specify that you only want the "leaves" of the code tree, in other words, you don't want to select codes that have child codes, using "`leavesOnly=TRUE`", and you can specify that you don't want to include the path, but instead only want the codes themselves, using "`includePath=FALSE`":

```
rock::export_codes_to_txt(input = parsedSources,
                          output = here::here("codes", "exported-codes.txt")
                          leavesOnly=TRUE,
                          includePath=FALSE);
```

To only select codes with a given parent, specify a value for "`onlyChildrenOf`", and to only select codes that match a given regular expression, specify it as "`regex`".

The `rock::export_codes_to_txt()` function will write a plain-text file to disk that can then be directly imported into the iROCK interface.

10.1.5 Analysing the results

10.1.6 (Re)Coding

10.1.7 Publishing the project

10.2 An extensive ROCK workflow

Below follows a more extensive workflow description. For the sake of completeness, this also includes common tasks in qualitative research that are unrelated to the ROCK. It is, after all, the extensive workflow.

10.2.1 Planning

10.2.1.1 Research questions

Like with any study, it is vital to have a clear research question. The research question determines which methods can be used. For example, not all research questions can be studies using qualitative research (and not all research questions can be studied using quantitative research). Typical research questions that require quantitative research are questions about associations or causality. Typical research questions that require qualitative research are questions about experiences, narratives, and contents of constructs. And if strong conclusions are desired, research syntheses are required, rather than a single study.

The research question is important because once it is clear, the required analysis approach can be determined, which allows determining the required coding approach, which allows determining how to collect the data. Because the research question is so fundamentally connected to all other aspects of the study,

one approach to clarify your research question is to think about what potential answers may look like.

One important implication of a research question is whether the coding will be deductive or inductive. Deductive coding uses predefined codes, and inductive coding uses codes created during the coding. Deductive ('closed') coding allows more transparency and reproducibility and enables procedures to minimize bias. The price the researcher pays for these advantages is less flexibility during coding. Inductive ('open') coding allows identifying patterns and categories in the data that could not be anticipated a priori. In that sense, inductive coding plays to the strengths of qualitative research: it imposes no constraints on analysis. The downside is that it cannot use tools to manage subjectivity; such tools inevitably impose structure and as such decrease the purely inductive nature of the coding.

In practice, coding is often a mix. Researchers rarely start collecting data in a field where no relevant theory exists. Therefore, those theories often shape the coding, in which case making that explicit by prespecifying a deductive coding structure aids transparency. This coding structure can then form the basis for the coding process, while still allowing coders to add more coding trees to the coding structure's root (for codes that cannot be captured by the prespecified codes and their definitions) and to add more codes as 'children' of prespecified codes.

10.2.1.2 Coding instructions

Unless there are no preconceived ideas about the coding process whatsoever, the coding process will inevitably require matching utterances to some definition. It is therefore important to have the relevant definitions available, in sufficiently clear and explicit formulations, as well as coding instructions. Coding instructions are important for decreasing undesirable, invisible subjectivity and bias and in the coding process. They explicitly capture the characteristics that a piece of data must satisfy to code it with a given code, including explicit guidelines for resolving edge cases (see section 12.2.3).

It is usually desirable to make sure the coding instructions are consistent over studies. Ideally, the exacty same coding instructions are used in all studies in a lab, department, or even institution (also see section 12.1.1).

If the qualitative study concerns humans, and therefore, the codes relate to constructs (e.g. psychological, sociological, or anthropological constructs), using a decentralized construct taxonomy (DCT) supports clear definitions that can consistently be applied over multiple studies. These are introduced in Chapter 12. For studies with humans, it is therefore strongly recommended to not proceed until a set of DCT specifications has been produced and the coding instructions have been generated from those DCTs.

If coding another type of content, it is still important to develop clear, unequivocal coding instructions before proceeding. The coding instructions should ideally be good enough to render individual coders more or less interchangeable.

10.2.1.3 A note on data management

- encryption
- password management

10.2.2 Data collection

The operational aspects of data collection vary with the type of data that are collected. We will cover two scenarios here: recording audio from individual interviews, group interviews or focus groups, and collecting existing data such as social media posts or archive materials.

10.2.2.1 Recording audio

- 2 recorders (one backup; redundancy)

10.2.2.1.1 Transcription into sources

- with group interviews, pay attention to distinguishing group members; make sure they introduce themselves

10.2.2.2 Collecting existing data

10.2.3 Source cleaning

Once a dataset has been collected, it is usually necessary to perform some cleaning. In a ROCK workflow, this cleaning includes rudimentary segmentation into *utterances* (see Chapter 6). In the ROCK specification, utterances are separated by a newline character: in other words, every utterance is on its own line. Utterances are the smallest codable unit, and as such, this is not a trivial step. The logic underlying the convention that utterances are separated by newline characters is that although sentences are themselves often hard to fully understand without context, at least they are often self-containing, whereas parts of a sentence are rarely comphehensible on their own.

