Illustrating Data Curation in NoRaRe with the Help of Templates

Johann-Mattis List
Chair for Multilingual Computational Linguistics
University of Passau

This study introduces a collection of templates that can be used to contribute data to the Database of Norms, Ratings, and Relations (NoRaRe) of words and concepts. The templates are intended to facilitate the process of dataset conversion and serve as a starting point for those who are interested to contribute data to the catalog. A first template structure with two sample datasets is introduced and discussed in more detail, pointing to those aspects of data curation that may lead to confusion among users who contribute the first time to the NoRaRe database.

1 Introduction

The Database of Norms, Ratings, and Relations (NoRaRe, Tjuka et al. 2022, https://norare.clld.org) offers a principled collection of data on speech norms for words across different languages, linked to identical concepts that are defined through the Concepticon catalog (List et al. 2025, https://concepticon.clld.org). While we are generally very happy with the NoRaRe data collection, given that it offers data from various sources in a unique and standardized form, we are aware that data access and data curation may cause frustration with some users, given that they require both basic knowledge about the commandline and quite solid knowledge about Python programming.

While we think that data access can be facilitated by providing more targeted tutorials, providing detailed tutorials on the curation of NoRaRe data may still be frustrating for potential contributors, given that the amount of data and scripts that has been accumulated in the original NoRaRe database by now is constantly growing and may easily overwhelm people who are new to the workflow of test-driven data curation

Computer-Assisted Language Comparison in Practice
Volume 8, Number 2
https://calc.hypotheses.org/8723 DOI: 10.15475/calcip.2025.2.2
Published under a Creative Commons Attributions 4.0 LICENSE
Published on 25/08/2025

(for details on these workflows, see Tjuka et al. 2023 on Concepticon and NoRaRe, and List et al. 2022 on Cross-Linguistic Data Formats in general). General information on how to add datasets to NoRaRe can be found in two studies introducing the data and the workflow for data curation (Tjuka et al. 2022 and Tjuka et al. 2023) and in a recent tutorial by Ahmedović (2025). In order to supplement these attempts to help people getting started with NoRaRe data curation, this study introduces a new approach, based on a template dataset. This dataset cannot only be used to illustrate different ways in which data can be added to the catalog, it can also be used as the starting point for those who want to contribute data to the original NoRaRe collection.

2 Illustrating Data Curation with Templates

Data curation in NoRaRe is based on two major components. On the one hand, we have a code package, PyNoRaRe (Forkel and List 2024, https://pypi.org/project/pynorare/), that can be invoked via the command line in order to convert individual datasets into the standardized tabular format required by NoRaRe. On the other hand, we have a folder structure, containing the original NoRaRe data, which is read and modified when invoking the code from the command line.

Since NoRaRe has grown drastically over time, listing by now close to 100 individual datasets (with new datasets being planned to be added during the next time), working with the entire NoRaRe database can feel cumbersome, given the amount of data that the repository stores in different folders and files. In order to cope with this problem, I have now created a first template repository that can be used to test and teach data curation in NoRaRe with targeted exemplary datasets that have been created specifically to illustrate the data curation process. While the template repository by now only contains two artificial sample datasets, illustrating two ways in which data can be curated in NoRaRe, I hope that we can extend the repository in the future by adding targeted illustrations.

3 Getting Started with NoRaRe

3.1 Installation

In order to get started with NoRaRe, you must make sure to install the PyNoRaRe package via pip in a fresh virtual environment (https://pypi.org/project/pynorare, List and Forkel 2024). Having installed the package, you may need additional packages for the handling of individual datasets, but most basic packages will be installed with this installation alone. In order to access NoRaRe data, you also need to download the NoRaRe database itself, along with the data underlying Concepticon. The easiest way to get started, is to download both packages with git. This can be done with the help of the following commands. The first command installs `pynorare`, the second command downloads the template collection — instead of the original NoRaRe data —, and the third command downloads the most recent version of the Concepticon reference catalog (Version 3.4.0).

```
$ pip install pynorare
$ git clone https://codeberg.org/digling/norare-template
$ git clone https://github.com/concepticon/concepticon-
data --depth 1 --branch v3.4.0
```

If you open the norare-template folder, you will find two folders (datasets and references) and two basic files (`datasets.tsv` and `norare.tsv`). The current version of the template repository will ignore the `norare.tsv` file. It describes individual columns of individual datasets, but it is not needed to guarantee the basic functionality of curated data in NoRaRe and maybe introduced in future modifications to the template. The file datasets.tsv contains a header and two entries that provide information on two datasets for which templates were created in the repository. We call these datasets `Template-0001-Base` and `Template-0002-Manual`, respectively, thereby following the naming convention of datasets in NoRaRe, which start with the author name (here replaced by `Template`), followed by the year of creation (here replaced by `0001` and `0002` to employ some numerical ordering of template data), and followed by a short string illustrating the main purpose of the data (`Base` illustrating basic mapping, and `Manual` illustrating how data can be mapped manually).

