R package 'imputeTestbench' as a Testbench to compare missing value imputation methods

by Neeraj Bokde, Kishore Kulat

Abstract This paper discusses about R package **imputeTestbench**, which provides a testbench to do comparison of methods used for missing data imputation. This package validates and compares a proposed imputation method with other default methods like historic mean and interpolation. The testbench is not limited to these methods. User can add or remove multiple numbers of methods in the existing methods in testbench. By default, testbench compares different imputation methods considering different error metrics *RMSE*, *MAE* or *MAPE*. Along with this, it allows user to add new error metrics as per their requirements. The simplicity of the package usage and significant reduction in efforts and time consumption in state of art procedure, adds valuable advantage to it. This paper explains the use of all functions in imputeTestbench package with the demonstration of examples.

Introduction

The *CRAN* repository is full of packages for missing value imputations. These packages can be categorized into EM based methods, nearest neighbour methods and few other miscellaneous methods. These packages have contributed in wide range of applications including areas like biomedical, intelligent traffic system, medical care, economics, ecology, etc. Out of this pool of such packages, `MICE`, `mi`, `Amelia` and `missForest` packages are well recognized and easy to use. `MICE` (Multivariate imputation by Chained equation) (Buuren and Groothuis-Oudshoorn, 2011) package considers data as Missing at Random (*MAR*) pattern and creates multiple imputations. This package uses various methods like Predictive mean matching (*PMM*), logistic regression, Bayesian polytomous regression and proportional odds model. The `mi` (Multiple imputation with diagnostic) (Su et al., 2011) package uses predictive mean matching technique and predicts the suitable values for missing data, whereas, the bootstrapping and EMB algorithm is applied on Missing at Random (*MAR*) data in the `Amelia` package (Honaker et al.). The `missForest` package (Stekhoven and Bühlmann, 2012) uses Random Forest algorithm for missing data imputations. Apart from this, CRAN is equipped with a large number of other imputation method packages as discussed in article CRAN Task View: Official Statistics & Survey Methodology. Many of these R packages are widely used in large number of applications and made the practical imputation methods more practical and user friendly. The proposed `imputeTestbench` R Package attempts to achieve next step by providing a testbench to compare multiple imputation methods with very less time consumption and little efforts.

While studying research articles related to comparison of different imputation methods, it is found that, almost all such articles follow a similar procedure to evaluate the superiority of one method over others. Zhu et al. (2011) proposed a mixture kernel based iterative estimation method for imputation of missing data and compared this method with four different imputation methods named, non-parametric iterative signal kernel method, non-parametric iterative signal with RBF kernel, the traditional kernel non-parametric missing value method and conventional frequency estimators. All these comparison of these methods with proposed method is done in terms of *RMSE* at different missing ratios. Table 1 presents the observation table of the imputation method comparison, where, first row indicates the variation in the percent missing value varying from `10` to `80%`. The first column indicates all imputation methods under study. T is the number iteration of imputation for respective methods and V is the mean of imputed values. Similarly, Table 2 shows cc as correlation coefficient and R as *RMSE* values for respective methods.

In a similar manner, Tak et al. (2016) proposed a spatial and temporal correlation based modified k-nearest neighbour method for missing data imputation. This proposed method is also compared with other methods like nearest historical data. The study was started with generation of missing ratio ranging from `0.1%` to `50%` and impute the missing data with nearest history (NH) method, bootstrapping based expectation maximization (B-EM) and the maximum likelihood estimation (MLE) method. Finally, the comparison of evaluation is done with reference to *RMSE*, *MAPE* and percent change in variance (*PCV*) is measured. Similarly, studies in articles (Oh et al., 2011), (Jörnsten et al., 2007), (Li et al., 2015), (Nguyen et al., 2013), (Sim et al., 2015), (Li et al., 2004) and (Ran et al., 2015) follow the same methodologies for missing data imputation performance evaluation. This comparison procedure can be explained with workflow diagram as in Figure 1, where variables *Min%* and *Max%* are minimum to maximum percent of missing values in dataset. *Interval*, *repetition* and other variables are discussed in detail in next section.

Method Names	10 T	Percent V	80 T	Percent 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 with reference to percents of missing values and number of iterations.

Method Names	10 T	Percent V	80 T	Percent V
Mixing	0.98	0.351	0.85	3.275
Poly	0.97	0.532	0.83	4.526
RBF	0.97	0.482	0.82	5.327
Normal	0.96	0.924	0.80	8.956
FE	0.95	0.678	0.81	7.622

Table 2: Comparison of imputation methods with reference to Correlation Coefficients (CC) and RMSE values (R).

