How to impute missing values?

Genevieve Robin, Imke Mayer, Aude Sportisse

26 juin 2020

Contents

If you have a dataset which contains missing values, it is relevant to impute the missing values, mainly for two reasons: (i) these values may be particularly interesting in themselves or (ii) the fully completed data is required to perform some estimation method that does not handle the missing data.

In this section we provide, for some of the main packages (the list is of course not thorough) to impute missing values, links to vignettes and tutorials, as well as a description of their main functionalities and reusable code. The goal is not to describe all the methods precisely, as many resources are already available, but rather to provide an overview of several imputation options. The methods we focus on are gathered in the table below.

Package	Data Types	Underlying Method	Imputation	Computational Time	Comments
softImpute	quantitative	low-rank matrix completion	single	+	Very fast, strong theoretical guarantees, regularization parameter to tune
mice	mixed	multivariate imputation by chained equations	multiple	-	Very flexible to data types, no parameter to tune
missForest	mixed	random forests	single	-	Requires large sample sizes, no parameter to tune

Package	Data Types	Underlying Method	Imputation	Computational Time	Comments
missMDA	mixed	component methods	single/multiple	+	Rank parameter to tune

```
library(Amelia)
library(mice)
library(missForest)
library(missMDA)
library(MASS)
library(softImpute)
library(dplyr)
library(tidyr)
library(ggplot2)
```

Description of imputation methods on synthetic data

Let us consider a gaussian data matrix of size n times p.

```
set.seed(123)
n <- 1000
p <- 10
mu.X <- rep(1, 10)
Sigma.X <- diag(0.5, ncol = 10, nrow = 10) + matrix(0.5, nrow = 10, ncol =
10)
X <- mvrnorm(n, mu.X, Sigma.X)
head(X)
```

```
##            [,1]        [,2]       [,3]       [,4]      [,5]       [,6]
## [1,]  1.5055007  0.67922116  2.7536194  1.2679330 0.6046278  1.3820507
## [2,]  1.0378387  0.37754233  1.1173218  2.5266558 1.8242035  1.2260849
## [3,] -0.6442869 -0.06921970 -1.1689235 -1.5794364 0.9841500 -0.1273577
## [4,]  0.5859960  0.65298888  0.4614630  1.5170295 1.6277217  1.0015291
## [5,]  3.1452516  0.48649309  1.9582524 -0.1270187 1.5389924  1.8349234
## [6,] -0.5447334  0.01552382 -0.2932727 -0.1295848 0.0662147 -1.3493665
##            [,7]       [,8]       [,9]      [,10]
## [1,]  2.2882116  1.6928612  1.2250392  0.7575341
## [2,]  0.3781322  1.3943540  1.1698080  0.6551008
## [3,]  0.4227255  0.6584061 -0.2414336  0.2056861
## [4,]  0.6614958 -0.1730071  1.9422992  1.1995797
## [5,]  0.1036019 -0.4067036  1.4189708 -0.9115867
## [6,] -0.2420110  0.4353963 -1.2832304  0.6058015
```

We introduce some missing (here MCAR) values in the data matrix. One uses the function **produce_NA** detailed in “amputation.R” available in the related R source code of “How to generate missing values?”.

```
source('amputation.R')

XproduceNA <- produce_NA(X, mechanism = "MCAR", perc.missing = 0.3)
XNA <- as.matrix(as.data.frame(XproduceNA$data.incomp))
```

softImpute

The softImpute package can be used to impute quantitative data. It fits a low-rank matrix approximation to a matrix with missing values via nuclear-norm regularization. A vignette is available online, as well as the original article (Hastie et al. 2015).

The **softImpute** function computes, based on an incomplete data set, a low-dimensional factorization which can be used to impute the missing values. The function is used as follows:

```
# perform softImpute
sft <- softImpute(x = XNA, rank.max = 2, lambda = 0, type = c("als", "svd"))
```

```
## Warning in simpute.als(x, J, thresh, lambda, maxit, trace.it, warm.start, :
## Convergence not achieved by 100 iterations
```

The main arguments are the following (more details can be found on the help page).

- x: the data set with missing values (matrix).
- rank.max: the restricted rank of the solution, which should not be bigger than min(dim(x))-1.
- lambda: the nuclear-norm regularization parameter.
- type: indicates the algorithm which should be used, among "svd" and "als". "svd" returns an exact solution, while "als" returns an approximate solution (in exchange for a faster computation time).

To compute the imputed data set based on the softImpute results, one may use the following code:

```
# compute the factorization
X.sft <- sft$u %*% diag(sft$d) %*% t(sft$v)
# replace missing values by computed values
X.sft[which(!is.na(XNA))] <- XNA[which(!is.na(XNA))]
```

To calibrate the parameter lambda, one may perform cross-validation, the code is given below.

```
cv_sft <- function(y,
                   N = 10,
                   len = 20) {
  y <- as.matrix(y)
  Y2 <- y
  Y2[is.na(Y2)] <- 0
  d <- dim(y)
  n <- d[1]
  p <- d[2]
  m <- sum(!is.na(y))
  lambda1.max <- max(svd(Y2)$d)
  lambda1.min <- 1e-3*lambda1.max
  grid.lambda1 <-
    exp(seq(log(lambda1.min), log(lambda1.max), length.out = len))
  ylist <-
    lapply(1:N, function(k)
      produce_NA(as.matrix(y), perc.missing = 0.2))$data.incomp
  res.cv <- lapply(1:N, function(k) {
    sapply(1:len,
           function(i) {
             yy <-produce_NA(as.matrix(y), perc.missing = 0.2)$data.incomp
             res <-
               softImpute(as.matrix(yy),
                          lambda = grid.lambda1[i], maxit = 1000)
             u <- res$u
             d <- res$d
             v <- res$v
             if (is.null(dim(u))) {
               res <- d * u %*% t(v)
             } else {
               res <- u %*% diag(d) %*% t(v)
             }
             imp <- as.matrix(yy)
             imp[is.na(yy)] <- res[is.na(yy)]
             return(sqrt(sum((res - y) ^ 2, na.rm = T)))
           })

  })
  res.cv <- colMeans(do.call(rbind, res.cv))
  l <- which.min(res.cv)
  lambda <- grid.lambda1[l]
  return(lambda)
}
```

