biokNN: A bi-objective imputation method for multilevel data in R

Maximiliano Cubillos

Abstract

The **biokNN** package focus on the imputation of missing values for multilevel datasets using a bi-objective k-Nearest Neighbors (biokNN) method. The package provides functions to produce single and multiple imputation for data with continuous variables along with visualization tools to analyze the structure of the missing values among classes and variables. This type of data is usually found in areas in which datasets show some form of natural clustering where lower-level units (e.g., students, employees) are nested with higher-level units (e.g., classrooms, departments). This article presents an overview of the package functionalities and examples of its implementation with simulated datasets.

1 Introduction

Most statistical and machine learning methods require that the input datasets are complete. However, most real-life datasets contain missing values limiting the application of such methods (Kowarik and Templ 2016). In these scenarios, imputation methods are recommended to avoid the efficiency loss and bias that result from removing rows with missing data (Garciarena and Santana 2017). Imputation consists of the estimation of values to replace the missing entries by using the observed data.

When data are structured in clusters or classes where lower-level units (e.g., students, employees) are nested with higher-level units (e.g., classrooms, municipalities, countries), they are said to have a multilevel structure. Failing to take into account this structure into the imputation process may lead to severe model and parameter misspecification (Enders, Mistler, and Keller 2016). Methods that integrate such structure into the imputation process are referred as multilevel imputation methods.

Multilevel imputation methods are usually based on linear models with fixed or varying intercepts. Two main approaches are commonly used to integrate this structure into the imputation modelling which are Joint Modelling (JM) and the Fully Conditional Specification (FCS) approach. In R, the JM approach is implemented in two packages: **pan** (Grund, Lüdtke, and Robitzsch 2016) and **jomo** (Quartagno, Grund, and Carpenter 2019). The package **mice** (Buuren and Groothuis-Oudshoorn 2010) implements several multilevel imputation methods including FCS approaches. The main methods implemented in **mice** are: two-level normal imputation (`2l.norm`), two-level normal imputation using pan (`2l.pan`), imputation at level-2 of the class mean (`2lonly.mean`), omputation at level-2 by Bayesian linear regression (`2lonly.norm`) and imputation at level-2 by Predictive mean matching (`2lonly.pmm`). Currently, these methods in the **mice** package are limited to numerical variables. Finally, the package **micemd** (Audigier and Resche-Rigon 2017) also provides multilevel imputation functions for binary and integer variables, but not for categorical variables with more than two classes.

One of the limitations of the imputation methods using linear multilevel modeling is that they require distributional and model specification. This *a priori* selection makes it hard for the users to remain flexible to perform estimation after the imputation is done, especially if different model specifications are to be compared in the inference phase. In that sense, users should aim to produce imputation models that are at least as general as the analysis model (Grund, Lüdtke, and Robitzsch 2018).

The **biokNN** package implements the bi-objective k-Nearest Neighbors (biokNN) imputation method proposed by Cubillos, Wulff, and Wøhlk (2021). The biokNN method is specifically designed to provide imputation for multilevel data using a bi-objective function that weights the estimations given by kNN and the averaged values of the values in the same class. In contrast with the other multilevel imputation methods available in R, this method is both simple and flexible since it does not require model specification. The biokNN method is particularly useful when imputation step needs to be as general as possible, for example when multiple analysis model are to be tested in the analysis phase. By design, this method admit arbitrary missing patterns and do not require specific joint distributional assumptions on the data.

The rest of this article is structured as follows. First, Section biokNN imputation algorithm presents a short overview of the biokNN algorithm. Next, Section Overview of biokNN shows the main functionalities of the **biokNN** R package, including a detailed description of it mains imputation functions. Section Example with simulated data ilustrates the application of the functions in the package focusing on its visualization tools, using a simulated multilevel dataset. Then, Section Example using `machine` use the `machine` dataset from the UCI Repository (Dua and Graff 2017) to illustrate the calibration process. Finally, the article presents an overall summary in Section Summary.