Additional information on individual datasets is provided in the following columns. When filling in the field `REFS`, it is important to know that this corresponds to a BibTeX-entry. The corresponding BibTeX-file can be found in the references folder (references.bib). The description of the data in the `NOTE` field will later be displayed on the NoRaRe website (https://norare.clld.org), as is the content of the other columns.

The datasets themselves rest in the folder datasets. Per dataset, we add a dedicated folder that should have the same name as the dataset ID as indicated in `datasets.tsv`. This means we find two folders in the current template, `Template-0001-Base` and `Template-0002-Manual`. Inside a dataset folder, there are two required files that need to be provided by the data curators. One file, called `norare.py`, contains the code that is needed to download the original data and convert it to the NoRaRe formats, and one file, consisting of the dataset ID, extended by `.tsv-metadata.json`, contains the CSVW specification that describes the content of all columns in the resulting NoRaRe dataset and — if applicable — how they relate to the original data.

3.2 Basic Principles of Data Curation

The basic procedure for adding datasets to NoRaRe consists in two steps, with both steps being automatized in such a form that they can be triggered one after another. The first step, called the download, consists in downloading the data and storing the data in a folder for raw data. The second step is the mapping step in which data are mapped to Concepticon. As a result of these two steps, a concept list is written to file, containing standardized data of concepts linked to Concepticon.

Both steps are triggered by adding a new folder to the collection of datasets in the original norare-data-folder. The name of this new folder should start with the name of the first author of the dataset, separated by a dash from the year, followed by a description of the kind of data, one must place two files in order to go ahead. Two files must be placed into this folder to get started. The first file is a metadata file that follows the CSVW standard (CSVW, https://csvw.org) and crucially provides information in JSON format for the names and content of the columns that the resulting TSV file in which the mapped concepts are added automatically later contains. The file itself should contain the name of the dataset, ending in `.tsv-metadata.json`. To describe the full structure of the file here would go too far. It seems sufficient to recommend to take an existing file from the other datasets that have already been added to NoRaRe and to adjust the columns in this file accordingly.

The second file is a Python script that triggers how data are downloaded and how data are mapped to Concepticon with the help of two functions. This file should be called `norare.py`. Essentially, this file should contain two functions, one called download and one called map. A minimal example fo these functions is shown below.

```python
def download(dataset):
    pass


def map(dataset, concepticon, mappings):
    pass
```

In the following, these commands will be illustrated in more detail.

3.3 Defining the Metadata

The metadata file that describes the structure of the TSV file that holds the mappings of a given resource to Concepticon plays a crucial role in the NoRaRe workflow. Its job is not only to handle the access to already mapped data. It also allows to provide active mappings between the columns of a sheet in the original data and the data that we want to create from it. Thus, the metadata sheet itself is crucial for the mapping procedure. In order to get started with such a file, it is best to start from a template. The file is named after a dataset, extended by the suffix `.tsv`, and extended by the suffix `-metadata.json`. Thus, in the case of our base template, `Template-0001-Base` would be the name of the dataset and the dataset folder, `Template-0001-Base.tsv` is the name of the data file that we want to produce (providing access to the mappings), and `Template-0001-Base.tsv-metadata.json` is the name of the metadata file, describing the data with the help of the CSVW specification. The file for our base template is shown below.

```json
{
    "url": "",
    "@context": ["http://www.w3.org/ns/csvw", {"@language": "en"}],
    "dc:title": "Template for Adding Data to NoRaRe.",
    "dc:source": "https://calclab.org/norare/example/",
    "dc:references": "Template",
    "dcat:keyword": ["template", "example"],
    "dc:description": "A template file that can be used as the basis for adding
                       new data to NoRaRe.",
    "dialect": {
        "delimiter": "\t",
        "encoding": "utf-8",
        "header": true
    },
    "tables": [{
        "tableSchema": {
            "columns": [
                {"name": "CONCEPTICON_ID", "datatype": "integer"},
                {"name": "CONCEPTICON_GLOSS", "datatype": "string"},
                {"name": "GERMAN", "datatype": "string", "titles": "word"},
                {"name": "GERMAN_FLOATS", "datatype": {"base": "float"},
                 "titles": "Floats"},
                {"name": "GERMAN_INTEGERS", "datatype": "nonNegativeInteger",
                 "titles": "Integers"},
                {"name": "GERMAN_STRINGS", "datatype": "string",
                 "titles": "Strings"},
                {"name": "GERMAN_JSON", "datatype": "json", "titles": "JSON"}
            ],
            "foreignKeys": [{
                "columnReference": "CONCEPTICON_ID",
                "reference": {
                    "resource": "../concepticon.tsv",
                    "columnReference": "ID"
                }
            }],
            "aboutUrl": "http://concepticon.clld.org/parameters/{CONCEPTICON_ID}"
        },
        "url": "Template-0001-Base.tsv"
    }]
}
```

