R package imputeTestbench to compare imputations methods for univariate time series

Article · November 2016

CITATIONS
0

READS
13

4 authors:

Neeraj Dhanraj Bokde
Visvesvaraya National Institute of Technology
17 PUBLICATIONS **7** CITATIONS
SEE PROFILE

K.D. Kulat
Visvesvaraya National Institute of Technology
69 PUBLICATIONS **99** CITATIONS
SEE PROFILE

Marcus W. Beck
United States Environmental Protection Age…
18 PUBLICATIONS **119** CITATIONS
SEE PROFILE

Gualberto Asencio Cortés
Universidad Pablo de Olavide
27 PUBLICATIONS **43** CITATIONS
SEE PROFILE

R package imputeTestbench to compare imputations methods for univariate time series

by Neeraj Bokde, Kishore Kulat, Marcus W Beck, Gualberto Asencio-Cortés

Abstract This paper describes the R package **imputeTestbench** that provides a testbench for comparing imputation methods for missing data in univariate time series. The **imputeTestbench** package can be used to simulate the amount and type of missing data in a complete dataset and compare filled data using different imputation methods. The user has the option to simulate missing data by removing observations completely at random or in blocks of different sizes. Several default imputation methods are included with the package, including historical means, linear interpolation, and last observation carried forward. The testbench is not limited to the default functions and users can add or remove additional methods using a simple two-step process. The testbench compares the actual missing and imputed data for each method with different error metrics, including *RMSE*, *MAE*, and *MAPE*. Alternative error metrics can also be supplied by the user. The simplicity of use and significant reduction in time to compare imputation methods for missing data in univariate time series is a significant advantage of the package. This paper provides an overview of the core functions, including a demonstration with examples.

Introduction

The CRAN repository includes many packages for imputing missing values. These packages have been used in a wide range of applications including biomedical research, civil and urban planning, design of medical care systems, and ecological research. Several packages are well recognized and easy to use, including **MICE**, **mi**, **Amelia**, and **missForest**. The **MICE** (Buuren and Groothuis-Oudshoorn, 2011) (Multivariate imputation via Chained equation) package provides several imputation methods for time series with data that are Missing at Random (*MAR*). Imputation methods used by **MICE** include Predictive mean matching (*PMM*), logistic regression, Bayesian polytomous regression, and proportional odds model. The **mi** (Multiple imputation with diagnostic) (Su et al., 2011) package also uses the predictive mean matching technique, whereas bootstrapping and the EMB algorithm can be used for *MAR* data in the **Amelia** package (Honaker et al.). An alternative approach is provided by the **missForest** package (Stekhoven and Bühlmann, 2012) that uses a Random Forest algorithm for imputation. Several other methods and packages are provided by CRAN as discussed in the CRAN Task View: Official Statistics & Survey Methodology. Given the amount and types of available methods, the **imputeTestbench** R Package is proposed as a testbench for efficient comparisons to inform the use of imputation methods.

Studies that have compared imputation methods have used similar approaches to evaluate the superiority of one method over another. For example, Zhu et al. (2011) proposed a kernel-based iterative estimation method for missing data and compared this method to a non-parametric iterative signal kernel method, non-parametric iterative signal with an RBF kernel, the traditional kernel non-parametric missing value method, and other conventional frequency estimators. The methods were compared by simulating different amounts of missing data, predicting the missing values with each method, and then comparing the predictions to the removed data using the standard root-mean squared error (*RMSE*). Table 1 reproduces the results, where the rows show *RMSE* for each imputation method at 10% and 80% missing data.

Similarly, Tak et al. (2016) proposed an imputation method based on a modified k-nearest neighbour approach that accounted for the effects of spatial and temporal correlation between observations. Missing observations were simulated by removing values from 0.1% to 50% of the complete data, and then imputed with the proposed method, the nearest history (NH) method, bootstrapping based expectation maximization (B-EM), and the maximum likelihood estimation (MLE) method. The imputation methods were compared using *RMSE*, mean absolute percent error (*MAPE*), and percent change in variance (*PCV*). Additional comparisons in Oh et al. (2011); Jörnsten et al. (2007); Li et al. (2015); Nguyen et al. (2013); Sim et al. (2015); Li et al. (2004); Ran et al. (2015) have used a similar workflow to compare the performance of imputation methods for missing data. This general procedure is summarized in Figure 1. The **imputeTestbench** package formalizes this approach by providing several functions that can greatly simplify the comparison of imputation methods.

	10%		80%	
Method names	*T*	*V*	*T*	*V*
Mixing	8	0.085	20	1.53
Poly	10	0.103	25	2.11
RBF	11	0.107	29	2.86
Normal	14	0.121	30	3.01
FE	13	0.117	29	2.59

Table 1: Comparison of imputation methods by varying the amount of missing data (10% and 80%) and number of iterations. Reproduced from Zhu et al. (2011). *T* is the number of iterations for each imputation method and *V* is the mean *RMSE* of the imputed values.