Then, the imputation procedure can be performed using the value of lambda computed with cross-validation (the other parameters are set to their default value):

```
lambda_sft <- cv_sft(XNA)
sft <- softImpute(x = XNA, lambda = lambda_sft)
X.sft <- sft$u %*% diag(sft$d) %*% t(sft$v)
X.sft[which(!is.na(XNA))] <- XNA[which(!is.na(XNA))]
head(X.sft)
```

```
##            [,1]       [,2]       [,3]       [,4]      [,5]       [,6]
## [1,]  1.5055007 0.67922116  2.7536194  1.8720242 0.6046278  1.3820507
## [2,]  1.0378387 0.92768670  1.2628175  2.5266558 1.8242035  1.2260849
## [3,] -0.6442869 0.47971427 -1.1689235 -1.5794364 0.9841500 -0.1273577
## [4,]  1.3117378 0.65298888  0.4614630  1.5170295 1.6277217  1.0015291
## [5,]  3.1452516 0.48649309  1.9582524 -0.1270187 0.3351728  1.8349234
## [6,] -1.3137655 0.01552382 -0.2932727 -1.5703715 0.0662147 -1.3493665
##            [,7]       [,8]       [,9]      [,10]
## [1,]  1.0987567  0.9848457  1.5025506  0.7575341
## [2,]  1.1180660  1.0312536  1.1698080  0.6551008
## [3,]  0.1504067  0.6584061 -0.2414336  0.2056861
## [4,]  0.6614958  0.9830745  1.9422992  1.1995797
## [5,]  0.1036019 -0.4067036  1.1354030 -0.9115867
## [6,] -0.2420110  0.4353963 -1.2832304  0.6058015
```

mice

The `mice` package implements a multiple imputation methods for multivariate missing data. It can impute mixes of continuous, binary, unordered categorical and ordered categorical data, as well as two-level data. The original article describing the software, as well as the source package (Buuren and Groothuis-Oudshoorn 2011) and example code are available online here.

The **mice** function computes, based on an incomplete data set, multiple imputations by chained equations and thus returns m imputed data sets.

```
mice_mice <- mice(data = XNA, m = 5, method = "pmm") #contains m=5 completed datasets.
#mice::complete(mice_mice, 1) #get back the first completed dataset of the five available in mice_res
```

The main arguments are the following (more details can be found on the help page).

- `data`: the data set with missing values (matrix).
- `m`: number of multiple imputations.
- `method`: the imputation method to use.

In this case the predictive mean matching method is performed. Other imputation methods can be used, type `methods(mice)` for a list of the available imputation methods.

We aggregate the complete datasets using the mean of the imputations.

```
IMP <- 0
for (i in 1:5) { IMP <- IMP + mice::complete(mice_mice, i)}
X.mice  <-  IMP/5  #5 is the default number of multiple imputations
head(X.mice)
```

```
##           X1          X2         X3         X4        X5         X6         X7
## 1  1.5055007  0.67922116  2.7536194  1.3995167 0.6046278  1.3820507  1.0980663
## 2  1.0378387  1.75344397  1.5339583  2.5266558 1.8242035  1.2260849  1.3165080
## 3 -0.6442869 -0.23069669 -1.1689235 -1.5794364 0.9841500 -0.1273577 -0.1871321
## 4  1.6995675  0.65298888  0.4614630  1.5170295 1.6277217  1.0015291  0.6614958
## 5  3.1452516  0.48649309  1.9582524 -0.1270187 0.9833513  1.8349234  0.1036019
## 6 -0.2437526  0.01552382 -0.2932727 -0.1846989 0.0662147 -1.3493665 -0.2420110
##           X8         X9        X10
## 1  1.3150305  1.3774491  0.7575341
## 2  1.9116179  1.1698080  0.6551008
## 3  0.6584061 -0.2414336  0.2056861
## 4  1.1445267  1.9422992  1.1995797
## 5 -0.4067036  0.9515503 -0.9115867
## 6  0.4353963 -1.2832304  0.6058015
```

missForest

The `missForest` package can be used to impute mixed-type data (continuous or categorical data).

The **missForest** function predicts missing values by training a random forest on the observed values of a data matrix. A vignette is available online as well as the original paper (Stekhoven and Buehlmann 2012).

```
forest <- missForest(xmis = XNA, maxiter = 20, ntree = 100)
```

The main arguments are the following (more details can be found on the help page).

- `xmis`: the data set with missing values (matrix).
- `maxiter`: maximum number of iterations to be performed given the stopping criterion is not met beforehand.
- `ntree`: number of trees for each forest.

```
X.forest<- forest$ximp
head(X.forest)
```

```
##              X1         X2         X3         X4        X5         X6
## [1,]  1.5055007 0.67922116  2.7536194  0.6672222 0.6046278  1.3820507
## [2,]  1.0378387 1.24797131  1.3617035  2.5266558 1.8242035  1.2260849
## [3,] -0.6442869 0.08544471 -1.1689235 -1.5794364 0.9841500 -0.1273577
## [4,]  1.0288242 0.65298888  0.4614630  1.5170295 1.6277217  1.0015291
## [5,]  3.1452516 0.48649309  1.9582524 -0.1270187 0.9286106  1.8349234
## [6,]  0.1312395 0.01552382 -0.2932727 -0.5195336 0.0662147 -1.3493665
##              X7         X8         X9        X10
## [1,]  1.1051579  1.0279821  1.2396038  0.7575341
## [2,]  1.1494255  1.4903604  1.1698080  0.6551008
## [3,] -0.3036121  0.6584061 -0.2414336  0.2056861
## [4,]  0.6614958  1.2188459  1.9422992  1.1995797
## [5,]  0.1036019 -0.4067036  0.8643055 -0.9115867
## [6,] -0.2420110  0.4353963 -1.2832304  0.6058015
```

missMDA

The `missForest` package serves to impute mixed-type data (continuous or categorical data).

The **imputePCA** function imputes missing values applying by using principal component methods. The missing values are predicted using the iterative PCA algorithm for a predefined number of dimensions. Some informations are available in the original article (Josse and Husson 2016).

```
pca <- imputePCA(X = XNA, ncp = 2, scale = TRUE, method = c("Regularized","EM"))
```

The main argument are the following (more details can be found on the help page).

- `X`: the data set with missing values (matrix).
- `ncp`: number of components used to to predict the missing entries.
- `scale`: if TRUE, it implies that the same weight is given for each variable.