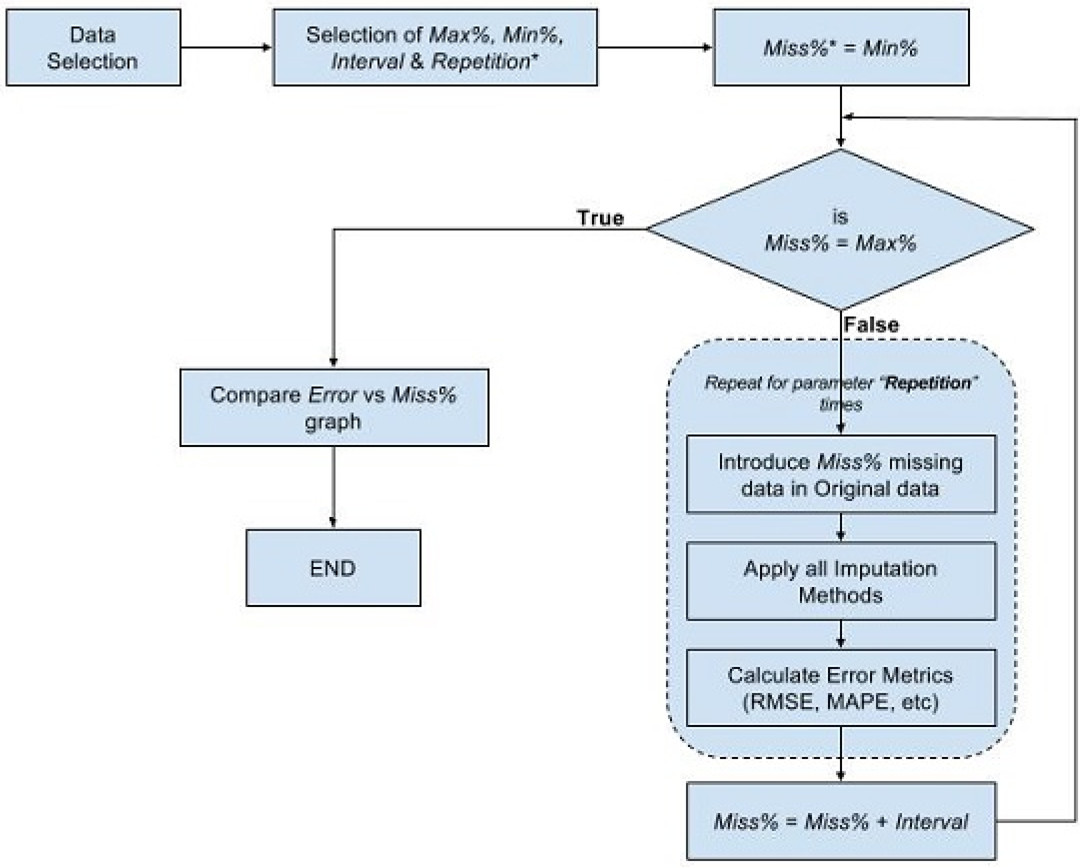

Data Selection
Selection of Max%, Min%, Interval & Repetition*
Miss%* = Min%
is Miss% = Max%
True
False
Compare Error vs Miss% graph
END
Repeat for parameter "Repetition" times
Introduce Miss% missing data in Original data
Apply all Imputation Methods
Calculate Error Metrics (RMSE, MAPE, etc)
Miss% = Miss% + Interval

Figure 1: Workflow diagram for methodologies for imputation methods comparison

Introduction to Package imputeTestbench

This section introduces imputeTestbench R package. Discussing about dependency, this package imports packages ggplot2 (Wickham, 2009), reshape2 (Wickham, 2007) and imputeTS (Moritz, 2015). The package imputeTestbench (Bokde, 2016) is a testbench tool to compare the missing value imputation methods. Many researchers have proposed about and still working towards the improvement of large numbers of imputation methods, such that missing values in a dataset can be filled suitably. But while evaluating the significance or the superiority of proposed method, the researchers need to regenerate or collect all existing benchmarked methods and then compare the results with that of proposed method. Most of the time, researchers performs the performance evaluation of the method manually by noting errors at different percentages of missing value for large number of repetitions for all error metrics on various methods under study. Obviously, this is very much time consuming and error prone procedure. Apart from this, the unavailability of a common platform for comparison, the question of accuracy of the performance evaluation is always present. To eliminate all these problems in the state of art methodologies, the imputeTestbench package is introduced. This package consists of five functions. One of them is error_functions() which is used to perform error analysis on imputed dataset, which includes *RMSE*, *MAE* and *MAPE* calculations. The other three functions, impute_errors(), append_method() and remove_method() are used as test bench tools for imputation techniques, whereas function plot_errors() adds a visual advantage to the testbench.

The impute_errors() function:

While performing imputation methods comparison study, the missing value pattern plays an important role for particular dataset. The performance of one method may vary significantly while changing such missing value patterns. The most of the imputation method analysis based literatures are primarily concerned with the more realistic scenarios where the data are 'Missing at Random (*MAR*)'. It can be said with MAR that the probability of a data point is missing can depend on the observed values of other data point but can not depend on the true values of missing data points themselves. In the impute_errors() function, the default missing values are introduced as missing at random (*MAR*) patterns. Also, it permit users to specify the missingness mechanism including a 'Missing Not At Random (*NMAR*)', where the probability of missingness depends on the true values of missing data points. It allows user to specify the desired missing data patches in dataset with *random* parameter. The impute_errors() function take eleven different parameters as input as discussed below. The syntax for this function is shown below:

```
# impute_errors(dataIn, missPercentFrom, missPercentTo, interval, repetition,
#               errorParameter, MethodPath, MethodName, random, startPoint, patchLength)
```

None of these parameters are mandatory for execution of this function, since all of them will be assumed with their default values. The detailed information of these parameters are as follows:

dataIn: It is an numeric time series data in any data format supported by R language. This data will be used by the function to do performance evaluation and compare imputation methods. It is desired that, this dataset should be only numeric without any missing values. If user let this parameter unused, the test bench will use default time series data with repeated pattern.

missPercentFrom/missPercentTo: These parameters indicate the minimum and maximum values of missing values respectively, those are to be introduced in the input data dataIn. The permissible range of these parameters ranges from 1 to 99 and value of missPercentFrom should be smaller than missPercentTo. If these parameters are not provided by users, the function will take its default values, missPercentFrom and missPercentTo as 10 and 80, respectively.

interval: It is an incremental interval parameter from value of missPercentFrom to missPercentTo. By default the interval value is taken as integer 10. With these three parameters missPercentFrom, missPercentTo and interval, user can decide the missing values and their variation pattern in the experiment. Suppose, missPercentFrom, missPercentTo and interval values are 20, 80 and 20 respectively. Then the function impute_errors() considers missing percent values as 20, 40, 60 and 80 and performs error analysis with reference to these values.

repetition: In the proposed testbench, missing values are placed randomly in original data. Hence, the performance evaluation with few repetitions may not be reliable and acceptable. In order to make robust evaluation, large number of repetition is expected. The parameter repetition is an

errorParameter	ErrorMetrics
1	RMSE
2	MAE
3	MAPE
4	Add New Metric

Table 3: Possible inputs for 'errorParameter'.

integer value provided by user, which decides how many time the error evaluation to be done and finally mean of all repeated samples is considered as final error. By default, the value of parameter `repetition` is set to `10`.

`errorParameter`**:** The `errorParameter` is the parameter which is refered to decide the error metrics which are to be used for imputation methods comparison. Generally, the most commonly used error metrics are Root Mean Square Errors (*RMSE*), Mean Absolute Percent Error (*MAPE*) and Mean Absolute Errors (*MAE*). But the types of error metrics are not limited to these three metrics. Tak et al. (2016) used Percent change in variance (*PCV*) as error metric along with *RMSE* and *MAPE*. Likewise, (Zhu et al., 2011), (Oh et al., 2011), (Li et al., 2015) and (Nguyen et al., 2013) used different error metrics like classification accuracy, logged RMSE (*LRMSE*), Normalised RMSE (*NRMSE*) and median Kolmogorov Smirnov (*KS*) test p value. Hence, the testbench should be capable enough to accommodate such new error metrics in the default functions. The Table 3 shows all the possible inputs for `errorParameter`.