To clean sources, we will use the `rock` package function `rock::clean_sources()` (for more details, see Section 8.3.1). We assume here that the data are located in a directory called `data` in your Project directory (see Section 10.1.1). In this

directory, we assume that the raw sources (i.e. the raw transcripts) are located in the subdirectory called `raw`. In addition, we will write the cleaned sources to another subdirectory of the `data` directory called `cleaned`.

The following command reads all files in the `data/raw` directory in your Project directory, applies the default cleaning operations (e.g. add a newline character following every sentence ending), and wrotes the cleaned sources to a directory called `data/cleaned` in your Project directory:

```
rock::clean_sources(input = here::here("data", "raw"),
                    output = here::here("data", "cleaned"));
```

Note that if you only want to read files that have a certain extension, such as `.txt` or `.rock`, you can specify this by specifying a regular expression to match against filenames as argument `filenameRegex`. For example, to only read both `.txt` files and `.rock` files, you would pass the regular expression `"\\.txt$|\\.txt$"`, to only read `.txt` files, the regular expression `"\\.txt$"`, and to only read files with the `.rock` extension, you would use this command:

```
rock::clean_sources(input = here::here("data", "raw"),
                    output = here::here("data", "cleaned"),
                    filenameRegex = "\\.rock$");
```

The `rock::clean_sources()` function has many other functions, which you can read about by requesting the manual page. You can do this by typing `?rock::clean_sources` in the R console (the bottom-left panel in RStudio).

10.2.4 Prepending utterance identifiers

10.2.5 Coding and segmentation

10.2.5.1 Automating coding and segmentation

In a sense, cleaning codes already applies some automatic segmentation: after all, the default

`rock::code_sources()`

10.2.6 Manual coding and segmentation

To manually code and segment, any software that can open and save plain text files can be used. To achieve this software independence was, after all, one of

the reasons the ROCK was developed. Most operating systems come with basic plain text editors (e.g. notepad on Windows; TextEdit on MacOS; and vim with most Unix systems), and many Free/Libre and Open Source Software (FLOSS) alternatives exist, such as the powerful Notepad++ for Windows and BBEdit for MacOS.

These editors can be used to open sources and add codes and section breaks. Many allow the creation of plugins that can further facilitate this, and in additon, plain text editors can be used in tandem with spreadsheet applications such as the FLOSS LibreOffice Calc. This allows having neatly a organised coding structure in a spreadsheet, which can then easily be copied to the clipboard and pasted in a source in a text editor. By using key combinations such as Alt-Tab (Windows) or Command-Tab (MacOS) the coder can quickly switch between the source and the code overview.

In addition, iROCK can be used, a rudimentary graphical user interface that simply allows appending predefined codes to utterances and inserting section breaks (for segmentation). More details are available in Chapter @(irock).

Finally, the ROCK standard enables development of a variety of other tools for specific use cases.

10.2.7 Inspecting coder consistency

10.2.8 Merging sources

`rock::merge_sources`

10.2.9 Viewing source fragments by code

`rock::collect_coded_fragments`

10.2.10 Recoding

Sometimes, it is desirable to change codes. For example, a set of codes that was initially obtained through inductive coding may, upon inspection, have a hierarchical structure. When using the ROCK, ideally, the originally coded sources remain in their original state so as to enable scrutiny of the coding process. Instead, the recoding is applied using a command that opens the sources, makes the changes, and then writes them to disk again.

At present, the `rock` R package has four functions to code and recode.

`rock::search_and_replace_in_sources`

10.2.11 Generating HTML versions

```
rock::export_to_html
```

10.2.12 Exporting the coding spreadsheet

Chapter 11

Using the ROCK for Epistemic Network Analysis

Epistemic Network Analysis (ENA) is a convention and software housed within the larger methodological framework of Quantitative Ethnography (Shaffer, 2017). ENA was originally developed for *modelling and comparing the structure of connections among various elements in a data set*. In case of qualitative data (narratives), connections in the data are generated from co-occurrences of codes within segments; these co-occurrences are visualized in a network. ENA is a useful tool if one is working with (*a large number of*) *variables in a single system* and can benefit from modelling complex structures in search of patterns in the data. To read more about ENA and Quantitative Ethnography, see Shaffer (2017).

Below we offer guidelines for using the ROCK to prepare data for use in ENA. *the ROCK provides help in the process of preparing and performing coding and segmentation, merging coded documents from multiple raters, and creating the qualitative data table (CSV file) necessary for making networks.* the ROCK will aid you the most if you are working with continuous narratives (e.g. semi-structured interviews, from more details see below: Planning Segmentation) and performing manual coding (as opposed to automated coding, for more see NCoder).

The following is a *step-by-step account of how to employ the ROCK in creating networks from your data*, but these steps are not in strict order as work processes are highly dependent on the project in question. the ROCK conventions will be illustrated with a worked example. In general, the guidelines will be structured as follows: theoretical considerations presented in separate sections, instructions for use in the ROCK, and the corresponding information from our worked example under each sub-section.