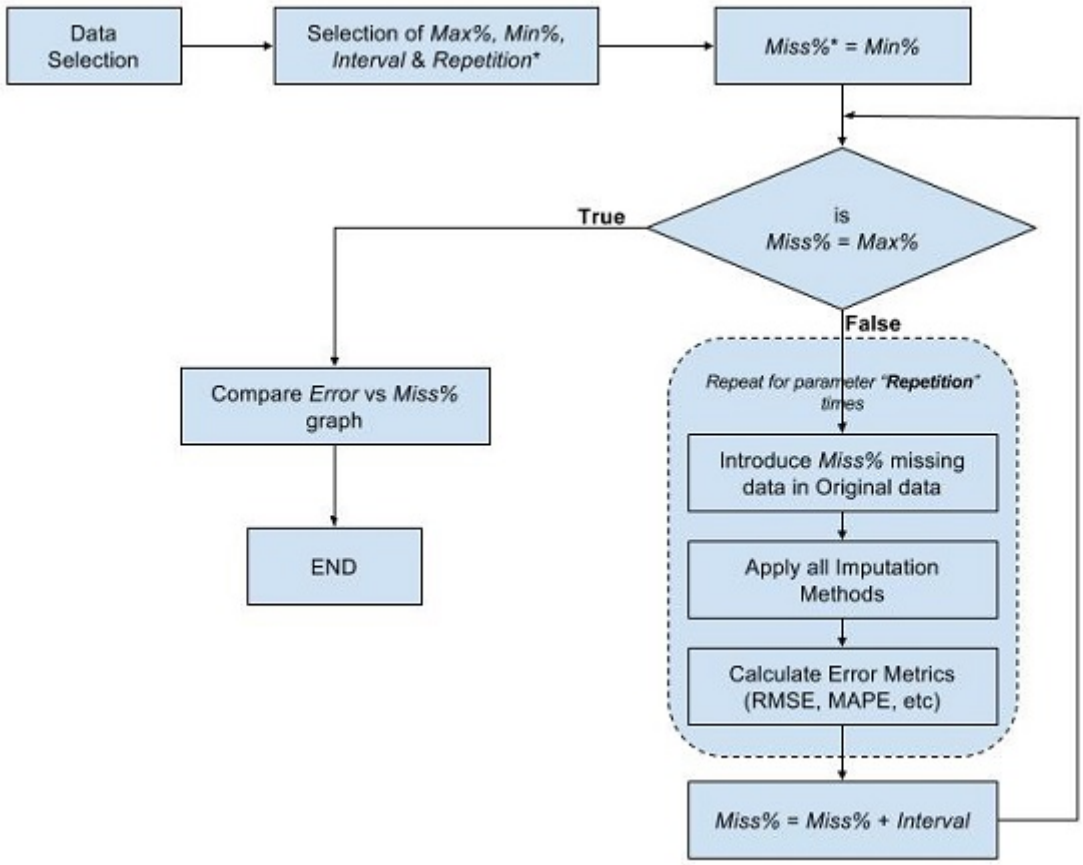

Figure 1: Workflow diagram for comparing imputation methods. *Min*% and *Max*% are minimum and maximum percent of missing values in the dataset. Details are described below.

Overview of imputeTestbench

This section introduces the **imputeTestbench** R package. The package **imputeTestbench** (Bokde and Beck, 2016) can be used to evaluate imputation methods for univariate time series by simulating missing data and comparing the predictions to the actual. Previous analyses have manually evaluated the performance of different methods by noting the errors at different percentages of missing values for several repetitions with different error metrics. This approach can be time consuming and prone to errors. Moreover, the relative accuracies of different performance evaluations is a concern given the unavailability of a common platform for comparison.

The **imputeTestbench** package is introduced to address the above issues. The package imports **dplyr** (Wickham and Francois, 2016), **forecast** (Hyndman, 2016), **ggplot2** (Wickham, 2009), **imputeTS** (Moritz, 2015), **reshape2** (Wickham, 2007), **tidyr** (Wickham, 2016), and **zoo** (Zeileis and Grothendieck, 2005). Five relevant functions are included in **imputeTestbench**. The primary function is `impute_errors()` which is used to evaluate different imputation methods for missing data that are randomly generated from a complete dataset. The method for generating missing data for imputation in the test or user-supplied dataset is of particular importance and different methods are provided by the `sample_dat()` function. The evaluation methods for the imputed data are in the `error_functions()` function. The remaining two functions, `plot_impute()` and `plot_errors()`, are used to visualize results and error summaries for the imputation methods. The package also allows users to include additional imputation methods and error functions as needed.

The `impute_errors()` function:

The `impute_errors()` function includes thirteen arguments as discussed below. This function evaluates the precision of different imputation methods based on changes in the amount and type of missing observations from the complete dataset. The default imputation functions included in `impute_errors()` are `na.approx()` (**zoo**), `na.interp()` (**forecast**), `na.interpolation()` (**imputeTS**), `na.locf()` (**zoo**), and `na.mean()` (**imputeTS**). None of the arguments are required since all of them include default or `NULL` values. The syntax is shown below.

```
impute_errors(dataIn = NULL, smps = "mcar", methods = c("na.approx",
  "na.interp", "na.interpolation", "na.locf", "na.mean"), methodPath = NULL,
  errorParameter = "rmse", errorPath = NULL, blck = 50, blckper = TRUE,
  missPercentFrom = 10, missPercentTo = 90, interval = 10,
  repetition = 10, addl_arg = NULL)
```

`dataIn`:

A `ts` (**stats**) object that will be evaluated. The input object is a complete dataset with no missing values. Missing observations are generated randomly for performance evaluation and comparison of imputation methods. The default dataset if `dataIn = NULL` is `nottem`, a time series object of average air temperatures recorded at Nottingham Castle from 1920 to 1930. This dataset is included with the base **datasets** package.

`smps`:

The desired type of sampling method for removing values from the complete time series provided by `dataIn`. Options are `smps = 'mcar'` for missing completely at random (default) and `smps = 'mar'` for missing at random. Both methods provide completely different approaches to generating missing data in time series, as described below.

`methods`:

Methods that are used to impute the missing values generated by `smps`. All five default methods are used unless the argument is changed by the user. For example, `methods = 'na.approx'` will use only `na.approx()` with `impute_errors()`. Methods not included with the default options can be added by including the name of the function in `methods` and providing the path to the script in `methodPath`. Additional arguments passed to each method can be included in `addl_arg` described below.

`methodPath`:

A character string for the path of the user-supplied script that includes one to many methods passed to `methods`. The path can be absolute or relative within the current working directory for the R session. The function sources the file indicated by `methodPath` to add the user-supplied function to the global environment.

`errorParameter`:

The error metric used to compare the true, observed values from `dataIn` with the imputed values. Commonly used error metrics are Root Mean Square Error (*RMSE*), Mean Absolute Percent Error (*MAPE*) and Mean Absolute Errors (*MAE*). These measures are included with **imputeTestbench** and can be used to evaluate the imputed observations by specifying `errorParameter = 'rmse'` (default), `'mape'`, or `'mae'`. Additional error measures can be provided as user-supplied functions where the first argument is observed values (numeric) and the second is the imputed values (numeric). The user-supplied function must return a single numeric value as the error measure. Examples below demonstrate the addition of a user-supplied function.

`errorPath`:

A character string for the path of the user-supplied script that includes one to many error methods passed to `errorParameter`.

`blck`:

The block size for missing data if the sampling method is at random, `smps = 'mar'`. The block size can be specified as a percentage of the total amount of missing observations in `interval` or as a number of time steps in the input dataset.

`blckper`:

A logical value indicating if the numeric value passed to `blck` is a percentage (`blckper = TRUE`) or a count of time steps (`blckper = FALSE`). This argument only applies if `smps = 'mar'`.

`missPercentFrom`, `missPercentTo`:

The minimum and maximum percentages of missing values, respectively, that are introduced in `dataIn`. Appropriate values for these arguments are 10 to 90, indicating a range from few missing observations to almost completely absent observations.

`interval`:

The interval of missing data from `missPercentFrom` to `missPercentTo`. The default value is 10% such that missing percentages in `dataIn` are evaluated from 10% to 90% at an interval of 10%, i.e., 10%, 20%, 30%, ..., 90%. Combined, these arguments are identifical to `seq(from = 10,to = 90,by = 10)`.

`repetition`:

The number of repetitions at each `interval`. Missing values are placed randomly in the original data such that multiple repetitions must be evaluated for a robust comparison of the imputation methods.