The first lines of this file, up to the `tables` key provide some basic metadata information on the dataset, allowing users to add some `description` of the data, a `title`, and `keywords`. The tables key provides access to the actual tables of the data. A table is defined by a schema (`tableSchema` and a `url` (the path to the file, identical with the file name in this case).

```
 "tables": [
   {
     "tableSchema": {},
     "url": "Template-0001-Base.tsv"
   }
  ]
```

The table schema consists of three objects, `columns`, `foreignKeys`, and `aboutUrl`. While the latter two can be left unchanged, the columns must be specified for the target dataset. Each column is represented by a dictionary of key-value pairs, of which three are regularly used in NoRaRe datasets, namely name, datatype, and titles. The key `name` refers to the target name of the column in the corresponding TSV file that one wants to create. The key `datatype` defines the type of the data in the cell. For the representation of datatypes, there are numerous options in CSVW, one can specify minimum and maximum values, patterns by regular expressions, different kinds of numbers, and boolean data. One can even — and this is important for all kinds of data where "normal" datatypes do not suffice — use JSON as a datatype. This offers the possibility to render complex objects, which comes in handy when dealing with concept relations (Bocklage et al. 2024).

The `titles` key in the column specification of CSVW metadata files plays an important role in NoRaRe, since it is used to provide a direct mapping between the columns in a source CSV file and the target CSV file produced by the mapping procedure of NoRaRe. Thus, in our sample metadata file, we have two target columns GERMAN and GERMAN_FLOATS, with the titles `word` and `Floats`, respectively, as shown below.

```
          {
            "name": "GERMAN",
            "datatype": "string",
            "titles": "word"
          },
          {
            "name": "GERMAN_FLOATS",
            "datatype": {"base": "float"},
            "titles": "Floats"
          },
```

In the internal semantics of the NoRaRe database, the name and titles attributes map the original data, the CSV file beispiel.tsv with the columns words, Strings, Integers, Floats, and JSON, to the NoRaRe representation of the data in the CSV file `Template-0001-Base.tsv` with the columns GERMAN (`words`), GERMAN_STRINGS (`strings`), GERMAN_INTEGERS (`Integers`), GERMAN_FLOATS (`Floats`), and GERMAN_JSON (`JSON`).

In order to create the metadata file, it has been shown to be the best strategy to use a template (as the one shown here), and to modify it subsequently, by adding or removing columns, and by modifying or refining the datatypes, target names, and titles. As a rule, a NoRaRe dataset should consist of two columns providing information on Concepticon mappings (CONCEPTICON_ID and CONCEPTICON_GLOSS), one column providing information on the actual words or concept glosses (GERMAN in our example), and one or more additional columns providing information on norms, ratings, or relations.

In order to understand how complex datatypes can be defined, it is recommended to study the information provided on the CSVW website (https://w3c.github.io/csvw/primer/#new-datatypes) or to check out the examples that we provide in NoRaRe itself. Obviously, datatypes would be a good example for a further extension of the NoRaRe templates that would, however, go beyond the state of this study.