The default value of this parameter is `1`. For first three values of `errorParameter`, fixed error metrics are assigned as shown in Table 3. The selection `errorParameter = 4`, allows user to select a new error metric other than already available in testbench. The syntax of this parameter to add new error metric is shown below.

```
errorParameter = c(4, “source(~/function.R)”, “label_name”)
```

where, `4` is the mandatory integer to indicate that a new error metric is to be attached. Second term is the location of the code for new error metric and third term is the label name for that error metric.

While introducing a new error metric, it is necessary to set the input parameters and return variables in a format desired by function `impute_errors()`. The code for new error metric should be in function form which is to be written in an R script. This function should contain two input parameters as original and imputed data series. Also, the function should return the error calculated between the input data serieses. The detailed explanation of this parameter is discussed with example in next section.

`MethodPath`**:** The `MethodPath` parameter allows user to add new imputation method along with already existing methods in testbench. In version 2.0.2 of `imputeTestbench` package, historic means and interpolation methods are existing methods. Suppose, user wants to add new method in testbench to compare its performance with former methods, `MethodPath` parameter is to be assigned. `MethodPath` as an input should be in string format of location of new method function in source format. The selection and operation methodology of this parameter is explained in example section.

`MethodName`**:** This parameter is also an input in string format which assigns a name to the new method introduced in testbench with `MethodPath` parameter. Since, this is not a mandatory parameter and it takes string *Proposed method* as input in default case.

With consideration of all these input parameters in default values, the `impute_errors()` function returns a list as shown below:

```
library(imputeTestbench)
a <- impute_errors()
a

#> $Parameter
#> [1] "RMSE Plot"
#>
#> $Missing_Percent
#> [1] 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8
#>
```

```
#> $Historic_Mean
#> [1] 0.4908711 0.6641892 0.8008869 0.9175113 1.0210202 1.1151930 1.2025538
#> [8] 1.2857627
#>
#> $Interpolation
#> [1] 0.5567764 0.7810250 0.9539392 1.1000000 1.2288206 1.3453624 1.4525839
#> [8] 1.5524175
```

The first member of list is the type of error metric used for comparison. In present case it is *RMSE* value. Whereas, the second member of the list is variation of percent of missing values in ascending order. Finally, remaining members in the list represent error values corresponding to the missing value percentages for different imputation methods. In this paper, this list with various members is referred as *Error profile*.

The `append_method()` function

In case of comparative study, comparison with 2-3 methods are not enough to describe the superiority of one method over others. It is the necessity to add more and more comparable methods in such study to describe the robustness of a particular method. When user needs to add more than one imputation methods in the testbench, the `append_method()`is to be used. The syntax for this function is as shown below.

```
# append_method(existing_method, dataIn, missPercentFrom, missPercentTo, interval,
#               repetition, errorParameter, MethodPath, MethodName)
```

Almost all of these input parameters for this function are similar to that of `impute_errors()` function with default values as well, except parameter `existing_method`. This function can not be applied directly on input. Rather, it is an additive function to the `impute_errors()` function to add new imputation methods in the testbench. This function takes error profile as input with parameter `existing_method` and generate another error profile with addition of new method in former error profile.

The `remove_method()` function

This function is applied on a error profile to remove the undesired method existed in the testbench. It takes error profile as input `existing_method` parameter along with `index_number` as index of imputation method, which is to be removed from the error profile. This function returns the new error profile with removal of unwanted method. The syntax of `remove_method()` function is:

```
# remove_method(existing_method, index_number)
```

The `plot_errors()` function

The functions `append_method()`, `remove_method()` and `impute_errors()` deal with error profiles. While the former two functions takes error profile as input as well as returns another modified error profile. These error values obtained in these profiles are enough to compare the imputation methods under study. But the proper visualization of these errors makes this comparison very easy. All the versions of imputeTestbench package before 2.0.2 have plotted the comparison analysis with the line plots, but this plot was accompanied with a limitation that the lines between the points for different values of missing percent values was misleading because the lines did not represent the real underlying values. To avoid this limitation, bar plot is introduced in 2.0.2 and onward versions of the package. Though the default plot is a bar plot which is able to present the comparison more clearly, the package also permit users to plot the line plot to show the approximate behaviors of imputation methods at different percentage of missing values with following code syntax.

```
# # Default bar plot presentation
# plot_errors(dataIn = error_profile)
#
# #  Line plot presentation
# plot_errors(dataIn = error_profile, plotType = 2)
```

The `plot_errors()` function is used to plot the error profiles in interactive graphical format. This function takes error profile as input value and returns the bar plot comparing different imputation methods.

The `error_functions()` function

This function is a collection of functions for error metrics, which are used to calculate the error between original input data and imputed data. In existing version 2.0.2 of `imputeTestbench` package, the available error metrics are *RMSE*, *MAE* and *MAPE*. But the noticeable thing is that, the proposed testbench is not limited to these metrics. User can attach new error metrics with `impute_errors()` and `append_method()` functions. The `error_functions()` function is not directly accessible for testbench users. Rather, the remaining functions in proposed package uses this function to measure errors for different error metrics in imputation methods.