Considering the default values, the `impute_errors()` function returns an `errprof` object as the *error profile* for the imputation methods:

```
library(imputeTestbench)
set.seed(123)
a <- impute_errors()
a
## $Parameter
## [1] "rmse"
##
## $MissingPercent
```

```
## [1] 10 20 30 40 50 60 70 80 90
##
## $na.approx
## [1] 0.84 1.33 1.95 3.01 3.80 4.89 6.61 8.39 9.94
##
## $na.interp
## [1] 0.78 1.11 1.44 1.65 1.90 2.06 2.34 2.57 2.96
##
## $na.interpolation
## [1]  0.84  1.35  2.00  3.02  3.98  5.04  6.76  8.52 10.15
##
## $na.locf
## [1]  1.7  2.7  3.8  5.2  6.3  7.8  9.3 10.5 11.4
##
## $na.mean
## [1] 2.6 3.8 4.7 5.4 6.1 6.6 7.2 7.7 8.2
```

The `errprof` object is a list with seven elements. The first element, `Parameter`, is a character string of the error metric used for comparing imputation methods. The second element, `MissingPercent`, is a numeric vector of the missing percentages that were evaluated in the input dataset. The remaining five elements show the average error for each imputation method at each interval of missing data in `MissingPercent`. The averages at each interval are based on the repetitions specified in the initial call to `impute_errors()` where the default is `Repetition = 10`. Although the print method for the `errprof` object returns a list, the object stores the unique error estimates for every imputation method, repetition, and missing data interval. These values are used to estimate the averages in the printed output and to plot the distribution of errors with `plot_errors()` shown below. All error values can be accessed as follows.

```
attr(a, 'errall')
```

Viewing results from `impute_errors()`

The `plot_errors()` function can be used to plot summaries of the error metrics for each method used in `impute_errors()`. This function uses the `errorprof` object as input and returns a graph of error values to compare the different imputation methods across the interval of missing data. Three plot types are provided by `plot_errors()` and are specified with the `plotType` argument. The default value is `plotType = 'boxplot'` that graphs the distribution of error values for each method and missing data interval using boxplot summaries (i.e., 25th, 50th, and 75th percentile shown by the box, whiskers as 1.5 times interquartile range, and outliers beyond). The boxplots are created using all error values stored in the `'errall'` attribute of the `errprof` object.

```
plot_errors(a)
```

The bar and line options for `plotType` show the average error values for each repetition. Similar information is shown as the `boxplot` option, although the range of error values for each imputation method and percent of missing observations is not shown.

```
plot_errors(a, plotType = 'line')
```

Sampling methods for missing observations

The `impute_errors()` function uses the `sample_dat()` function to remove observations for imputation from the input dataset. The `sample_dat()` function removes observations using one of two methods that are relevant for univariate time series. Observations can be removed following a missing completely at random (MCAR) or missing at random (MAR) sampling scheme with the appropriate `smps` argument. The MCAR sampling scheme assumes all observations have equal probability of being selected for removal and is appropriate for univariate time series that are not serially correlated (i.e., no temporal dependence). Conversely, the MAR sampling scheme selects observations in blocks such that the probability of selection for a single observation depends on whether an observation closer in time was also selected. The MAR scheme is appropriate for time series with serial correlation. For example, missing data may occur in univariate time series if monitoring equipment fail for a period of time or data are not collected on the weekends. The `sample_dat` function has the following syntax:

```
sample_dat(datin, smps = "mcar", repetition = 10, b = 50, blck = 50,
  blckper = TRUE, plot = FALSE)
```

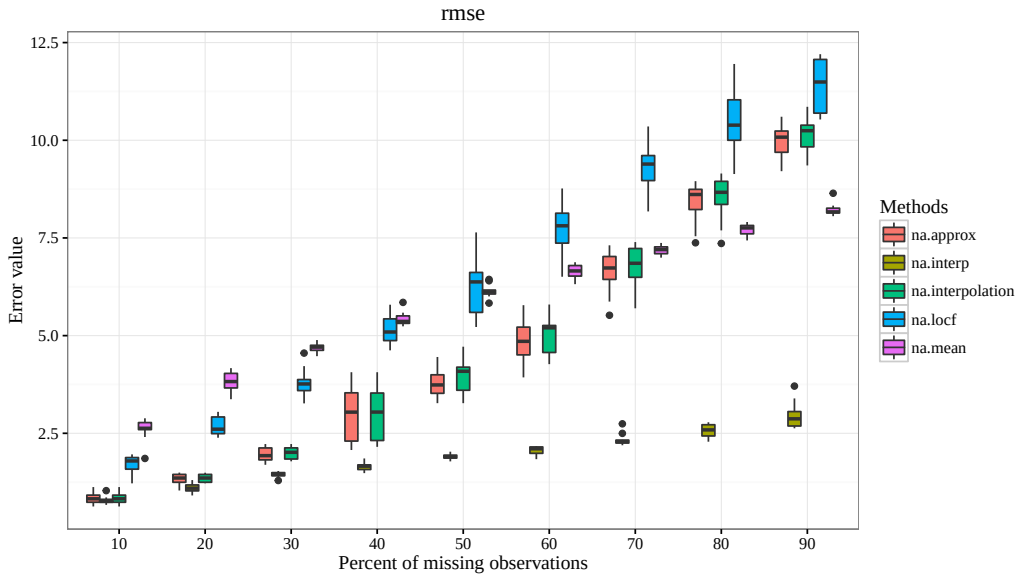

Figure 2: Distribution of error values for each imputation method and interval of missing observations.

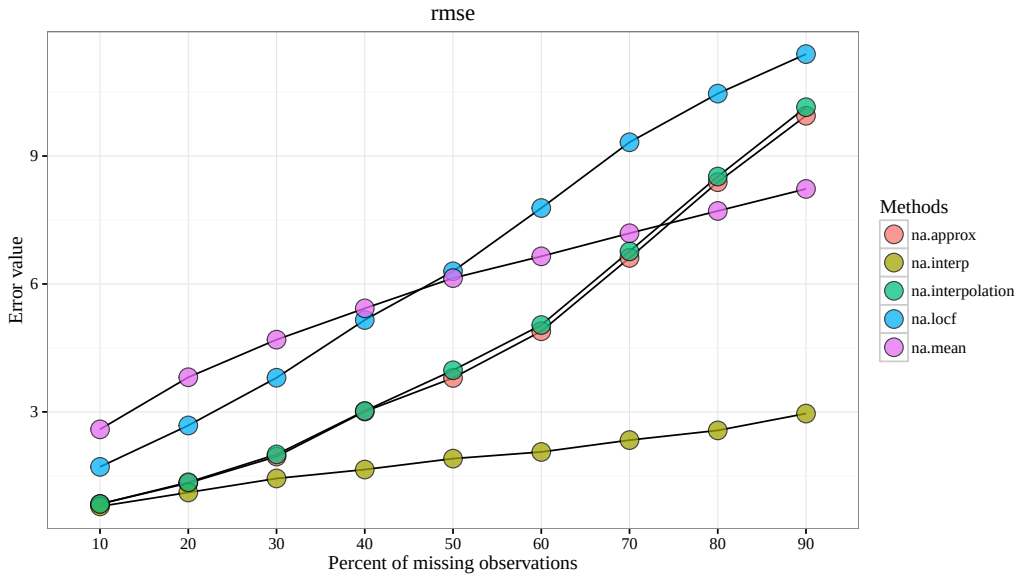

Figure 3: Average error values for each imputation method and interval of missing observations. The `line` option is used for `plot_errors()`.