11.1 Starting point

These guidelines assume that the researcher is familiar with the basic tenets of Quantitative Ethnography (QE) and ENA (although important terms will be clarified). The starting point of the guidelines also presupposes that the researcher is working with an *anonymized database of raw, qualitative data* that was collected in a systematic manner during a project where the research question, sub-questions, methods, and sampling have all been established. The guidelines do not provide advice on research design.

As a preliminary step, please install the required software as explained in sections 8.1 and 8.2 in Chapter 8.

Our example

Very succinctly, we were interested in modelling cognitive and behavioral patterns in patient decision-making processes regarding choice of therapy, i.e. for a certain diagnosis, what sources of information are considered, what specific decisions are made during the patient journey, and what conceptual framework the patient has concerning illness causation. We wanted to know what cognitive patterns underlie the decision to use different types of medicine (conventional and non-conventional). To read more about the research questions and design (methods, sampling, etc.), please see the methods section of @zorgo_patient_2018 and @zorgo_qualitative_2018, and the methodological considerations in @zorgo_epistemic_2019.

11.2 Planning coding

11.2.1 What is a code?

One aim of QE is to localize patterns within a community of practice (culture or subculture), which may be referred to as "Discourse" (capitalization intended). To do this, the researcher gathers "discourse", that is, data from the scrutinized community, such as transcripts from interviews or focus groups, field notes from observations, etc. "Codes" (capitalization intended) can be defined as culturally relevant and meaningful aspects of a Discourse, the elements that the researcher wishes to address in the process of analysis; these elements will constitute the nodes of the network model. Finally, "codes" are manifestations of these elements that one identifies in their data, i.e. evidence for Codes within the narratives. For a more elaborate description of the QE framework see: xxx.

11.2.2 Types of coding

As with coding qualitative data in general, there are several decisions one needs to make. Should I code with a predetermined set of codes (deductive coding) or should I allow for the codes to emerge from the narratives as I progress in analysis (inductive coding)? Both manners of coding have their advantages and disadvantages (for more details see Smith and Osborn (2008), Denzin and Lincoln (2000) and Babbie (2007)), the ROCK enables the researcher to employ one or the other, or even both.

Another consideration, with both deductive and inductive coding, is whether the set of codes should be hierarchical or not. A hierarchy would imply that some codes constitute part of other, more abstract codes, such as the parent code `fruit` containing the child code `banana`. If codes are not arranged hierarchically, they would still constitute a single analytical system in light of the research question, but cannot be conceptualized as containing one another, such as `fruit`, `dairy`, `meat`, `grains`, and `vegetables`. Again, the ROCK supports both hierarchical and non-hierarchical constructs.

11.2.3 How to represent codes in the ROCK

The general format for representing codes in the ROCK is placing the code name (e.g. fruit) in between two square brackets, for example: `[[fruit]]`. If the code name contains two or more words, we suggest using an underscore to separate them, e.g.: `[[exotic_fruit]]`; we also suggest keeping the code names concise but informative.

Both hierarchical and non-hierarchical inductive coding can be generated in the above format with the help of the Interface for the ROCK (the iROCK) platform (see below: Coding and Segmentation).

Both inductive and deductive hierarchical codes necessitate a greater-than sign to signal their place in the overall structure, for example: `[[fruit>banana]]` connotes a two-level hierarchy; `[[fruit>exotic>banana]]` connotes three levels.

Hierarchical and non-hierarchical deductive codes need to be specified before coding begins and listed in a file designated specifically for this. Deductive codes may be structured in several code clusters or trees (for more detail see: xxx). Non-hierarchical deductive coding should follow the above format for the ROCK codes, for example, the codes in the previous example would look like this:

```
[[fruit]]
[[dairy]]
[[meat]]
[[grains]]
```

```
[[vegetables]]
```

Hierarchical and non-hierarchical deductive codes are essentially a list of codes, in the above format, placed into a separate file, preferably with a `.rock` extension (see previous section: General background and introduction: `.rock` file format).

Our example

In our case, *Discourse* refers to patterns in cognition and behavior among patients using biomedicine only, and patients using non-conventional medicine to treat their illness(es). We can also consider the individuals in these two groups as all belonging to larger groups delimited by their “primary diagnosis”. In our case, we began by choosing four diagnoses (D1-4): Diabetes (D1), Musculoskeletal diseases (D2), Digestive diseases (D3), and Nervous system diseases (D4). Thus, every individual in our study belongs to a group indicating their primary diagnosis (D1-4) and their choice of therapy. Individuals can be grouped further based on other characteristics (for more on this subject, see below: Designating Attributes).

Our *discourse* consists of transcripts from semi-structured interviews conducted with patients belonging to one of the four types of diagnosis groups and representing different choices of therapy (for more on the latter, see below: Designating Attributes).

