MRP-Kit: A 'grammar' of multilevel regression with post-stratification and implementation in R*

Lauren Kennedy† Mitzi Morris‡ Jonah Gabry§ Rohan Alexander¶
Dewi Amaliah‖

11 March 2021

Abstract

In this paper we define a 'grammar' for multilevel regression with post-stratification (MRP) and implement this grammar in the R Package `MRPKit`. Our grammar is centered around the following components: datasets; correspondences; and models. Survey datasets are linked by correspondences and models act on these correspondences. Our grammar, and its implementation, provides a detailed workflow when conducting MRP that will be useful for researchers and in teaching.

Keywords: multilevel regression with post-stratification, R, reproducibility, statistics.

1 Introduction

Multilevel regression with post-stratification (MRP) is a statistical approach in which surveys are related to each other using a statistical model. This is important because known biases in one survey can be adjusted by another in a statistically reasonable way. This enables better use of non-representative surveys, additional information, and propagation of uncertainty. However, it can be difficult to use MRP due to this need to relate (at least) two different datasets. This package defines a grammar, or list of underlying rules, for MRP and then describes an R package, `MRPKit`, that implements this grammar.

At its core, MRP is a mapping between a survey object and a population object. It is from this mapping that the power of MRP exists, but it can be difficult to establish this mapping in a reproducible way that fits into a typical applied statistics workflow. Making this implementation easier and more reproducible is important as interest in, and the use of, MRP increases. It can be difficult even for those experienced with MRP to ensure there are no mistakes in this mapping and creating a well-defined workflow for MRP is an important contribution to enhancing the reproducibility of MRP analysis.

We first define a grammar of MRP, which we define as the underlying rules and principles that are common to every analysis based on MRP. This grammar is based around the following components: 'datasets'; 'correspondences'; and 'models.' Typically MRP requires at least two datasets - usually a survey object and a post-stratification object, which we implement using the `SurveyData` method. The questions in each datasets are then aligned by establishing a correspondence between each possible answer to a particular question, and we implement this using the `SurveyQuestion` method. Having linked each question, the surveys themselves can be linked and we implement this using the `SurveyMap` method. Finally, that correspondence can be modelled, and used to generate estimates. In this way, our grammar and package implement an entire statistical workflow for MRP that is largely agnostic of the specifics of each component.

*Code and data are available at: https://github.com/mitzimorris/mrp-kit.

†Monash University, lauren.kennedy729@gmail.com
‡Columbia University
§Columbia University
¶University of Toronto
‖Monash University

The survey object would typically be a regular survey, such as a political poll of, say, 1,000 respondents, but it could also be a larger survey, such as the Canadian Election Survey, or similar. The post-stratification object would typically be a larger survey, such as, in the case of the US, the American Community Surveys, or a census. The correspondences need to be established for each possible answer for each question, for instance, the age-groups used in both surveys need to be homogenized. Finally, the model would typically be a Bayesian hierarchical model, but in principle any variety could be used.

Our grammar and package complements existing packages such as `survey` Lumley (2010), and `DeclareDesign` (Blair et al. 2019). These packages are focused on the designing, implementing, and simulating from, surveys. Instead, ours is focused on what is needed for MRP, and we draw on their packages where possible.

The remainder of this paper is structured as follows: Section 2 reviews similar packages and contributions and places ours within that context. Section 3 discusses the grammar and the core aspects of MRP as implemented in `MRPKit`. Section 4 discusses some of the implementation issues and technical notes related to the decisions that were made. Section 5 provides two examples of the package in use, one using SOMETHING, and the other using SOMETHING ELSE. Finally, Section 6 provides a summary discussion, some cautions and weaknesses, as well as notes about next steps.

2 Review of related packages

The most common alternative at the moment to this package is for users to do all aspects of the MRP workflow themselves. While there is nothing inherently difficult about MRP, the implementation can be difficult. In particular, preparing and matching different levels between surveys can be time consuming and potentially introduce undocumented errors.

The `survey` package Lumley (2010) is well-established and allows a user to specify a survey design, such as clustered, stratified, etc, generate summary statistics, and conduct sub-population analysis. The `DeclareDesign` package (Blair et al. 2019) is a newer package that is based around a grammar of research design aspects including: data generation, creating treatment and control groups, and sampling. There is also an existing package called `MRP` available here: https://github.com/gelman/mrp. However, it does not appear to have been updated in some time.

The main alternative approach is for MRP to be conducted on a case-specific basis. This means that a researcher who is interested in MRP estimates writes the code needed to obtain, clean, prepare, analyze and ultimately interpret the estimates. Again, while there is nothing inherently wrong with this approach, successfully conducting MRP requires dealing with a large number of small issues. Each of these is small in their own right but getting them wrong can have large effects that are potentially unnoticed on the estimates. It is also easy to overlook steps, or to not appropriately document them. The `MRPKit` package is prescriptive about the steps that must be taken, and ensures that all aspects are documented.

3 Components and grammar

The `MRPKit` package has the following key components: `SurveyData` objects, which for most users will be a collection of two surveys where one is larger than the other; a `SurveyQuestion` object, that holds the mapping for one question or demographic between the survey and post-stratification dataset; and a `SurveyMap` object, which holds the mapping between a set of items in a survey and a post-stratification dataset. These are expanded on in the following sub-sections before being brought together.

3.1 SurveyData objects

A `SurveyData` object represents a survey and its metadata. The survey itself is a data frame and importantly has a series of metadata. The survey metadata consists of the following:

- per-column questions: a list of strings
- per-column allowed response values: a list of character vectors

- per-column survey weights: a vector of numeric weights
- survey design: a string that specifies the survey design using `lme4` style formula syntax.