`datin`:

Input numeric vector, inherited from `dataIn` from `impute_errors`.

`smps, repetition, blck, blckper`:

Arguments that are inherited as is from `impute_errors` indicating the sampling type (`smps`), number of repetitions for each missing data type (`repetition`), block size (`blck`), and block type as percentage or count (`blckper`).

`b`:

Numeric indicating the total amount of missing data as a percentage to remove from the complete time series. The values passed to `b` within `impute_errors` are those defined by `missPercentFrom`, `missPercentTo`, and `interval`.

`plot`:

Logical indicating if a plot is returned that shows one repetition of the sampling scheme defined by the arguments (see Figure 4).

The MCAR sampling scheme is used if `smps = 'mcar'` in the call to `impute_errors()`. The only relevant arguments for MCAR are `missPercentFrom`, `missPercentTo`, and `interval` that define the amount of data to remove as a percentage of the total. The amount of data to remove for each interval is passed to the `b` argument in `sample_dat`. The MAR sampling scheme requires additional arguments to control the block size for removing data in continuous chunks, in addition to the total amount of data to remove. The block size argument, `blck`, can be given as a percentage or as number of observations in sequence. The type of block size passed to `blck` is controlled by `blckper`, where `blckper = TRUE` indicates a percentage and `FALSE` indicates a count for `blck`. For example, if the total sample size of the dataset is 1000, `b = 50`, `blck = 20`, and `blckper = TRUE` means half the dataset is removed (`b = 50`, 500 observations) and each block will have 100 observations (20% of 500). For both percentages and counts, the blocks are automatically selected until the total amount of missing data is equal to that specified by `b`. Final blocks may be truncated to make the total amount of missing observations equal to `b`. The starting location of each block is selected at random and overlapping blocks are not uniquely counted for the required sample size given by `b`.

The `sample_dat()` function includes an optional `plot` argument. Although the function is primarily used within `impute_errors` to generate missing data, it can be used independently to visualize different sampling schemes. Figure 4 shows some examples of sampling completely at random (MCAR) and at random (MAR).

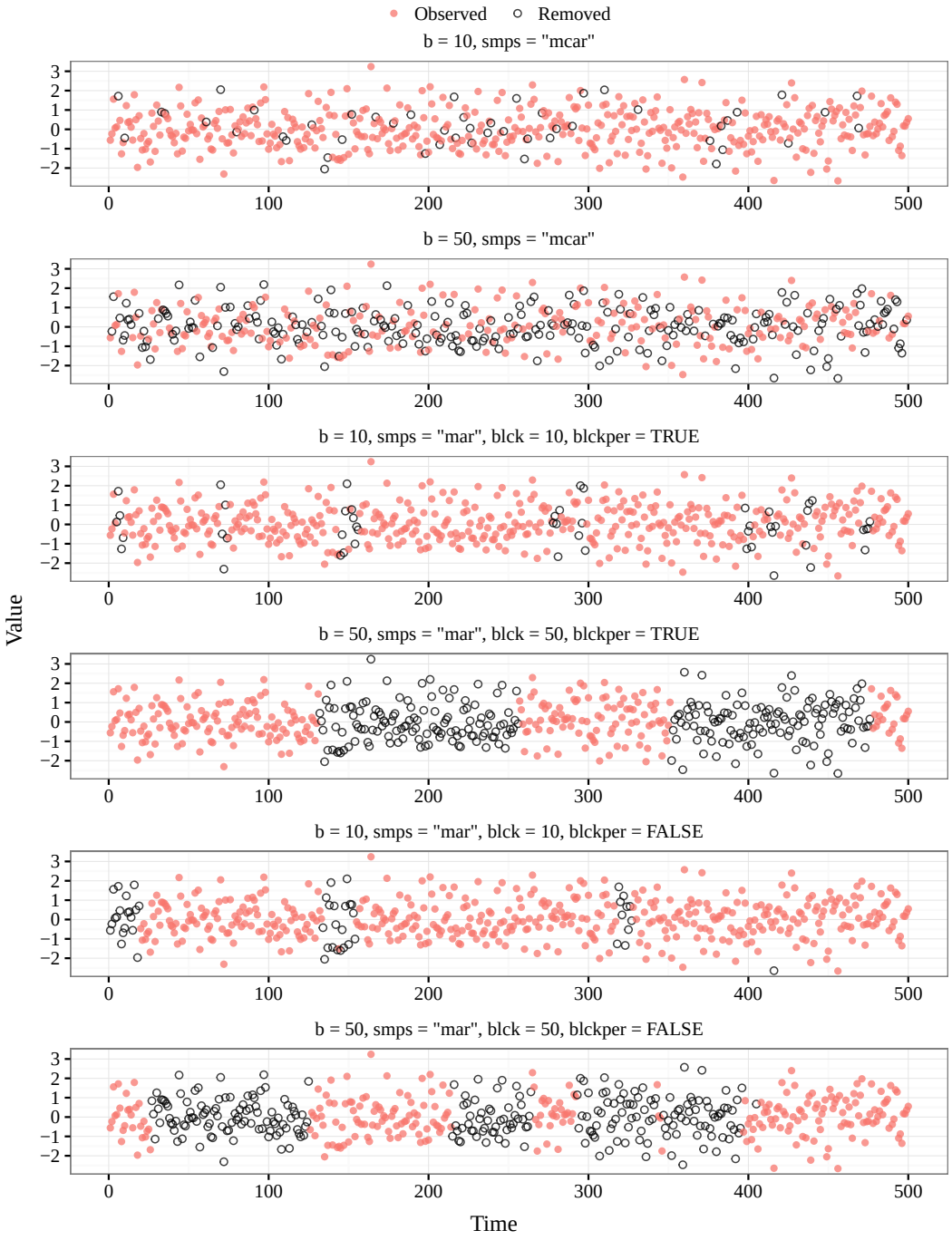

Figure 4: Examples of sampling schemes for missing data provided by `sample_dat()`. Values to be removed and imputed are shown as open circles and the data to be kept are in red. From top to bottom, sampling is completely at random (MCAR) with 10% missing, MCAR with 50% missing, sampling at random (MAR) with 10% missing and block size 10% of total missing, MAR with 50% missing and block size 50% of total missing, MAR with 10% missing and block size of ten observations, and MAR with 50% missing and block size of fifty observations.