Demonstration of `imputeTestbench` package with examples

This example is to demonstrate how the *imputTestbench* package can be used as testbench to compare different imputation methods. This testbench always initiates with `impute_errors()` function, which consumes time series data and various other input parameters as discussed in above section. Let us consider as data string from `iris` dataset available in R repository for testing purpose.

```
data("iris")
x <- iris[2]
```

All the functions in namespace of the package `imputeTestbench` will be loaded with function as shown below.

```
library(imputeTestbench)
```

The `impute_errors()` function with error metric *RMSE* and other default imputation methods returns error profile as states below.

```
aRMSE <-  impute_errors(dataIn = x, missPercentFrom = 10, missPercentTo = 90,
                        interval = 5, repetition = 1, errorParameter = 1)
aRMSE

#> $Parameter
#> [1] "RMSE Plot"
#>
#> $Missing_Percent
#>  [1] 0.10 0.15 0.20 0.25 0.30 0.35 0.40 0.45 0.50 0.55 0.60 0.65 0.70 0.75
#> [15] 0.80 0.85 0.90
#>
#> $Historic_Mean
#>  [1] 0.0811589 0.1122208 0.1430093 0.1723577 0.1927798 0.2095538 0.2481437
#>  [8] 0.2828763 0.3163933 0.3747002 0.4510670 0.5181335 0.5284983 0.5551195
#> [15] 0.5713803 0.5692042 0.4894895
#>
#> $Interpolation
#>  [1] 0.1104536 0.1201927 0.2216604 0.1821039 0.1854724 0.1904381 0.2096028
#>  [8] 0.2596374 0.2379075 0.2763627 0.2585859 0.3423258 0.5830952 0.3575392
#> [15] 0.5312250 0.4095921 0.4190465
```

```
plot_errors(aRMSE)

#>      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
#> [1,]  1.5  4.5  7.5 10.5 13.5 16.5 19.5 22.5 25.5  28.5  31.5  34.5
#> [2,]  2.5  5.5  8.5 11.5 14.5 17.5 20.5 23.5 26.5  29.5  32.5  35.5
#>      [,13] [,14] [,15] [,16] [,17]
#> [1,]  37.5  40.5  43.5  46.5  49.5
#> [2,]  38.5  41.5  44.5  47.5  50.5
```

This function allows user to add one more imputation method to the default methods: historic means and interpolation methods. In following example, a random number imputation method has to be introduced to existing error profile. The procedure to add new method consists of very easy steps. First of all, create a new R script file in RStudio (RStudio Team, 2015) and write the function for the imputation method which is to be added in the testbench. It is required to make sure that, this function should be able to take time series data with missing values as input and should return the time series data with the imputed values as shown below.

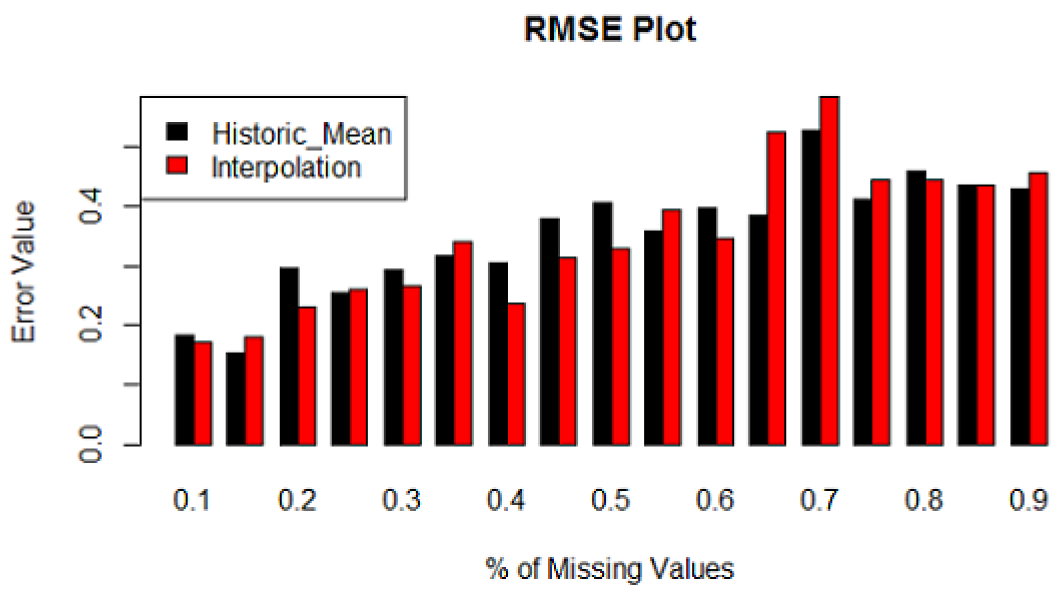
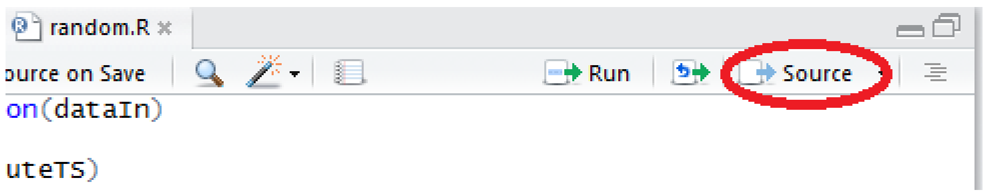


Figure 2: *RMSE* Comparison plot for *aRMSE*

```
#================================================================
# A sample function to impute the missing data randomly
#================================================================
sss <- function(In)
{
  library(imputeTS)
  set.seed(1)
  In <- ts(In)
  out <- na.random(In)
  return(out)
}
#================================================================
```

Finally, save this R script in appropriate location as per user convenience. User can source the location of this function with the use of button *source* in RStudio as shown in Figure 3.



```
on(dataIn)

uteTS)
```

Figure 3: 'Source' button in RStudio

Copy this location and provide as input string to `MethodPath` parameter.

```
aMAPE <- impute_errors(dataIn = x, missPercentFrom = 10, missPercentTo = 90,
                        interval = 5, repetition = 1, errorParameter = 3,
                        MethodPath = "source('~/bokde-kulat/bokde-kulat/SupportiveCodes/random.R')",
                        MethodName = "Proposed_Method")
aMAPE

#> $Parameter
#> [1] "MAPE Plot"
#>
#> $Missing_Percent
#>  [1] 0.10 0.15 0.20 0.25 0.30 0.35 0.40 0.45 0.50 0.55 0.60 0.65 0.70 0.75
#> [15] 0.80 0.85 0.90
#>
#> $Historic_Mean
#>  [1]  0.6683288  1.1287067  1.7487150  2.3775415  2.9452188  3.5552057
#>  [7]  4.7108808  5.7799708  6.9702120  8.8931386 11.4985381 13.9124195
#> [13] 14.5619992 15.5985299 16.4202949 16.4678784 13.8077818
#>
#> $Interpolation
#>  [1]  1.055898  1.274439  2.765005  2.536758  2.663360  2.955889  3.571324
#>  [8]  4.393177  4.435320  5.993678  5.589545  8.268802 16.305541  9.013878
#> [15] 15.045940 10.875242 10.060838
#>
#> $Proposed_Method
#>  [1]  1.413084  1.891483  2.565993  3.068246  3.739231  4.145203  4.555982
#>  [8]  5.080328  5.557194  5.973131  5.576042  6.083130  6.860961 12.377778
#> [15] 12.863721 13.090605 11.678625
```

```
 plot_errors(aMAPE)
```

```
#>      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
#> [1,] 1.5  5.5  9.5  13.5 17.5 21.5 25.5 29.5 33.5 37.5  41.5  45.5  49.5
#> [2,] 2.5  6.5  10.5 14.5 18.5 22.5 26.5 30.5 34.5 38.5  42.5  46.5  50.5
#> [3,] 3.5  7.5  11.5 15.5 19.5 23.5 27.5 31.5 35.5 39.5  43.5  47.5  51.5
#>      [,14] [,15] [,16] [,17]
#> [1,] 53.5  57.5  61.5  65.5
#> [2,] 54.5  58.5  62.5  66.5
#> [3,] 55.5  59.5  63.5  67.5
```