2 biokNN imputation algorithm

In the following we present a short overview of the biokNN algorithm and how it works. First, three input parameters are required: the number of neighbors `k`, the weighting parameter `alpha`, and the number of iterations `nIter`. As an starting point, the missing values are imputed by randomly assigning a sample for each variable, and it is set as the start solution `X0`. Then for each iteration it proceed with two parts. First, the algorithm updates the neighbor assignment by computing the distance matrix between the observations with at least one missing value in the original dataset `X` and the rest of the observations. For each observation in the set of indeces with missing values (`M`) the distances are sorted and the `k` observations with the smaller distance are selected. Second, the imputed values `w_iv` are updated. In this step, each imputed value is updated individually using a weighted average between the neighbors mean value `w_neigh` and the class mean value `w_class`.

```
Input: X  dataset with missing values at indexes (i, v) in M
Input parameters: k, alpha, nIter
X_0 initial dataset imputed using random samples
X <- X0
while(iter < nIter)
    Update neighbors assignment:
    for each i in M
        compute distance between i and all observations in X
        sort the computed distances
        select the k observations with the smallest distances
    end for
    Update the imputation for each missing value (i, v)
    for each (i, v) in M
        compute the class mean value w_class
        compute the neighbors mean value w_neigh
        assign w_iv <- alpha*w_neigh + (1-alpha)*w_class
    end for
end while
Output: X dataset with imputed values
```

3 Overview of biokNN

The biokNN package is available on Github (biokNN on Github) and consists of two imputation and four visualization functions. Table 1 shows an overview of the functions of the package with a small description of it usage.

Table 1: Overview of the functions in biokNN.

function.	Description
biokNN.impute	Returns a complete dataset using the biokNN imputation method
biokNN.impute.mi	Returns m complete datasets for multiple imputation
calibrate	Allows to select the best pair of parameters in the biokNN method
create.multilevel	Creates a simulated dataset with a multilevel structure
pattern.plot	Plots the pattern of the missing values by class and by variable
missing.plot	Plots the frequency of the missing values by class and by variable
target.boxplot	Plots a boxplot of the observations by class using a specific variable

The function that implements the biokNN imputation method is `biokNN.impute()`, which receives the following parameters:

- **data**: A dataframe with missing values. The function requires data continuous variables and one complete class-variable containing the class of each observation.
- **className**: The name of the variable containing the classes.
- **nIter**: Number of iterations of the built-in biokNN algorithm. Default is 10.
- **weight**: Weight parameter. Default is 0.5.
- **k**: Number of neighbors. Default is 10.
- **distance**: Distance function used to calculate the neighbor assignment. It can be "Euclidian" (default), "gower," or "Manhattan."

In order to select the two parameters of the method, which are the weight paramenter (α) and the number of neighbors (k), the package provides a function that returns the pair of values that provides best imputation accuracy. The user can select the set of values to iterate for both parameters (for example, `c(0.1, 0.5, 0.9)` for α, and `c(10, 20, 30)` for k). The method will randomly generate extra missing values for a given percentage, from the original data with missing entries, and select the pair that minimizes the RMSE. It is implemented in the function `calibrate()` which takes the following parameters:

- **data**: A dataframe with missing values. The data requires continuous variables and one complete class-variable containing the class of each observation.
- **prop_valid**: Proportion of values to make missing to perform the calibration. Default is 0.2.
- **nIter**: Number of iterations of the built-in biokNN algorithm. Default is 10.
- **weight_space**: vector with the values in which the weight parameter iterates. Default is `seq(0, 1, 0.1)`.
- **k_space**: vector with the values in which k iterates. Default is `c(10, 15, 20)`.

Up to this point, the imputation techniques using biokNN have been described for single imputation. Multiple imputation (MI) (Rubin 1987) uses the distribution of the observed data to estimate a set of likely values of the data that are missing rather than a single value (Wulff and Ejlskov 2017). MI estimates the complete dataset `m` times accounting for the random components in the estimation. Then, the `m` imputed datasets are pooled together to produce inference across them. The MI extension of biokNN is implemented in the function `biokNN.impute.mi()`.

In the following sections we present two examples to ilustrate the functions of **biokNN**. First, we present an example using a simulated multilevel dataset, including a function that generates allows to generate the data integrated in the package. This example focuses on the visualization functions. Then, we present an example using a dataset from the UCI repository to focus on multiple imputation.