The single imputation step requires tuning the number of dimensions used to impute the data. We use the function **estim_ncpPCA** which estimates the number of the dimensions using a cross-validation.

```
ncp.pca <- estim_ncpPCA(XNA)$ncp
pca <- imputePCA(XNA, ncp = ncp.pca)
X.pca <- pca$comp
head(X.pca)
```

```
##              X1          X2         X3         X4        X5         X6
## [1,]  1.5055007  0.67922116  2.7536194  1.2726210 0.6046278  1.3820507
## [2,]  1.0378387  1.33062614  1.3160679  2.5266558 1.8242035  1.2260849
## [3,] -0.6442869 -0.05433620 -1.1689235 -1.5794364 0.9841500 -0.1273577
## [4,]  1.1114432  0.65298888  0.4614630  1.5170295 1.6277217  1.0015291
## [5,]  3.1452516  0.48649309  1.9582524 -0.1270187 0.7727156  1.8349234
## [6,] -0.2003831  0.01552382 -0.2932727 -0.1650226 0.0662147 -1.3493665
##               X7         X8         X9        X10
## [1,]  1.26132766  1.2308518  1.2264664  0.7575341
## [2,]  1.35708790  1.3301561  1.1698080  0.6551008
## [3,] -0.06973992  0.6584061 -0.2414336  0.2056861
## [4,]  0.66149581  1.0973240  1.9422992  1.1995797
## [5,]  0.10360187 -0.4067036  0.7301720 -0.9115867
## [6,] -0.24201096  0.4353963 -1.2832304  0.6058015
```

Comparaison on synthetic data

We compare the methods presented above for different percentage of missing values and for different missing-data mechanisms:

- Missing Completely At Random (MCAR) if the probability of being missing is the same for all observations
- Missing At Random (MAR) if the probability of being missing only depends on observed values.
- Missing Not At Random (MNAR) if the unavailability of the data depends on both observed and unobserved data such as its value itself.

The cause of missingness shoud be studied before making a choice of imputation method.

We compare the methods in terms of MSE, i.e.:

$$MSE(X^{imp}) = \frac{1}{n_{NA}} \sum_i \sum_j 1_{X^{NA}_{ij}=NA} (X^{imp}_{ij} - X_{ij})^2$$

where $n_{NA} = \sum_i \sum_j 1_{X^{NA}_{ij}=NA}$ is the number of missing entries in X^{NA}.

Note that in order to evaluate this error, we need to know the true values of the missing entries.

```
MSE <- function(X, Xtrue, mask) {
  return(sqrt(sum((as.matrix(X) * mask - as.matrix(Xtrue) * mask) ^ 2) / sum(mask)))
}
```

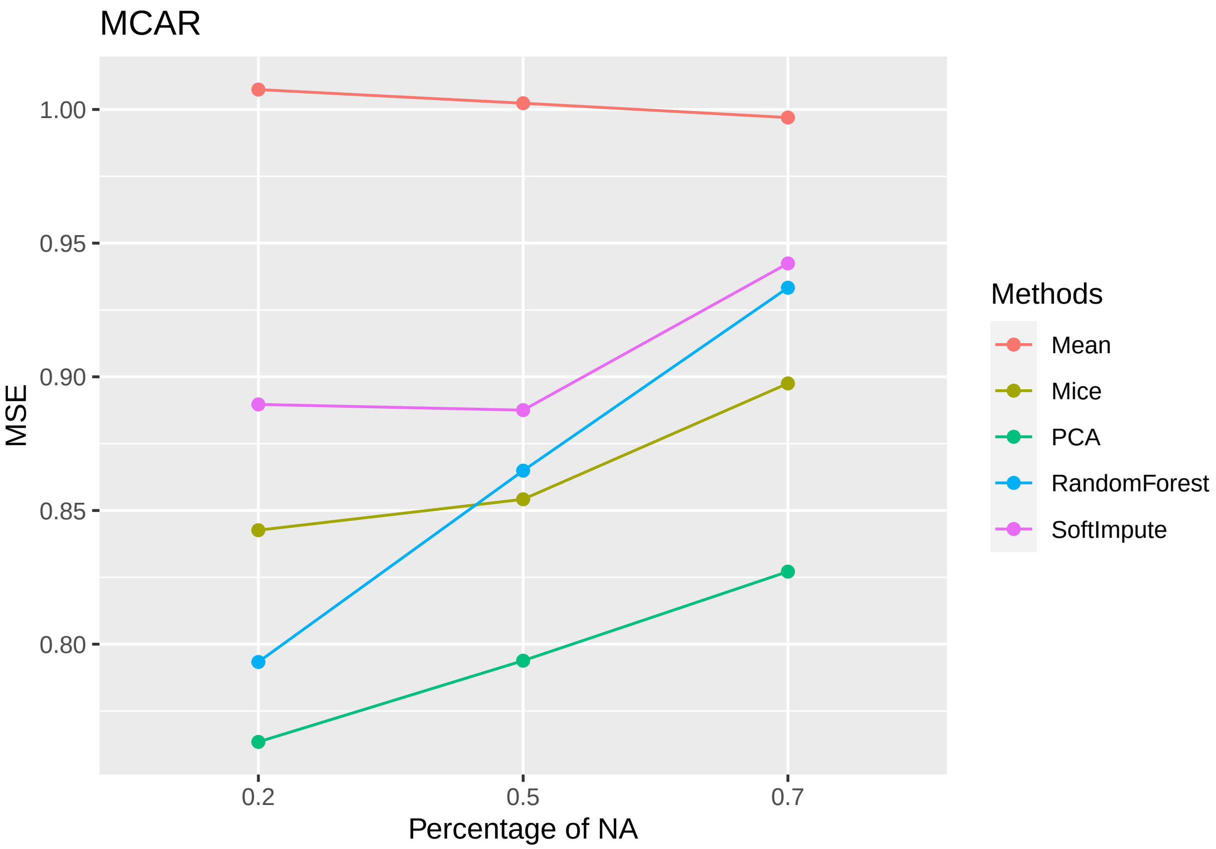
The function **HowToImpute** compares the methods above with the imputation by the mean (the benchmark method). It computes the results (aggregation of the results for several simulations) of the methods for different percentage of missing values and missing-data mechanisms. The arguments are the following.