The *Codes* we were interested in encompass the three main areas of interest within the project: sources of information (epistemology), concepts of illness causation (ontology), and decisions in the patient journey (behavior). We employed both deductive and inductive coding. We coded the above three areas of interest with a predetermined set of hierarchically organized codes, on three levels of abstraction, comprising 52 low-level *codes* in total. The complete code tree can be accessed here: xxx. Our inductive coding only concerned illnesses. As interviewees also spoke about comorbidities during the interview, we found it important to distinguish among primary diagnosis and other, specific comorbidities the patient is referring to within the narrative.

11.3 Planning Segmentation

11.3.1 What is discourse segmentation?

Segmentation, according to QE, is the process of dividing data up into sensible structures, meaningful parts. There are different levels and modes in which one can segment narratives; these segments will be important in the creation

of a network because connections are formed based on the number of code co-occurrences within the designated segments.

Following the QE framework, there are three important levels of segmentation to consider: the smallest unit of segmentation (utterance), a middle level (stanza), and a high level (unit). At this point we will address the first two of these, units will be dealt with later (see below: Creating Networks).

An utterance is the smallest entity of analysis in a narrative. This can be one sentence (e.g. a semi-structured interview's utterances are sentences articulated by the interviewee) or more than one sentence (e.g. one remark made by one participant in a focus group). An utterance can also be one line or one entry in a field journal, for example. In any case, coding will occur at this level.

Albeit coding occurs on the level of utterances, co-occurrences are computed based on a higher level of segmentation, the stanza. A stanza is a level of discourse structure composed of one or more utterances that occur in close proximity and discuss the same topic (i.e. recent temporal context). Stanza size reflects how much content the researchers consider indicative of psychological proximity. Researchers who are only interested in tightly connected concepts may prefer shorter stanzas, however, if the research topic concerns broader, more complex issues, researchers may want to define larger stanza sizes. Stanza size crucially determines analysis results, thus the rules for segmentation should not be arbitrary and should be made transparent. In order to explore various versions of segmentation, more than one stanza-type can be employed (i.e.: multiple ways of defining stanza length and multiple identifiers). Furthermore, stanzas constitute merely one way of segmenting data on the middle level, one may want to utilize many forms of section breaks (for details see: Cognitive Interviews).

11.3.2 Continuous and discontinuous narratives

Qualitative data can come in many forms and have varying characteristics, an anthropological field journal presents us with very different text compared to a focus group or an interview, for example. Thus far ENA has mainly been used for discontinuous data, namely, teams of people performing tasks in a common virtual reality or performing virtual tasks in a shared physical reality (see: Ruis et al. (2018)). Similar to focus group situations, these studies worked with data that was supplied by several participants and on several, discrete occasions. Naturally occurring "turns of talk" among participants provide for excellent segmentation, for example, students discussing how to accomplish a common task in a chatroom (Bressler et al., 2019).

Continuous narratives are distinguished from discontinuous narratives by the lack of naturally occurring possibilities for segmentation; such text may originate from the transcript of a semi-structured interview or an audio dairy. Because there are no "turns of talk" (or they are between interviewer and interviewee and

yield little contextual information), and the whole text may be intricately connected internally, demarcating stanzas becomes a challenge. Similar problems may occur with the smallest unit of analysis, especially if it is defined as "one sentence". The verbatim transcription of spoken speech comes with inherent subjectivities; as a sentence in speech may persist across vast reaches, the transcriber makes many judgement calls in punctuation. Yet, as co-occurrences are computed based on stanza, length of utterance is not decisive in this particular case.

11.3.3 How to represent segmentation in the ROCK

Utterances are represented in the ROCK by something called an "utterance identifier" (UID). It is one line in the ROCK file (a line being defined as zero or more characters ending with a line ending). When the ROCK reads a certain file containing text and utterance identifiers, it splits each file at the line endings (with newline characters). This parsing is necessary to perform the coding of each utterance in a file and be able to work with that information later on. An example of a UID is: [[uid=73ntnx8n]], each utterance receives a unique identifier. Utterances may be grouped together with the aid of section breaks, one of which is the stanza; the ROCK uses this particular format: <<stanza-delimiter>>, where the name "stanza delimiter" may be changed according to the segmentation needs of the project.

Our example

In our project, an *utterance* is defined as one sentence. Each sentence constitutes one line in the ROCK; each utterance/line receives a unique identifier. Regarding the *stanza*, we employed the generic definition above, but we had three different raters perform segmentation autonomously based on their judgement of psychological proximity. Their three identifiers were: <<stanza-delimiter-low>>, <<stanza-delimiter-mid>>, and <<stanza-delimiter-high>>. The names indicate the level of knowledge each rater had concerning the scrutinized research topic; "low" signified a (naïve) rater not connected to the research project, only privy to the interview transcripts. "Mid" was employed by a research assistant with a significant amount of prior knowledge on research objectives and codes, while the "high" delimiter was used by the principle investigator. Thus, we had three different stanza-types for all interview transcripts in order to explore which stanza-type creates the best models.