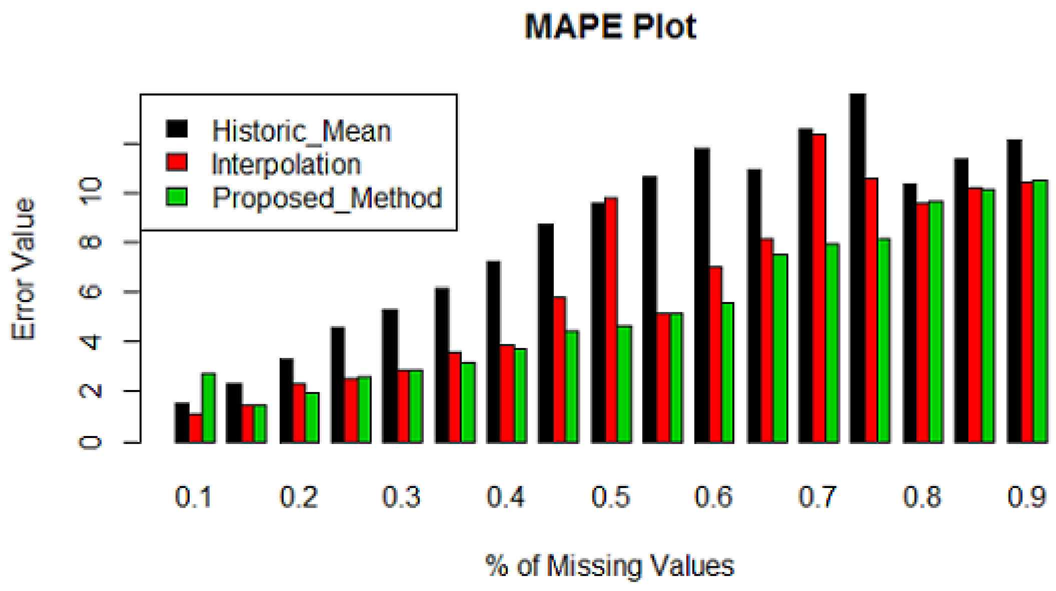
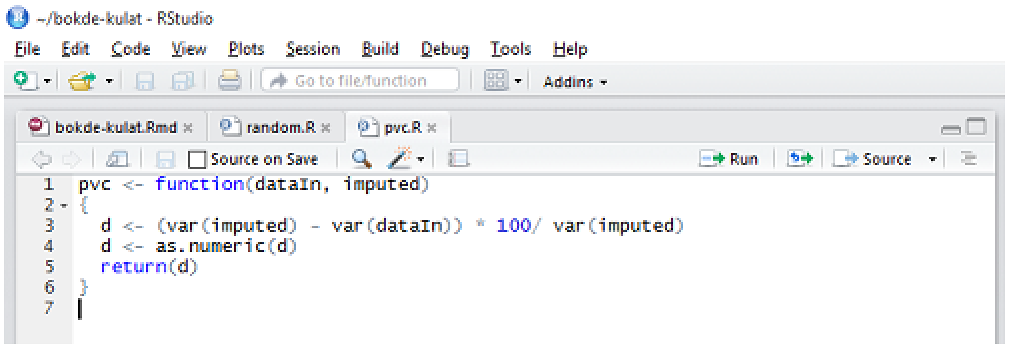


Figure 4: *MAPE* Comparison plot for *aMAPE*

This error profile 'aMAPE' compares the *MAPE* values for three computing methods, since 'MAPE' is selected as error metric with parameter '*errorParameter = 3*'. In similar way, suppose user wants to attach a new error metric to the testbench, same procedure as that of adding new imputation method is to be followed. The function for new error metric should have two inputs. First of these inputs should be the original time series data and the later one should be the modified time series data, which is processed with imputation method. For these two inputs, the function should return the error values corresponding to the newly proposed error metric. The sample program for new error metric is as shown in Figure 5.



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

Figure 5: The sample program for new error metric 'PCV'

Here 'PCV' is used as error metric, which is a percent change in variance and can be formulated as shown in equation (1).