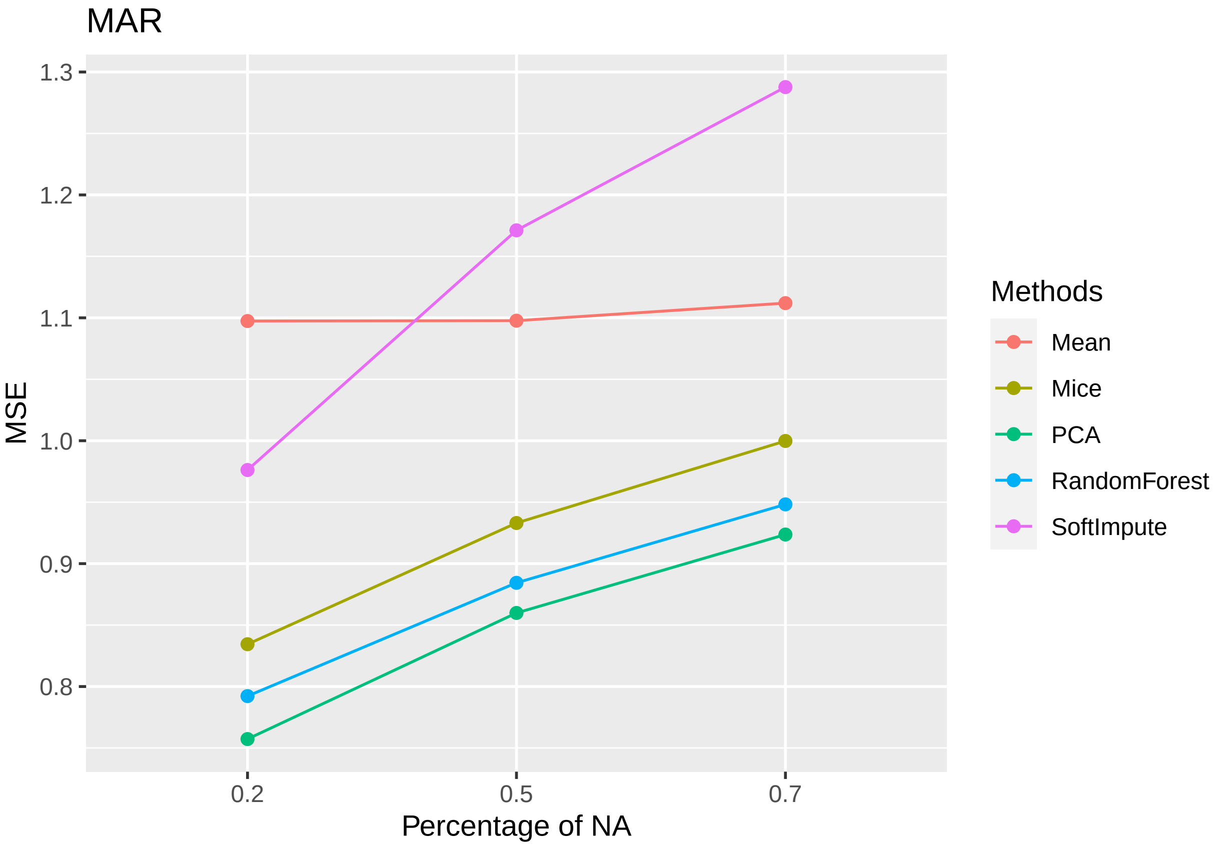
- `X`: the data set with missing values (matrix).
- `perc.list`: list containing the different percentage of missing values.
- `mecha.list`: list containing the different missing-data mechanisms ("MCAR","MAR" or "MNAR").
- `nbsim`: number of simulations performed.

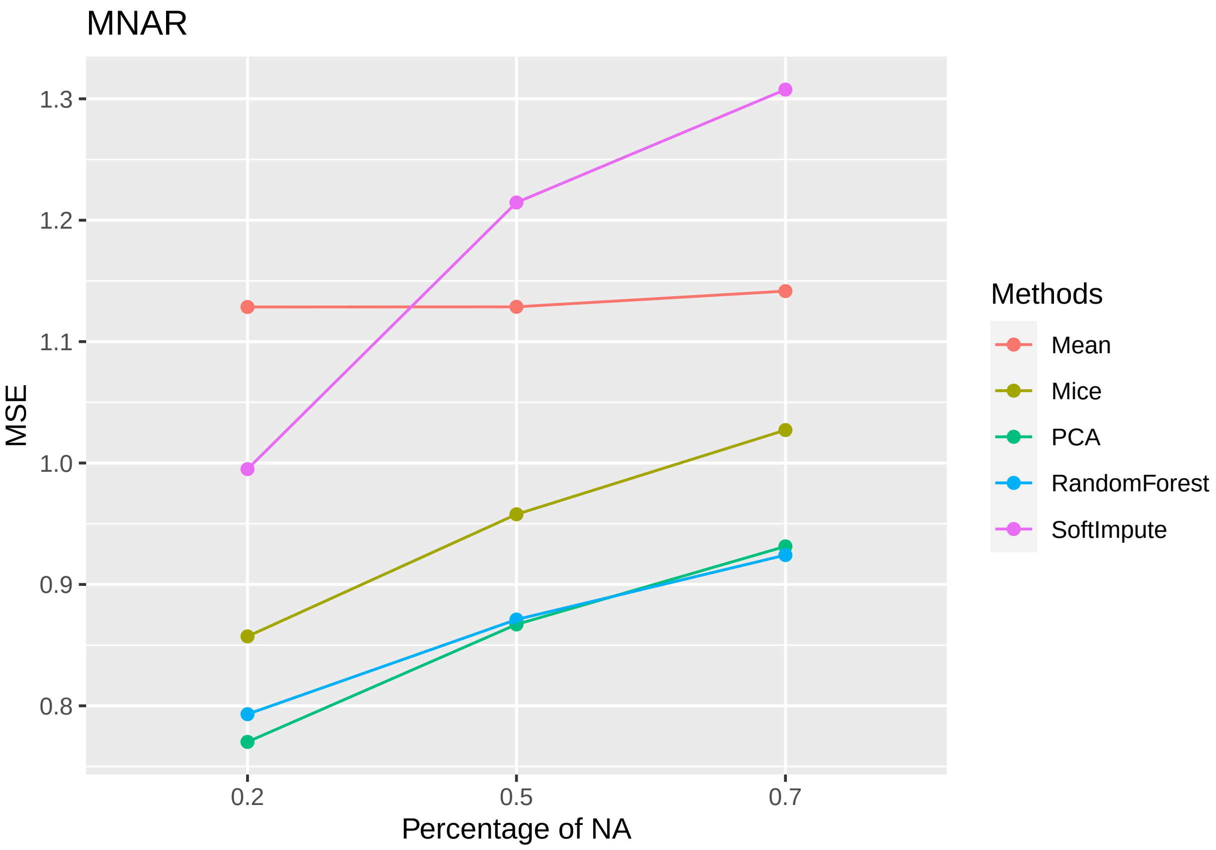
It returns a table containing the mean of the results for the simulations performed.