4 Example with simulated data

Multilevel data can be simulated by using linear regression with varying intercept and varying slopes. Such models consider a target variable y_{ij} that depends linearly on an independent variable X_{ij}, and a class variable, where $i \in \{1...n\}$ is the ith observation and $j \in \{1...Q\}$ is the jth class. The class variable contains the assignment of Q classes, each one with Q_s observations each. The model can be formulated as follows:

$$
\begin{aligned}
y_{ij} &= \beta_{0j} + \beta_{1j} X_{ij} + \epsilon \\
\beta_{0j} &\sim \mathcal{N}(\mu_0, \tau_0) \\
\beta_{1j} &\sim \mathcal{N}(\mu_1, \tau_1) \\
\epsilon &\sim \mathcal{N}(0, \sigma)
\end{aligned}
$$

where τ_0 and τ_1 represent the random effect of the intercepts and slopes among classes, respectively; and σ corresponds to the overall random error of the model. The parameters μ_0 and μ_1 represent the average effect of the intercept and the slope, respectively. In this case there is one independent variable X_1, but it can be extended to multiple regression by including an intercept and slope distributions for each independent variable.

The **biokNN** package provides a function called `create.multilevel()`, which generates a complete dataset using the linear varying slopes model. The function allows to select the number of classes, the number of independent variables, the average number of observations per class, the variance of the number of observations per class, and specify the parameters of the intercept and slope distributions. In this example, we consider `nClass=20` classes, `nVars=1` a single independent variable, and `classMean=20` observations per class (by setting the variance `classSD` to zero).

```r
set.seed(12345)
df <- create.multilevel(nClass = 50, nVars = 1,
                        classMean = 20, classSD = 0,
                        beta0 = 0, tau0 = 3,
                        beta = c(1), tau = c(1),
                        sigma2 = 1)
df[1:10, ]
```

```
##    class          y          X
## 1      1  1.4569678  1.1480914
## 2      1  3.7997005  0.4550137
## 3      1  2.2342012  2.0219464
## 4      1  1.9624434 -0.6952704
## 5      1  1.4219792  1.6039653
## 6      1  2.3180081  1.3517073
## 7      1  2.5399288  0.4395146
## 8      1  2.9099682  1.5716217
## 9      1  3.5364914 -1.7412532
## 10     1 -0.9015718 -1.4840148
```

We can generate missing values randomly by using the function `ampute()` from the **mice** package. In this example, we consider a 20% of missing values in the continous variables (both ´y´ and ´x´), while the class variable has to be complete. We assume a Missing Completely at Random (MCAR) pattern:

```
set.seed(12345)
df_obs <- df
clust_var <- df_obs$class
df_obs$class <- NULL
df_miss <- ampute(df_obs, prop = 0.1, mech = "MCAR")$amp
df_miss$class <- clust_var
df_miss <- df_miss %>% select(class, everything())
df_obs <- df_obs %>% mutate(class = clust_var) %>% select(class, everything())
df_miss[1:10, ]
```

```
##    class          y          X
## 1      1  1.4569678  1.1480914
## 2      1  3.7997005  0.4550137
## 3      1  2.2342012  2.0219464
## 4      1  1.9624434 -0.6952704
## 5      1  1.4219792  1.6039653
## 6      1  2.3180081  1.3517073
## 7      1  2.5399288  0.4395146
## 8      1  2.9099682  1.5716217
## 9      1  3.5364914 -1.7412532
## 10     1 -0.9015718 -1.4840148
```

We can explore the structure of the missing values by plotting the entries by class and by observation. We can use the function `pattern.plot()` to obtain a plot with the entries by variable. The white boxes represent the missing values, and the graduation of colors between white and black represents the value of the observed observation. In this example, all classes (20) have the same number of observations (20).

```
pattern.plot(df_miss, class)
```

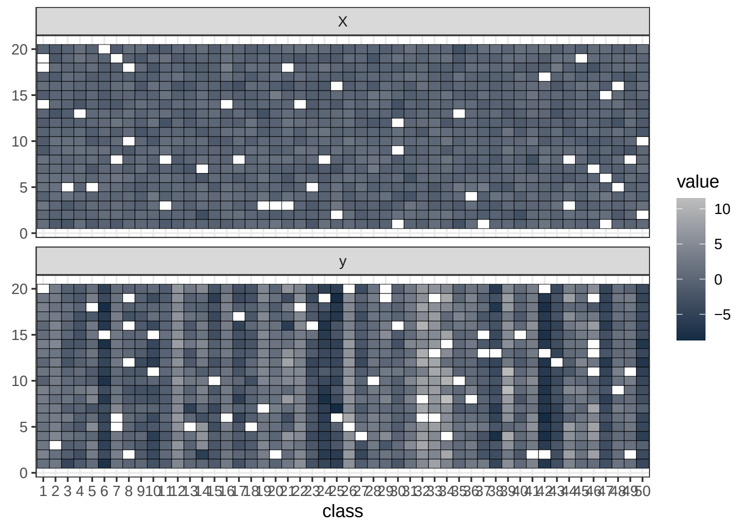

Figure 1: Missing values pattern by class and by variable.