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

The source location of this error metric can be captured as similar to 'MethodPath' parameter. Example below presents the implementation of new error metric in the testbench.

```
aPCV <- impute_errors(dataIn = x, missPercentFrom = 10, missPercentTo = 90,
                       interval = 5, repetition = 1,
                       errorParameter = c(4,
                                          "source('~/bokde-kulat/bokde-kulat/SupportiveCodes/pvc.R')",
                                          "PCV Error"),
                       MethodPath = "source('~/bokde-kulat/bokde-kulat/SupportiveCodes/random.R')",
                       MethodName = "Random Method")
aPCV

#> $Parameter
#> [1] "PCV Error"
#>
#> $Missing_Percent
#>  [1] 0.10 0.15 0.20 0.25 0.30 0.35 0.40 0.45 0.50 0.55 0.60 0.65 0.70 0.75
#> [15] 0.80 0.85 0.90
#>
#> $Historic_Mean
#>  [1]     -3.611079     -7.111418    -11.911132    -18.121771    -23.414147
#>  [6]    -27.753133    -39.568167    -53.458586    -66.377794   -100.245432
#> [11]   -175.722952   -289.489980   -339.734877   -460.263471   -711.006784
#> [16]   -738.836925 -2258.911111
#>
#> $Interpolation
#>  [1] -6.009205e-02 -6.733185e+00  8.324751e+00 -1.750050e+01 -1.726283e+01
#>  [6] -2.534953e+01 -3.437303e+01  1.980240e+00 -2.062579e+01 -7.664376e+01
#> [11] -6.288260e+01 -1.581101e+02 -3.274078e+02 -2.212806e+02 -6.927485e+02
#> [16] -3.513842e+02 -1.108459e+03
#>
#> $Proposed_Method
#>  [1]      5.266549      5.634083      5.766577      4.525516      4.361102
#>  [6]      5.206880      4.673078      3.290308      4.122781      2.583929
#> [11]    -50.591183    -58.681769    -72.934018   -398.114952   -591.780572
#> [16]   -638.492617 -1880.065239

plot_errors(aPCV, args.legend = list(x = "bottomleft", bg = "NA"))

#>      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
#> [1,]  1.5  5.5  9.5 13.5 17.5 21.5 25.5 29.5 33.5  37.5  41.5  45.5
#> [2,]  2.5  6.5 10.5 14.5 18.5 22.5 26.5 30.5 34.5  38.5  42.5  46.5
#> [3,]  3.5  7.5 11.5 15.5 19.5 23.5 27.5 31.5 35.5  39.5  43.5  47.5
#>     [,13] [,14] [,15] [,16] [,17]
#> [1,]  49.5  53.5  57.5  61.5  65.5
#> [2,]  50.5  54.5  58.5  62.5  66.5
#> [3,]  51.5  55.5  59.5  63.5  67.5
```

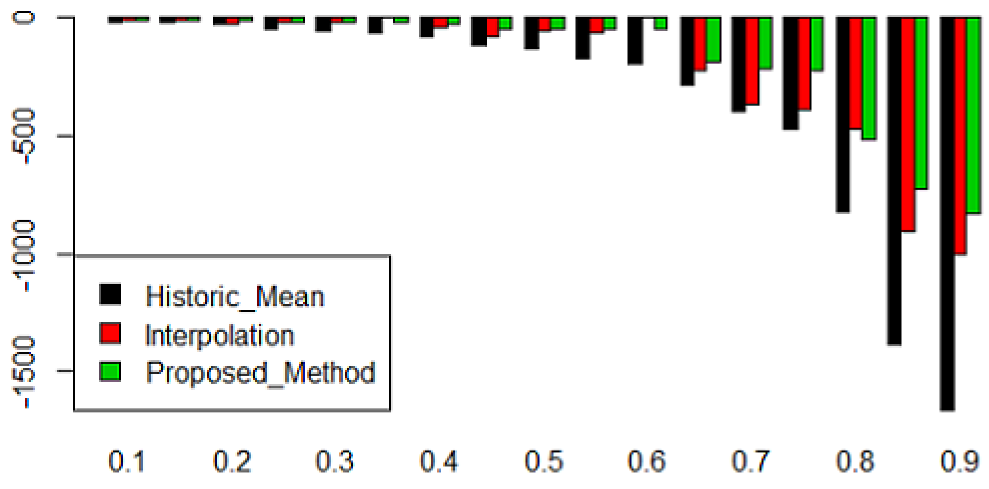


Figure 6: *PCV* Comparison plot for *aPCV*

These examples are explaining the basic working of testbench. But, if user wants to add more and more imputation methods to testbench, the `import_errors()` function is not sufficient. This can be done with `append_errors()` function. The working of this function is very much similar to that of `impute_errors()` function. This function takes an error profile as input and add new method of imputation to it. In following example, a function representing a new imputation method labeled as `Random Method` is sourced in addition to error profile `aMAPE`.

```
bMAPE <- append_method(existing_method = aMAPE, dataIn = x, missPercentFrom = 10,
                        missPercentTo = 90, interval = 5, repetition = 1,
                        errorParameter = 3,
                        MethodPath = "source('~/bokde-kulat/bokde-kulat/SupportiveCodes/random1.R')",
                        MethodName = "Random Method")
bMAPE

#> $Parameter
#> [1] "MAPE Plot"
#>
#> $Missing_Percent
#>  [1] 0.10 0.15 0.20 0.25 0.30 0.35 0.40 0.45 0.50 0.55 0.60 0.65 0.70 0.75
#> [15] 0.80 0.85 0.90
#>
#> $Historic_Mean
#>  [1]  0.6683288  1.1287067  1.7487150  2.3775415  2.9452188  3.5552057
#>  [7]  4.7108808  5.7799708  6.9702120  8.8931386 11.4985381 13.9124195
#> [13] 14.5619992 15.5985299 16.4202949 16.4678784 13.8077818
#>
#> $Interpolation
#>  [1]  1.055898  1.274439  2.765005  2.536758  2.663360  2.955889  3.571324
#>  [8]  4.393177  4.435320  5.993678  5.589545  8.268802 16.305541  9.013878
#> [15] 15.045940 10.875242 10.060838
#>
#> $Proposed_Method
#>  [1]  1.413084  1.891483  2.565993  3.068246  3.739231  4.145203  4.555982
#>  [8]  5.080328  5.557194  5.973131  5.576042  6.083130  6.860961 12.377778
#> [15] 12.863721 13.090605 11.678625
#>
#> $‘Random Method‘
#>  [1]  0.913122  1.561226  2.269093  3.000680  3.676465  4.331168  5.378775
#>  [8]  6.237326  7.013975  8.035999 11.880977 12.858238 13.315708 20.611162
#> [15] 21.399744 21.675406 15.934054
```

```
plot_errors(bMAPE)
```

```
#>      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
#> [1,]  1.5  6.5 11.5 16.5 21.5 26.5 31.5 36.5 41.5  46.5  51.5  56.5
#> [2,]  2.5  7.5 12.5 17.5 22.5 27.5 32.5 37.5 42.5  47.5  52.5  57.5
#> [3,]  3.5  8.5 13.5 18.5 23.5 28.5 33.5 38.5 43.5  48.5  53.5  58.5
#> [4,]  4.5  9.5 14.5 19.5 24.5 29.5 34.5 39.5 44.5  49.5  54.5  59.5
#>      [,13] [,14] [,15] [,16] [,17]
#> [1,]  61.5  66.5  71.5  76.5  81.5
#> [2,]  62.5  67.5  72.5  77.5  82.5
#> [3,]  63.5  68.5  73.5  78.5  83.5
#> [4,]  64.5  69.5  74.5  79.5  84.5
```