The `plot_impute()` function

An additional plotting function available in **imputeTestbench** is `plot_impute()`. This function returns a plot of the imputed values for each imputation method in `impute_errors()` for one repetion of sampling with `sample_dat()`. The plot shows the results as a single facet for each method with the points colored as not filled or filled (i.e., original data not removed and filled data that were removed). An optional argument, `showmiss`, can be used to show the original values as open circles that were removed from the data. It should be noted that the plot from `plot_errors()` is a more accurate representation of the abilities of each method. The `plot_impute()` function shows results for only one simulation and missing data type (e.g., `smps = 'mcar'` and `b = 50`). This function is useful as a simple visualization of the sampling scheme for the missing values and the relative abilities of each method for imputation.

```
plot_impute(showmiss = T)
```

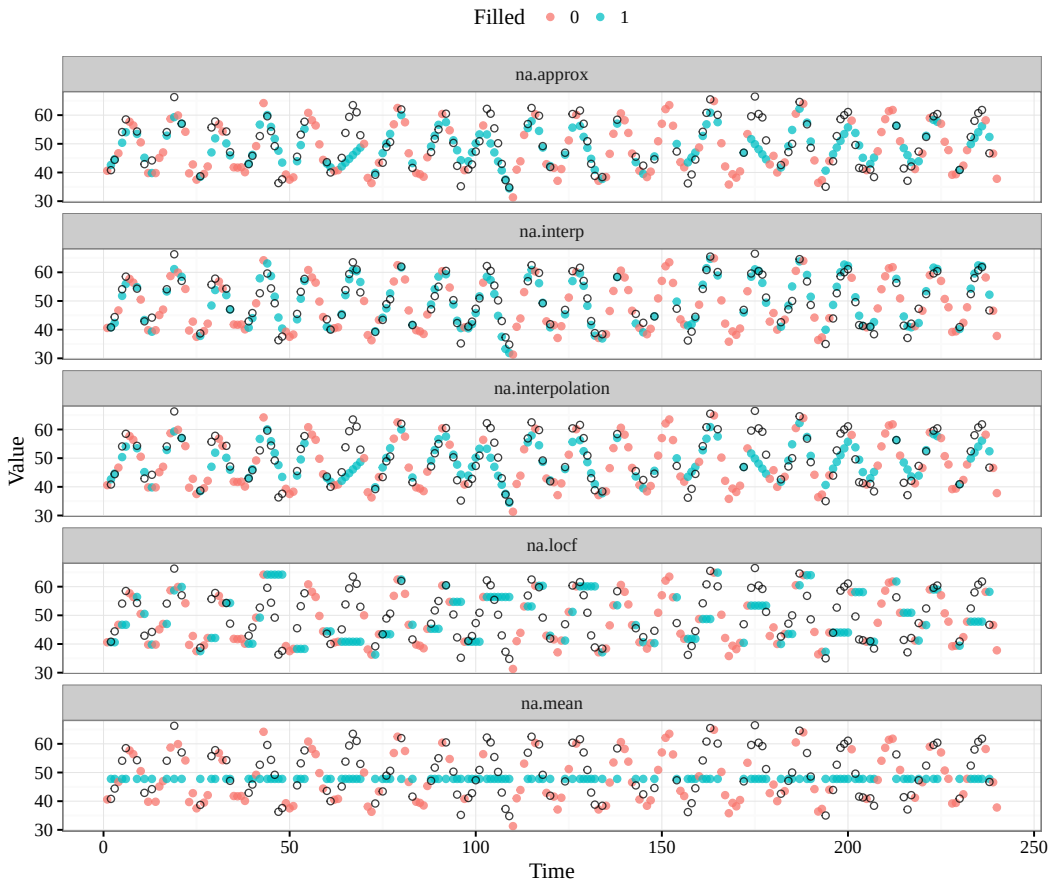

Figure 5: Output from the `plot_impute()` function that shows the data that were not removed (red), removed (open circles), and imputed (blue).

Adding error metrics and imputation methods

The `error_functions()` function is a collection of error metrics that are used to evaluate differences between the original and imputed data. This function is used internally within `impute_errors()` to compare results from the imputation methods. As described above, the available error metrics are *RMSE*, *MAE*, and *MAPE*. However, the proposed testbench is not limited to these metrics and alternative functions can be provided by the user. An alternative error metric can be used by specifying the path to the function using the `errorPath` argument in `impute_errors()`. The `errorParameter` argument must also be changed to the name of the function in `errorPath`. The user-supplied function should accept two arguments as input, the first being the observed time series and the second being the imputed data for comparison. The function must return a single numeric value that is the result of comparing the two input vectors.

Similarly, imputation methods can be added to `impute_errors()` by providing a path for an R script to the `methodPath` argument. The added imputation method must also be added as a character string to the `methods` argument. User-supplied imputation functions should have one required argument for the input `ts` object and should return an imputed vector of observations of the same length as the input object. Examples of adding error metrics and additional imputation methods are provided in the next section.

Additional arguments for user-supplied imputation functions can also be included. These arguments can be passed to `impute_errors` using the `addl_arg` argument, as can any arguments for the default imputation methods. The additional arguments are passed as a `list` of lists to the `addl_arg` argument, where the list contains one to many elements that are named by the methods in `methods`. The elements of the list are lists with arguments that are specific to each imputation method. For example, the default function `na.mean` has an additional `option` argument that specifies the algorithm to use for missing values, where possible values are `"mean"`, `"median"`, and `"mode"`. This argument can be changed from the default `option = "mean"` with `addl_arg` in `impute_errors`, as shown below. Arguments to user-supplied imputation functions can be changed similarly.

```
# changing the option argument for na.mean
impute_errors(addl_arg = list(na.mean = list(option = 'mode')))
```

Demonstration of imputeTestbench with examples

This example demonstrates how **imputTestbench** can be used to compare different imputation methods. The testbench is always initiated with the `impute_errors()` function, which uses time series data and additional arguments as discussed in the previous section. The example uses the `austres` dataset available in the **datasets** package. The `impute_errors()` function with the *RMSE* error metric and five default imputation methods returns the error profile below:

```
aus <- datasets::austres
ex <- impute_errors(dataIn = aus)
ex
## $Parameter
## [1] "rmse"
##
## $MissingPercent
## [1] 10 20 30 40 50 60 70 80 90
##
## $na.approx
## [1]  1.9  3.1  3.6  5.1  6.0  8.5 12.2 19.7 31.0
##
## $na.interp
## [1]   2.4   3.8   9.8  14.8  20.1  24.1  29.1  55.0 191.7
##
## $na.interpolation
## [1]   2.5   3.8   9.3  12.4  16.3  20.0  23.4  44.7 177.1
##
## $na.locf
## [1]  19  32  47  68  89 123 168 286 574
##
## $na.mean
## [1]  430  611  759  867  936 1064 1123 1257 1380
plot_errors(ex, plotType = 'line')
```