11.4 Designating Source and Cases

11.4.1 What is a source and how is it represented in the ROCK?

A source is a file with content to code (or coded content); it can contain the transcript of an interview or a focus group discussion, or even a list of twitter posts. Sources comprise one or more utterances from one or more participants of a study. Sources should be plain-text files and can bear any name, although they should be kept concise, as these will be displayed in the ENA interface later on. Information relevant to the study can also be displayed in the name, such as: “5-female-30s”, indicating this is the fifth interview and it is with a female participant in her 30s.

11.4.2 What is a case and how is it represented in the ROCK?

A case signifies a participant, a provider of data within a study. This can be a person, a family, an organization, or any other unit of research. In case of individual interviews, the source and the case may be identical, but it is important to distinguish between these as one source can contain data from many cases. For example, a focus group transcript constitutes a source, while the six participants connote separate cases within the source. Each case receives a unique identifier (case id, CID) and is represented in the ROCK with two square brackets and a designated name, for example: `[[cid=alice]]`. Naturally, in anonymized studies it is preferred to have an alias of some sort, it can even be a number.

11.5 Designating Attributes

11.5.1 What is an attribute?

Cases can be supplemented with characteristics or variables; we refer to these as “attributes” (ENA term: metadata). For each participant you may want to collect additional data, such as demographic variables, or even conduct a survey in addition to an interview, for example, and record the answers respondents provide. You may want to register aspects, such as the date the interview was conducted, the researcher who led the focus group, or the sequence audio diaries were recorded in. Attributes essentially allow you to group participants in various ways (thus creating different networks) and enable other types of analysis through flexibly changing the sets of data you want to see a network for (i.e. conditional exchangeability), for more see below: Creating Networks.

11.5.2 How to record attributes in the ROCK

There are two main ways you can record attributes for cases in your study using the ROCK. One is to place this information into the source directly. For example, you open the plain-text file of your semi-structured interview and enter the attributes above or below narrative. The other option is to create a separate .rock file containing the attributes of all participants. In either case, the format for entering attribute-related information is the following:

```
---
ROCK_attributes:
  -
    caseId: 1
    sex: female
    age: 50s
  -
    caseId: 2
    sex: male
    age: 30s
---
```

The above displays the aggregated version of recording attributes (illustrated with two cases). The list of entries begins with three dashes, followed by the attributes listed in the manner displayed, and ends with three dashes. In later phases of data preparation, the ROCK will read this information and assign it to the appropriate case.

Our example

For each interview we recorded the following *attributes*: interview date, interviewer ID, interviewee ID, interviewee sex, age, and level of education, diagnosis type (D1-4), specific illness, comorbidities, illness onset, time of diagnosis, and therapy choice (treatment type concerning primary diagnosis: biomedicine only, complementary use of non-conventional medicine, alternative use of non-conventional medicine). For complementary and alternative medicine (CAM) users we also registered type of CAM use, attendance in CAM-related courses, disclosure of CAM use to conventional physician and the employed CAM modalities. For users of solely biomedicine, reason for rejecting CAM was also coded (deductively).

To summarize, here is a list of terms we have discussed thus far and some examples for each term:

Term	Explanation	Example
Discourse	Patterns within a community of practice (culture or subculture)	E.g.: patterns in cognition and behavior among those using biomedicine only, and and those using non-conventional medicine to treat their illness(es)
discourse	Data from the scrutinized community	E.g.: transcripts of semi-structured interviews conducted with patients
Code	Culturally relevant and meaningful aspects of a Discourse	E.g.: sources of information, concepts of illness causation, and decisions in the patient journey
code	Manifestations of these elements that one identifies in their data, i.e. evidence for Codes	E.g.: hierarchical, three levels of abstraction, 52 low-level, e.g. the ROCK non-hierarchical code format, e.g.: `[[exotic_fruit]]` the ROCK hierarchical code format, e.g.: `[[fruit>banana]]`
Segmentation	The process of dividing data up into sensible structures, meaningful parts	• Utterance (e.g.: one sentence) • the ROCK utterance identifier (UID) format: `[[uid=73ntnx8n]]` • Stanza (e.g.: psychological proximity, recent temporal context) • the ROCK section break format, e.g.: `<<stanza-delimiter>>`
Discontinuous narratives	Text containing naturally occurring “turns of talk” among participants	E.g.: students discussing how to accomplish a common task in a chatroom
Continuous narratives	Text lacking naturally occurring possibilities for segmentation	Semi-structured interviews, audio diaries, etc.

11.6 Coding and Segmentation

11.6.1 How is coding performed with the ROCK?

Although manual coding can be performed within a qualitative data table in a spreadsheet (for more detail see: xxx), when conducting hermeneutic analysis with a high number of codes, this is unwieldly. For this reason, we developed the Interface for ROCK (iROCK), an online user platform, consisting of a file that combines HTML, CSS, and javascript to provide a rudimentary graphical user interface. Because iROCK is a standalone file, it does not need to be hosted on a server, which means that no data processing agreements are required (as per the GDPR). The iROCK interface allows raters to upload a source, a list of codes, and segmentation identifiers. Coding is performed by dragging and dropping codes upon utterances at the end of their line. Once coding is finished, the coded sources can be saved. There are a few preparatory steps you need to take before you can start coding your sources.