3.4 Downloading Original Data

The download command -- when triggered with actual code -- takes the variable dataset as input. This variable will be passed from the commandline when calling `norare download DATASET`, where DATASET stands for the name of the dataset that one wants to add. The dataset variable itself is an object that offers additional functions that allow for a convenient downloading of data and storing the downloaded data in a folder raw, without having to do this explicitly in the code. Thus, one can extend the base function as shown below, in order to download the TSV file `example.tsv` from the URL https://calclab.org/examples/example.tsv.

```
def download(dataset):
    dataset.download_file(
       'https://calclab.org/examples/example.tsv',
       'beispiel.tsv')
```

This command can be initiated by typing the following command in the terminal.

```
$ norare --norarepo=norare-data --repos=concepticon-data download
Template-0001-Base
```

This will download the respective file from the website and store it in a folder raw, assigning it the name `beispiel.tsv` in that very folder. There are more possibilities to download data. For example, you can download and unzip a file directly, using the `dataset.download_zip` command, that takes three arguments as input: the original URL of the dataset (ending in `.zip`), the target name that you want to give to the data, and the file that you want to extract from the repository itself. An example is included in the template `Template-0001-Base`, where the code for download actually downloads two files, the file `examples.tsv` and the file `beispiel2.tsv` from the zipped archive example.zip.

```python
def download(dataset):

dataset.download_file('https://calclab.org/examples/example.tsv',
    'beispiel.tsv')

dataset.download_zip("https://calclab.org/examples/example.zip",
                         "example.zip",
                         "beispiel2.tsv")
```

If you do not want to download data, or if you do not need to download data, you can leave the map-command empty, by adding a `pass` statement under the function.

```python
def download(dataset):
    pass
```

In addition, you can place the data in their original form into the raw folder and submit it to the repository. This may be useful in those cases where you have small datasets that can be openly shared.

3.5 Mapping Data Automatically

Data in NoRaRe must be mapped to Concepticon, since the links to Concepticon are the major way to compare information on particular concepts across languages. There are basiclly two ways in which you can proceed in order to map your concepts. On the one hand, you can use the integrated functions for automated concept mapping that come along with NoRaRe and Concepticon itself. On the other hand, you can use custom procedures to map your concepts, or you could even manually map your concepts to Concepticon, provide the information in the original data or in the `raw` folder or take the information from projects like Concepticon, and later create the NoRaRe dataset without using the automated mapper shipped along with NoRaRe.

As an example for the typical usecase where we start from some dataset providing norms or ratings in some language, let us look at the template `Template-0001-Base` that builds on automated mapping with the help of the following map-command.

```
def map(dataset, concepticon, mappings):

    sheet = dataset.get_csv(
        'beispiel.tsv',
        delimiter="\t",
        dicts=True
        )

dataset.extract_data(
        sheet,
        concepticon,
        mappings,
        gloss='GERMAN',
        language='de'
        )
```

The map function here takes three arguments as input, `dataset`, `concepticon`, and `mappings`. As a NoRaRe user, you do not necessarily need to care for their internal structure, since they are provided by the commandline procedure. The argument dataset is the same complex object providing access to various routines that we also used in the download procedure. The `concepticon` argument provides access to the PyConcepticon API (Forkel et al. 2024, https://pypi.org/project/pyconcepticon). This means one can access any datapoint and concept list that Concepticon offers in the version that one selects from the commandline. It also means one can enrich a given dataset from Concepticon with additional data in NoRaRe. The `mappings` argument provides access to the mappings from Concepticon. These mappings are extracted from all concept lists that are linked by a respective Concepticon version and allows to check, to which Concepticon ID and Concepticon Gloss a given word in a given language with a given part of speech (if available) is linked in the Concepticon project. The mappings in this form provide the core of all the mappings used in NoRaRe.

The code that we added to implement the map functions consists itself of two functions, both provided by the dataset object. First, we extract the data from the CSV-file `beispiel.tsv`, stored in the raw folder (the file that we just downloaded), with the help of the `dataset.get_csv` function (where we indicate that the separator of the CSV-file is a tabstop). The resulting sheets object is a list of ordered dictionaries in