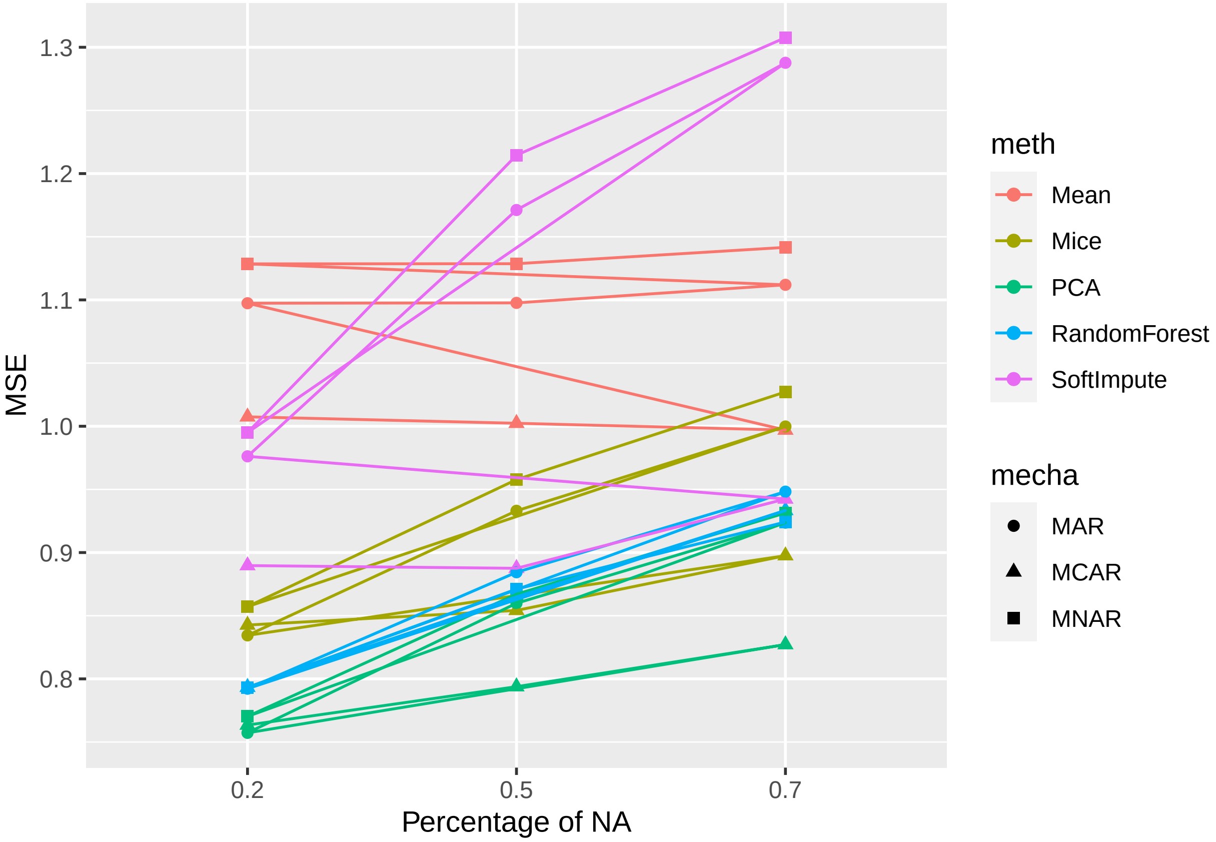
```
perc.list = c(0.2, 0.5, 0.7)
mecha.list = c("MCAR", "MAR", "MNAR")
res <- HowToImpute(X, perc.list = c(0.2, 0.5, 0.7), mecha.list = c("MCAR", "MAR", "MNAR"), nbsim = 2)
```

```
##            0.2 MCAR  0.5 MCAR  0.7 MCAR   0.2 MAR   0.5 MAR   0.7 MAR  0.2 MNAR
## X.pca     0.7634465 0.7938293 0.8271351 0.7572672 0.8598614 0.9236692 0.7702829
## X.forest 0.7933375 0.8648899 0.9333274 0.7922273 0.8843877 0.9482164 0.7931253
## X.mice    0.8426320 0.8541917 0.8975362 0.8344313 0.9330601 0.9997968 0.8572073
## X.soft    0.8896821 0.8875365 0.9424094 0.9761686 1.1711742 1.2877166 0.9949814
## X.mean    1.0074463 1.0023387 0.9969785 1.0973872 1.0976519 1.1119450 1.1285159
##            0.5 MNAR  0.7 MNAR
## X.pca     0.8670415 0.9313367
## X.forest 0.8710677 0.9241768
## X.mice    0.9577506 1.0271624
## X.soft    1.2145881 1.3076324
## X.mean    1.1286550 1.1416066
```


MCAR
MSE
1.00
0.95
0.90
0.85
0.80
0.2
0.5
0.7
Percentage of NA
Methods
Mean
Mice
PCA
RandomForest
SoftImpute


MAR
MSE
1.3
1.2
1.1
1.0
0.9
0.8
0.2
0.5
0.7
Percentage of NA
Methods
Mean
Mice
PCA
RandomForest
SoftImpute


MNAR
MSE
1.3
1.2
1.1
1.0
0.9
0.8
0.2
0.5
0.7
Percentage of NA
Methods
Mean
Mice
PCA
RandomForest
SoftImpute


MSE
1.3
1.2
1.1
1.0
0.9
0.8
0.2
0.5
0.7
Percentage of NA
meth
Mean
Mice
PCA
RandomForest
SoftImpute
mecha
● MAR
▲ MCAR
■ MNAR

Comparison on real data

We will now compare the methods on real data set taken from the UCI repository (Dua and Graff 2017). In the present workflow, we propose a selection of several data sets:

- Seeds (221x7)
- Wine Quality - Red (1599x11)
- Wine Quality - White (4898x11)
- Slump (103x9)
- Movement (360x90)
- Decathlon (41x10)