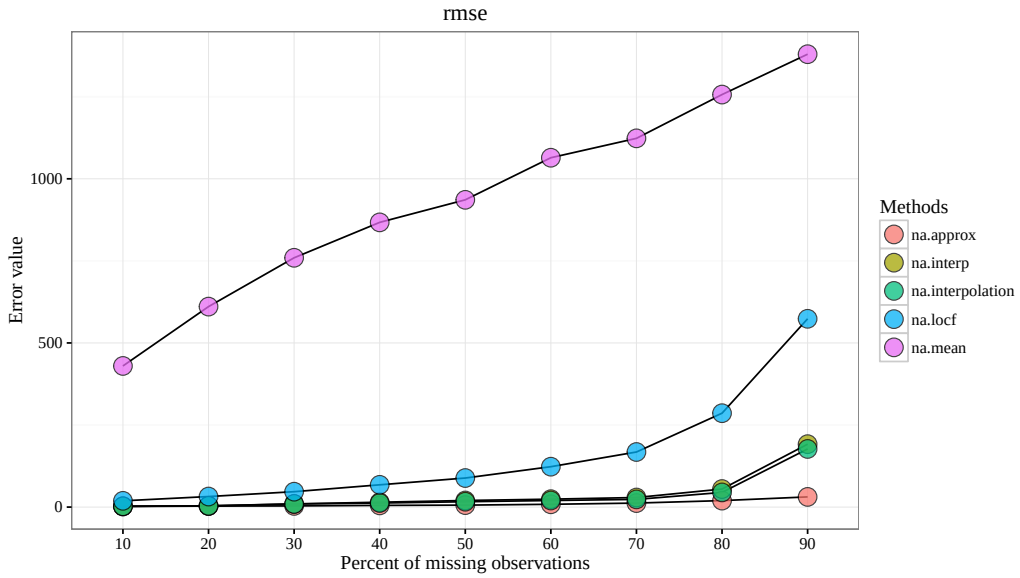

Figure 6: RMSE comparison of imputations using the `austres` dataset.

The `austres` dataset is a `ts` object of Australian population in thousands, measured quarterly from 1971 to 1994 (Brockwell and Davis, 1996). The `plot_errors()` function shows that all imputation methods had larger error values with additional missing observations, as expected, and that the `na.mean` imputation method had the largest error values. Differences between the error values can be understood by viewing a sample of the imputed data with `plot_impute()`. The example belows shows an example of imputed values using the `na.approx`, `na.locf`, and `na.mean` functions at 10% and 90% missing observations using MCAR sampling.

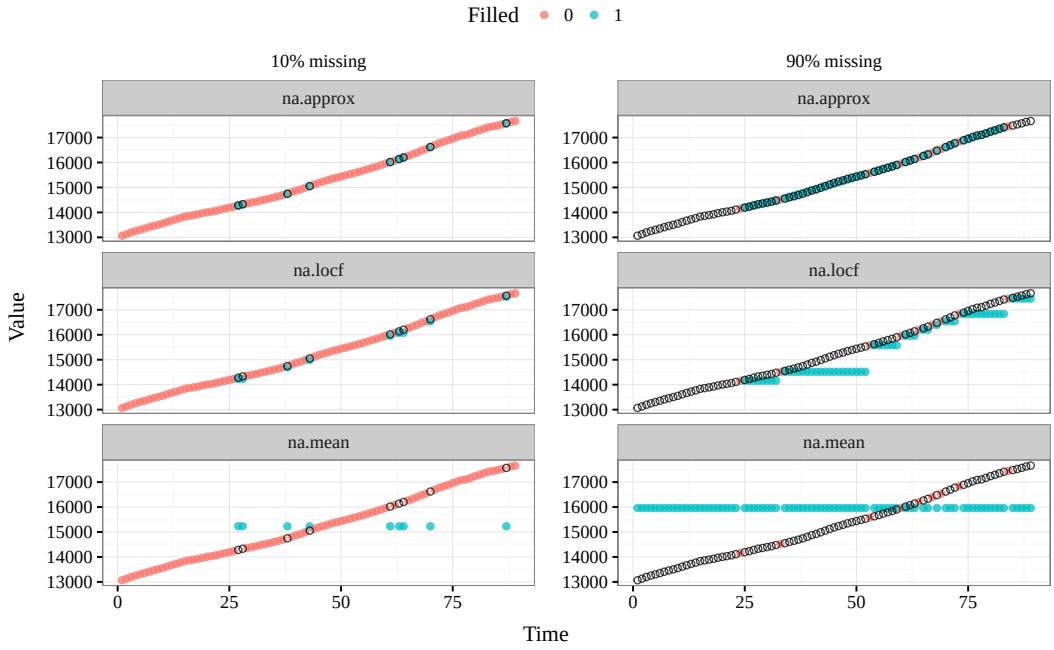

Figure 7: An example from `plot_impute()` of imputed values for the `austres` dataset using `na.approx`, `na.locf`, and `na.mean`. The left and right columns shows 10% and 90% missing data with MCAR sampling.

Reasons for differences in error values between the methods are apparent from `plot_impute`. The `austres` data is a serially correlated time series that increases linearly throughout the time period. The `na.mean` function performs poorly because it does not capture the linear increase through time. Conversely, `na.locf` and `na.approx` perform equally well for small percentages of missing data but error values diverge for larger percentages. These trends are shown in Figure 6 and verified in Figure

7. As such, differences in error values between methods relate to the characteristics of the dataset and the interpolation method used by each function.

