RE-Pract 2018 Survey: Coding Instructions

Introduction

Generally, there are two types of questions allowing free text answers (**free text questions**):

- questions expecting short free text answers (**short free text questions**; asking for a specific information item), and
- questions expecting long free text answers (**long free text questions**; asking for reasoning/elaboration).

The **coding procedures** to be followed depend on the type of question concerned, and they are detailed below. This document is accompanied by a **coding supplement**. The coding supplement summarizes the results of the preliminary algorithmic coding and spells out the rules used to produce these results.

Please direct your **questions and comments** to:

Corinna Coupette (corinna.coupette@campus.lmu.de)
Daniel Mendez (daniel.mendez@tum.de)

Caution: The csv files mentioned below are distributed with semicolons as separators and double quotes as quotechars so that they can be readily read by the standard Excel configuration in Europe, and performing the validation steps in Excel will likely be easier for you than working with the csv files directly. If you desire a different format, please contact us.

Short Free Text Questions

Short free text questions can be grouped into two categories:

- **List-supplementing questions:**
  Respondents are shown a list of options in a previous question. They choose "Other (please specify)" and enter a short free text.
- **List-supplanting questions:**
  Respondents are asked to enter a short free text directly, with no prior list of options or guidance regarding what values to specify.

Both types of questions are preprocessed using similar procedures, and the coding of responses should result in the **assignment of exactly one code to each answer**.

Preprocessing with Python

Start with the **original data export**. For each short free text question:

1. Produce a **csv file** containing just the *response identifiers* ("lfdn") and the *responses* to the specific question.
2. Design **regular expressions** to assign exactly one code to each response, deriving the codes from the data (i.e., from data exploration) or from external resources.
   a. For **list-supplementing questions**, use the original list of codes for the supplemented variable as a starting point for the codes to assign. If deemed necessary after data exploration, add new codes for responses that cannot reasonably be subsumed under any of the existing codes.
   b. For **list-supplanting questions**, use existing taxonomies (if available) and data exploration to come up with the codes (and the rules).

   Note: The design of the regular expressions is an iterative process best performed within Jupyter Notebook (or some other interactive programming environment supporting data exploration). It is likely to overfit the data at hand – but this is acceptable if we are primarily looking to make our code assignment process more reproducible.
3. Produce one **csv file** containing *response identifiers*, *responses*, and *assigned codes* (**code file**), and one **csv file** containing only the header (*<name of response identifier column>,old_code,new_code,reasoning*) (**correction log**).
4. Prepare a **one-page summary** containing (1) variable name, variable type, the question, and any previous answer options (if applicable); (2) a quantitative or visual summary of the algorithmic coding results; and (3) the regular expressions used in the algorithmic coding process (**supplement**).
5. Distribute the code file, the correction log, and the supplement to the expert(s) tasked with code validation.

RE-Pract 2018 Survey: Coding Instructions

Code Validation

To perform code validation for an individual short free text question, start with:

1. one csv file containing the results of the algorithmic coding process (**code file**);
2. one csv file containing only the header (*<name of response identifier column>,old_code,new_code,reasoning*) (**correction log**); and
3. one sheet of summary information (**supplement**).

To validate the codes assigned to the responses of a single short free text question:

1. **Inspect** the **supplement** and **annotate** any questions or *a priori* objections (in the supplement).
2. **Open** the code file and the correction log, and
   - for each row in the **code file**,
   - if you **object** to an assigned code,
   - **enter** the *response identifier*, the *original code*, the *corrected code*, and a short summary of *your reasoning* in the **correction log**, making sure not to break the csv file in the process.

   Caution: Please do not correct the assigned code in the code file itself. When suggesting correction codes, try to reuse existing codes as much as possible and to maintain consistency across all codes. The main purpose of coding short free text answers is aggregation (to some extent via abstraction), so creating many specific custom codes that suit only few responses is counterproductive.
3. **Email** (1) the **annotated supplement**, and (2) the **filled-in correction log** to:
   Corinna Coupette (corinna.coupette@campus.lmu.de)
   Daniel Mendez (daniel.mendez@tum.de)

Example:

```
In an imaginary survey, v_42 asks respondents to name their favorite meal component;
the algorithm performs coding to enable grouping by types of produce.

Original code file:

lfdn;v_42;v_42_coded
1001,Tomato,Vegetable

Your correction in the correction log:

lfdn;old_code;new_code;reasoning
1001;Vegetable;Fruit;"Tomatoes classified as vegetables; scientifically they are fruit"
```

Long Free Text Questions

As the coding of long free text questions is fundamentally different from the coding of short free text questions, we will refer to it as **tagging**. Tagging differs from coding in two important respects:

- The tags are not a set of disjoint concepts but rather organized as a **forest of facets** (a hybrid of the facet and tree taxonomies); each facet is represented by a hierarchical tree of tags, which may be several levels deep.
- **More than one tag may be assigned** to each answer.