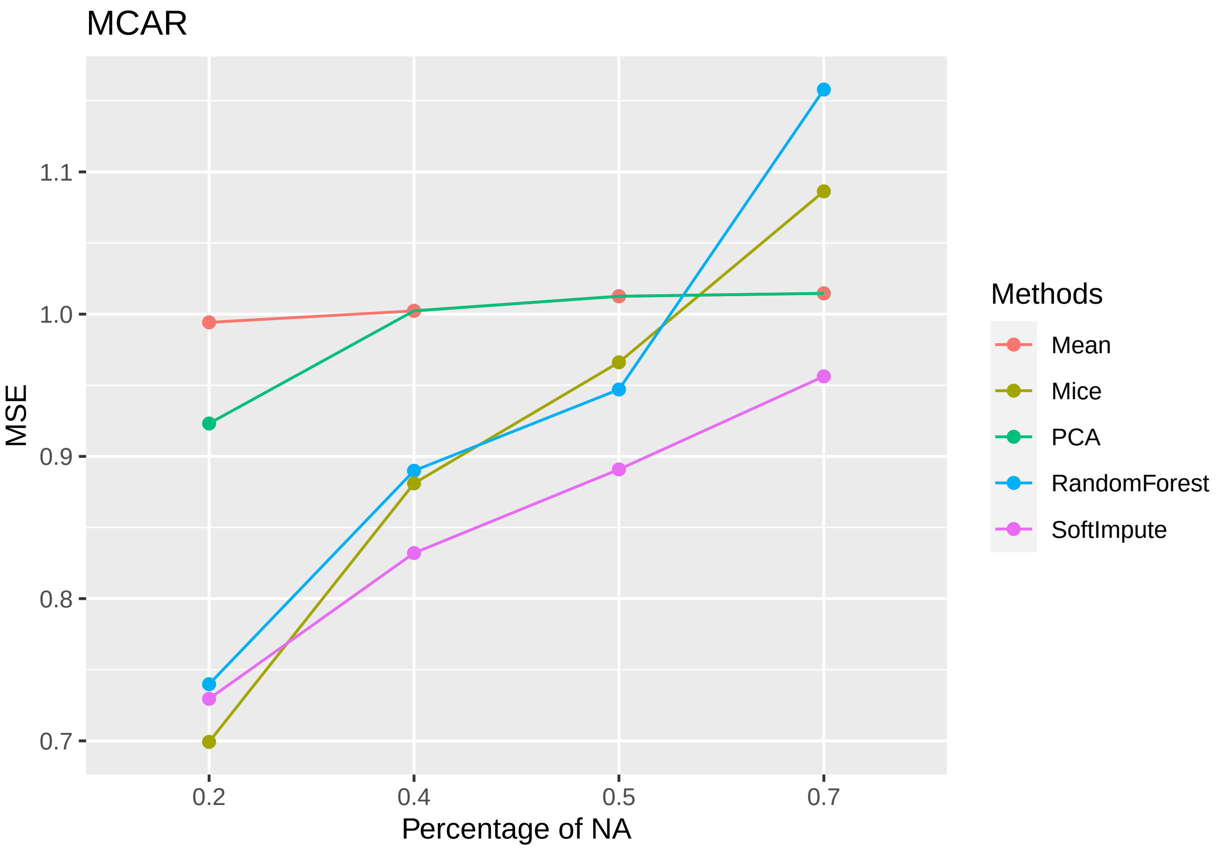
But you can replace the data.frame `don` with any dataset you want to test the methods on.

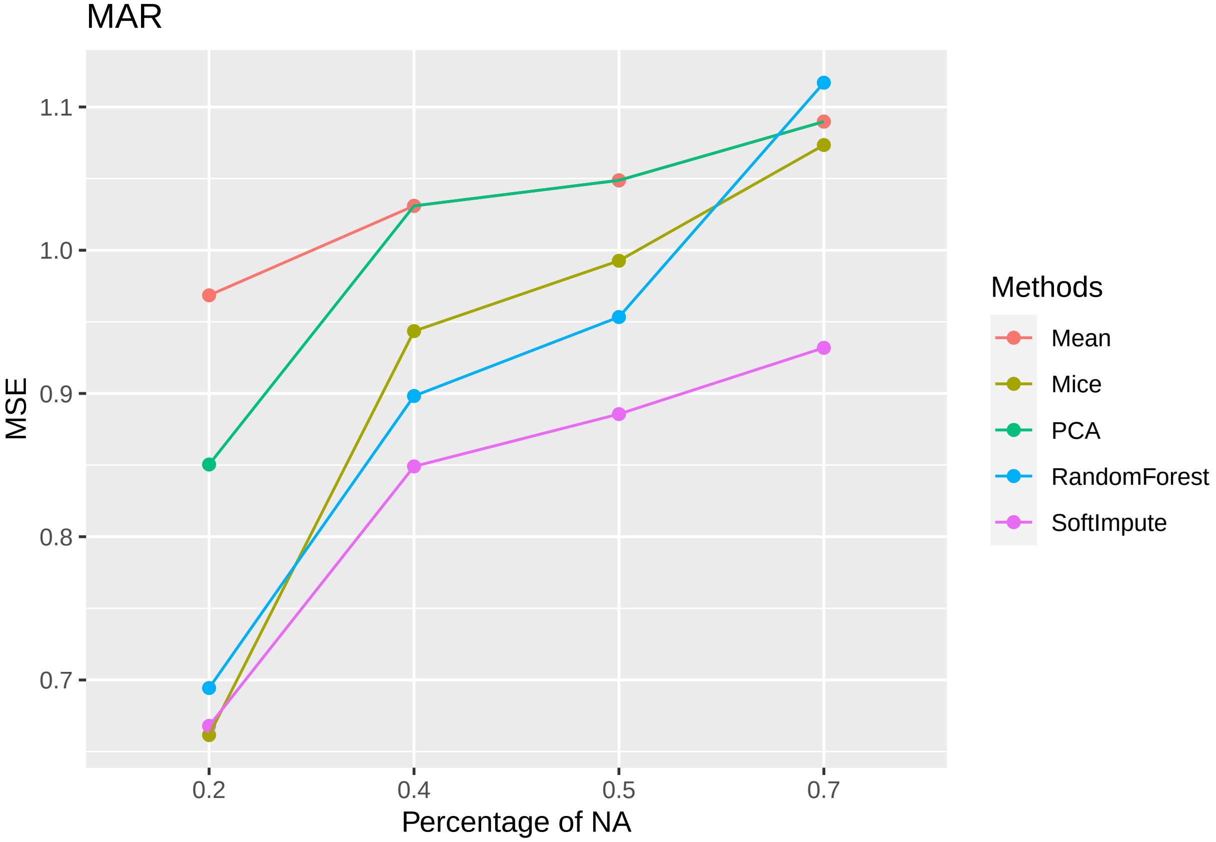
```
name_data <- "slump"

if (tolower(name_data) == "seeds"){
 don <-read.csv("https://archive.ics.uci.edu/ml/machine-learning-databases/00236/seeds_dataset.txt", sep
 don <- don[, -ncol(don)]
}
if (tolower(name_data) == "wine_red"){
  don <-read.csv("https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red
  don <- don[, -ncol(don)]
}
if (tolower(name_data) == "wine_white"){
  don <-read.csv("https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-whi
  don <- don[, -ncol(don)]
}
if (tolower(name_data) == "slump"){
  don <-read.csv("https://archive.ics.uci.edu/ml/machine-learning-databases/concrete/slump/slump_test.da
  don <- don[, -ncol(don)]
}
```