As described above, imputation methods supplied by the user can be added to `impute_errors()`. The example below demonstrates the addition of a random number imputation method to the error profile. An R script file must be created for adding and saving the function. Additional functions can be added to the script as needed. User-supplied functions for imputation should use time series data with missing values as input and return the time series data with the imputed values as shown below.

```
# A sample function to randomly impute the missing data
library(imputeTS)
sss <- function(In){
  out <- na.random(In)
  out <- as.numeric(out)
  return(out)
}
```

The path where the R script is saved is used as an input string to the `methodPath` argument. The name of the new function is added to the `methods` argument, including any of the default methods used by `impute_errors`. Results are shown below and in Figure 8.

```
ex <- impute_errors(dataIn = aus, methodPath = 'SupportiveCodes/sss.R',
                        methods = c('na.mean', 'na.locf', 'na.approx', 'sss'))
ex
## $Parameter
## [1] "rmse"
##
## $MissingPercent
## [1] 10 20 30 40 50 60 70 80 90
##
## $na.mean
## [1]  401  594  763  894  936 1034 1150 1255 1353
##
## $na.locf
## [1]  19  33  49  66  89 111 184 263 477
##
## $na.approx
## [1]  1.6  2.7  3.9  4.7  6.0  7.4 10.5 16.1 26.6
##
## $sss
## [1]  582  795  978 1218 1221 1549 1497 1414 1674
plot_errors(ex, plotType = 'line')
```

An error metric can be added similarly. The following example shows use of the percent change in variance (*PCV*, Tak et al., 2016) as an alternative error metric:

$$PCV = \frac{var(\bar{V}) - var(V)}{var(V)} \tag{1}$$

Error is estimated as the difference between the variance of the imputed data, $var(\bar{V})$, and variance of the missing data, $var(V)$, divided by the variance of the missing data. The user-supplied error function must include two arguments as input, the first being a vector of observed values and the second being a vector of imputed missing values equal in length to the first. The function must also return a single value as a summary of the errors or differences.

```
# the pcv error function
pcv <- function(dataIn, imputed)
{
  d <- (var(imputed) - var(dataIn)) * 100/ var(imputed)
  d <- as.numeric(d)
  return(d)
}
```

As before, the new error function should be saved as an R script. The file path is added to the `errorPath` argument and the error function name is added to the `errorParameter` argument for the `impute_errors` function. Results are shown below and in Figure 9.

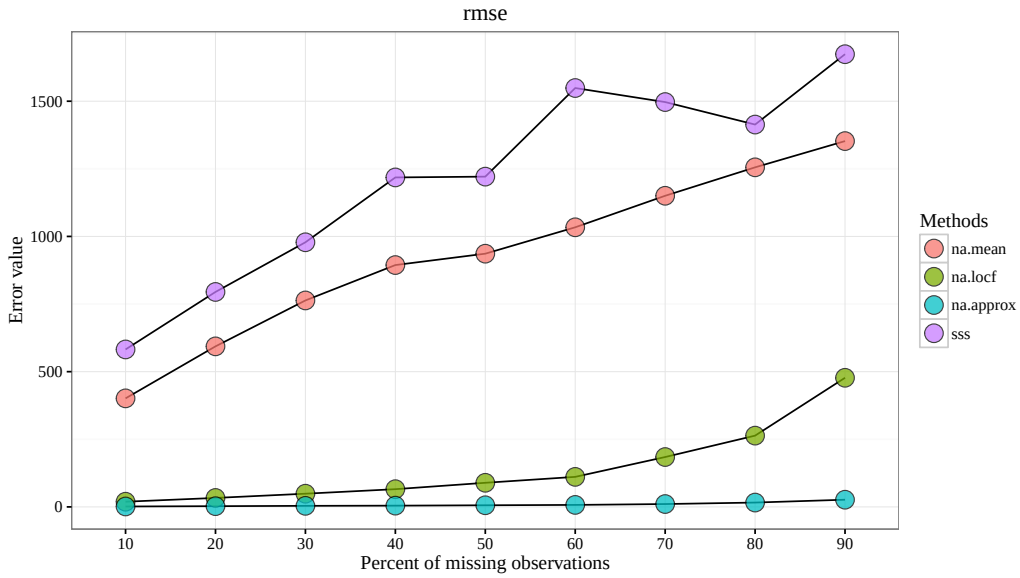

Figure 8: Adding a new imputation method to fill missing data in the `austres` dataset.

```
ex <- impute_errors(dataIn = aus, errorPath = 'SupportiveCodes/pcv.R',
  errorParameter = 'pcv')
ex
## $Parameter
## [1] "pcv"
##
## $MissingPercent
## [1] 10 20 30 40 50 60 70 80 90
##
## $na.approx
## [1]  -0.49  -1.56  -0.75  -2.38  -2.91 -10.86  -9.93 -23.28 -66.28
##
## $na.interp
## [1]  -0.0048  -0.1405  -0.0333  -0.3056  -0.3367  -1.8588  -1.5596
## [8]  -5.0646 -22.2181
##
## $na.interpolation
## [1]  -0.0092  -0.1337  -0.0252  -0.2842  -0.3964  -1.6820  -1.4542
## [8]  -4.9024 -23.7908
##
## $na.locf
## [1]   0.083  -1.143  -0.198  -1.451  -3.230  -4.726  -6.559  -9.041
## [9] -42.841
##
## $na.mean
## [1]   -11   -29   -43   -69  -110  -166  -258  -452 -1128
plot_errors(ex, plotType = 'line')
```

Summary

This paper described the **imputeTestbench** (Bokde and Beck, 2016) R package which works as a testbench to compare imputation methods for missing data. The usability of this package was demonstrated by the examples above. By default, the testbench compares existing imputation methods (`na.approx()`, `na.interp()`, `na.interpolation()`, `na.locf()`, and `na.mean()`) using *RMSE*, *MAE* or *MAPE* error metrics. Along with the default methods, the package allows users to include additional imputation methods for comparison. As such, this package can support imputation methods compiled with C, Fortran, C++, Java, Python or Matlab languages with the help of R packages like **Rcpp** (Eddelbuettel and François, 2011), **rJava** (Urbanek, 2016), **rPython**, and **matlabr** (Muschelli, 2015). Imputation methods can also be evaluated with alternative error metrics other than those provided with

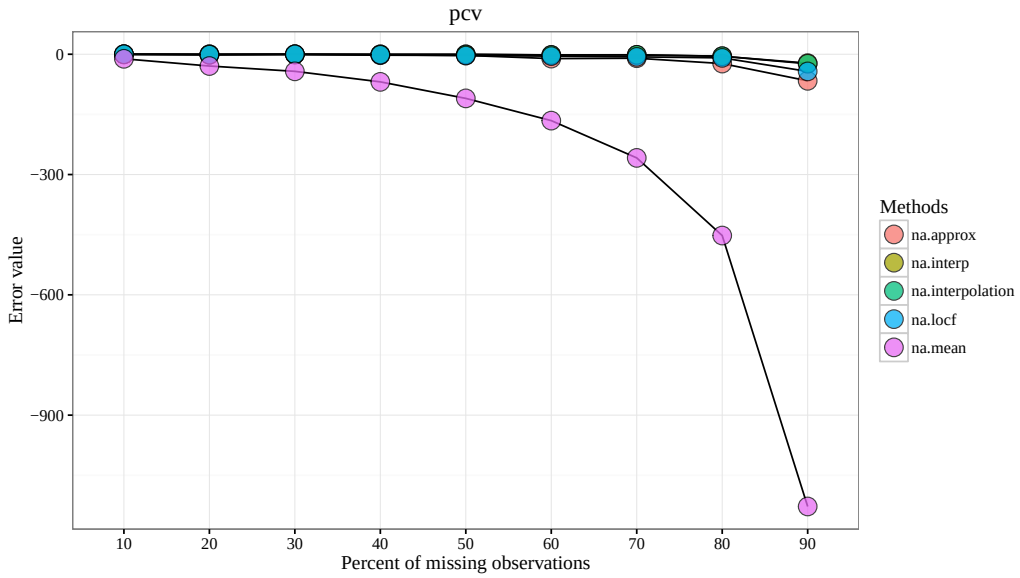

Figure 9: Adding a new error parameter to compare imputation methods for missing data in the `austres` dataset.