Another useful visualization is to have the number of missing values per class, particularly to detect strctures like Missing at Random (MAR) or Missing Not at Random (MNAR), in which the missing values can be unbalanced and concentrate in some of the classes. Using `missing.plot()` we can obtain a barplot with the number of missing values per class per variable:

```r
missing.plot(df_miss, "class")
```

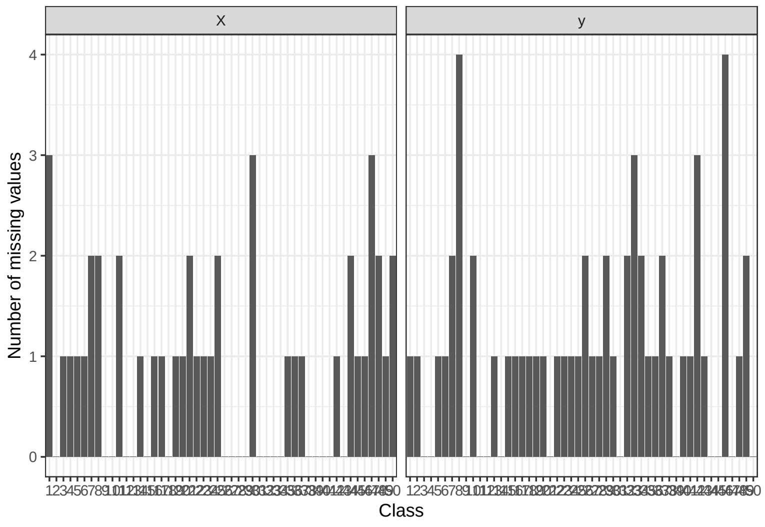

Figure 2: Amount of missing values by class and by variable.

In a multilevel dataset, the correlation between classes and between observations within the same class define how suitable the biokNN method can be, and influence the selection of the weight parameter. In order to get an idea of how similar/different observations are among classes, we can use `target.boxplot()` to get a boxplot for each class for an specific target variable:

```r
target.boxplot(df_miss, "y", "class")
```

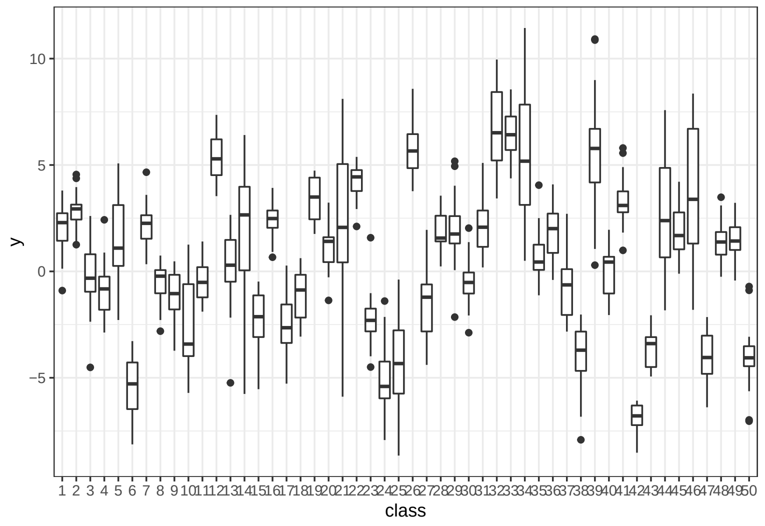

Figure 3: Amount of missing values by class and by variable.

Before the imputation step, biokNN requires to select two parameters: the weight (`weight`) and the number of neighbors (`k`). This can be done either by using prior or expert knowledge or by experiments with different settings. To facilitate this task, the function `calibrate()` can receive the amount of missing values to be taken from the original dataset to compare imputation accuracy, and the search space of the two parameters. For example, we can use 10% extra missing values to test, and try two valus for the weight and for k. The function returns a vector in which the first value correspond to the weight and the second to k, of the best combination:

```
set.seed(123456)
(calibrate(df_miss,
            prop_valid = 0.1,
            weight_space = c(0.1, 0.9),
            k_space = c(10, 20)))
```

```
## [1]  0.9 10.0
```