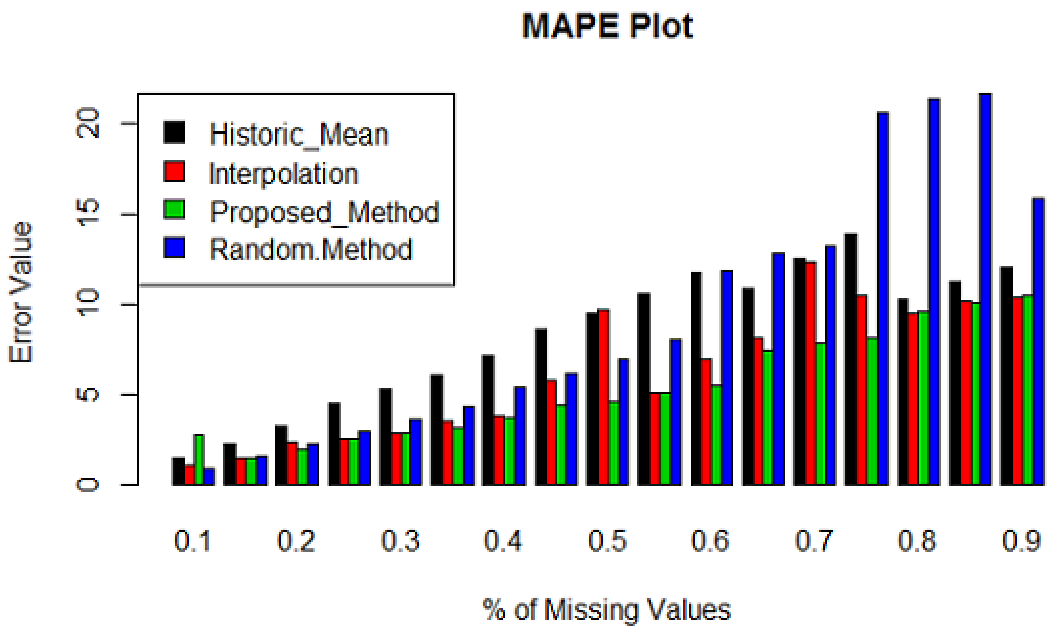


Figure 7: *MAPE* Comparison plot for *bMAPE*

Along with this, the testbench allow users to remove the unwanted imputation method from the error profile. This is done with the `remove_errors()` function as shown in following example.

```
cMAPE <- remove_method(existing_method = bMAPE, index_number = 2)
cMAPE
```

```
#> $Output for cMAPE
#>
```

```
#> $Parameter
#> [1] "MAPE Plot"
#>
#> $Missing_Percent
#> [1] 0.10 0.15 0.20 0.25 0.30 0.35 0.40 0.45 0.50 0.55 0.60 0.65 0.70 0.75 0.80 0.85 0.90
#>
#> $Historic_Mean
#> [1]  0.6683288  1.1287067  1.7487150  2.3775415  2.9452188  3.5552057  4.7108808  5.7799708
#> [9]  6.9702120  8.8931386 11.4985381 13.9124195 14.5619992 15.5985299 16.4202949 16.4678784
#> [17] 13.8077818
#>
#> $Proposed_Method
#> [1]  1.413084  1.891483  2.565993  3.068246  3.739231  4.145203  4.555982  5.080328
#> [9]  5.557194  5.973131  5.576042  6.083130  6.860961 12.377778 12.863721 13.090605
#> [17] 11.678625
#>
#> $‘Random Method‘
#> [1]  0.913122  1.561226  2.269093  3.000680  3.676465  4.331168  5.378775  6.237326
#> [9]  7.013975  8.035999 11.880977 12.858238 13.315708 20.611162 21.399744 21.675406
#> [17] 15.934054
```

```
plot_errors(cMAPE)
```

```
#>      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
#> [1,]  1.5  5.5  9.5 13.5 17.5 21.5 25.5 29.5 33.5  37.5  41.5  45.5
#> [2,]  2.5  6.5 10.5 14.5 18.5 22.5 26.5 30.5 34.5  38.5  42.5  46.5
#> [3,]  3.5  7.5 11.5 15.5 19.5 23.5 27.5 31.5 35.5  39.5  43.5  47.5
#>      [,13] [,14] [,15] [,16] [,17]
#> [1,]  49.5  53.5  57.5  61.5  65.5
#> [2,]  50.5  54.5  58.5  62.5  66.5
#> [3,]  51.5  55.5  59.5  63.5  67.5
```

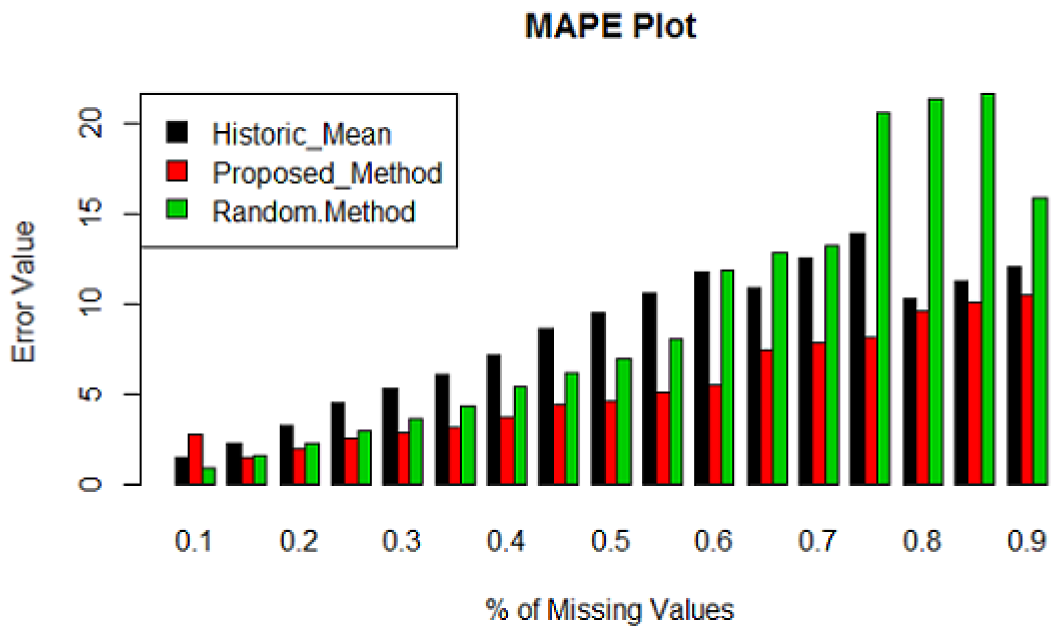


Figure 8: *MAPE* Comparison plot for *cMAPE*

Similarly, this package permit users to obtain graph plot with following code:

```
plot_errors(dataIn = cMAPE, plotType = 2)
```