Some examples of survey designs include:

- `~.`: a random sample
- `~ (1|cluster)`: a one stage cluster sample
- `~ stratum`: a stratified sample

These `SurveyData` objects are the surveys that MRP will bring together. A user can use the function `add` to create a survey object by providing a CSV file location. The user then needs to identify the data types of each column. Typically, there will need to be two survey objects, where one will be the survey that is of interest for its response variables, such as political opinion, and another will be a survey that is to be used for post-stratification.

3.2 SurveyQuestion objects

A `SurveyQuestion` object holds the mapping for one question or demographic between the survey and post-stratification dataset. For instance, if both the survey and post-stratification dataset contain a question about age, it may be that answers are expressed as a single integer representing years and so mapping them is relatively straight-forward. This is done with the function `new`, for instance:

```
SurveyQuestion$new(
  name = "age",
  col_names = c("age1","age2"),
  values_map = list(
    "18" = "18",
    "19" = "19",
    "20" = "20"
    )
)
```

If ages have been aggregating into age-groups then it may not be the case that both the survey and the post-stratification dataset have groups that directly correspond. In that case, a user needs to specify how the age-groups fit together. This is also done with the function `new`, for instance:

```
SurveyQuestion$new(
  name = "age",
  col_names = c("age1","age2"),
  values_map = list(
    "18-25" = "18-35",
    "26-35" = "18-35",
    "36-45" = "36-55",
    "46-55" = "36-55",
    "56-65" = "56-65",
    "66-75" = "66+",
    "76-90" = "66+"
    )
)
```

In this example, the age-groups do not neatly match, however they can be made to match. For instance, the 36-45 and 46-55 age-groups in the survey correspond to the 36-55 age-group in the post-stratification dataset.

If there is not a clean correspondence between the survey and post-stratification datasets, then the user needs to decide which age-group to assign to.

3.3 SurveyMap objects

A `SurveyMap` object holds the mapping between a set of items in a survey and a population dataset. That is, it draws on two `SurveyData` objects, and a `SurveyQuestion` object, and brings the two `SurveyData` objects together following the correspondences provided in the `SurveyQuestion` object.

The label is the item label in each dataset and the values is a list of all possible values. The values for the survey and post-stratification must be aligned, i.e., the lists must have the same number of elements and the values at index i in each list are equivalent. If there is a meaningful ordering over the values, they should be listed in that order, either descending or ascending.

One of the fundamental issues when conducting MRP is to ensure that the survey and post-stratification datasets are aligned. This object ensures that is the case before a user conducts their analysis.

For instance, if a user decides to use age-group and gender, then the map would specify the correspondences in age-group and also in gender. If they then also decided to use education, then the map would also specify that correspondence. They would then use the function `add` to include that third question in the map.

The function `validate` is used on `SurveyMap` objects to assess whether the proposed map is appropriate. This means that an error occurs if there is a mismatch, and a warning occurs if there are variables that are available in either the survey or post-stratification datasets that have not been mapped and hence would be unavailable to model.

After the `SurveyMap` object has been validated then the function `mapping` is used to prepare it to be modelled.

A model is fit to a `SurveyMap` object using the function `fit`. After a model has been fit it can be used to get MRP-based estimates using `predictify()`. And finally the estimates can be visualized using `visify()`.

3.4 Putting them together

We provide examples of implementation in Section 5 , however, we want to explain putting the three objects together here and to briefly bring the three objects together.

To begin with, one needs two datasets. These would be created using `SurveyObj$new`, specifying the questions, responses, weights, and design.

Having established both a survey dataset and a post-stratification dataset, a user need to specify how the questions in the survey dataset correspond to the questions in the post-stratification dataset. This is done on a question-by-question basis using `question$new` assigning a name for that question, the labels that are used in the survey and post-stratification datasets, and the values. A user does this in such a way that the values are matched between the two datasets.

The user now need to put these questions together and does this using `SurveyMap$new` and specifying the survey object, the post-stratification object, and the questions are to be used. A user can validate this map with `validate`, and potentially add additional questions using `add`, delete some existing ones using `delete`, or potentially replace a question using `replace`. Once a user is happy with the correspondence to be used they can establish a mapping using `mapping`.

The relationship can then be modelled using `fit`. There are a variety of options that are available as built-into the package, including `rstanarm::stan_glmer`, and `brms::brm`. A user could also specify their own modelling approach if they wanted to. In any case, once a model has been fit to the survey dataset, it can be used to create estimates based on the post-stratification dataset with `predictify`, which takes a fitted model as an argument. Typically a user is interested in some type of high-level estimates, for instance on a province/state basis, and `collapsify` provides the functionality to do this. Finally, the estimates on that collapsed basis can be visualized with `visify`

4 Implementation and technical notes

5 Vignette

5.1 Simulated data

5.2 Using CCES data

5.3 Using your own data

6 Summary and discussion summary

6.1 On the importance of scaffolding for workflows

6.2 Next steps and cautions

References

Blair, Graeme, Jasper Cooper, Alexander Coppock, and Macartan Humphreys. 2019. "Declaring and Diagnosing Research Designs." *American Political Science Review* 113: 838–59. https://declaredesign.org/paper.pdf.

Lumley, Thomas. 2004. "Analysis of Complex Survey Samples." *Journal of Statistical Software* 9 (1): 1–19.

———. 2010. *Complex Surveys: A Guide to Analysis Using r: A Guide to Analysis Using r*. John Wiley; Sons.

———. 2020. "Survey: Analysis of Complex Survey Samples."