The package provides two imputation functions that uses the biokNN algorithm, one for single and other for multiple imputation. For single imputation, we can use `biokNN.impute()` to obtain a complete dataset:

```
df_imp <- biokNN.impute(df_miss,
                        className = 'class',
                        nIter = 10,
                        weight = 0.9,
                        k = 10,
                        distance = "gower")
df_imp[1:10, ]
```

```
##    class          y          X
## 1      1  1.4569678  1.1480914
## 2      1  3.7997005  0.4550137
## 3      1  2.2342012  2.0219464
## 4      1  1.9624434 -0.6952704
## 5      1  1.4219792  1.6039653
## 6      1  2.3180081  1.3517073
## 7      1  2.5399288  0.4395146
## 8      1  2.9099682  1.5716217
## 9      1  3.5364914 -1.7412532
## 10     1 -0.9015718 -1.4840148
```

For multiple imputation, we use `biokNN.impute.mi()` which returns a list with `m` (Default is `m=5`) complete datasets which can be pooled into a single model estimation:

```
df_imp_mi <- biokNN.impute.mi(df_miss,
                              className =  'class',
                              m = 3)
str(df_imp_mi)
```

```
## List of 3
##  $ :'data.frame':    1000 obs. of  3 variables:
##   ..$ class: Factor w/ 50 levels "1","2","3","4",..: 1 1 1 1 1 1 1 1 1 1 ...
##   ..$ y    : num [1:1000] 1.46 3.8 2.23 1.96 1.42 ...
##   ..$ X    : num [1:1000] 1.148 0.455 2.022 -0.695 1.604 ...
##  $ :'data.frame':    1000 obs. of  3 variables:
##   ..$ class: Factor w/ 50 levels "1","2","3","4",..: 1 1 1 1 1 1 1 1 1 1 ...
##   ..$ y    : num [1:1000] 1.46 3.8 2.23 1.96 1.42 ...
##   ..$ X    : num [1:1000] 1.148 0.455 2.022 -0.695 1.604 ...
##  $ :'data.frame':    1000 obs. of  3 variables:
##   ..$ class: Factor w/ 50 levels "1","2","3","4",..: 1 1 1 1 1 1 1 1 1 1 ...
##   ..$ y    : num [1:1000] 1.46 3.8 2.23 1.96 1.42 ...
##   ..$ X    : num [1:1000] 1.148 0.455 2.022 -0.695 1.604 ...
```

5 Example using `machine`

In this section we ilustrate the use of the functions with a dataset from the UCI repository. The second variable of this dataset was removed since it contains categorical values and does not add relevant information regarding this example. We can get the dataset by using:

```
##    adviser X125  X256 X6000 X256.1 X16 X128 X198 X199
## 1   amdahl   29  8000 32000     32   8   32  269  253
## 2   amdahl   29  8000 32000     32   8   32  220  253
## 3   amdahl   29  8000 32000     32   8   32  172  253
## 4   amdahl   29  8000 16000     32   8   16  132  132
## 5   amdahl   26  8000 32000     64   8   32  318  290
## 6   amdahl   23 16000 32000     64  16   32  367  381
## 7   amdahl   23 16000 32000     64  16   32  489  381
## 8   amdahl   23 16000 64000     64  16   32  636  749
## 9   amdahl   23 32000 64000    128  32   64 1144 1238
## 10  apollo  400  1000  3000      0   1    2   38   23
```

The class variable seems to be `adviser`. We again generate missing values randomly using a 20% of missingess.

```
set.seed(12345)
df_obs <- machine
clust_var <- df_obs$adviser
df_obs$adviser <- NULL
df_miss <- ampute(df_obs, prop = 0.2, mech = "MCAR")$amp
df_miss$adviser <- clust_var
df_miss <- df_miss %>% select(adviser, everything())
df_obs <- df_obs %>% mutate(adviser = clust_var) %>% select(adviser, everything())
df_miss[1:10, ]
```

```
##    adviser X125  X256 X6000 X256.1 X16 X128 X198 X199
## 1   amdahl   29  8000 32000     32   8   32  269  253
## 2   amdahl   29  8000 32000     32   8   32  220  253
## 3   amdahl   29  8000 32000     32   8   32  172  253
## 4   amdahl   29  8000 16000     32   8   16  132  132
## 5   amdahl   26  8000 32000     64   8   32  318  290
## 6   amdahl   23 16000    NA     64  16   32  367  381
## 7   amdahl   23 16000 32000     64  16   32  489  381
## 8   amdahl   23 16000 64000     64  16   32  636  749
## 9   amdahl   23 32000 64000    128  32   64 1144 1238
## 10  apollo  400  1000  3000      0   1    2   38   23
```