Here, the `remove_errors()` function took error profile `bMAPE` and `index_number` parameter as input. The value `2` for `index_number` suggests the index of interpolation method, which is to be removed from the error profile. In this way, the `imputTestbench` package allows and assists user to compare different imputation methods with different error metrics under desired conditions.

So far, the comparison analysis is done in the paper is based on random patterns missing data. Apart from this, the `impute_errors()` function permit users to introduce the missing values dependent on realistic conditions (which are not random) and to compare it with all possible methods. The function code written below shows how to introduce the missing data patterns in dataset and to proceed for further analysis.

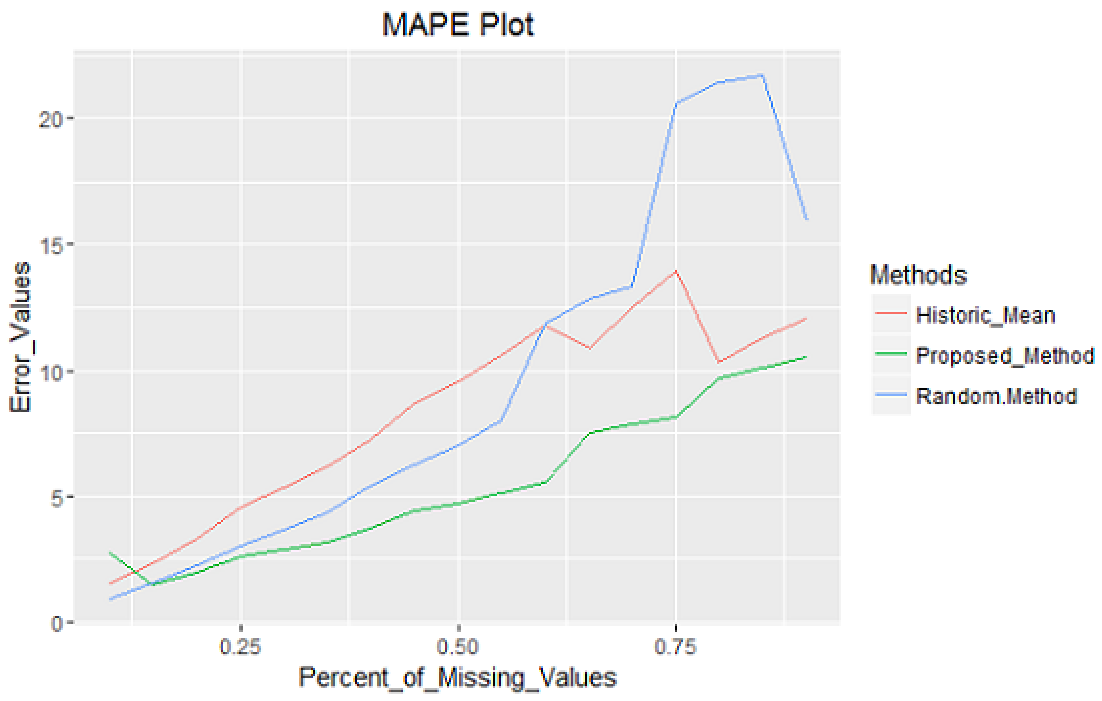


Figure 9: *MAPE* Comparison plot (line plot) for *cMAPE*

```
zRMSE <- impute_errors(dataIn = x, missPercentFrom = 10, missPercentTo = 90,
                                                interval = 5,
                                                repetition = 1,
                                                errorParameter = 1,
                                                random = 0,
                                                startPoint = c(10, 25, 40),
                                                patchLength = c(5, 7, 10))
zRMSE

#> $Parameter
#> [1] "RMSE Plot"
#>
#> $Missing_Percent
#> [1] 0.1466667
#>
#> $Historic_Mean
#> [1] 0.1940842
#>
#> $Interpolation
#> [1] 0.1987771
```

Here, three patches are introduced at indexes 10, 25 and 40 of lengths 5, 7 and 10 respectively. The output of the function returns the comparison of error values for each method for given missing values. Along with this, the missing value percentages are calculated in the results. This feature allows user to make proposed package more adaptive as per the required and more realistic scenarios.

Summary

This paper discussed about **imputeTestbench** (Bokde, 2016) R package which works as a testbench to compare missing data imputation methods. The usability of this package is demonstrated with an example. By default, this testbench compares a proposed method by user with existing imputation methods (historic means and interpolation methods) with *RMSE*, *MAE* or *MAPE* as error metrics. Along with default methods in the testbench, it allows to add new various imputation methods in comparison. This package may support the 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). Also, the example discussed about the procedure to add new error metric in addition to existing error metrics. The simple working of testbench to add and remove various imputation methods, makes it more robust and useful. The results discussed in this paper are performed using R 3.2.4. The **imputeTestbench** package is available at CRAN repository with URL https://cran.r-project.org/package=imputeTestbench.

Bibliography

N. Bokde. *imputeTestbench: Test Bench for Missing Data Imputing Models/Methods Comparison*, 2016. URL https://cran.r-project.org/web/packages/imputeTestbench/index.html. R package version 1.1.0. [p]

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

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/. [p]

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

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

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. [p]

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. [p]

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

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

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. [p]

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. [p]

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. [p]

RStudio Team. *RStudio: Integrated Development Environment for R*. RStudio, Inc., Boston, MA, 2015. URL http://www.rstudio.com/. [p]

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. [p]

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

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. [p]

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. [p]

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

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

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

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. [p]

Neeraj Bokde
Visvesvaraya National Institute of Technology, Nagpur
North Ambazari Road, Nagpur
India
neerajdhanraj@gmail.com

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