the package. As such, the simple architecture of **imputeTestbench** to add or remove multiple methods and error metrics makes it a robust and useful tool to evaluate existing and proposed imputation techniques. The results discussed in this paper are performed using R 3.3.1. The **imputeTestbench** package is available on CRAN (https://cran.r-project.org/web/packages/imputeTestbench/index.html).

Bibliography

N. Bokde and M. W. Beck. *imputeTestbench: Test Bench for Missing Data Imputing Models/Methods Comparison*, 2016. URL `https://cran.r-project.org/package=imputeTestbench`. R package version 3.0.0. [p3, 13]

P. J. Brockwell and R. A. Davis. *Introduction to Time Series and Forecasting*. Springer-Verlag New York, 1996. ISBN 978-1-4757-2526-1. [p11]

S. Buuren and K. Groothuis-Oudshoorn. mice: Multivariate imputation by chained equations in r. *Journal of statistical software*, 45(3), 2011. [p1]

D. Eddelbuettel and R. François. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011. URL `http://www.jstatsoft.org/v40/i08/`. [p13]

J. Honaker, G. King, M. Blackwell, et al. Amelia ii: A program for missing data. [p1]

R. J. Hyndman. *forecast: Forecasting functions for time series and linear models*, 2016. URL `http://github.com/robjhyndman/forecast`. R package version 7.2. [p3]

R. Jörnsten, M. Ouyang, and H.-Y. Wang. A meta-data based method for dna microarray imputation. *BMC bioinformatics*, 8(1):109, 2007. [p1]

D. Li, J. Deogun, W. Spaulding, and B. Shuart. Towards missing data imputation: a study of fuzzy k-means clustering method. In *Rough sets and current trends in computing*, pages 573–579. Springer, 2004. [p1]

H. Li, C. Zhao, F. Shao, G.-Z. Li, and X. Wang. A hybrid imputation approach for microarray missing value estimation. *BMC genomics*, 16(Suppl 9):S1, 2015. [p1]

S. Moritz. *imputeTS: Time Series Missing Value Imputation*, 2015. URL `https://CRAN.R-project.org/package=imputeTS`. R package version 0.4. [p3]

J. Muschelli. *matlabr: An Interface for MATLAB using System Calls*, 2015. URL `https://CRAN.R-project.org/package=matlabr`. R package version 1.1. [p13]

C. D. Nguyen, J. B. Carlin, and K. J. Lee. Diagnosing problems with imputation models using the kolmogorov-smirnov test: a simulation study. *BMC medical research methodology*, 13(1):1, 2013. [p1]

S. Oh, D. D. Kang, G. N. Brock, and G. C. Tseng. Biological impact of missing-value imputation on downstream analyses of gene expression profiles. *Bioinformatics*, 27(1):78–86, 2011. [p1]

B. Ran, H. Tan, J. Feng, Y. Liu, and W. Wang. Traffic speed data imputation method based on tensor completion. *Computational intelligence and neuroscience*, 2015:22, 2015. [p1]

J. Sim, J. S. Lee, and O. Kwon. Missing values and optimal selection of an imputation method and classification algorithm to improve the accuracy of ubiquitous computing applications. *Mathematical Problems in Engineering*, 2015, 2015. [p1]

D. J. Stekhoven and P. Bühlmann. Missforest—non-parametric missing value imputation for mixed-type data. *Bioinformatics*, 28(1):112–118, 2012. [p1]

Y.-S. Su, M. Yajima, A. E. Gelman, and J. Hill. Multiple imputation with diagnostics (mi) in r: Opening windows into the black box. *Journal of Statistical Software*, 45(2):1–31, 2011. [p1]

S. Tak, S. Woo, and H. Yeo. Data-driven imputation method for traffic data in sectional units of road links. *IEEE Transactions on Intelligent Transportation Systems*, PP(99):1–10, 2016. ISSN 1524-9050. doi: 10.1109/TITS.2016.2530312. [p1, 12]

S. Urbanek. *rJava: Low-Level R to Java Interface*, 2016. URL https://CRAN.R-project.org/package=rJava. R package version 0.9-8. [p13]

H. Wickham. Reshaping data with the reshape package. *Journal of Statistical Software*, 21(12):1–20, 2007. URL http://www.jstatsoft.org/v21/i12/. [p3]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2009. ISBN 978-0-387-98140-6. URL http://ggplot2.org. [p3]

H. Wickham. *tidyr: Easily Tidy Data with 'spread()' and 'gather()' Functions*, 2016. URL https://CRAN.R-project.org/package=tidyr. R package version 0.6.0. [p3]

H. Wickham and R. Francois. *dplyr: A Grammar of Data Manipulation*, 2016. URL https://CRAN.R-project.org/package=dplyr. R package version 0.5.0. [p3]

A. Zeileis and G. Grothendieck. zoo: S3 infrastructure for regular and irregular time series. *Journal of Statistical Software*, 14(6):1–27, 2005. doi: 10.18637/jss.v014.i06. [p3]

X. Zhu, S. Zhang, Z. Jin, Z. Zhang, and Z. Xu. Missing value estimation for mixed-attribute data sets. *Knowledge and Data Engineering, IEEE Transactions on*, 23(1):110–121, 2011. [p1, 2]

Neeraj Bokde
Visvesvaraya National Institute of Technology, Nagpur
North Ambazari Road, Nagpur
India
neeraj.bokde@students.vnit.ac.in

Kishore Kulat
Visvesvaraya National Institute of Technology, Nagpur
North Ambazari Road, Nagpur
India
kdkulat@ece.vnit.ac.in

Marcus W Beck
USEPA National Health and Environmental Effects Research Laboratory, Gulf Ecology Division
1 Sabine Island Drive, Gulf Breeze, FL 32651
USA
beck.marcus@epa.gov

Gualberto Asencio-Cortés
Universidad Pablo de Olavide
ES-41013, Sevilla
Spain
guaasecor@upo.es