11.6.2 Creating code clusters or trees

Depending on your research design, the code structure, and the amount of codes you are working with, you may want to have separate code clusters or discrete, hierarchically organized trees. You may also want to assign different raters to specific code clusters/trees, or have several raters use the same code structure to perform autonomous coding (this allows for triangulation or even inter-rater reliability testing, for more on this subject see: xxx). In these instances, you end up with multiple coded versions of a source, for example, interview number 1 (cid=1) is coded by two raters (R1 and R2), so you end up with two coded versions of Case 1. In a situation where R1 and R2 are autonomously coding different sections of the whole code structure, they will need separate code trees or clusters. Thus, the preparation of code lists depends on how many ways the whole code structure is divided among raters: each code group (cluster or tree) should be listed in its own .rock file. Naturally, if your code structure is not divided up amongst raters then one, all-inclusive list suffices (even if that list is used by multiple raters).

11.6.3 Preparing sources for coding

In order to perform coding, ROCK needs to “clean” your sources, i.e. parse the plain-text files according to one sentence per line (or however utterance is defined). ROCK also needs to add UIDs to each line in order to be able to merge codes from various raters in a later phase of the process (see below: Merging Coded Sources). Hence, you will need four directories, all of which need to be retained: original-sources; cleaned-sources; sources-with-UIDs; coded-sources. Below are the steps for preparing your data for coding:

1) Copy raw sources into "`original-sources`" folder (plain-text)
2) Locate the R chunk called "`# Preparing and cleaning sources`" and run it. This will clean the sources and save them to the "`clean-sources`" directory (and convert them to .rock format).
3) Locate the R chunk called "`prepend-utterance-ids`" and run it. This will load the cleaned sources, prepend the UIDs, and save them to the "sources-with-uids" directory.

You can safely repeat these steps; they will not overwrite existing files. When the files have appeared in the sources-with-UIDs folder, they are ready for coding.

11.6.4 Using the iROCK interface

The iROCK platform can be found at: https://r-packages.gitlab.io/rock/iROCK/. After you arrive to the website, you will see a ribbon at the top; by clicking on "Sources" you can upload the file you wish to code, "Codes" and "Section Breaks" can also be uploaded here. (Thus, coding and segmentation can be conducted simultaneously.) Perform coding by dragging and dropping the appropriate codes from your uploaded list to the end of each utterance. When coding is complete, download the file to your computer and place it into the "coded-sources" folder.

Our example

We divided up our code structure into three main areas that were reflected in the research question and the complete code tree as well (three high-level codes): epistemology, ontology, and behavior. The low-level codes belonging to these three parent codes were given to three different raters, each of whom specialized in one specific code tree. The three raters performed coding separately; none of their codes overlapped. There was a fourth rater who inductively coded the illnesses present in narratives. Each rater downloaded their coded files to a shared folder housed by a secure cloud storage (we use GDPR-compliant Sync, available at: https://www.sync.com/); the four raters had their own folder where they dropped every source they coded. Subsequently, the coded sources were copied to the "coded-sources" folder where all sources received a separate directory, comprising four versions of the source (three versions of coding with parent code trees and one version of inductively coded illnesses). Segmentation was performed by two of the above four researchers and one naïve rater; each used a <> matching their level of expertise (low, mid, high). Thus, in the end, *we had five versions of a source: 3 coded with parent code trees (including segmentation; high-level), 1 coded with illnesses (including segmentation; mid-level), and 1 segmented by a naïve rater (low-level).* Because we had so many versions of one source, neither of which was complete on its own, we needed to merge these files (see below).

11.7 Merging Coded Sources (if necessary)

If the research design and protocol call for multiple raters coding all sources, sources need to be merged into a master document. This is necessary because the ENA interface (to be used for creating the networks) will require a Comma-Separated Values (CSV) file to be uploaded, which contains all sources, attributes, utterances, codes, and segmentation, together a referred to as a "qualitative data table" with rows and columns that are ontologically consistent (Shaffer, 2017). This master document can be produced with ROCK by locating the R chunk called "`# Merging sources`" and running it. Provided the attributes were listed in a separate `.rock` file, use the R chunk called "`# Reading merged sources`" to create the CSV file comprising the master document with attributes added. Merging coded sources may also be required if the project is later revisited with a different set of codes by the same researcher, or if many researchers are collaborating on the same project non-synchronously.

11.8 Creating Networks

The CSV file can be uploaded to the ENA interface (available at: http://www.epistemicnetwork.org/) or can be further processed with rENA (available at: https://cran.r-project.org/web/packages/rENA/index.html). A tutorial on how to apply the QE framework and employ ENA software can be found here: http://www.epistemicnetwork.org/resources/. You may benefit from reading ENA tutorials and worked examples in preliminary phases of your research, as there are other questions that need to be addressed that may, for example, influence planning discourse segmentation in your project.