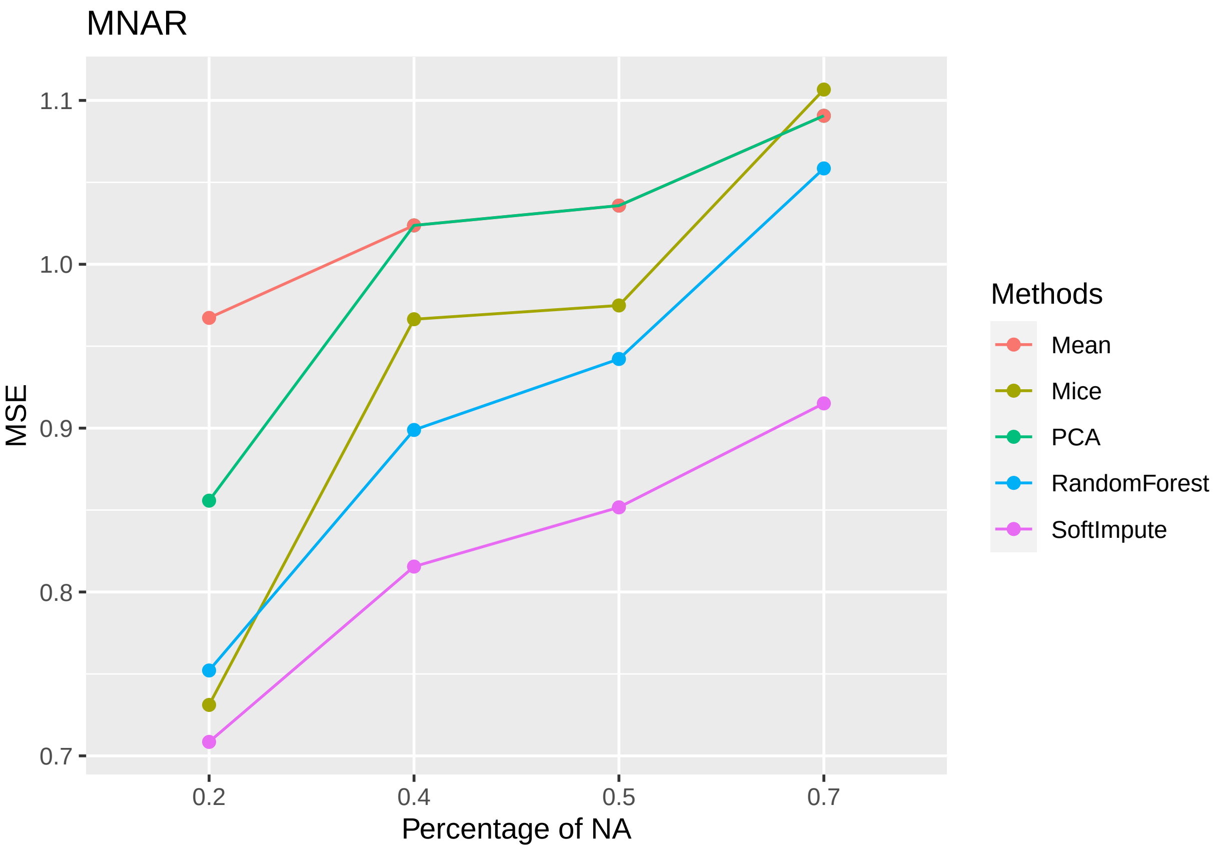
You can choose to scale data prior to running the experiments, which implies that the variable have the same weight in the analysis. Scaling data may be performed on complete data sets but is more difficult for incomplete data sets. Let us keep in mind that it is more realistic to not scale data.