For the purposes of storage, tags will be Strings with the following structure:
<level1>:<level2>:<level3>_<level4>, where all elements but the first level identifier are optional.
Examples of tags:

- **reasoning:relevance** (to characterize an answer to a question inquiring about reasons for positive evaluations);
- **what:challenge:process_prioritization** (to characterize the content of a one-sentence paper summary).

RE-Pract 2018 Survey: Coding Instructions

Preprocessing with Python

Start with the **original data export**. For each long free text question:

1. Produce a **csv file** containing just the *response identifiers* ("lfdn") and the *responses* to the specific question.
2. Design **regular expressions** to assign tags to each response, deriving the tag structure (i.e., the facets, the tags, and their hierarchical organization) from the data or from external resources.
   Note: The design of the regular expressions is an iterative process best performed within Jupyter Notebook (or some other interactive programming environment supporting data exploration). It is likely to overfit the data at hand – but this is acceptable if we are primarily looking to make our code assignment process more reproducible.
3. Produce one **long-form csv file** containing *response identifiers*, *responses*, and *assigned codes*, with one row per response and assigned code (**code file**).
   Note: It is likely that the regular expressions will leave some responses without any tags. Make sure to sort the long-form csv file by response identifiers and append the unmatched responses at the end of the file.
4. Prepare a **one-page summary** containing (1) variable name, variable type, and the question; (2) a quantitative summary of the algorithmic coding results showing clearly the tagging taxonomy (e.g., a screenshot of a multi-level-grouped pandas DataFrame); and (3) the regular expressions used in the algorithmic coding process (**supplement**).
5. Distribute the code file and the supplement to the expert(s) tasked with code validation.

Code Validation

To perform code validation for an individual long free text question, start with:

1. one csv file containing the results of the algorithmic coding process (**code file**); and
2. one sheet of summary information (**supplement**).

To validate the codes assigned to the responses of a single long free text question:

1. **Inspect** the **supplement** and **annotate** any questions or *a priori* objections (in the supplement).
   Note: You may object to the tags themselves, the way they are organized, or the rules mapping responses to tags.
2. **Open** the code file and **for each unique response identifier** in the code file:
   - For each row containing the response identifier, if you *object to a tag*:
     - if the tag you object to is the only tag assigned to the response in question: **change the tag**;
     - else if there are further tags for the response in question: **delete the row**;
   - If you encounter a *row without a tag* (usually only towards the end of the file), **add a tag**;
   - If you find a *tag missing*, **add a row** containing the *response identifier*, the *response*, and the *tag*, right below the last row that contains a tag for the response identifier in question.

   Caution: When you introduce a tag that does not appear in the original tagging scheme, you are automatically adding to the tag taxonomy. Make sure your addition preserves the consistency of the tagging scheme, both as an abstract scheme and in its concrete application to the data at hand.
3. **Email** (1) the **annotated supplement**, and (2) the **corrected code file** to:
   Corinna Coupette (corinna.coupette@campus.lmu.de)
   Daniel Mendez (daniel.mendez@tum.de)

Example:

```
In an imaginary survey, respondents are asked to rate produce
and to give reasons for particularly negative ratings.
The algorithm performs coding to enable grouping by characteristic aspects of reasoning.

Original code file:

ProduceID,;lfdn;reasoning;Tag
42,100;"Too healthy; takes hours to prepare; expensive";reason:nothealthy
42,100;"Too healthy; takes hours to prepare; expensive";reason:tooexpensive

Code file after your corrections (you correct one tag and add another tag):

ProduceID,lfdn;reasoning;Tag
42,100;"Too healthy; takes hours to prepare; expensive";reason:toohealthy
42,100;"Too healthy; takes hours to prepare; expensive";reason:toocomplicated
42,100;"Too healthy; takes hours to prepare; expensive";reason:tooexpensive
```