Chapter 12

Using the ROCK for Decentralized Construct Taxonomies

12.1 Introduction to Decentralized Construct Taxonomies

When studying humans, one must deal with the somewhat challenging fact of life that one often does not study natural kinds. The objects of study are generally variables that are assumed to exist in people's psychology, usually called constructs. Those constructs are not assumed to exist as more or less modular, discrete entities (Peters and Crutzen, 2017). Instead, these constructs concern definitions that enable consistent measurement and consistent manipulation of certain aspects of the human psychology, without the pretense that the constructs are somehow clearly distinguished from other constructs.

As a consequence, data collection and analysis in research with humans differs fundamentally from data collection in sciences that do deal with natural kinds. Specifically regarding qualitative data, this lack of natural kinds further complicates the challenges that come with having humans code rich, messy data. Human perception and processing is flawed enough as it is. Without the existence of discrete, modular, objectively existing entities to code, the coding instructions become the only tangible foothold coders can rely on.

Therefore, being able to engage in the scientific endeavour with any degree of consistency over studies requires unequivocal communication about the constructs under study. However, many theories do not provide sufficiently explicit definitions of the described constructs. Instead, there is often much room for

interpretation: room that manifests as heterogeneity in constructs' definitions, operationalizations, and instructions for coding the constructs.

It has been argued that this heterogeneity is a feature, not a bug. … …

To facilitate unequivocal references to specific definitions of constructs, combined with coherent instructions for operationalisation and coding, Decentralized Construct Taxonomy specifications (DCTs) were developed. DCTs are simple plain text files in the YAML format that specify, for one or more constructs:

- A unique identifier for the construct, the Unique Construct Identifier (UCID);
- A human-readable label (title / name) for the construct (which doesn't need to be unique, as the identifier is already unique);
- An exact definition of the construct;
- Instructions for developing a measurement instrument to measure the construct;
- Instructions for coding measurement instruments as measurement instruments that measure this construct;
- Instructions for developing a manipulation to change the construct;
- Instructions for coding manipulations as manipulations that change this construct;
- Instructions for generating qualitative data pertaining to this construct;
- Instructions for identifying when qualitative data pertains to this construct and then coding it as such.

12.1.1 Consistency over studies

DCT specifications can easily be re-used in different studies, for example in all studies in the same lab, in the same faculty, or organisation.

12.2 Creating a DCT

12.2.1 Thinking about constructs

Creating a DCT requires knowing which construct you want to describe and what exactly the construct is and is not. This seems trivial - most psychologists rely on the assumption that they have sufficient tacit knowledge of the constructs they work with. Hwoever, because this knowledge never has to be made explicit, this assumption is never tested. Producing a DCT for a construct confronts one with ecactly how much one knows about a construct. Based on our experience, this is usually depressingly little.

The reason for this is that theories and the textbooks describing them usually do not provide clear definitions, either. In fact, that is one of the causes of the heterogeneity that exists. To a degree this is inevitable because constructs are not directly observable, and often do not represent natural kinds. But to a degree it can be remedied - by being very explicit about a construct's definition, by producing a DCT. Thus, while producing a DCT may not necessarily be easy, it is definitely worthwhile.

When creating DCTs, it is important to keep in mind that there are no objectively wrong or right "answers". After all, the constructs do not correspond to natural kinds. Various definitions can co-exist without any of them being wrong or right. In fact, since the constructs do not correspond to more or less discrete or modular entities anyway, one could argue that they are all 'wrong' (or are all 'right'). Given that at present, most constructs lack clear, explicit definitions, any explicitation is progress. And DCTs can always be updated or adjusted by updating their UCID. If you end up iterating through several versions, that's clear evidence that there was room for improvement in your original, implicit, definitions.

When creating a DCT, it doesn't matter where you start. If you have a pretty clear idea about the construct's definition, you start by making that explicit. But it's possible that while there are a number of measurement instruments for the construct (e.g. questionnaires), there is no clear definition available. In that case, you can start with the measurement instruments, too, and first complete the instruction for developing measurement instruments by deriving common principles from the measurement instruments you have.

In any case the process will be iterative. Eventually, you will complete at the the definition of the construct, and probably at least two of the instructions (either the instruction for developing measurement instruments and for coding measurement instruments; or for developing manipulations and for coding manipulations; or for eliciting ('developing') qualitative data and for coding qualitative data). As you complete these sections, you will probably need to update other sections to make sure everything stays coherent.

On the surface, producing a DCT just consists of putting stuff in words. After all, you just need to type in the construct's name, definition, and add the instructions that allow you (and others) to work with the construct. This can be done within an hour. Most time is not spent on specifying the DCT in a file, but on arricing at definitions and instructions that you and your colleagues agree on. However, that is time well-spent.