We can use the `calibrate()` function to perform more specific search of parameters. For example, we can select sparce values as a first step and then perform a second search around the values that perform the best in imputation accuracy. In this function, we can set the parameter `print = TRUE` to print the parameters values and the RMSE obtained by using them:

```
set.seed(12345)
(calibrate(df_miss,
            prop_valid = 0.3,
           weight_space = c(0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1),
           k_space = c(10),
           print = TRUE))
```

```
## [1] "0.1,10,2500.37438055025"
## [1] "0.2,10,2449.675932956"
## [1] "0.3,10,2439.70434131216"
## [1] "0.4,10,2372.44305175299"
## [1] "0.5,10,2356.43084031269"
## [1] "0.6,10,2310.97199048813"
## [1] "0.7,10,2266.64852265491"
## [1] "0.8,10,2241.8805114179"
## [1] "0.9,10,2214.90510265878"
## [1] "1,10,2275.75260725361"

## [1]  0.9 10.0
```

In this case, the minimum RMSE was obtained by the pair of weight and k (0.9, 10). We can now look around the value 0.9 to see if we can improve the performance:

```
set.seed(12345)
(calibrate(df_miss,
            prop_valid = 0.3,
           weight_space = c(0.85, 0.87, 0.92, 0.95),
           k_space = c(10),
           print = TRUE))
```

```
## [1] "0.85,10,2267.56338979713"
## [1] "0.87,10,2165.70114930746"
## [1] "0.92,10,2270.66964810911"
## [1] "0.95,10,2156.13775644213"

## [1]  0.95 10.00
```

6 Summary

In this article we present an implementation in R of the bi-objective kNN (biokNN) imputation method proposed by Cubillos, Wulff, and Wøhlk (2021). This method is suited for continous datasets with a multilevel structure, in which the observations of a dataset are nested within higher-level units contained in a class variable. The **biokNN** package provides visualization tools to make easier the analysis of missing values with this structures. Compared with other imputation methods specifically designed for multilevel data available in R, this method does not require any model specification which makes it simple and generic.

7 References

Audigier, Vincent, and M Micemd Resche-Rigon. 2017. "Multiple Imputation by Chained Equations with Multilevel Data." *R Package*.

Buuren, S van, and Karin Groothuis-Oudshoorn. 2010. "Mice: Multivariate Imputation by Chained Equations in R." *Journal of Statistical Software*, 1–68.

Cubillos, M., J. Wulff, and S. Wøhlk. 2021. "A Bi-Objective k-Nearest Neighbors Based Imputation Method for Multilevel Data." *Submitted*.

Dua, Dheeru, and Casey Graff. 2017. "UCI Machine Learning Repository." University of California, Irvine, School of Information; Computer Sciences. http://archive.ics.uci.edu/ml.

Enders, Craig K, Stephen A Mistler, and Brian T Keller. 2016. "Multilevel Multiple Imputation: A Review and Evaluation of Joint Modeling and Chained Equations Imputation." *Psychological Methods* 21 (2): 222.

Garciarena, Unai, and Roberto Santana. 2017. "An Extensive Analysis of the Interaction Between Missing Data Types, Imputation Methods, and Supervised Classifiers." *Expert Systems with Applications* 89: 52–65.

Grund, Simon, Oliver Lüdtke, and Alexander Robitzsch. 2016. "Multiple Imputation of Multilevel Missing Data: An Introduction to the R Package Pan." *Sage Open* 6 (4): 2158244016668220.

———. 2018. "Multiple Imputation of Missing Data for Multilevel Models: Simulations and Recommendations." *Organizational Research Methods* 21 (1): 111–49.

Kowarik, Alexander, and Matthias Templ. 2016. "Imputation with the r Package VIM." *Journal of Statistical Software* 74 (7): 1–16.

Quartagno, Matteo, Simon Grund, and James Carpenter. 2019. "Jomo: A Flexible Package for Two-Level Joint Modelling Multiple Imputation." *R Journal* 9 (1).

Rubin, Donald B. 1987. "Multiple Imputation for Survey Nonresponse." New York: Wiley.

Wulff, Jesper N, and Linda Ejlskov. 2017. "Multiple Imputation by Chained Equations in Praxis: Guidelines and Review." *Electronic Journal of Business Research Methods* 15 (1): 41–56.