Python, that represents cells as key-value pairs, with column names as keys, and cell content as values, as shown below for clarity.

```
[
    OrderedDict(
        {
            'word': 'Hand',
            'Floats': '1.2',
            'Integers': '1', 'Strings':
            'eine Hand',
            'JSON': '{"name": "hand"}'
        }
    ),
    OrderedDict(
        {
            'word': 'Fuß',
             'Floats': '1.3',
             'Integers': '2',
             'Strings': 'ein Fuß',
             'JSON': '{"name": "Fuß"}'
        }
    ),
    OrderedDict(
        {
            'word': 'Stadt',
            'Floats': '1.5',
            'Integers': '3',
            'Strings': 'eine Stadt',
            'JSON': '{"name": "city"}'
        }
    ),
    OrderedDict(
        {
            'word': 'Apfel',
            'Floats': '1.5',
            'Integers': '3',
            'Strings': 'eine Stadt',
            'JSON': '{"name": "city"}'
        }
    ),
    OrderedDict(
        {
            'word': 'Arm',
            'Floats': '1.2',
            'Integers': '1',
            'Strings': 'eine Hand',
            'JSON': '{"name": "hand"}'
        }
    )
]
```

Then, we use the command dataset.extract_data to map the data automatically to Concepticon and only retain those lines in the original data that can be mapped. The `extract_data`-command itself employs the mappings between the column names of the original data and the column names of the target CSV file that was discussed before in §3.3. Taking the sheet that we just extracted before as input, as well as the `concepticon` object, and the mappings, that the map-function receives from the commandline usage of NoRaRe, the method uses the information on the `gloss` field (which refers to the name that this column will receive in the target language, thus aiming at the German word forms in the original column words in our sample data) and the `language` (represented for the major languages by a two-letter code) in order to assess which of the Concepticon concept receives the highest score in the automated mapping procedure. If no mapping is found, this word form will not be written to the target file. Furthermore, each word form is mapped to maximally one Concepticon concept set.

In order to check how well the automated mapping procedure works, one just has to trigger the command of the base template, passing the paths to the NoRaRe template folder and the Concepticon data folder along with the map command and the name of the dataset one wants to map, as shown below.

```
$ norare --norarepo=norare-template --repos=concepticon map
Template-0001-Base
```

The resulting mapping will be written to the file `Template-0001-Base.tsv`, which is the file that we have already described through our CSVW metadata file. The first four columns of this file are shown below in Table 1.

CONCEPTICON_ID	CONCEPTICON_GLOSS	GERMAN	GERMAN_FLOATS
1277	HAND	Hand	1.2
1301	FOOT	Fuß	1.3
1320	APPLE	Apfel	1.5
1391	TOWN	Stadt	1.5
1673	ARM	Arm	1.2

Table 1: Output of the automated concept mapping.

As can be seen, the automated mapping procedure identifies Concepticon glosses for all five German words and writes all columns that we defined in the metadata file to the new file. While these mappings themselves are considerably easy to achieve, it is always recommended to be careful with the trust in automated mappings. While we currently assume that the errors fall below a margin of 10%, we have not yet carried out detailed error statistics. Our trust in the mapping algorithm is rather based on our concrete experience in using the algorithm to preprocess large Concepticon concepts lists (Tjuka et al. 2023).

3.6 Mapping Data Explicitly

While the automated mapping procedure described in the previous section works well and sufficiently in most cases, there may be situations in which one does not want to resort to automated mapping. On the one hand, one might have access to better mappings, for example, produced by manual data curation. On the other hand, one might want to make use of a different than the standard method to produce the mappings in question.

The NoRaRe data curation workflow allows for this flexibility, which we illustrate in the template dataset `Template-0002-Manual`, as part of our initial template collection for NoRaRe. This template uses the same metadata file (with the exception that the file name has changed to the name of the template). It also employs the same download routine. What differs, is the mapping routine, which makes use of the PySem package (List 2025, https://pypi.org/project/pysem/), introduced in List (2022), which can be installed pip (`pip install pysem`). The modified mapping routine that makes explicit use of the `to_concepticon` function in PySem is illustrated below.

```
def map(dataset, concepticon, mappings):

    sheet = dataset.get_csv(
            'beispiel.tsv',
            delimiter="\t",
            dicts=True
            )
    # get mapping from old to new column names
    s2t = {str(c.titles): c.name for c in dataset.columns if c.titles}

    table = []
    for row in sheet:
        maps = to_concepticon(
                [
                    {
                        "gloss": row["word"],
                        },
                    ],
                language="de"
                )
        if maps[row["word"]]:
            cid, cgl, pos, sim = maps[row["word"]][0]
            table += [{
                "CONCEPTICON_ID": cid,
                "CONCEPTICON_GLOSS": cgl,
                s2t["word"]: row["word"],
                s2t["Floats"]: row["Floats"],
                s2t["Integers"]: row["Integers"],
                s2t["Strings"]: row["Strings"],
                s2t["JSON"]: row["JSON"]
                }]
    dataset.write_table(table)
```