By discussing the constructs you work with and the varying definitions that everybody uses, you achieve consensus. If you don't manage to achieve consensus about a given construct, that's fine of course - simply create two DCTs for two different constructs. You can even give them the same name - as long as they have different identifiers (UCIDs). If after these discussions, all researchers and their supervised students within your lab use the DCTs you produced, all

research will be consistent. Of course, researchers without DCTs will often assume such consistency as well. And if they are right, the process of producing DCTs should be effortless. If the process proves more cumbersome, clearly it was necessary.

12.2.2 Creating a DCT file

To create a DCT file, you can use any software that can create plain text files, such as Notepad, Textedit, Notepad++, BBEdit, Vim, Nano, or the RStudio IDE. A DCT file contains one or more DCT specifications, delimited by a line containing only three dashes (“`---`”). This is an example of an extremely simple DCT specification:

```
---
dct:
  dctVersion: 0.1.0
  version: 1
  id: chair_75vl264q
  label: "Chair"
  definition:
    definition: "A piece of furniture designed to support a sitting human."
  measure_dev:
    instruction: ""
  measure_code:
    instruction: ""
  manipulate_dev:
    instruction: ""
  manipulate_code:
    instruction: ""
  aspect_dev:
    instruction: ""
  aspect_code:
    instruction: "Objects that have legs and a surface that was designed for humans to
  rel:
    id: furniture_75vl25k8
    type: "semantic_type_of"
---
```

This example only specifies the UCID, name (label), definition, and instructions for coding, as well as one relationship to another construct with UCID “`furniture_75vl25k8`” that this construct is apparently a type of. These relationships are parsed when the `rock` package reads a set of DCT specifications, and they are used to build a hierarchical tree of constructs (i.e. a deductive

coding structure). You could omit these relationships of course, if you will not need to collapse codes or fragments based on higher levels in the hierarchy.

12.2.3 Description of edge cases

Clear definitions are most valuable when edge cases are encountered. For example, most people will have little difficulty in identifying 'chairs' and agreeing whether an object is a chair even without first explicitly communicating about and calibrating the definitions they use. It is with edge cases such as seating furniture with one, two, or three legs, or furniture that seats two or three people, where unclear definitions become problematic.

For example, a definition of a chair could be "A piece of furniture designed to support a sitting human". In this case, a bicycle would fall under this definition, and in a qualitative study, would therefore be coded as a `[[chair]]`. This example is easily solved by updating the definition to "A piece of static furniture designed to support a sitting human". However, in this definition, a bar stool with one leg would also be coded as `[[chair]]`, which in this case might fall beyond the intended definition. Describing all specific edge cases explicitly in the definition may make the definition unwieldy.

Therefore, the specific instructions in a DCT normally discuss edge cases explicitly, referring the user to alternative codes where appropriate. For example, the coding instructions for coding a piece of qualitative data as `[[chair]]` could include the sentence "Note that furniture without back and arm support and having three legs or less should not be coded as `[[chair]]` but instead as `[[stool]]`.".

Thus, coding instructions are often most useful if they do not only describe the core of a construct, but if they pay special attention to the periphery of a construct's definition. Coding errors often concern ambiguity, and coding instructions should not add to this ambiguity.

12.3 Coding with DCTs

When coding with DCTs, you code slightly differently than when you code without DCTs. Regular codes are simply delimited by two square brackets, e.g. `[[chair]]`. However, if you use DCTs, you specify this in the code: `[[dct:chair_75vl264q]]`. You can still combine this with inductive coding, for example for indicating that an important subtype of chairs are the thrones: `[[dct:chair_75vl264q>throne]]`. Like normal inductive codes, you can keep on nesting such subcodes infinitely to indicate ever more precise subconstructs, if need be (although one level will usually suffice).

12.4 Analysing DCT-coded sources

Chapter 13

Using the ROCK for Cognitive Interviews

Content goes here

Chapter 14

References

Bibliography

Babbie, E. (2007). *The Practice of Social Research.* Wadsworth, Belmont.

Bressler, D., Bodzin, A., Eagan, B., and Tabatabai, S. (2019). Using epistemic network analysis to examine discourse and scientific practice during a collaborative game. *Journal of Science Education and Technology*, 28(5):553–566.

Denzin, N. and Lincoln, Y. (2000). *Handbook of Qualitative Research.* Sage Publications, Thousand Oaks.

Peters, G.-J. Y. and Crutzen, R. (2017). Pragmatic nihilism: how a Theory of Nothing can help health psychology progress. *Health Psychology Review*, 11(2).

Ruis, A. R., Rosser, A. A., Quandt-Walle, C., Nathwani, J. N., Shaffer, D. W., and Pugh, C. M. (2018). The hands and head of a surgeon: Modeling operative competency with multimodal epistemic network analysis. *American Journal of Surgery*, 216(5):835–840.

Shaffer, D. (2017). *Quantitative Ethnography.*

Smith, J. A. and Osborn, M. (2008). Interpretative phenomenological analysis. In Smith, J. A., editor, *Qualitative psychology: A practical guide to research methods*, pages 53–80. Sage Publications, London.