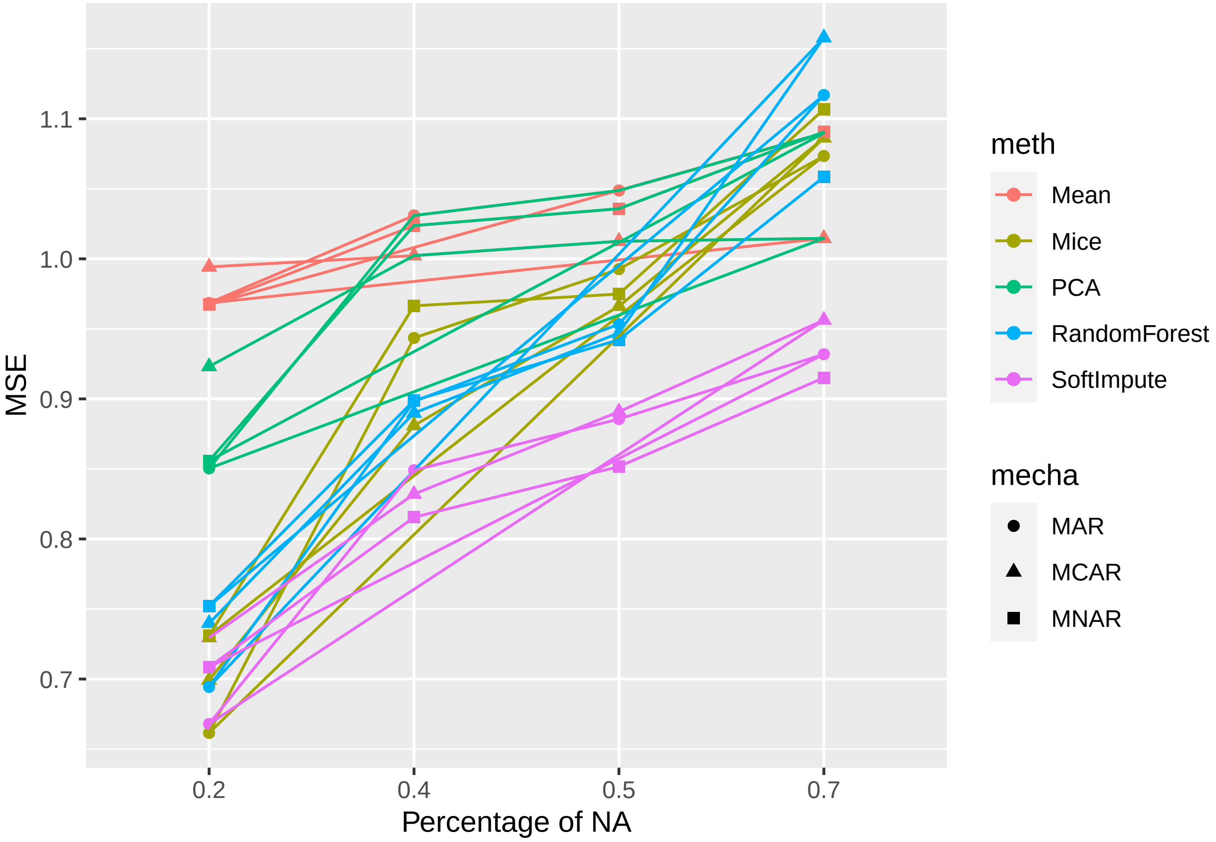
```
scale <- TRUE
if(scale){
  meanX <- apply(don, 2, mean)
  don <- t(t(don) - meanX)
  etX <- apply(don, 2, sd)
  don <- t(t(don)/etX)
}
```

We can then apply the HowToImpute function. The results are presented below.


MCAR
MSE
1.1
1.0
0.9
0.8
0.7
0.2
0.4
0.5
0.7
Percentage of NA
Methods
Mean
Mice
PCA
RandomForest
SoftImpute


MAR
MSE
1.1
1.0
0.9
0.8
0.7
0.2
0.4
0.5
0.7
Percentage of NA
Methods
Mean
Mice
PCA
RandomForest
SoftImpute


MNAR
MSE
1.1
1.0
0.9
0.8
0.7
0.2
0.4
0.5
0.7
Percentage of NA
Methods
Mean
Mice
PCA
RandomForest
SoftImpute


MSE
1.1
1.0
0.9
0.8
0.7
0.2
0.4
0.5
0.7
Percentage of NA
meth
Mean
Mice
PCA
RandomForest
SoftImpute
mecha
MAR
MCAR
MNAR

Session info

```r
sessionInfo()
```

```
## R version 3.6.2 (2019-12-12)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS Mojave 10.14.3
## 
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRlapack.dylib
## 
## locale:
## [1] fr_FR.UTF-8/fr_FR.UTF-8/fr_FR.UTF-8/C/fr_FR.UTF-8/fr_FR.UTF-8
## 
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base     
## 
## other attached packages:
##  [1] mltools_0.3.5       gdata_2.18.0        ggplot2_3.3.0      
##  [4] tidyr_1.0.2         dplyr_0.8.5         softImpute_1.4     
##  [7] Matrix_1.2-18       MASS_7.3-51.4       missMDA_1.16       
## [10] missForest_1.4      itertools_0.1-3     iterators_1.0.12   
## [13] foreach_1.4.8       randomForest_4.6-14 mice_3.8.0         
## [16] Amelia_1.7.6        Rcpp_1.0.3         
## 
## loaded via a namespace (and not attached):
##  [1] gtools_3.8.2         tidyselect_1.0.0     xfun_0.12           
##  [4] purrr_0.3.3          lattice_0.20-38      colorspace_1.4-1    
##  [7] vctrs_0.2.4          generics_0.0.2       htmltools_0.4.0     
## [10] yaml_2.2.0           rlang_0.4.5          pillar_1.4.3        
## [13] withr_2.1.2          foreign_0.8-72       glue_1.3.1          
## [16] lifecycle_0.2.0      stringr_1.4.0        munsell_0.5.0       
## [19] gtable_0.3.0         mvtnorm_1.1-0        codetools_0.2-16    
## [22] leaps_3.1            evaluate_0.14        labeling_0.3        
## [25] knitr_1.27           doParallel_1.0.15    parallel_3.6.2      
## [28] broom_0.5.5          backports_1.1.5      scales_1.1.0        
## [31] flashClust_1.01-2    scatterplot3d_0.3-41 farver_2.0.3        
## [34] digest_0.6.25        stringi_1.4.6        ggrepel_0.8.2       
## [37] grid_3.6.2           tools_3.6.2          magrittr_1.5        
## [40] tibble_2.1.3         cluster_2.1.0        crayon_1.3.4        
## [43] FactoMineR_2.3       pkgconfig_2.0.3      data.table_1.12.8   
## [46] assertthat_0.2.1     rmarkdown_2.1        R6_2.4.1            
## [49] nlme_3.1-142         compiler_3.6.2
```

References

Buuren, Stef van, and Karin Groothuis-Oudshoorn. 2011. “mice: Multivariate Imputation by Chained Equations in R.” *Journal of Statistical Software* 45 (3): 1–67. http://www.jstatsoft.org/v45/i03/.

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.

Hastie, Trevor, Rahul Mazumder, Jason D Lee, and Reza Zadeh. 2015. "Matrix Completion and Low-Rank Svd via Fast Alternating Least Squares." *The Journal of Machine Learning Research* 16 (1). JMLR. org: 3367–3402.

Josse, Julie, and François Husson. 2016. "missMDA: A Package for Handling Missing Values in Multivariate Data Analysis." *Journal of Statistical Software* 70 (1): 1–31. doi:10.18637/jss.v070.i01.

Stekhoven, Daniel J., and Peter Buehlmann. 2012. "MissForest - Non-Parametric Missing Value Imputation for Mixed-Type Data." *Bioinformatics* 28 (1). Oxford Univ Press: 112–18.