While the routine to load the data from the CSV file remains the same here, the difference lies in the way in which the target NoRaRe dataset is written to a table. Here, the code first explicitly extracts the information on the mapping from source to target column headers, storing them in the dictionary `s2t`. It then creates an empty list called `table` and afterwards iterates over all individual entries in the source table, mapping all entries automatically with the help of PySem's modified mapping routine. This table is a list of dictionaries whose keys correspond to the new column headers that we want to write to the target CSV file defined by the CSVW metadata file. The table itself can then be written to the target file with the help of the command `dataset.write_table`. This command takes the table as input and takes essentially care of all the rest. This means, among others, that only those columns that were defined in the CSVW metadata file will be written to the target spreadsheet. Columns defined in the metadata but not present as keys in the dictionary will be left empty. For this reason, it is important to check the resulting data thoroughly, since spelling errors can easily slip in when creating and modifying the metadata.

The method to write a table explicitly outlined here can be used in all those cases where the NoRaRe data one wants to produce differs from the standard datasets that one encounters so far in NoRaRe. Allowing for this flexibility with new datasets that have not been encountered before has the advantage that it allows us to explore new datatypes for NoRaRe and later decide if we write new regular routines to map them. When dealing with concept networks, for example, it may well be that we add a more targeted routine in the future, even if for now we add them explicitly in NoRaRe.

4 Creating New Data from Templates

4.1 Checklist for the Creation of NoRaRe Datasets

NoRaRe datasets require more integration beyond the dataset folder (whose structure was described before in due detail). Table 2 gives a short checklist that can be used when creating new NoRaRe datasets, indicating all those places where things need to be modified.

This checklist falls short in describing the data in the file `norare.tsv`. This means the data can be accessed via the NoRaRe API and individual scripts can be written to integrate the data in scientific programming routines. Only by adding information on the individual columns in `norare.tsv`, however, we can make sure that the data can be identified and compared with similar datapoints. The details of how to add this column-specific information are not discussed here, since our initial templates concentrate for now only on the dataset creation. In the future, we may add more examples that also show how the data can be further integrated.

Operation	File	Note
Create dataset folder.	`datasets/DATASET`	Follow specific naming conventions.
Create Python file in dataset folder	`datasets/DATASET/norare.py`	Add `download` and `map` commands.
Create CSVW metadata file in dataset folder.	`datasets/DATASET/DATASET.tsv-metadata.json`	Use `titles` and `names` to link column names.
Add dataset to the list of datasets.	`datasets.tsv`	Fill in all fields, pay attention to the ID and REFS.
Add reference to the bibliography.	`references/references.bib`	Use the key that was used in the REFS column of `datasets.tsv`.
Download dataset.	`norare download DATASET`	Run the command to make sure the download routine works as expected.
Map dataset.	`norare map DATASET`	Run the command to make sure the mapping proceeds as expected.
Validate dataset.	`norare validate DATASET`	Run the command to make sure the data validates (also check manually by inspecting the TSV file).

Table 2: Checklist for the different steps needed to integrate a dataset in NoRaRe.

4.2 Basic Tips for Generating Derived Datasets

If you want to derive your own datasets from one of the templates introduced above, the first step that I can recommend would consist in locating the data, ideally finding a regular URL from which they can be downloaded, and a reference that can be cited. As next step, you would determine the name of the dataset, as a combination of hte family name of the first author, along with the year of the publication, and a short name that introduces the data. With this information, you can then create a folder in the norare-template directory, add a first draft `norare.py` script and copy and paste one of the sample metadata files that we provide in the `norare-template` folder. Having determined how to download the data (if regular download does not work and the data are distributed with open licenses, you can also simply paste them to the `raw` folder of the dataset directory), one would then elaborate how to map the data to Concepticon. When working with automated mapping procedures, one would first determine the relation between the columns of the original spreadsheet and the columns that one wants to define in the target table. After adding these relations to the metadata file, one could start experimenting with an initial mapping routine. If the data are more complex in nature, or if mappings are also available independently, one would have to write code that loads the data and converts them to the tables along with mappings to Concepticon, as illustrated in §3.6. Before running any mapping or download commands, the reference would have to be provided in BibTeX format and the dataset would have to be described properly in the CSV file that stores informatio on all individual datasets.

Having created a dataset that passes both individual and general tests, it will be straightforward to copy-paste the folder and the modified lines to the original NoRaRe data, currently curated on GitHub (https://github.com/concepticon/norare-data). From there, one would then make a pull request and hope for a quick and productive review process by the NoRaRe team.

5 Outlook

In the future, we hope to find time to add more templates to NoRaRe in order to illustrate how particular types of data can be handled. While NoRaRe itself provides plenty examples of how data has been handled in the past by us, we are aware that it may be confusing for new contributors to build directly on these examples when trying to integrate their own data into the NoRaRe catalog. Since we hope to be able to integrate quite a few more datasets on concept relations in the future, we will need better tutorials that help contributors to get started with NoRaRe. The templates can be understood as a first step towards this goal. We hope that we also will find time to follow up with tutorials that illustrate more broadly how NoRaRe can be put to active use.

References

Ahmedović, Mira (2025): Handling Non-Standard Datasets in NoRaRe: A Practical Guide. Computer-Assisted Language Comparison in Practice 8.1. 17–23. https://doi.org/10.15475/calcip.2025.1.3

Bocklage, Katja and Di Natale, Anna and Tjuka, Annika and List, Johann-Mattis (2024): Representing the Database of Semantic Shifts by Zalizniak et al. from 2024 in Cross-Linguistic Data Formats. Computer-Assisted Language Comparison in Practice 7.1. 25-35. https://doi.org/10.15475/calcip.2024.1.4

Robin Gower (2021): CSV on the Web. Stirling: Swirrl. https://csvw.org

List, Johann-Mattis and Tjuka, Annika and Blum, Frederic and Kučerová, Alžběta and Barrientos Ugarte, Carlos and Rzymski, Christoph and Greenhill, Simon J. and Robert Forkel (2025): CLLD Concepticon [Dataset, Version 3.3.0]. Leipzig: Max Planck Institute for Evolutionary Anthropology. https://concepticon.clld.org

List, Johann-Mattis (2022): How to Map Concepts with the PySem Library. Computer-Assisted Language Comparison in Practice 5.1. 1-5. https://calc.hypotheses.org/3193

List, Johann-Mattis and Hill, Nathan W. and Forkel, Robert (2022): A new framework for fast automated phonological reconstruction using trimmed alignments and sound correspondence patterns. In: Proceedings of the 3rd Workshop on Computational Approaches to Historical Language Change. Association for Computational Linguistics 89-96. https://aclanthology.org/2022.lchange-1.9

Forkel, Robert and Rzymski, Christoph and List, Johann-Mattis (2024): PyConcepticon [Python library, Version 3.1.0]. Leipzig: Max Planck Institute for Evolutionary Anthropology. https://pypi.org/project/pyconcepticon

Forkel, Robert and List, Johann-Mattis (2024): PyNoRaRe [Python library, Version 1.1.0]. Passau: MCL Chair at the University of Passau. https://pypi.org/project/pynorare

List, Johann-Mattis (2025): PySem: Python library for handling semantic data in linguistics [Software, Version 1.2.1]. Leipzig: Max Planck Institute for Evolutionary Anthropology. https://pypi.org/project/pysem

Tjuka, Annika and Forkel, Robert and List, Johann-Mattis (2022): Linking norms, ratings, and relations of words and concepts across multiple language varieties. Behavior Research Methods 54.2. 864–884. https://doi.org/10.3758/s13428-021-01650-1

Tjuka, Annika and Forkel, Rober and List, Johann-Mattis (2023): Curating and extending data for language comparison in Concepticon and NoRaRe [version 2; peer review: 2 approved]. Open Research Europe 2.141. https://doi.org/10.12688/openreseurope.15380.3

Supplementary Material
NoRaRe template data are curated on Codeberg (https://codeberg.org/digling/norare-template, Version 0.1) and archived with Zenodo (https://doi.org/10.5281/zenodo.16902395).
Funding Information
This project has received funding from the European Research Council (ERC) under the European Union's Horizon Europe research and innovation programme (Grant agreement No. 101044282). The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.